The Unpredictable Attacker
Does the "Honeymoon Effect" Hold for Exploits?

Rebecca Gelles
University of Pennsylvania

Project in Collaboration with:
Matt Blaze Sandy Clark

Abstract:

The goal of the research presented in this paper is to expand on research which examines the time it takes for new security vulnerabilities to be discovered after a new release of a computer program. In particular, this research seeks to build upon previous research examining time from release to vulnerability discovery by doing additional research into the time from release to exploitation, in order to test whether the previously discovered pattern holds. To do so, it is necessary to discover when new vulnerability exploits are created, which is no easy task. Unfortunately, because of the nature of exploit data, to this date we have been left only with new questions to answer. As such, the approach of this paper will be to discuss research methods tried thus far, problems encountered, and ideas for continuing research, rather than on data analyses and conclusions.

Introduction:

One of the most difficult problems in the area of computer security is that it is at this time extraordinarily difficult, even impossible, to know whether a program is secure or even whether it is secure enough to be released. As such, program creators must be constantly aware of the possibility of new attacks, and data is always at a major risk of being stolen. This problem also makes it difficult for security specialists to find effective monitoring solutions—if you cannot predict when behavior will occur, you must either be constantly vigilant or risk a successful attack. And constant vigilance can be costly. Thus, knowing how long it takes for vulnerabilities in programs to be discovered and exploited can be a very effective tool in computer security.

For a long time, security vulnerabilities have been assumed to behave in the same manner as other computer bugs, and have thus been approached in the same way. However, recent research indicates that this may not be the case. In fact, where the number of new bugs found in software has been shown to be high initially and decrease as time goes on, it appears that the number of vulnerabilities found in programs show a pattern almost directly opposite that of bugs. That is, initially vulnerabilities are found slowly, but as time goes on more and more are uncovered.

Unlike other bugs, which are themselves inherent problems within software that need to be fixed, vulnerabilities are only problems within the context of exploits: if an exploit that takes advantage of a particular vulnerability can be or has been developed, that vulnerability becomes dangerous. Thus, it seems important to also examine the time it takes for exploits to be discovered, rather than vulnerabilities alone. This raises the question of what the lifecycle is of software with respect to exploits themselves, rather than to vulnerabilities, and the related question of the time link between the

discovery of a vulnerability and the development of an exploit.

One of the most significant problems faced in a task such as this is effective data collection. Gathering data about vulnerabilities is, while not easy, not extraordinarily difficult, as there are websites like BugTraq that carefully record all discovered vulnerabilities. However, gathering data on exploits is inherently difficult because, while some exploits are developed by legitimate employees in the security field trying to test their systems in order to improve them, and others are created by curious individuals who want to know if they can, some exploits are created by criminals, and, as such, are not likely to be shared. This means that gathering data on when they were created is difficult. As a result, one of our goals has been to learn enough about the patterns of exploit-creators to figure out whether the unshared exploits are likely to have been created around the same time as the shared. But this is no easy task, and so even after ten weeks of work and significant amounts of data collection, we have been left with no clear results.

Background:

Since the release of the important book, "The Mythical Man-Month" [2], it has been well-established that program errors are found in high frequency shortly after the release of the program, but see their frequency of discovery decrease as time goes on. Although bugs continue to be found, at a certain point the rate of discovery is deemed low enough for the software to be released. In fact, this model is the basis for Software Reliability models.

However, although for a long time security vulnerabilities were just considered a specific type of bugs, recent research indicates that this may not be the case. In the paper "Familiarity Breeds Contempt: The Honeymoon Effect and the Role of Legacy Code in Zero-Day Vulnerabilities," [1] produced by my mentor, Matt Blaze, and others, the authors argue that vulnerabilities actually display a far different pattern of discovery that other bugs, and that programs experience a "honeymoon period" where no vulnerabilities are found, and then new vulnerabilities start to be discovered gradually, with the pace increasing as time goes on. However, their research in this area is limited in a few ways that new research found in this paper hopes to address: first, that they only consider the first few vulnerabilities discovered, and so it is unclear whether this pattern continues, or whether eventually new vulnerabilities cease being discovered or are discovered at a slower rate; second, that they focus on the time to discover vulnerabilities after release but not the related time to exploit vulnerabilities after release; and finally, that it does not attempt to gather data specifically based on attacker, focusing instead on more easily found data produced by outside observers and legitimate security companies. The research found herein aims to fill in these crucial gaps.

In aid of the final point, it is helpful to gain a more thorough understanding of both how attackers behave and what motivates attacker behavior. To this end, the book "Kingpin," [6] one of the most thorough analyses of the criminal world on the internet, is an especially valuable resource. In particular, it notes how most of the data theft was achieved by just a few criminals and then sold to others around the world, and how the methods of gathering the data used by these criminals were generally the same exploits that they employed for long periods of time, or, when those failed, new exploits taking advantage of the same vulnerabilities. This insight raises

interesting questions, such as whether the work that goes into preventing vulnerabilities is worth the effort. The researcher in "The Honeymoon Effect" [1] found that quality of code appeared to have no impact on the rate of vulnerability discovery, and, in fact, that well-tested legacy code was even more dangerous. This might lead one to wonder whether it is entirely necessary to attempt to produce secure code initially, or if resources should instead be directed towards figuring out effective patches once a vulnerability is known. If it is true that, as in "Kingpin," [6] attackers frequently reuse known attacks and vulnerabilities, making patches which cut off their current method of entry might be more useful than trying to prevent them from finding a new method. Doing otherwise might be similar to building ten more feet of wall around a castle when the invading army got in through the drains still leading in from the outside: it might make you feel secure to prevent the attackers from coming up with a new mode of attack, but they are more likely to stick with the method that they know is effective.

Prediction of attacker behavior can also be enhanced by understanding the general interplay between attacker and defender, even in areas beyond computer security. One paper whose research finds attack-response patterns similar to those seen in computer security is "Pattern in Escalations in Insurgent and Terrorist Activity," [4] which studies, in the context of the military situation in Afganistan, the case in which a less institutionalized "insurgent," which could be considered similar to cyber criminals, adapts more quickly than the organized but less adaptive respondents (here, probably security specialists and law enforcement) and manages to have an advantage despite being apparently weaker.

"The Honeymoon Effect" [1] is not the only paper to examine this problem of discovery rate of security vulnerabilities. In "Milk or Wine: Does Software Security Improve with Age?" [5] researchers examined both when security vulnerabilities were found in OpenBSD and whether these vulnerabilities were found in foundational, "legacy" code. They found that, in general, many of the vulnerabilities were from the foundational code. They posited a number of explanations for this, including that the newer code was better and more secure or that there had simply been more time to examine the old code. However, they also acknowledged that the percentage of vulnerabilities found in legacy code was proportional to the percentage of legacy code still in use as compared to newer code. Still, this result seems initially counterintuitive in the "vulnerabilities are bugs" scenario: presumably, bugs have been found and fixed, and so older code would actually be more secure, as it would have been tested and patched more times than other code. Unfortunately, since this paper focuses exclusively on OpenBSD, it is impossible to know if we can extrapolate these results to the more general scenario, especially considering OpenBSD places an especially strong focus on security.

Data and Analysis:

Since the data gathering was still in process and the data interpretation had barely begun at the time my portion of the project was completed, it is difficult at this point to show what has been found, and what the data show. However, I will endeavor to discuss at this point the data retrieval process and what data has thus far been found.

The first data retrieval process involved both updating the data used in the paper "The Honeymoon Effect" [1] and

rendering future updates unnecessary. The information in question is a list of vulnerabilities in various programs found on the website Security Focus [7]—a website that the researchers in "The Honeymoon Effect" [1] had found to be very useful, as it contained both government-collected data and additional entries collected from other sources. The previous data retrieval mechanism was a two-part web scraper program which first collected the links to each entry on the website, and then collected individual portions of the data from each linked entry. However, the program created a new database each time, rather than simply updating the previous one with new entries, which both disrupts the data interpretation process and takes far too long—the program literally took days to run, as it had to visit over 50000 separate web pages. To solve this problem, my goal was to create a version of the program that simply updated the previous list, so that whenever the program was run the data would be up-to-date. As I had never worked with a web scraper before, figuring out the most effective mechanism to do this was surprisingly difficult.

Next, the goal was to make this updating process occur daily and automatically. For this endeavor, I had to learn how to write a bash script which would run the programs, and then write a cron job which would run the bash script daily. Now, finally, the update process was automated and consistent, and it was time to gather the additional data from new sources needed for the new focus of the research on exploits.

Unfortunately, attackers don't exactly publicize it when they create a new successful exploit, as letting the program creators know that the problem exists would allow them to patch it, cutting off their new source. Luckily, there exists the field of penetration testing, where defenders mimic attackers and seek to exploit security vulnerabilities in a system in order to help programmers figure out what they need to fix. Their behavior may not be exactly like that of normal attackers, but hopefully it will be similar enough to approximate attacker behavior. One good source of data from both penetration testers and another group of people—those who like to break into programs for fun but do not use what they find maliciously—is known as Metasploit, which is a program full of already-created exploits that can be used to get into systems. Fortuitously for our purposes, each Metasploit exploit module includes its date of creation. Even better, it contains the identification number of the vulnerability it exploits, the same number found in the Bugtraq databases. To take advantage of this source, I created another web scraper to pull this data.

Metasploit also contains the date the vulnerability was discovered, which created a new problem: occasionally the Metasploit and Security Focus [7] data disagreed. And due to the form the data was in, it was too difficult to try to compare these automatically, and an automatic comparison might not have provided us with a reason why the sources disagreed. So a manual comparison was necessary instead.

In the course of this research, we noticed that the speed of discovery of the first vulnerability, and first exploit, were both increasing as time went on; that is, the "honeymoon period" was getting shorter. One goal of our research, then, was to figure out why. One posited reason was that there were simply more people working on exploits now than there were previously. In order to make an attempt to test this hypothesis, I created a program to tell me more about the Metasploit module authors: how many there were, how many exploits each had created, when each had started, and

what kind of modules (modules were based on what kind of operating system the vulnerabilities were found in, what the line of attack was, etc.) they worked on. The last turned out to be relatively irrelevant, because for most Metasploit authors who had created more than one exploit, almost all of them had created exploits for multiple modules.

The data found in this process was as follows:

# exploits written	# authors with this total	yr of first	yr of last
>20	4	2005	2011
15-20	1	2011	2011
10-14	2	2007	2011
5-9	13	varies	varies

This is a very rough approximation. Essentially, most of the exploits were written by the same four authors, all of whom had been writing from the creation of Metasploit in 2005 to the present. The only other author with more than 15 exploits had written all of them in 2011. There were two more who had written a significant number, both beginning in 2011. Of those who had written at least five, the threshold I set for whether finding exploits was something they were particularly interested in, and thus whether they were likely to be working on finding exploits for any given program, and as a result have an impact of the rate of exploit discovery, many had begun early but stopped writing, and many had only begun recently. The information obtained was unclear.

There were two particular problems with the data. First, anything created for Metasploit was likely not to entirely reflect the attacker market, both because the Metasploit authors were generally not the same people as the attacker, and because the people writing exploits for Metasploit were writing specifically for Metasploit, and so the number of authors was limited. Second, on Metasploit if an exploit has already been discovered there is no way to tell if another person was discovering it separately. This latter point is important: for example, it is possible some of the authors who appear to have stopped writing may have continued to search for exploits but have gotten slower than others searching, and are thus never fast enough to be the author of new exploits. In addition, the sample size of the data appears to be too small to be useful.

However, looking back at the black market seen in "Kingpin," it is unclear whether these problems are actually real. In particular, "Kingpin" describes a world in which a very small group of attacker churns out most of the exploits that exist. This actually seems to be reflected in the Metasploit data. So perhaps what seem to be problems are just reflections of the nature of the field.

Conclusion:

Unfortunately, at the time my component of this research project was completed, we had not reached any true conclusions, or completed any careful analysis, or even completed the data collection process. Instead, we had simply discovered more questions that needed to be answered. Further steps in the research that need to be taken are a comparison of program release dates to exploit discovery times, a more thorough analysis of the exploit authors data, and an additional attempt to delve into data about the attackers themselves rather than simply a subsection of penetration testers. Hopefully, this additional research will be able to shed light on this highly complicated problem.

Acknowledgements:

I would like to thank the DREU program for making this research experience possible and for helping me connect with my mentor, Professor Matt Blaze. I would also like to thank Carleton College's Robert J. Kolenkow and Robert A. Reitz fund for helping me to even consider such a project, and for providing the initial funding to make the experience possible. I would of course like to thank my mentor for all of his help throughout the summer, and for making sure to choose a project that allowed me to actually feel like I accomplished something despite my relative lack of familiarity with the field, and his graduate student Sandy Clark for helping to answer all of my many questions and provide suggestions and guidance when necessary. I would finally like to thank all of the graduate students in the Distributed Systems Laboratory for being willing to talk to me about anything and making this a really wonderful summer experience.

References:

[1] Matt Blaze, Sandy Clack, Stefan Frei, and Jonathan Smith. Familiarity Breeds Contempt: The Honeymoon Effect and the Role of Legacy Code in Zero-Day Vulnerabilities. In *Proceedings, 26th Annual Computer Security Applications Conference*, pages 251-260, 2010.

[2] Frederick P. Brooks. *The Mythical Man-Month: Essays on Software Engineering, 20th Anniversary Edition*. Addison-Wesley Professional, August 1995.

[3] CVE. Common vulnerabilities and exposures, 2011.

[4] Neil Johnson, Spencer Carran, Joel Botner, Kyle Fontaine, Nathan Laxague, Philip Nuetzel, Jessica Turnley and Brian Tivnan. Pattern in Escalations in Insurgent and Terrorist Activity. *Science,* 333(6038):81-84, July 2011.

[5] Andy Ozment and Stuart E. Schechter. Milk or wine: does software security improve with age? In *USENIX-SS'06: Proceedings of the 15th conference on USENIX Security Symposium*, Berkeley, CA, USA, 2006. USENIX Association.

[6] Kevin Poulson. *Kingpin.* New York: Crown Publishers, 2011

[7] Security Focus. Vulnerabilities Database, 2011.