# Determining vulnerability resolution time by examining malware proliferation rates

Jeremy D. Seideman
The Graduate School and University Center
City University of New York
New York, USA
Email: jseideman@gc.cuny.edu

Bilal Khan
Dept. of Math & Comp. Science
John Jay College, CUNY
New York, USA
Email: bkhan@jjay.cuny.edu

Ghassen Ben Brahim
Computer Science Dept.
Prince Mohamed Univ.
Al-Khobar, Saudi Arabia
Email: gbrahim@pmu.edu.sa

***Abstract*—One of the ways that malware infects is by exploiting weaknesses in computer systems, often through conditions in software. When this happens, software and operating system vendors must repair these vulnerabilities by patching their software. However, vendors can release patches but cannot force users to apply them. Malware attempts to proliferate without regard to the state of the infected system; it is only once that the malware infection is stopped that we can truly say that systems are patched to eliminate that exploit. By examining appearance and disappearance of malware types, as determined through dynamic analysis of malware samples, classified by behavioral profiles correlated with a timeline of discovery dates, we can determine a more real-world average time for effective patch times, as opposed to the time it takes for a vendor to release a patch for a discovered vulnerability.**

***Keywords*—Malware, Patch Time, Vulnerability Resolution, Malware Emergence, Malware Trends**

## I. Introduction

In the wild, malicious software, or *malware*, spreads throughout target systems through many methods, and often those methods are chosen by the malware writer to take advantage of some vulnerability in the target operating system. Various vulnerabilities often crop up in operating systems and require patches in order to protect users from their effects. It becomes the responsibility of the operating system manufacturer to maintain the security of their products (in the case of a FOSS[1] operating system such as GNU/Linux, this support may be provided by the community or by a software distributer), although this often can take some time to actually happen. The fact that some manufacturers lag behind in their patching has caused the internet security community to implement standards in the reporting of vulnerabilities, in order to give manufacturers a chance to patch those vulnerabilities before giving the malware community a chance to exploit them. Even with standards and frameworks in place, the fact that tools such as metasploit [1] continue to be useful indicates that even with patches available, many operating systems remain vulnerable.

The race between malware and vulnerability exploit writers and software vendors has created a large amount of business for both those malware writers, security firms who identify them and attempt to disclose them and the software developers who must then rectify problems [2]. It is in the overall best interest for everyone (aside from the malware writers, of course) that vulnerabilities are identified and patched, and for those patches to be applied in a timely manner. Therein lies the problem – while patches can be released to eliminate vulnerabilities in software, there is no real way to force those patches to be applied, thus leaving systems vulnerable to more attacks.

This paper is organized as follows. Section II discusses the motivation behind this research. Section III describes our method of acquisition and analysis. Section IV discusses the results of our analysis. Section V explores what the results mean to the software industry, and shows the possibility for further work using our technique.

## II. Motivation

There are several different definitions of what constitutes a vulnerability, based on several factors, including whether or not the vulnerability is intentional or whether it is caused by program error, human error, or any other of a variety of factors, and how that factor can lead to attack [2], [3], [4], [5], [6], but for our purposes, we will define a vulnerability as *a condition by which an attack can be made, deliberately, on a computer system.* Specifically, we are examining vulnerabilities in software[2]. Microsoft has performed detailed analysis of industry-wide vulnerability discovery and disclosure, and has noted that while application vulnerability (i.e. an attack vector that relies on a specific software package that is installed on the computer) has decreased over the past three years, operating system vulnerability (i.e. an attack vector that exploits weaknesses in the underlying OS architecture) has remained fairly constant [7]. However, it has also been shown that while OS vulnerabilities tend to be patched faster, application vulnerabilities tend to have a longer response time [3] and that the number of discovered vulnerabilities has increased rapidly, although this could be due to increased software publishing and a decline in quality as a result in the increased pace of the software market [8].

There are various statistics regarding how quickly a vulnerability can be patched. Arora *et al.* [5] show that some vulnerabilities are actually patched before being announced, and some are patched later, and that patch times range from approximately 101 days to approximately 126 days, depending

[1]Free, Open Source Software

[2]While hardware vulnerabilities can exist inasmuch as there is a certain level of software within the hardware, this is beyond the scope of this research.

978-1-4673-2480-9/13/$31.00 ©2013 IEEE

on how they are disclosed. Further research by Arora *et al.* [8] shows that time to disclosure of a vulnerability can affect the time it takes for a patch to be released, and that this time can be reduced even further through market competition and pervasiveness of the vulnerable product. Sharma *et al.* show that patches are released faster in open source operating systems than in Windows [3].

Research has shown a definite timeline of how a vulnerability can be exploited, which tracks vulnerability from when it is developed until it is effectively dead, whether this is through a patch, attrition, or lack of use [2]. The timeline goes from the actual creation of the vulnerability during development, through disclosure and correction, as well as the eventual death. The vulnerability can be tracked and dated, and may or may not be known in secret, to the public, or readily exploitable [6]. This leads to different types of disclosure – a vulnerability can be made known to the world at large, or may first be disclosed to the software manufacturer to correct. Often this is the ideal case, so that the solution can be distributed in a timely manner [9] and be given the appropriate analysis by experts. The need for appropriate response is so great that recommendations were made by a dedicated task force to the United States Government regarding the disclosure and life cycle of vulnerabilities [4].

Arora *et al.* go on to show that a published but unpatched vulnerability will increase in the number of attacks on hosts as time increased, while a published but patched vulnerability will have relatively few attacks upon those hosts, remaining relatively flat. Overall the life cycle of a vulnerability shows that attacks will rise before publication, and will peak shortly before a patch is released [5]. Based on this, we can assume that a vulnerability will lead to an increase of malware propagation that takes advantage of behavior that will affect the vulnerable portions of software, and that a patch will lead to a drop in those attacks, as that patch becomes more prevalent in the software ecosystem.

The aim of this research, therefore, is to examine how malware propagation and survival is related to patch times. We posit that, despite relatively rapid response times from OS manufacturers to patch vulnerabilities, malware proliferation and infection rates will show that these patches are used and installed rather slowly and that malware writers can continue to exploit known vulnerabilities for quite some time.

## III. Method

Our analysis of patching times involved looking at a corpus of malware and the discovery date of each sample therein. The discovery date was key in determining the amount of time that was involved in the patching process, as it allowed us to construct a model of malware emergence rates.

### A. Acquisition

There are many ways to access a collection of malware samples for research. Aside from setting up an on-network collection, such as with Nepenthes [10] or Dionaea [11], or with a Honeypot [12], there also exist large-scale databases of malware samples that have already been collected and labeled. Examples include Offensive Computing [13] and VX Heavens [14]. As we required a set of known malware to perform an historical analysis, we selected VX Heavens since it provides both a full database of samples and a snapshot of the database that can be accessed in a single download. Those samples have been labeled by Kaspersky Antivirus (KAV).

Symantec [15] provides a large amount of information regarding malware within its database, known as Threat Explorer [16]. The data collected from the database included discovery dates, equivalent names, technical descriptions, and other identifying information. However, these information pages were indexed by the name that each sample was assigned by Symantec. The VX Heavens corpus, however, was indexed by KAV names and we therefore needed to map one to the other. Using a custom email server and submission engine, we were able to send malware samples to VirusTotal [17], an internet-based service that allows for simultaneous parallel scanning of files. The report generated by VirusTotal contains the detected name for each AV engine used to scan a file, so when submitting a known malicious sample, the result report describes which AV engines picked up the sample as malicious and what that engine labeled the sample. While some of the scanners used by VirusTotal may not have been able to detect some samples, particularly newer ones, this did not have an effect on our data collection since we were using known and labeled malware for our corpus.

With the collection of reports, we were able to map each sample to its applicable Symantec name and other information that includes alternate names and discovery dates. While that other information may be useful in other analyses, we simply collected it for the time being.

### B. Reduction

Since we are examining patch times based on malware property emergence, it was crucial that we had a corpus of malware that we were able to track on a timeline. For this reason, we stripped out any samples for which we could not determine a Symantec name equivalent (thus preventing us from determining a discovery date) or for which no discovery date was listed. This left us with 32,573 malware samples to analyze.

### C. Dynamic Analysis

The Norman Sandbox [18] is a software package that allows for dynamic analysis of malware samples (i.e. it records system side effects that can only be viewed through sample execution [19]). Using Sandbox, the entire corpus of dated malware was analyzed and recorded, and we then grouped the properties recorded into one of eleven categories. Any sort of behavior within one of those categories (e.g. Registry Read) would indicate that the categorized behavior was present (in the case of the example, Registry activity). We then aggregated the data into a report that, for each sample, included the sample's name, Symantec name and discovery date, and a Boolean value of whether or not a property class was exhibited. Some other file metadata was included, such as file size, but was not considered in this study.

### D. Emergence of Malware

In order to see how each property emerged into the wild, we examined our corpus, and calculated, by date, when each

property appeared. Instead of just looking at the individual dates of first appearance, though, we instead looked at the number of samples that exhibited a property within a specific time frame. By comparing this number to the total number of samples that appeared within that time frame, we were able to assemble a picture of how many of the malware samples that appeared during that time frame exhibited those properties. An example of this is in Figure 1. This particular graph examines malware that causes changes to the Windows Registry. In that figure, we are looking at samples that appeared within a particular year. Essentially this means we are, on December 31 of any given year, looking at all malware samples that were released in that year.

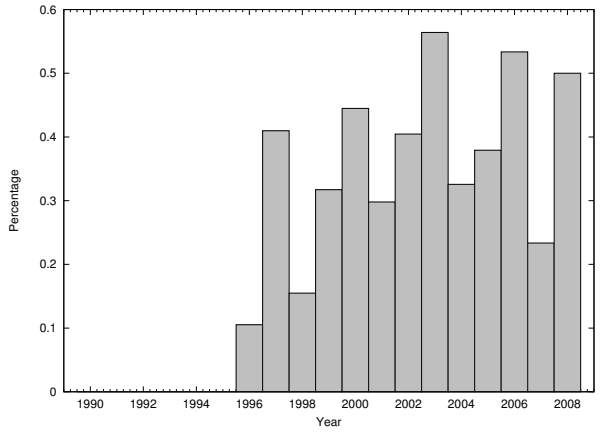


Fig. 1. Percentage of malware emerging per year that exhibits changes to the registry.

We felt that we could provide a better picture of the state of malware development if we were to provide a "sliding window" image of malware emergence – instead of looking solely at the malware that appeared within a year, we instead provided a moving rate of emergence of each property. This better allowed us to see how the landscape of malware changed over time – we could see how the rate changed, based on a preceding amount of time. Figure 2 is an example of a sliding window graph, with a window size of 365 days. In other words, we are looking at a moving sequence of malware that emerged within the previous year (as the window size was 365 days) at each date, as opposed to an instantaneous figure at the end of each year.

### *E. Determination of Appearance of Vulnerability and Patch*

In order to refine the rates of change that we saw in terms of malware emergence, we then took the derivative of each emergence curve, based on a rate of change of 30 days, giving us the graph in figure 3. We took the derivative of each curve in order to see how the emergence rates changed with each new sample. We were concerned with how the derivative curve crossed the 0 line, moving from positive to negative or vice-versa, showing us how the emergence rate was changing.

Using the derivative curves, we then applied an algorithm we designed to determine the peak of each malware emergence cycle, and the trough of that cycle. By using this algorithm, we were able to eliminate any ambiguity in terms of the curves –

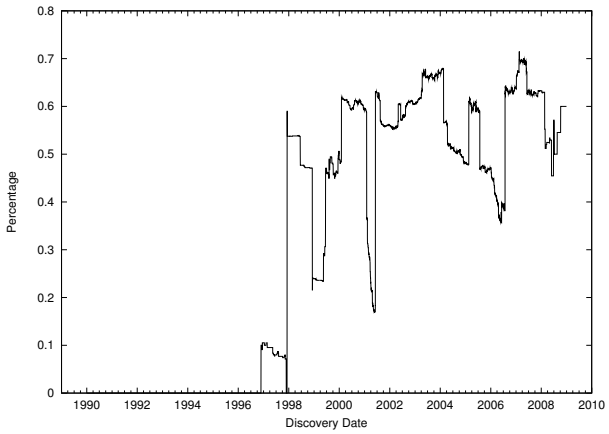


Fig. 2. Percentage of malware emerging over the previous 365-day period that exhibits changes to processes.

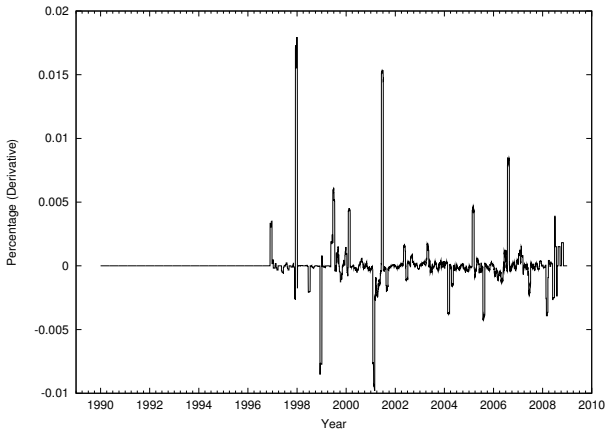


Fig. 3. Derivative of Sliding Window percentage curve for malware exhibiting process changes.

we were able to define and determine the exact dates for the start and end of each cycle so that we can determine how long that cycle actually was.

*a) Determination of a threshold value.:* The first thing we did was determined a threshold value, $T$, the average of the absolute values of the peaks and troughs, which we used as a cut-off value. As the graphs may have small peaks or small troughs, we wanted to eliminate the "blips" in the curves that would represent small-scale infections. Rather, we were interested in what we can safely determine: the emergence of malware that exploits a vulnerability.

*b) Isolation of relevant data points.:* Once we have determined our $T$, we then reset our values such that, for each date $x$, we set the data point to either 1, 0, or $-1$, based on the value of $f(x)$ at that date with respect to $T$:

$$f_T(x) = \begin{cases} -1 & \text{if } f(x) \leq -T \\ 0 & \text{if } |f(x)| < T \\ 1 & \text{if } f(x) \geq T \end{cases}$$

Leaving us with a set of definite peaks and troughs, ordered

by date. At this point, though, we do not have the start of each, but rather the date range where each peak or trough plateaus.

*c) Determination of date interval endpoints.:* Next, we take the peaks and troughs and filter them to determine the start and end of each. For each maximal sequence of $n$ $1's$, we replaced it with a 1 followed by $n-1$ $0's$. Similarly, we did the same for the troughs, looking for sequences of $-1's$ instead. Now, we have a start date for each peak and trough.

*d) Identification of Peak and Trough.:* Finally, we found the time interval represented by each peak-trough cycle. Discarding all dates where $f_T(x) = 0$, we look at sequences of $1, -1$, and determine the time difference between the two. We can then record this time difference, and use it to calculate average time intervals between peaks and troughs. We can also create a histogram of the frequencies of the time intervals. Figure 4 is a graphical representation of the intervals between a peak and a trough. In that figure, "Interval Point" refers to the value of $f_T(x)$, which was set and filtered earlier in our analysis.

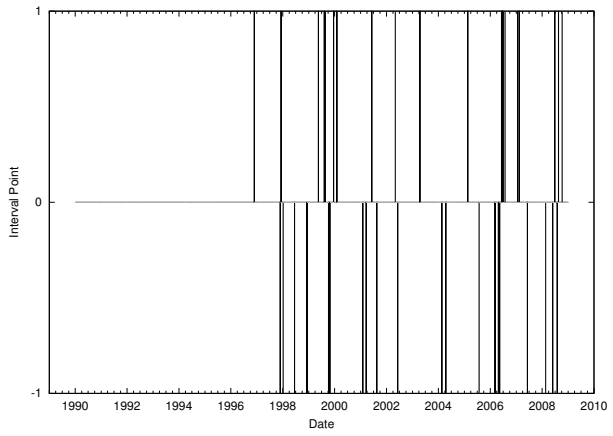


Fig. 4. Endpoints of patch times for malware exhibiting changes to processes.

## IV. Results

Each peak–trough cycle represents an *outbreak* – a rise and then fall in malware exhibiting a specific characteristic or behavior. Looking at previous vulnerability patch models as mentioned earlier, we can correlate the peak with the disclosure of a vulnerability, and the trough with the patch for the vulnerability being released and installed by the majority of the user base, thus limiting the vulnerability's exploit potential. Examining the patch times we derived, we can then graph the frequencies we observed. Figure 5 shows the frequencies of patch times for outbreaks of malware which exhibit changes to the file system. The frequency shown is the absolute number of each patch time as it occurred in our analysis.

We were able to create a histogram for each type of behavior that we examined. The set of histograms we created is summarized in Table I which shows, for each type of behavior, the number of outbreaks that were found, and the average length of each. The average length of the outbreak is determined by the distances between the peaks and troughs of the emergence rates of malware that exhibit that behavior.

We see that some types of behavior (e.g. file system changes, registry changes, and network activity) take much longer to stop acting as exploitable vulnerabilities, while others (e.g. network services, file infectors and setting changes) are seemingly patched up faster. There are many reasons this could be, such as the fact that some vulnerabilities might not lead to obvious system problems. Exploits targeting those vulnerabilities would be more "hidden" from the user, thus reducing the probability that a user would act upon a security issue immediately. On the other hand, some vulnerabilities may cause the user to seek out and apply an update or a patch since the attack up on their system may interfere more visibly with their daily usage.

We have also noticed a large amount of variability among the lengths of the outbreaks. This may be due to the fact that variable conditions within computer systems may prevent some infections from taking hold, thus limiting the amount of similar malware that may be created. Alternatively, this may again be due to the fact that malware effects may have differing levels of visibility to computer users, thus changing the time until an impact is noticed that causes a user to apply a patch or update.

| Characteristic | Outbreaks | Average Patch Time | $\sigma$ |
|---|---|---|---|
| File System Changes (%) | 10 | 145.80 | 138.22 |
| Registry (%) | 9 | 158.67 | 139.23 |
| Network (%) | 9 | 200.56 | 189.07 |
| Security (%) | 7 | 82.71 | 68.93 |
| Process (%) | 10 | 152.10 | 133.56 |
| Setting Changes (%) | 12 | 89.42 | 90.89 |
| Network Services (%) | 8 | 176 | 189.84 |
| Spread via E-mail (%) | 2 | 90.50 | 42.5 |
| Spread via LAN or WAN (%) | 5 | 158.60 | 132.18 |
| Spread via File Infector (%) | 5 | 52.20 | 51.32 |
| Spread via P2P (%) | 6 | 94.83 | 69.13 |

TABLE I. Average times for patches to become pervasive.

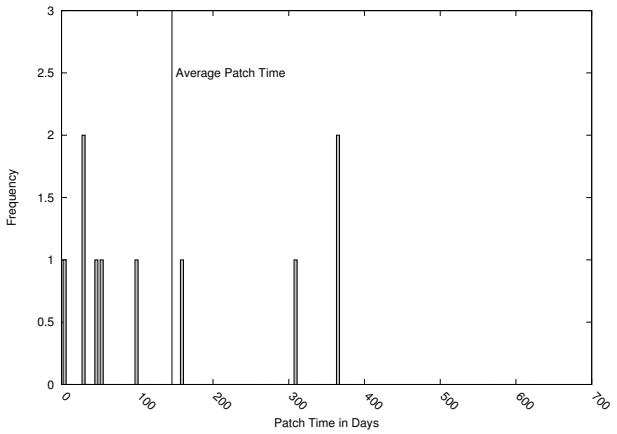


Fig. 5. Histogram of patch times for malware exhibiting file system changes.

## V. Conclusion and Future Work

We have presented a method by which, through the examination of the emergence rates of malware, we can determine the beginnings and endings of outbreaks of malware that exploit various characteristics of computer systems. Our method relies on a classification of malware based on the type of behavior that it exhibits. Since multiple malware samples can be released that exploit the same vulnerabilities and therefore

exhibit similar behavior, our method uses the emergence rates of these groups and their eventual decline to define those start and end points.

Despite that some patches can be released rapidly in response to discovered vulnerabilities, our results show that some of these patches are adopted and installed slowly. In other words, this means that despite there being a solution to a vulnerability, machines are still vulnerable and therefore are possible targets. Looking at the landscape as a whole, even though individual machines are immune to certain infections, it can be said that the population as a whole is still vulnerable. This is analogous to biological disease – someone may be immune to a disease but that disease is not "eradicated" as long as the population remains vulnerable. Unfortunately, as previously stated, there is no real way to "force" people to install updates to computers. While operating systems can be designed to automatically apply updates in the default case, a user can always change a setting and prevent that from happening. Additionally, if a computer is already infected, then further updates may not be applied as a side effect of the infection.

The information we have gathered regarding patch cycles can be used to further our study of malware, especially within a single family or related to a single vulnerability type. Using our algorithm, we can determine a vulnerability life cycle and then we can examine a peak-trough cycle by date. We can then determine if the samples within that cycle are, in fact, members of the same malware family (there are several ways to determine if the samples are part of the same family). This sort of study could be done for any behavior, vulnerability, or malware type, to help us see the type of infections that are prevalent and what type of vulnerabilities they exploit, hopefully leading to techniques that encourage better software development and operating system patch habits.

## REFERENCES


[1] "Metasploit," 2012, http://www.metasploit.com.

[2] A. Cencini, K. Yu, and T. Chan, "Software Vulnerabilities: Full-, Responsible-, and Non-Disclosure," 2005, http://www.cs.washington.edu/education/courses/csep590/05au/whitepaper_turnin/software_vulnerabilities_by_cencini_yu_chan.pdf.

[3] G. Sharma, A. Kumar, and V. Sharma, "Windows operating system vulnerabilities," *International Journal of Computing and Corporate Research*, vol. 1, no. 3, 2011, http://www.ijccr.com/November2011/13.pdf.

[4] J. T. Chambers and J. W. Thompson, "National Infrastructure Advisory Council Vulnerability Disclosure Framework: Final Report and Recommendations by the Council," 2004, http://www.dhs.gov/xlibrary/assets/vdwgreport.pdf.

[5] A. Arora, R. Krishnan, R. Telang, and Y. Yang, "Impact of vulnerability disclosure and patch availability - an empirical analysis," in *In Third Workshop on the Economics of Information Security*, 2004, http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.81.9350.

[6] A. Ozment, "Improving vulnerability discovery models," in *Proceedings of the 2007 ACM workshop on Quality of protection*, ser. QoP '07. New York, NY, USA: ACM, 2007, pp. 6–11, http://doi.acm.org/10.1145/1314257.1314261. [Online]. Available: http://doi.acm.org/10.1145/1314257.1314261

[7] "The evolution of malware and the threat landscape – a 10-year review: key findings," 2012, http://download.microsoft.com/download/1/A/7/1A76A73B-6C5B-41CF-9E8C-33F7709B870F/Microsoft_Security_Intelligence_Report_Special_Edition_10_Year_Review_Key_Findings_Summary.pdf.

[8] A. Arora, C. M. Forman, and R. Telang, "Competitive and strategic effects in the timing of patch release," 2005, http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.79.6891.

[9] "ISC Security Vulnerability Disclosure Policy," 2012, http://www.isc.org/security-vulnerability-disclosure-policy.

[10] "Nepenthes - finest collection," 2010, http://nepenthes.carnivore.it/.

[11] "dionaea – catches bugs," 2012, http://dionaea.carnivore.it/.

[12] "The Honeynet Project," 2010, http://www.honeynet.org.

[13] "Offensive Computing: Community Malicious code research and analysis," 2010, http://www.offensivecomputing.net.

[14] "VX heavens," 2010, http://vx.netlux.org.

[15] "Symantec," 2012, http://www.symantec.com/index.jsp.

[16] "Threat explorer - spyware and adware, dialers, hack tools, hoaxes and other risks," 2012, http://www.symantec.com/security_response/threatexplorer/.

[17] "VirusTotal," 2008, http://www.virustotal.com.

[18] "Norman Sandbox," 2009, http://www.norman.com/technology/norman_sandbox/.

[19] J. Seideman, "Recent advances in malware detection and classification: A survey," The Graduate School and University Center of the City University of New York, Tech. Rep., 2009.