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

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.