Journal of Multimedia Information System VOL. 4, NO. 2, June 2017(pp.57-64): ISSN 2383-7632(Online)
http://dx.doi.org/10.9717/JMIS.2017.4.2.57

Extended Linear Vulnerability Discovery Process

HyunChul Joh*,1

Abstract

Numerous software vulnerabilities have been found in the popular operating systems. And recently, robust linear behaviors in software vulnerability discovery process have been noticeably observed among the many popular systems having multi-versions released. Software users need to estimate how much their software systems are risk enough so that they need to take an action before it is too late. Security vulnerabilities are discovered throughout the life of a software system by both the developers, and normal end-users. So far there have been several vulnerability discovery models are proposed to describe the vulnerability discovery pattern for determining readiness for patch release, optimal resource allocations or evaluating the risk of vulnerability exploitation. Here, we apply a linear vulnerability discovery model into Windows operating systems to see the linear discovery trends currently observed often. The applicability of the observation form the paper show that linear discovery model fits very well with aggregate version rather than each version.

Key Words: Software vulnerability, Risk assessment, Linear vulnerability discovery model.

I. INTRODUCTION

Due to the lack of widely accepted standards and definitions in the information technology research area, researchers in the field are frequently confused while doing peer reviews. Since this paper is all about the software vulnerability related materials, first, we are trying to define the word vulnerability.

The software vulnerability is a subset of vulnerability in general, so the software vulnerability should inherit the characteristics what the general vulnerability has. According to the Collins English dictionary (http://www.collinslanguage.com) "Someone who is vulnerable is weak and without protection, with the result that they are easily hurt physically or emotionally." In other words, it represents a susceptibility to malevolent manipulations.

Even though the concept is crystal clear, it is not that simple to define the software vulnerability due to the lack of standards in the field; there's no widely accepted definition for the word currently [1-2]. Yet, there are many definitions proposed, and here are some of them:

- "Security flaws, defects, or mistakes in software that can be directly used by a hacker to gain access to a system or network" [3]
- "Weakness in an information system, system security procedures, internal controls, or implementation that could be exploited or triggered by a threat source" [4]
- "Weakness in the security system which might be exploited by malicious users causing loss or harm" [5]
- "A vulnerable system is an authorized state from which an unauthorized state can be reached using authorized state transitions; a vulnerability is a characterization of a vulnerable state which distinguishes it from all non-vulnerable states" [6]
- "Defect which enables an attacker to bypass security measures"[7]

So far, there are not many literatures discussing the terminology in depth. As one of the early works, Otwell and Aldridge [8] examined the treatment of vulnerability at the 1988 Risk Model Builders' Workshop. They say that defining the word of vulnerability formally is proven to be a complex task while showing the several proposed definitions from the researchers in the workshop. Some of them are:

- "Weaknesses that allow a threat to compromise the security (confidentiality, integrity, or availability) of an asset" [9]
- "Achievable bad events", which "implies that the protections against them are nonexistent, insufficient, or insufficiently protected" [10]

Manuscript received June 12, 2017; Revised June 23, 2017; Accepted June 27, 2017. (ID No. JMIS-2017-0012)
Corresponding Author (*): HyunChul Joh, Room# 06-415, 50 Gamasilgil, Hayangup Gyeongsan Gyeongbuk, South Korea
Dept. of Computer Eng., Kyungil University, 38428 Korea, *joh@kiu.kr*, +82-53-600-5563
1Dept. of Computer Eng., Kyungil University, *joh@kiu.kr*

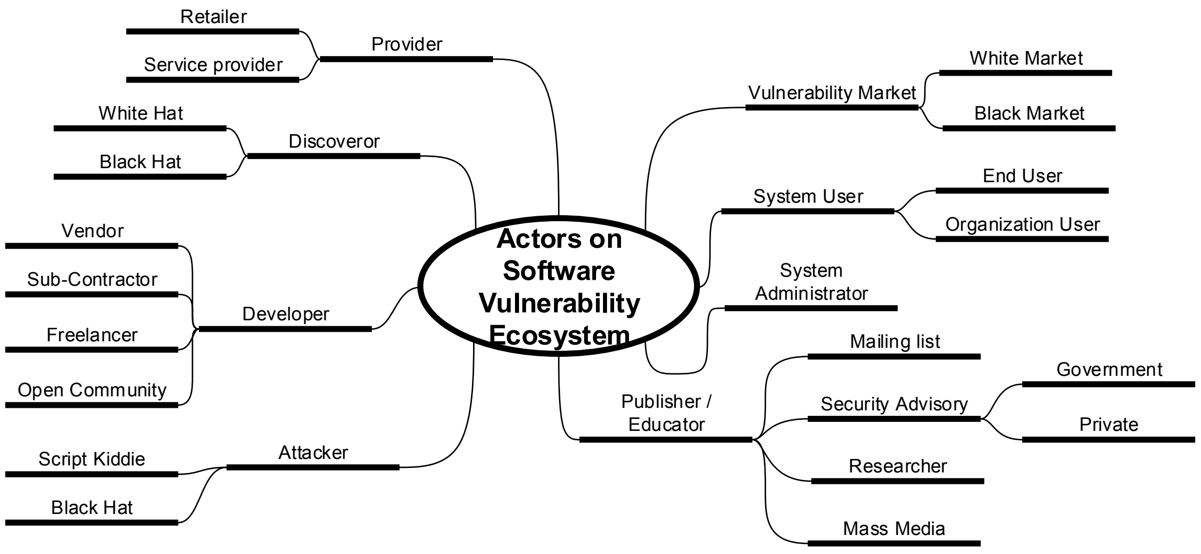

Fig. 1. Major factors influencing on software vulnerability ecosystem

- "The ability of an agent to cause an attack event" [11]

A decade later, in 1998, Krsul [1] defined the software vulnerability in his doctoral dissertation as "an instance of an error in the specification, development, or configuration of software such that its execution can violate the security policy." And another decade later, in 2007, Ozment supports the Krsul's definition with some minor modification; after the modification, the definition is read as "an instance of [*a mistake*] in the specification, development, or configuration of software such that its execution can violate the [*explicit or implicit*] security policy" [2]. Ozment made two changes. The first one is that the *mistake* is used instead of the *error* since in software engineering, an *error* is already defined as "the amount by which the result is incorrect" [12]. The second is that he put the *explicit or implicit* in the modified definition to emphasize the fact that all systems have a security policy whether it is explicit or not.

Meanwhile, after showing the definitions, Otwell and Aldridge [13] stated that it is clear that all the researchers have the same general conception of vulnerability and differ mainly how vulnerabilities of a particular system are specified and measured, and also clear that "more vulnerable" means "easier to adversely affect" and "less vulnerable" is better, other things being equal.

In this paper, we follow the definition from CVE website (http://cve.mitre.org/about/terminology.html): *an information security vulnerability is a mistake in software that can be directly used by a hacker to gain access to a system or network*. Just like what Frei stated in his dissertation [14], we also only consider vulnerabilities listed in the CVE directories. Hence, it does make sense for this paper to use the definition from CVE since all the vulnerability datasets used in this paper have CVE identification numbers.

II. ACTORS ON SOFTWARE VULNERABILITY ECOSYSTEM

There are many players in the software vulnerability ecosystem. According to Breukers [15] the vulnerability ecosystem is representing all the relationships among the vulnerability lifecycle including vulnerability discovery, exploitation, disclosure and patching of a software vulnerability, combining of interaction of all the actors and mechanisms.

So far, many researchers have proposed similar vulnerability lifecycle model including Joh and Malaiya [16]. In the vulnerability life journey, there are major events such as birth of vulnerability, discovery, internal disclosure, public disclosure, exploitation, script, patch available and death.

Software developers are the creators of security vulnerabilities in software systems. This could be a commercial or governmental vendor, sub-contractor, freelancer, or an open source community. Unsafe or careless programming behaviors cause the security defects as shown in Figure 1.

There are largely two types of vulnerability discoverer: white hat and black hat. When white hats discover the security vulnerability, they follow the responsible disclosure practice, which usually means a full disclosure under the all stakeholders' agreement of a period of time for developing patches on the vulnerability before publishing

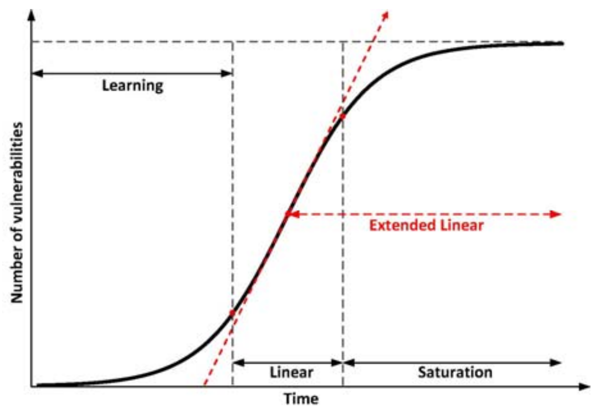

Fig. 2. S-shaped discovery process and extended linear phase

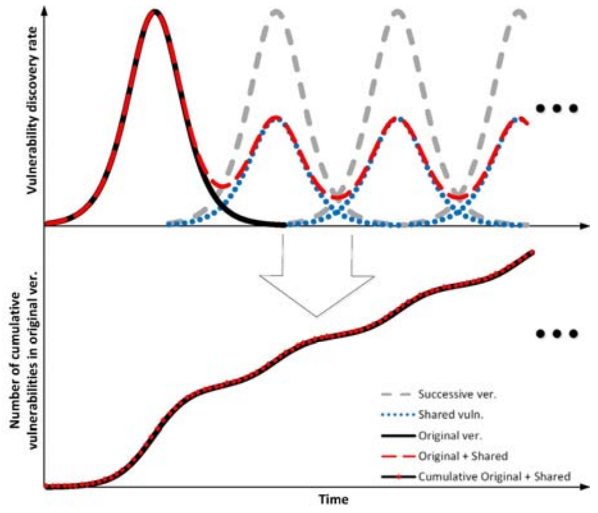

Fig. 3. The extended linear phase is caused by shared vulnerabilities in successive versions

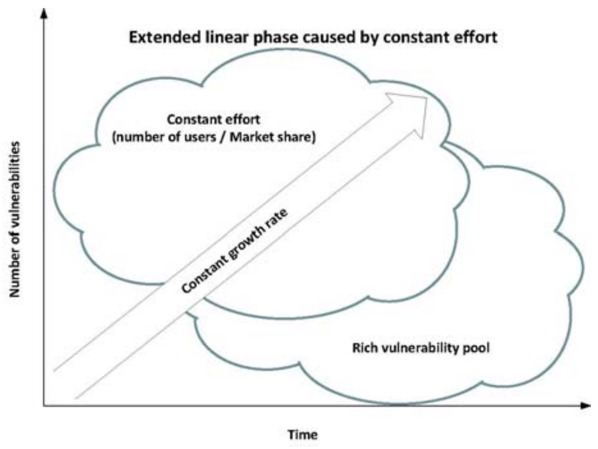
Extended linear phase caused by constant effort
Constant effort
(number of users / Market share)
Constant growth rate
Rich vulnerability pool
Number of vulnerabilities
Time

Fig. 4. The extended linear phase is caused by constant effort put on a system with a rich vulnerability pool

the details. On the other hand, if black hats detect the vulnerability, they use the information for their own goods.

Radianti et al. [17] empirically shows that there is indeed vulnerability black markets along with the white markets run by security companies. Whether it is a black or white, the markets give motivations and incentives for the vulnerability hunters.

Many commercial software vendors directly sell their products online, but often retailers and service providers do those businesses for the software producers. The product buyers could be home users or organizations. For the home users, they need to install the products and patches in their systems by themselves. In organizations, usually specialized administrators do the jobs.

Administrators' roles are very important for defending systems efficiently against malicious users and attackers. They need to decide when to install the newly released security updates because some patches or updates cause problems which not exist before.

The terminology of "script kiddie" is frequently used to distinguish from "black hat" who is able to create a hacking tools and able to analyze system's security holes. The script kiddie uses scripts or programs made by other skilled hackers to attack computer systems and networks.

III. EXTENDED LINEAR VULNERABILITY DISCOVERY PROCESS

Recently, robust linear behaviors in software vulnerability discovery process have been noticeably observed among the many popular systems having multi-versions released. Schryen [18] empirically examined vulnerability detection growth processes in seventeen software systems. He found that 14 out of the 17 systems show a significant linear or, at least, piecewise linear correlation between time and the number of cumulative published vulnerabilities, but without a deep investigation why the linear processes are prevalent. While showing the results, the author disproves the S-shape logistic vulnerability discovery pattern proposed by [19].

In Figure 2, the solid S-shaped line shows the shape of the vulnerability discovery process in AML [20] with the three distinctive phases. In the long run, for a software system, the vulnerability discovery process should look like the S-shape pattern when all the source codes with market effort put on it are reflected. During the release period, the vulnerability discovery rate gradually increases. At this phase, called learning phase, the software is gaining market share gradually and installed bases is small. In the linear phase, the discovery rate reaches the maximum due to the popularity, and finally, in the saturation phase, vulnerability discovery rate slows down.

However, under certain circumstances, the S-shape could be distorted, occasionally, seriously. The length of the second phase could be extended as long as new code is injected with certain levels of popularity lasted among users, so that the final phase tends to appear significantly later.

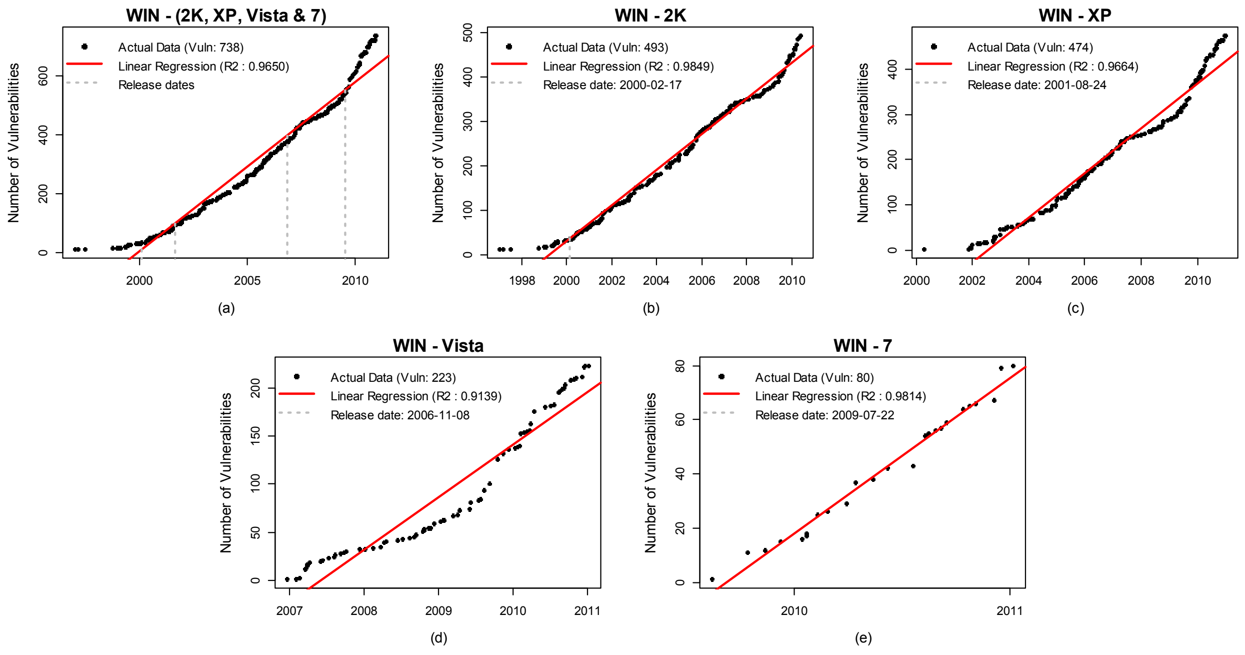

Fig. 5. Linear vulnerability growth trends. Black dots represent actual data points and the red lines are linear model fittings. Vertical dotted lines are released dates in the first graph.

Sometimes, after a clear saturation phase, new vulnerabilities are found. When this happens repeatedly, a discovery process forms a stairway-like pattern. Yet another, the first phase could not be seen at all. It is possible that combinations of above cases are coming out altogether. Among the mutant S-shapes above, here, mainly, the reason behind the extended linear phase is examined which currently appears notably. Other mutations also can be surmised based on the presentations.

The red dashed line, in Figure 2 highlights an extended linear phase. The first possible reason for this could be code sharing throughout the successive versions. New versions of software systems usually based on the previous version. When the product is getting popular, the number of users is also getting increasing. As a result, vulnerabilities originated from the earlier version starts to be found in the later version.

Moreover, new chunk of codes added into a new version introduces new vulnerabilities. When those software upgrades or patches go on and on, the extended linear phase could be resulted. Figure 3 shows this behavior. The original idea of sharing vulnerability is already introduced by [21]. In the figure, the vulnerability discovery rate for the original software system has been almost hit the saturation phase, marked by the solid black line (the first bell shape hump), but due to the shared vulnerabilities in successive versions (the grey dashed lines), the vulnerability discovery rate for the original product continually rises. The slope in the number of cumulative vulnerabilities is mainly influenced by how many codes are shared between the successive versions. Hence, as long as new versions, sharing codes with the previous version, are released with an enough market share, the extended linear phase will be observed.

The second reason could be, for a software system, the constant number of users with a vulnerability pool having a sheer amount of vulnerabilities which continually discovered with a constant rate due to a balanced effort, not increasing nor decreasing, put on the system, such as number of users. In this case, it will take some time proportional to the size of the vulnerability pool to be exhausted which causes a longer linear phase with a bigger pool. The concept is described in Fig. 4.

Table 1. Shared number of vulnerabilities and percentages.

B A	2K (191)	XP (113)	Vista (49)	Seven (8)
2K 2000-02-17	493 100%	300 60.85%	99 20.08%	28 5.67%
XP 2001-08-24	300 63.29%	474 100%	158 33.33%	58 12.23%
Vista 2006-11-08	99 44.39%	158 70.85%	223 100%	72 32.28%
Seven 2009-07-22	28 35%	58 72.5%	72 90%	80 100%

III. Observations

The software systems examined for the linear trend here is Windows operating systems. The datasets are minded at NVD (http://nvd.nist.gov) on January 2011. Table 1 shows the release dates for the software systems with the number of vulnerabilities shared among the successive versions in

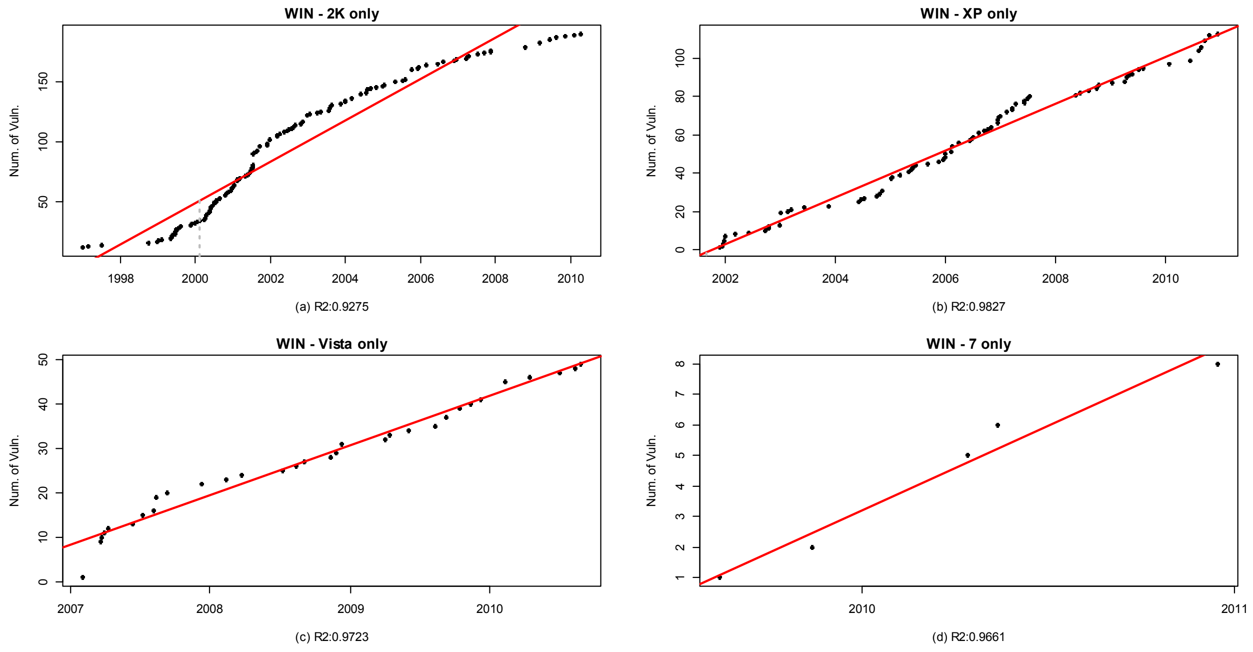

Fig. 6. Linear vulnerability growth trends by version with unique vulnerabilities in each version. Black dots represent actual data points and the red lines are linear model fittings.

each OS version. In the table, for the percentages, it should be read as *A* is sharing *X%* with *B*, where *A* and *B* are the row and column respectively as marked.

The oldest Windows OS is 2K and the newest one is Seven in the table. It could be conjecturable that the code sharing is higher with adjacent versions than others based on the shared number of vulnerabilities. 90% of vulnerabilities in Windows Seven is sharing with Vista, 70.85% of vulnerabilities in Vista is from XP, and 63.29% of the vulnerabilities in XP is from the previous version, which uncovers that the Windows OSes are continually built on top of its ancestors closely.

Plots in Figure 5 shows the linear model fittings with their R2 values. In all cases, the linear patterns are significantly observed and the linear fittings are well performed. We do not see any saturation phase at the end of the data periods.

Figure 6 shows the number of unique vulnerabilities in each specific version. Their R2 values can be found at each sub-caption. First, it is observed that the number of vulnerabilities have been dramatically reduced in each sub-plot compared to its entire vulnerability counterpart sub-plot from Figure 5 due to the removing the shared vulnerabilities. The noteworthy thing is that the learning phases start to appear more clearly in Figure 6. Also, the third phase tends to come out in Win 2K when its market share has been encroached by its successive version which proves that the extended linear phenomena is due to the code sharing with the popular successive versions. Especially, Win 2K reveals the saturation phase with unique vulnerabilities while their counterpart sub-plots for the entire vulnerabilities do not.

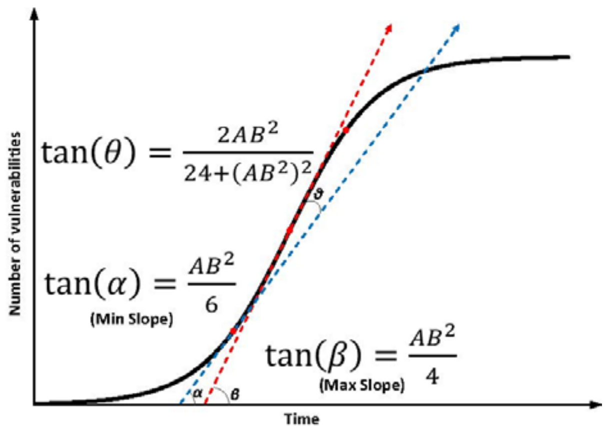
$\tan(\theta) = \frac{2AB^2}{24+(AB^2)^2}$
$\tan(\alpha) = \frac{AB^2}{6}$
(Min Slope)
$\tan(\beta) = \frac{AB^2}{4}$
(Max Slope)
Number of vulnerabilities
Time

Fig. 7. Estimated Max/Min slopes by AML

Table 2. AML model fitting parameters and fitting results

AML para. A	AML para. B	AML para. C
100E-06	932.7873	0.0475
R2	Min Slope	Max Slope
0.9908	0.1455	0.2183
Transition Point1	Mid point	Transition Point2
2004-03-29	2008-02-03	2011-12-11

The following equation represents the simple linear discovery model where *S* represents the slope or discovery rate and *k* is y-axis intersection which does not have a clear meaning.

$$\Omega(t) = k + S \times t$$

Now, predicting the exact discovery rate or slope for the extended linear phase is not an easy task. However, we could achieve fairly easily a probable scope of the rate falling into the ranges from the maximum and minimum slopes estimated by AML model fitting.

Figure 7 demonstrates the maximum ($tan(\beta)$) and the minimum ($tan(\alpha)$) slopes during the linear phase in the AML model. Consequently, the difference (θ) between the two slopes can be achieved. A and B are from the AML parameters. The maximum slope is on the tangent line of the mid-point whereas the minimum slope exists on either of the two transition points. Hence, in some degree, it is possible to estimate the current extended linear vulnerability discovery rate for the multi-version software systems.

When we apply the method to get the slope from the linear fitting for the aggregated version in Figure 5, the result is 0.1569453. If we conduct the same analysis with other operating systems such as OSX, we could compare the slopes among different OSes. Table 2 shows the AML model fitting parameters on the aggregated version, fitting result with R2 value, Min and Max possible slopes, and the three Transition points (TP1, MP, TP2) [22].

TP1 by AML model fitting are estimated a bit later than the time point supposed to be, due to the code evolutions. When we consider currently observed strong linear trends across the discovery patterns, transition points, especially MP and TP2, in Table 2 should not be accurate estimations because of the continuous software evolving which should trigger shifting of transition points.

IV. DISCUSSION

Although upper and lower boundaries for linear rates could be estimated and there are already some VDMs available, it would be nice to estimate the linear rate more precisely with more less complex relationship for taking advantage of the newly appeared linear pattern.

By its nature, a quantitative vulnerability discovery model requires empirical observations on the relationship between a growth trend in actual data and a set of factors believed to influence on the trend. At the beginning of the investigation, usually, the relationship is unknown or unclear, so that researchers generate some assumptions providing a starting point which are reasonable, or observed vaguely from the actual data but are not confirmed in a scientific way yet. For the starters, we have also some intuitive and vaguely observed assumptions that might influence on the slopes in the linear model which could be *i)* Skills of programming team & maturity of vendor *ii)* Number of installations with code sharing *iii)* Source code edit frequency *iv)* Software type.

First, attitude of a vendor and its developers toward secure programming practice should effect on the degree of slope. Experts agree that developers' skills are important factors influence on quality of products although there have not been good references quantitatively conducted. Therefore, skill of programming team should be in inverse proportion to the slope value. Also vendor's maturity in its field should also be matter for products' quality. The better skills developers have in the more security related mature environment, the gentler slope should be produced.

Second, it is intuitive that the more number of installations causes the more number of vulnerabilities discovered. This is because, as the AML model already has claimed, a system would be more thoroughly tested with a bigger group of users or testers which will demand more number of vulnerabilities found. Along with this intuition, as long as popular enough successive versions are released regularly, the saturation phase will not be seen. Therefore, there should be positive growth correlations between the slope and the number of installations backed up by code sharing with successive versions. As a result, code sharing also effect on the slope. The more codes are shared, the steeper slope should be appeared from the originated version.

Third, Zimmermann et al. [23] empirically examined the effectiveness of classical software metrics to predict vulnerabilities and assess how well the measurements perform on Windows Vista. They measure the correlations between the metrics and the number of vulnerabilities. The study shows that all the correlations are less than 0.3, which is considered as small effect size. However, among them, the correlation between the frequency of source code editing and the number of vulnerabilities claims the highest value. Hence, the more frequently developers edit source code, the better chances that vulnerabilities are introduced.

Lastly, the software type matters. For example, the growth rates and slopes for popular software systems such as Web browsers and OSes should be steeper than other types of systems due to the number of users. Hence, there should be some empirical guidelines categorizing software systems and endowing with certain weights associated with numbers. Software systems could be grouped into OSes, Web browsers, Web servers, Web applications, DBMSs, etc. After the classification, proper weights need to be associated.

Acknowledgement

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT & Future Planning (NRF-2015R1C1A1A02036448)

References

[1] I. V. Krsul, "Software vulnerability analysis," PhD dissertation, Purdue University, West Lafayette, IN, USA. Advisor: E. H. Spafford, 1998.

[2] A. Ozment, "Improving vulnerability discovery models," *in Proceedings of the 2007 ACM workshop on Quality of protection*, NewYork, pp. 6–11, 2007.

[3] J.A. Wang, M. Guo, H. Wang, M. Xia and L. Zhou, "Environmental Metrics for Software Security Based on a Vulnerability Ontology," *in Proceedings of the third IEEE International Conference on Secure Software Integration and Reliability Improvement*, pp. 159-168, 2009.

[4] R. Kissel, "Glossary of Key Information Security Terms," NIST IR 7298, 2006

[5] C.P. Pfleeger and S.L. Pfleeger, *Security in Computing*. 3rd ed., Prentice Hall PTR, 2003.

[6] W.R. Cheswick and S.M. Bellovin, Firewalls and Internet Security: Repelling the Wily Hacker. Reading, MA: Addison-Wesley, 1994.

[7] E.E. Schultz Jr., D.S. Brown and T.A. Longstaff, "Responding to Computer Security Incidents," Lawrence Livermore National Laboratory, 1990.

[8] K. Otwell and B. Aldridge, "The role of vulnerability in risk management," *in proceedings of Computer Security Applications Conference*, pp.32-38, 1989

[9] H. Mayerfeld, "Definition and Identification of Assets as The Basis for Risk Management," *in Proceedings of 1988 Computer Security Risk Management Model Builders Workshop*, pp.21-34, 1988

[10] N. Lewis, "Using Binary Schemas to Model Risk Analysis," *in Proceedings of 1988 Computer Security Risk Management Model Builders Workshop*, pp.35-48, 1988

[11] D. Snow, "A General Model for the Risk Management of ADP Systems," *in Proceedings of 1988 Computer Security Risk Management Model Builders Workshop*, pp.145-162, 1988

[12] *IEEE standard glossary of software engineering terminology*, IEEE Standard 610.12-1990, 1990

[13] K. Otwell and B. Aldridge, "The role of vulnerability in risk management," *in Proceedings of Computer Security Applications Conference*, pp.32-38, 1989

[14] S. Frei, "Security Econometrics - The Dynamics of (In)Security", Ph.D. dissertation, ETH Zurich, ISBN 1-4392-5409-5, 2009

[15] Y.P. Breukers, "The Vulnerability Ecosystem: Exploring vulnerability discovery and the resulting cyberattacks through agent-based modelling," M.S. Thesis, Delft University of Technology, Aug. 22, 2016

[16] H. Joh and Y. K. Malaiya, "Defining and Assessing Quantitative Security Risk Measures Using Vulnerability Lifecycle and CVSS Metrics," *in Proceedings of the 2011 International Conference on Security and Management*, pp. 10-16, 2011

[17] J. Radianti, E. Rich, and J. Gonzalez, "Vulnerability black markets: Empirical evidence and scenario simulation," *in Proceedings of the 42nd Hawaii International Conference on System Sciences*, pp.1-10, 2009

[18] G. Schryen, "Security of open source and closed source software: An empirical comparison of published vulnerabilities," *in Proceedings of the 15th Americas Conference on Information Systems*, 6-9 Aug., 2009

[19] O. Alhazmi, Y.K. Malaiya and I. Ray, "Security vulnerabilities in software systems: A quantitative perspective," *Lecture Notes in Computer Science of Data and Applications Security XIX*, vol.3654, pp.281–294, 2005

[20] O. Alhazmi and Y.K. Malaiya, "Application of Vulnerability Discovery Models to Major Operating Systems," *IEEE Transactions on Reliability*, vol.57, pp.14-22, 2008

[21] J. Kim, Y.K. Malaiya and I. Ray, "Vulnerability Discovery in Multi-Version Software Systems," *in Proceedings of the 10th IEEE High Assurance Systems Engineering Symposium*, Washington, DC, USA, pp.141-148, 2007

[22] O. Alhazmi and Y.K. Malaiya, "Prediction Capabilities of Vulnerability Discovery Models," *in Proceedings of Reliability and Maintainability Symposium*, pp. 86-91, 2006

[23] T. Zimmermann, N. Nagappan and L. Williams, "Searching for a Needle in a Haystack: Predicting Security Vulnerabilities for Windows Vista," in Proceedings of the 2010 Third International Conference on Software Testing, Verification and Validation, pp.421-428, 2010.

Author

HyunChul Joh is an assistant professor in the Department of Computer Engineering at Kyungil University since March 2014. From 2012 to 2014, he was a GIST college laboratory instructor in division of liberal arts and sciences at Gwangju Institute of Science and Technology in Korea. His research focuses on modeling the discovery process for security vulnerabilities and risk metrics. He received his Ph.D. and M.S. in computer science from Colorado State University in 2011 and 2007 respectively. He also received a B.E. in information and communications engineering from Hankuk University of Foreign Studies in Korea, 2005.