WORKSHOP ON THE ECONOMICS OF INFORMATION SECURITY
June 2010

An Empirical Analysis of Exploitation Attempts based on Vulnerabilities in Open Source Software

Sam Ransbotham
Carroll School of Management, Boston College, Chestnut Hill, MA 02467, sam.ransbotham@bc.edu

For open source software, security attention frequently focuses on the discovery of vulnerabilities prior to release. The large number of diverse people who view the source code may find vulnerabilities before the software product is release. Therefore, open source software has the potential to be more secure than closed source software. Unfortunately, for vulnerabilities found after release, the benefits of many having viewers may now work against open source software security. Attackers may be more likely to exploit discovered vulnerabilities since they too can view the source code and can use it to learn the details of a weakness and how best to exploit it. I examine the diffusion of vulnerabilities in open source software compared with closed source software. Empirical analysis of two years of security alert data from intrusion detection systems indicates that open source software vulnerabilities are at greater risk of exploitation, diffuse more rapidly, and have greater volume of exploitation attempts.

Key words: information security; vulnerability disclosure; information technology policy; empirical analysis

1. Introduction

The current world economy relies heavily on computerized information systems; security problems threaten this infrastructure. While earlier attention focused on issues such as system development (Baskerville 1993, e.g.), researchers increasingly turn to managerial action (Straub and Welke 1998, Lohmeyer et al. 2002), organizational context (Dhillon and Backhouse 2001), and economic incentives (Gordon and Loeb 2002, 2006, Gordon et al. 2010) to understand the process of security compromise (Ransbotham and Mitra 2009). From any perspective, security remains important.

While there are many paths to insecurity, vulnerabilities in software are an unfortunately common problem. Research to reduce the effect of software vulnerabilities focuses on four main stages. First, before software is released, a great deal of research emphasizes improved development prac-

tices to reduce vulnerabilities. Second, after the software is released, incentive structures for vulnerability discovery efforts continue to be proposed and debated (e.g. Ransbotham et al. 2010, Kannan and Telang 2005, Schechter 2004, Ozment 2004, Bohme 2006). Third, once a vulnerability is discovered, there are important policy decisions about the best way to disclose the vulnerability to vendors, information systems professionals and the public (e.g. Arora et al. 2004b,a, Choi et al. 2005, Cavusoglu et al. 2005b, Li and Rao 2007, Cavusoglu et al. 2007). Finally, once vendors develop patches, trade-offs remain about the optimal patching policy (e.g. August and Tunca 2008, Arora et al. 2006, Cavusoglu et al. 2008).

One important choice software developers make is the availability of the source code. Developers can choose two basic options for software source code visibility— closed source or open source. In closed source software, the developers do not make the source code publicly visible; in open source software, the source code is publicly available for viewing. Many proponents of open source software believe that by making the source code available, the software product can be more secure. They reason that the more people who view the code, the more likely that vulnerabilities are discovered *before* the software is released. This is summarized and often repeated as "given enough eyeballs, all bugs are shallow" (Raymond 1997).

However, despite development and testing efforts, not all vulnerabilities in software will be found before release. Instead, some will inevitably be found afterwards. While considerable research and commentary has focused on the pre-release stage benefits of open source software, the post-release stage is important as well. Once a vulnerability is found in released software, the same mechanism which increases security before release may make security worse after release— potential attackers have access to view the source code containing the vulnerability. Because potential attackers can view the code, their effort required to design code to exploit the vulnerability is reduced.

In this research, I investigate exploitation attempts on vulnerabilities in open source and closed source software products. I use an empirical analysis of log data from intrusion detection systems to examine the risk, diffusion and volume of exploitation attempts. The log data spans two years

(400 million alerts) and is generated by 960 clients of a managed security service provider. Relative to vulnerabilities in closed source software products, my analysis indicates that vulnerabilities in open source software products (a) have a greater risk of exploitation, (b) diffuse earlier and wider, and (c) have greater overall volume of exploitation attempts. My research contributes to our understanding of software vulnerabilities in three ways. First, I consider the effects of wide viewing of source code in the vulnerability exploitation phase: while analysis of the pre-release stage of open source software is common, the exploitation stage receives much less attention. Second, I contribute one of the few large scale empirical examinations of software vulnerabilities: while analytical models are common, there are relatively few empirical analyses using real data across multiple organizations (e.g. Ransbotham et al. 2010, Ransbotham and Mitra 2009). Empirical studies so far focus predominantly on the important topic of vulnerability creation during development (Meneely and Williams 2009, Meneely et al. 2008), not the post-release exploitation process. Third, the theoretical insights from the analysis provide important guidance for disclosure policy makers.

The rest of the paper is organized as follows. Section 2 provides the research context and the theoretical background for four hypotheses about the diffusion, risk and volume of exploitation attempts from open source software. Next, Section 3 describes the data and methodology used to test the hypotheses. Section 4 details the results of the empirical analysis. Finally, Section 5 summarizes the contribution of the study and suggests future research to build on this study.

2. Theoretical Background

2.1. Vulnerability Discovery and Disclosure

While developers try to eliminate security vulnerabilities before software is released, both security professionals and attackers continue to find vulnerabilities. As an indication, the NVD published 5,632 vulnerabilities in 2008 and 5,733 in 2009. There is little evidence to suggest that developers can completely eradicate vulnerabilities before release.

Thus, the impact of these inevitable vulnerabilities depends largely on who discovers them first. If security professionals find vulnerabilities, they typically share this information with other professionals through organizations such as CERT. Using various policies (Arora et al. 2004b), these

organizations attempt to get vulnerabilities corrected and the vulnerability threat reduced. On the other hand, if attackers find vulnerabilities, they quickly attempt to exploit them for their own gain.

This high level overview illustrates two key aspects of the vulnerability life-cycle. First, attackers and defenders have different reward and incentives. Much recent research attention has focused on ways to evaluate and manipulate incentive structures, particularly for security professionals. Instead of freely sharing, market mechanisms have been proposed (Schechter 2004, Ozment 2004) to increase rewards and incentives for responsible disclosure. These market mechanisms may have inherent weaknesses (such as incentive to leak information, Kannan and Telang 2005) or may provide a time advantage to defenders (Ransbotham et al. 2010). Second, many different disclosure policies exist and have been analyzed (e.g. Arora et al. 2004a,b, Choi et al. 2005, Cavusoglu et al. 2005b, Bohme 2006, Li and Rao 2007, Cavusoglu et al. 2007).

Regardless of who discovers the vulnerability or what mechanism is used for discovery, the discovery starts a research and development (R&D) race between attackers and defenders (Ransbotham et al. 2010). Attackers pour effort into developing methods and tools for exploiting the vulnerabilities, and then begin to use and distribute them. Conversely, defenders work to correct the vulnerability, to inform affected entities, to release updated software and to encourage users to install updates. Instead of focusing on the pre-release benefits of open source software, my hypotheses focus on how the public visibility of source code affects this exploitation/defense race.

2.2. Open Source Software

Open source software is widely held to be more secure than closed source software. The core of the argument is that with open source code, many people have the potential to find and correct an error. This is summarized as "given enough eyeballs, all bugs are shallow" (Raymond 1997). While researchers have attempted to quantify and measure this effect (e.g. Schryen and Rich 2010), it is inherently complex. Software projects differ in complexity, features, scope, and user base; the number and severity of vulnerabilities may be linked to these differences. Therefore, attributing

the vulnerabilities to the open/closed choice is difficult. Furthermore, recent research suggests that there may be diminishing returns to increased number of users in the context of software (Rescorla 2005) or other community build artifacts (Constant et al. 1996, Kane and Ransbotham 2009). Therefore, many closed source projects could already have "enough eyeballs" and open source projects could have more than enough. In fact, recent empirical research finds limited differences in vulnerabilities disclosed in each type (Schryen 2010), but also finds some evidence of more frequent disclosures in open source (Schryen 2009).

Open source software presents two additional challenges to post-release security. First, the open nature of the source code eliminates any benefits of private disclosure. Private disclosure, if leakage is avoided (Kannan and Telang 2005), can give defenders some time advantage (Ransbotham and Mitra 2009). However, because changes to the source code are visible, they are publicly disclosed by definition, making it "too easy for hackers to figure out how to defeat the security" (Lawton 2002, p. 19). Even if open source software patches more rapidly than closed source (Arora et al. 2005), modifications to the source code are already visible. Second, many open source software projects are themselves used as components of other software products. Even if the vulnerability itself is corrected, the downstream software products are still at risk until they incorporate the correction and release a patch. For example, OpenSSL is a widely used open source library used for secure socket communication. However, after corrections are made in OpenSSL, products that in turn use OpenSSL must also develop, test, and release their products. While possible in closed source software (e.g. drivers, libraries), this component nature that is common in open source projects increases the window of opportunity that attackers have.

2.3. The Risk of Exploitation Attempt

First, open source software may increase the risk of exploitation attempts based on a vulnerability. Since attackers can view the source code, their effort and time to develop an exploit is reduced for open source software. Because their effort is reduced, the expected value of exploitation is strictly increased because of reduced cost. Similarly, because the time is reduced, the expected value is

increased because of increased probability of exploit success. In general innovation competition, information and time advantage is important (Bloch and Markowitz 1996, Fudenberg et al. 1983). In the context of security, the time advantage means that attackers will find fewer systems patched and defended against the vulnerability. Faced with limited resources, the rational attacker will focus on opportunities with the highest expected value of exploitation first. Therefore, I hypothesize that:

HYPOTHESIS 1. *A target firm will face a greater risk of exploitation for vulnerabilities in open source software than for vulnerabilities in closed source software.*

2.4. The Diffusion of Exploitation Attempts

Second, like other innovation (Rogers 2003), exploitation knowledge diffuses through the attacker community. After vulnerabilities are discovered, expert attackers build tools to exploit the vulnerability (Ransbotham and Mitra 2009). As these tools diffuse through the attacker community, more and more firms will experience the exploitation attempt. Then, as the potential targets install countermeasures, the value of the tool will diminish; new tools based on new vulnerabilities will offer more reward. Therefore, exploitation attempts will follow the traditional S-curve of technology diffusion.

To model the diffusion process of attacks, I use the following notation. Let $N(t)$ be the cumulative number of target firms affected at time t where t is measured from the time a vulnerability is discovered. Let P be the height of the S-curve, or the maximum number of target firms in the population affected by the vulnerability (referred to as the penetration of the diffusion process). Let T_h be the time t when half of the target systems are affected by the vulnerability. R is the slope of the S-curve which is dependent on factors such as the type of vulnerability, complexity of developing exploits, and the impact of the vulnerability on systems. As such, $N(t)$ is modeled using the following familiar form of the S-curve.

$$N(t) = \frac{P}{1 + e^{-R(-t-T_h)}} \tag{1}$$

Similar to the reasoning in Hypothesis 1, I expect that the diffusion of exploitation attempts based on open source vulnerabilities will occur sooner than exploits based on closed source vulnerabilities. The reduced time and effort for attackers to develop exploits translates to earlier diffusion. Consequently, the S-curve in Equation 1 is shifted to the left for vulnerabilities in open source software. Therefore, I hypothesize that:

HYPOTHESIS 2. *The start of the diffusion of attacks through the population of target firms will be accelerated for vulnerabilities in open source software as compared with vulnerabilities in closed source software.*

Because of the time advantage, attackers will attempt exploits on a larger number of firms as well. The rational attacker will only stop when the expected value of the next exploitation attempt is non-positive. The time advantage means that attackers have longer before potential targets can implement countermeasures. Consequently, the target population of expected unprotected systems will be greater, leading to a taller height (P) of the S-curve for the diffusion process. Therefore, I hypothesize that:

HYPOTHESIS 3. *The diffusion of attacks through the population of target firms will have greater penetration for vulnerabilities in open source software than for vulnerabilities in closed source software.*

2.5. Volume of Exploitation Attempts

Finally, another metric to evaluate the relative impact of the open versus closed source choice is the overall volume of exploitation attempts. Vulnerabilities are typically found by expert attackers; they are then quickly incorporated into automated tools usable by anyone (Ransbotham and Mitra 2009). While expert discovery will be seen in the risk of first exploitation analysis, tools are responsible for the volume of exploitation attempts.

Open source code affects tools from both a supply and demand side. First, expert attackers invest their time and effort into tools that the attacker community will find most valuable. Tool value comes from increase probability of successful attack. Because the source code is visible, expert

attackers require less effort and time to create tools; this time reduction increases the probability of successful attack. Therefore, more tools will be available. Second, adopters of the tools are similarly rational. They will adopt and spend effort on tools with the greatest expected value. The compounded result is greater adoption of a greater number of tools; these tools increase the number of exploitation attempts that a firm will experience. Therefore, I hypothesize that:

HYPOTHESIS 4. *A target firm will have more attacks for vulnerabilities in open source software than for vulnerabilities in closed source software.*

3. Data and Methodology

To evaluate these hypotheses, I merge information from three sources: 1) detailed alert log data from 960 firms using intrusion detection systems managed by a security service provider; 2) detailed vulnerability data from the National Vulnerability Database (NVD); and 3) manual classification of the software products associated with each vulnerability.

3.1. Intrusion Detection System Alert Logs

Firms install an intrusion detection system (IDS) to prevent potentially malicious traffic from entering their network. An IDS monitors incoming traffic and looks for known patterns of suspicious traffic. These known patterns are called *signatures*. When an IDS detects a signature, it stops the traffic and records an alert in a log. As a result, the IDS provides two main functions. First, it prevents malicious traffic from entering a network in real-time. Second, the logs provide the opportunity for later detailed analysis. As a result, IDS logs offer insight into attacker behavior that has been exploited by researchers. Cavusoglu et al. (2005a), Ransbotham et al. (2010), and Ransbotham and Mitra (2009) present more detail on intrusion detection systems and offer examples of their prior use in security research.

For the empirical analysis, I use alert logs generated by intrusion detection systems installed at 960 clients of a managed security service provider, SecureWorks. This dataset provides a unique research opportunity because it contains real alert data (as opposed to data from a research setting) from a large number of firms with varied infrastructure across many industries. The alert database

contained over four hundred million alerts generated during 2006 and 2007. My analysis is based on a summary dataset of the number of alerts generated every day during the two year period of my analysis, grouped by target firm and individual vulnerability.

3.1.1. National Vulnerabilities Database My second main data source is the National Vulnerabilities Database (NVD 2008). The NVD consolidates several public vulnerability data sources such as CERT, Bugtraq, XForce and Secunia. Security exports assess each vulnerability in the NVD using a Common Vulnerability Scoring System (CVSS) (Mell and Romanosky 2008, Mell et al. 2006). The CVSS is an open, mature, and well-established (e.g. Frei et al. 2006, Jones 2007, Kotenko and Stepashkin 2006, Ransbotham et al. 2010) definition of the fundamental characteristics of a vulnerability. Despite its shortcomings, it is objectively scrutinized by many interested parties and uniformly applied to all vulnerabilities. Details of the CVSS scoring system used in my analysis are in Mell and Romanosky (2008). It is important for the analysis that I insure that the effects I see are due to the open source nature of the software product and not due to characteristics of the vulnerability itself. The uniform scoring system, along with other data in the NVD, provides several control variables. Based on the coded vulnerability attributes, the CVSS also generates a single score to convey the urgency and priority of the vulnerability and provide an indication of the likely potential damage from the vulnerability. However, instead of using the aggregate score, I use the individual vulnerability attributes as controls since a summarized score loses the more detailed information available from the components of the score.

I use the following control variables in my empirical analysis derived from the information available through the NVD.

Complexity: Once the attacker has access, vulnerabilities require varying degrees of complexity to exploit; the NVD uses an ordinal scale of low, medium, or high complexity. I code low complexity as the base level and include two indicator variables, $MediumComplexity$ and $HighComplexity$. The complexity score is based on the difficulty of exploitation and not on the difficulty of detection and deterrence. Attackers can exploit vulnerabilities of low complexity without additional information gathering, specialized access, or high skill levels, while vulnerabilities classified as medium or

high complexity require varying and greater levels of skill, specialized access conditions, possible social engineering, and specialized situations that can often be deterred by the target firm. Thus, medium and high complexity vulnerabilities are progressively harder to exploit than low complexity vulnerabilities.

Signature: I include an indicator variable, $Signature$, that is set to 1 if a signature was available at the time that the vulnerability was disclosed to the public, 0 otherwise. The signature represents an additional method that attackers can use to learn how to exploit a vulnerability.

Impact: The potential impact of the vulnerability would likely affect the diffusion, risk and volume as attackers may prefer to spend resources on vulnerabilities which have certain categories of impact. The potential impact of a vulnerability is categorized by experts as affecting the disclosure of confidential information ($impact\_conf$), the integrity of data ($impact\_integrity$), or the availability of system resources ($impact\_avail$). For the analysis, I use an indicator variable for each impact category that is set to 1 if the potential for the specific impact (confidentiality, integrity and availability) is present, 0 otherwise. A vulnerability can be coded by experts to have multiple potential impacts.

Type: attackers may find exploitation easier or rewards larger based on the type of flaw. The NVD classifies vulnerabilities into seven different types based on the specific software flaw that the vulnerability represents. These are (i) incorrect access privileges ($type\_access$), (ii) failure to handle incorrect input ($type\_input$), (iii) shortcomings in the design of software ($type\_design$), (iv) insufficient response to unexpected conditions ($type\_exception$), (v) weak configuration of settings ($type\_config$), (vi) errors due to sequencing of events ($type\_race$), and (vii) uncategorized vulnerability types ($type\_other$). Separate indicator variables are included for the first four types; however, there were insufficient configuration and race type vulnerabilities and these were grouped into the base type.

Patch: I include an indicator variable, $Patch$, that is set to 1 if a patch was available at the time the vulnerability was disclosed to the public, 0 otherwise.

Age: I also include the *Age* of vulnerability (log transformed) at the time of the analysis, measured as the number of days since the vulnerability was disclosed.

3.1.2. Manual Classification of Software Products My final data source is a manual classification of software products. While the NVD database lists software products affected by a vulnerability, it does not specify if the software products are open or closed source. At the time of my analysis, the NVD contained vulnerabilities affecting 13,101 distinct software products. I was able to find definitive information that 3,369 software products (25.72%) were open source. Similarly, I found that 3,121 products (23.82%) had closed source licenses. I did not find definitive information about the remaining 6,611 products (50.46%). Despite the large percentage, qualitative analysis of the unknown licenses indicates that they are not well-known or highly used products. Some vulnerabilities affect more than one product; if a vulnerability affected a mix of open and closed source products, I classified the vulnerability based on the code where the vulnerability was located (if available). For example, a vulnerability in OpenSSL could affect both open and closed source software that use OpenSSL. In this case, the vulnerability is an "open source" vulnerability because attackers can see the source code for the vulnerability even if they cannot see the source code for the closed source products affected. This approach to classification has been used before but only with a limited set of vendors (Christey and Martin 2007)[1]. My key focal variable is an indicator variable, *OpenSource*, that is set to 1 if the source code for a vulnerability is open source and 0 otherwise. Subsequent analysis does not include NVD entries without definitive classification.

I match the signatures for each unique vulnerability in the intrusion alert logs with detailed information in the NVD. The matching is done through a CERT assigned unique ID that links the databases together. Not all NVD entries link to signatures in the intrusion detection system logs; conversely, not all signatures in the intrusion detection system link to NVD entries. These links are particularly sparse for older vulnerabilities. While the links are incomplete, I compiled a sample of 883 vulnerabilities with matching IDS signatures. Of these, 359 (40.66%) are vulnerabilities in

[1] This classification process was difficult and time consuming. No other researcher should have to take the effort to do a similar classification to build on these results. A project to make this information publicly available is underway.

Table 1 Sample Descriptive Statistics

		Open Source		Closed Source	
Variable	Value	Count	Percentage	Count	Percentage
Exploited	No	329	91.64%	457	87.21%
	Yes	30	8.36%	67	12.79%
Access Required	Requires Local	50	13.93%	63	12.02%
	Requires Adjacent	3	0.84%	8	1.53%
	Network	306	85.24%	453	86.45%
Complexity	Low	187	52.09%	245	46.76%
	Medium	131	36.49%	225	42.94%
	High	41	11.42%	54	10.31%
Authentication	Not required	337	93.87%	508	96.95%
	Required	22	6.13%	16	3.05%
Confidentiality Impact	No	104	28.97%	105	20.04%
	Yes	255	71.03%	419	79.96%
Integrity Impact	No	103	28.69%	94	17.94%
	Yes	256	71.31%	430	82.06%
Availability Impact	No	69	19.22%	73	13.93%
	Yes	290	80.78%	451	86.07%
Vulnerability Type	Access	12	3.34%	34	6.49%
	Input	116	32.31%	151	28.82%
	Design	63	17.55%	82	15.65%
	Exception	48	13.37%	45	8.59%
	Environmental	1	0.28%	1	0.19%
	Configuration	6	1.67%	7	1.34%
	Race	6	1.67%	6	1.15%
	Other	6	1.67%	9	1.72%
Patch Available?	No	140	39.00%	220	41.98%
	Yes	219	61.00%	304	58.02%
Signature Available?	No	348	96.94%	380	72.52%
	Yes	11	3.06%	144	27.48%

open source software. Table 1 presents descriptive statistics on the vulnerability data for both open and closed source software products.

4. Results

The sample consists of 883 vulnerabilities with their alert signatures matched to the NVD database and a known license. In the alert database, attackers attempted to exploit only 97 (11%) of the 883 vulnerabilities during the time period of the study. To evaluate the hypotheses, I analyze three empirical models—the risk of an exploitation attempt (Section 4.1), the diffusion of exploitation attempts (Section 4.2), and the volume of exploitation attempts (Section 4.3).

4.1. Risk of First Exploitation Attempt

I analyze the risk of first exploitation attempt from a vulnerability on a firm through a proportional hazard model (Cox 1972, Kalbfleisch and Prentice 2002). In the hazard model, the failure event is the first observed exploitation attempt of a vulnerability. The hazard model integrates information for vulnerabilities that were never exploited, incorporates the evolving risk of exploitation, and handles truncation of observation caused by the end of the study period. The hazard model estimates the risk, $h(t)$, through the following equation:

$$h(t) = lim_{\Delta t->0} \frac{Pr(t + \Delta t > T > t | T > t)}{\Delta t} \tag{2}$$

The Cox proportional hazard model consists of two parts: the baseline hazard function describing how the risk of first exploitation attempt changes over time when all covariates are at the mean level; and a parameter for each covariate that describes how the baseline hazard changes in response to explanatory covariates. In the Cox model, the baseline hazard is not affected by the covariates, and the parameters are assumed to have a multiplicative effect on the baseline hazard. Because firms have underlying unobserved heterogeneity, I stratify the analysis to help incorporate unobserved firm specific vulnerability to exploitation attempts. In Equation 3, the baseline hazard, $h_0(t)$, for firm i and vulnerability j is adjusted by both covariates ($x_{i,j}$) and the firm specific stratification parameter (ν_i).

$$h_{i,j}(t) = h_0(t) e^{(\beta x_{i,j} + \nu_i)} \tag{3}$$

For this analysis, I construct a data set that contains for each firm and vulnerability combination, the day of first attempt to exploit the vulnerability (960 firms and 883 vulnerabilities for a total of 847,126 observations). The model considers a firm at risk of exploitation attempt on the date that the vulnerability was published. Risk ends when either the first exploitation attempt is observed or upon censoring at the end of the study period (12/31/2007). To evaluate Hypothesis 1, I incorporate the focal and control variables as explanatory variables in the hazard model.

Table 2 describes the results of the proportional hazard analysis. Model 0 describes the effects of only the control variables on the risk of exploitation (baseline hazard). Model 1 enters the

focal indicator variable *OpenSource* and supports Hypothesis 1. The coefficient of the open source variable ($\beta_{12} = 0.259$, $p < 0.001$) indicates that the risk of first attack increases if the underlying software product is open source. This is consistent with the argument that information available from the source code makes it easier for attackers to exploit a vulnerability. Similarly, signature availability increases the risk of first attack ($\beta_{11} = 1.095$, $p < 0.001$) providing additional evidence that attackers can also gain information about how to exploit a vulnerability by examining a signature. It is interesting that high complexity vulnerabilities ($\beta_{10} = 0.169$) are *more* likely to be exploited. It is not clear why; perhaps the exclusivity possible by exploiting a complex vulnerability offsets the additional effort required. More research is needed to understand this finding.

4.2. Diffusion of Attacks

Next, to evaluate Hypothesis 2 and Hypothesis 3, I model the diffusion of exploitation attempts through the sample of firms (see Ransbotham et al. 2010). For each of the exploited vulnerabilities and for each day in the research period, I calculated the cumulative number of firms that experienced exploitation attempts based on that vulnerability until that day and built a panel with 83,806 observations. No observations are included prior to the publish date of the vulnerability. To examine the impact of open source licensing on the diffusion of attacks, I estimate Equation 1 with covariates using non-linear least squares estimation. In Equation 4 below, penetration (P), rate (R) and delay (D) are linear functions of the focal and control variables.

$$N(t) = \frac{P}{1 + e^{(-Rt - D)}} \tag{4}$$

where

$$P = \beta_0^P + \beta_1^P OpenSource + control\_variables \tag{5}$$

$$R = \beta_0^R + \beta_1^R OpenSource + control\_variables \tag{6}$$

$$D = \beta_0^D + \beta_1^D OpenSource + control\_variables \tag{7}$$

The term RT_h in Equation 1 is incorporated in the constant term β_0^D in Equation 7. Hypothesis 2 and Hypothesis 3 predict that the coefficients of the OpenSource indicator variable (β_1^P and β_1^D) in Equation 5 and Equation 7 are positive and negative, respectively.

Table 2 Hazard Analysis of Risk of Exploitation of a Vulnerability

	Variable	Model 0	Model 1
β_1	Confidence Impact	-0.133^{***}	-0.143^{***}
		(0.027)	(0.027)
β_2	Integrity Impact	0.074^{*}	0.107^{***}
		(0.031)	(0.030)
β_3	Availability Impact	0.361^{***}	0.356^{***}
		(0.034)	(0.034)
β_4	Vuln: Access	-34.994^{***}	-41.942^{***}
		(0.013)	(0.014)
β_5	Vuln: Input Validation	0.083^{***}	0.062^{*}
		(0.024)	(0.024)
β_6	Vuln: Design	-0.387^{***}	-0.411^{***}
		(0.034)	(0.034)
β_7	Vuln: Exception	-0.011	-0.052
		(0.033)	(0.032)
β_8	Patch	-0.022	-0.031
		(0.018)	(0.018)
β_9	Complexity: Medium	-0.138^{***}	-0.122^{***}
		(0.023)	(0.023)
β_{10}	Complexity: High	0.187^{***}	0.169^{***}
		(0.025)	(0.025)
β_{11}	Signature	0.979^{***}	1.095^{***}
		(0.020)	(0.020)
β_{12}	OpenSource		0.259^{***}
			(0.019)
	Log likelihood	-78594.73	-78519.29
	Wald χ^2	$9.86x10^6$	$1.41x10^7$

Cox proportional hazard model of 12,661 exploitation attempts across 847,126 observations of 883 vulnerabilities in 960 firms; robust standard errors in parentheses; analysis stratified across 960 firms; two-tailed significance levels: $^{*}p < 0.05$; $^{**}p < 0.01$; $^{***}p < 0.001$

Table 3 details the results of the exploitation attempt diffusion using non-linear least-squares estimation of parameters. Model 0 describes the effects of the control variables only on the overall penetration (P), rate of diffusion (R), and delay (D). Model 1 introduces the focal indicator variable *OpenSource*. I find support for Hypothesis 2 that attackers attempt exploitation on open source products earlier ($\beta_{12} = -13.992$, $p < 0.001$). I also find support for Hypothesis 3 that attackers attempt exploitation on a larger number of firms for open source software products ($\beta_{12} = 46.995$, $p < 0.001$).

The non-linear parameter estimates in Table 3 are not straightforward to interpret. To clar-

Table 3 Diffusion of Exploitation Attempts through a Sample of Firms

		Model 0			Model 1		
	Variable	P	R	D	P	R	D
β_0	Constant	85.388^{***}	-0.038^{***}	9.208^{***}	167.409^{***}	-0.206^{***}	117.796^{***}
		(3.139)	(0.004)	(0.722)	(3.697)	(0.028)	(13.203)
β_1	Confidence Impact	-17.262^{***}	0.016^{***}	-2.463^{***}	86.082^{***}	-0.363^{***}	26.190^{***}
		(2.433)	(0.002)	(0.388)	(3.419)	(0.040)	(3.014)
β_2	Integrity Impact	-31.458^{***}	-0.027^{***}	1.466^{***}	-193.342^{***}	-0.620^{***}	-9.945^{***}
		(2.328)	(0.002)	(0.364)	(3.792)	(0.067)	(1.449)
β_3	Availability Impact	-49.810^{***}	-0.079^{***}	-6.166^{***}	-112.635^{***}	-0.406^{***}	-99.260^{***}
		(3.084)	(0.004)	(0.621)	(3.292)	(0.047)	(11.023)
β_4	Vuln: Access	-38.443^{***}	-0.037^{***}	3.584^{**}	6.702	1.545^{***}	26.222^{***}
		(4.546)	(0.003)	(1.095)	(14.877)	(0.001)	(1.095)
β_5	Vuln: Input Validation	9.002^{***}	-0.031^{***}	0.205	65.027^{***}	0.321^{***}	-0.590
		(1.139)	(0.002)	(0.193)	(1.198)	(0.036)	(0.335)
β_6	Vuln: Design	275.381^{***}	0.106^{***}	2.116^{***}	4.030^{**}	0.123^{***}	15.300^{***}
		(4.075)	(0.007)	(0.261)	(1.811)	(0.017)	(1.638)
β_7	Vuln: Exception	$7.6e4^{***}$	0.110^{***}	5.414	201.462^{***}	0.582^{***}	13.919^{***}
		(4.0e6)	(0.007)	(53.274)	(4.369)	(0.067)	(1.459)
β_8	Patch	57.854^{***}	0.043^{***}	0.833^{***}	32.821^{***}	-0.150^{***}	-2.746^{***}
		(1.130)	(0.003)	(0.228)	(1.117)	(0.018)	(0.510)
β_9	Complexity: Medium	42.489^{***}	0.020^{***}	-1.561^{***}	98.721^{***}	0.220^{***}	-0.466
		(1.402)	(0.001)	(0.205)	(1.488)	(0.026)	(0.410)
β_{10}	Complexity: High	7.933^{***}	-0.026^{***}	-1.664^{***}	-1.211	-0.462^{***}	1.709^{*}
		(1.398)	(0.003)	(0.265)	(1.349)	(0.052)	(0.593)
β_{11}	Signature	51.607^{***}	0.001^{***}	-0.002	133.025^{***}	1.370^{***}	-31.127^{***}
		(1.187)	(0.001)	(0.088)	(1.437)	(0.149)	(3.554)
β_{12}	OpenSource				46.995^{***}	0.251^{***}	-13.992^{***}
					(1.261)	(0.029)	(1.708)

83,806 daily observations of vulnerabilities exploited in at least one of 960 firms. Nonlinear regression on the cumulative number of affected firms, $N(t) = \frac{P}{1+e^{(-Rt-D)}}$. Penetration ($P$), rate ($R$), and delay ($D$) are modeled as linear functions of the above covariates. Robust standard errors in parentheses; two-tailed significance levels: $^{*}p < 0.05$; $^{**}p < 0.01$; $^{***}p < 0.001$

ify the estimates, Figure 1 graphs the resultant curves at the mean values of the covariates and illustrates the differences in open source versus closed source software. The figure shows both the acceleration in diffusion and overall increase in penetration. The graph strongly supports Hypothesis 2 and Hypothesis 3. It illustrates that exploitation attempts on open source software occur approximately three days sooner and the overall penetration is increased by approximately 50%. The figure particularly clarifies the parameter estimate for *OpenSource*. Based on Table 3, the effect of *OpenSource* appears to be much larger— 14 days sooner. However, because of differences

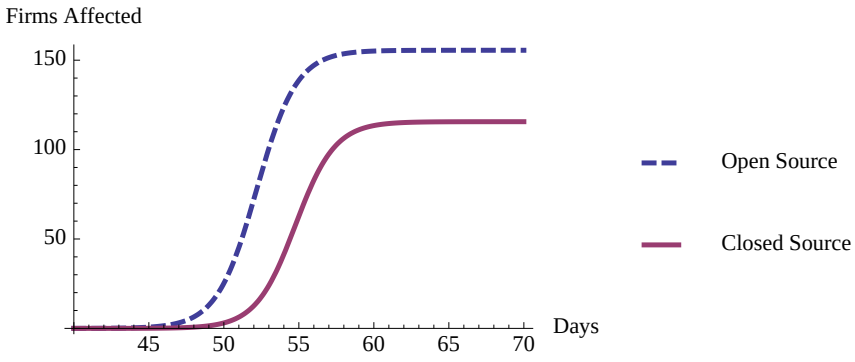

Figure 1 Diffusion of Exploitation Attempts

in the types of vulnerabilities reported for open source and closed source, the increase strictly due to *OpenSource* is smaller. This analysis encourages an interesting topic for future research—there may be underlying theoretical reasons for the differences in types of vulnerabilities reported. These differences may lead to differences in diffusion patterns. Interestingly, once diffusion starts, it is rapid with the curves almost vertical; the rate of diffusion of open source products is slightly greater. Overall, the rapid diffusion underscores the importance of additional time to implement deterrence measures. Interestingly, the availability of a signature reduces delay ($\beta_{11} = -31.127$, $p < 0.001$) and increases penetration ($\beta_{11} = 133.025$, $p < 0.001$), indicating that attackers can use signatures in a similar way that open access to source code may be used, reverse engineering a vulnerability to develop attack tools.

4.3. Volume of Exploitation Attempts

To evaluate Hypothesis 4, I use a two-stage Heckman model to analyze the number of alerts generated by a vulnerability for a specific firm. I construct a data set that has for each firm (960 firms) and each vulnerability (883 vulnerabilities), the number of alerts generated on each day of the research period. The data set is not balanced; I do not include observations prior to the publication of the vulnerability. Therefore, some vulnerabilities have longer periods for which data is available, and the total number of observations is 896,407. Also, many vulnerabilities are never exploited during the study period and ordinary least squares estimation will ignore the selection bias. The two-stage model incorporates selection bias in the volume of attacks. In the first

stage, I use a selection model to investigate vulnerability attributes that affect overall likelihood of exploitation. I control for all vulnerability covariates and further include monthly fixed effects based on vulnerability publication date to control for possible changes in exploitation propensity over the two-year sample period. In the second stage, I estimate the number of alerts per day (with a natural log transformation). In this analysis, I control for all vulnerability covariates (including log-transformed number of days since vulnerability disclosure). I include monthly fixed effects based on alert date to control for changes in attack behavior over time and include 960 firm fixed effect indicators to control for potential differences in a firms inherent risk of attack. In the two models below, i indexes a firm, k indexes a vulnerability, and $E_k = 1$ if vulnerability k is ever exploited in the alert data, 0 otherwise. F_i^2 are firm fixed effect dummies, while M_k^1 in stage 1 is the month fixed effects based on the vulnerability publication date, and M_t^2 in stage 2 is the month fixed effect based on the attack date.

$$E_k = \alpha^1 + \beta_1^1 OpenSource + \beta_2^1 ln(Age_k) + M_k^1 + control\_variables \tag{8}$$

$$ln(V_{i,k,t}) = \alpha^2 + \beta_1^2 OpenSource + F_i^2 + M_t^2 + control\_variables \tag{9}$$

Table 4 describes the results from the two stage Heckman analysis. In stage 2 (Model 1), the coefficient of the *OpenSource* variable is significant and positive ($\beta_{13} = 0.148$, $p < 0.001$), indicating an increase in exploitation attempt volume for vulnerabilities in open source software. Thus, the results support Hypothesis 4. The results from stage 1 (Model 1) provide additional support for Hypothesis 1, and indicate that open source vulnerabilities have an increased likelihood of exploitation ($\beta_{13} = 0.072$, $p < 0.001$). Interestingly, the availability of signatures increases the likelihood of vulnerability exploitation ($\beta_{11} = 0.832$, $p < 0.001$ in stage 1), providing additional evidence that attackers learn to exploit a vulnerability by reverse engineering the associated signatures.

Because exploitation attempts are count variables, alternative models such as poisson or negative binomial could be used. Although unreported, these models are consistent with the Heckman results; the coefficient for the *OpenSource* variable is positive in the poisson ($\beta = 0.882$, $p < 0.001$) and negative binomial ($\beta = 0.611$, $p < 0.001$) models.

Table 4 Volume of Exploitation Attempts Per Firm

		Model 0		Model 1	
	Variable	Stage 1	Stage 2	Stage 1	Stage 2
β_0	Constant	-0.136^{***}	0.687^{***}	-0.115^{***}	0.678^{***}
		(0.009)	(0.023)	(0.009)	(0.023)
β_1	Confidence Impact	-0.274^{***}	0.120^{***}	-0.270^{***}	0.122^{***}
		(0.005)	(0.004)	(0.005)	(0.004)
β_2	Integrity Impact	0.510^{***}	-0.186^{***}	0.505^{***}	-0.188^{***}
		(0.005)	(0.005)	(0.005)	(0.005)
β_3	Availability Impact	-0.058^{***}	-0.001	-0.055^{***}	-0.005
		(0.005)	(0.004)	(0.005)	(0.004)
β_4	Vuln: Access	-0.830^{***}	0.242^{***}	-0.826^{***}	0.190^{***}
		(0.009)	(0.008)	(0.009)	(0.003)
β_5	Vuln: Input Validation	-0.037^{***}	0.131^{***}	-0.032^{***}	0.131^{***}
		(0.004)	(0.003)	(0.004)	(0.027)
β_6	Vuln: Design	-0.125^{***}	-0.092^{***}	-0.124^{***}	-0.094^{***}
		(0.005)	(0.003)	(0.005)	(0.003)
β_7	Vuln: Exception	-0.315^{***}	-0.133^{***}	0.320^{***}	-0.206^{***}
		(0.006)	(0.004)	(0.006)	(0.004)
β_8	Patch	-0.070^{***}	-0.031^{***}	-0.068^{***}	-0.042^{***}
		(0.003)	(0.002)	(0.003)	(0.002)
β_9	Complexity: Medium	-0.176^{***}	-0.065^{***}	-0.177^{***}	-0.049^{***}
		(0.004)	(0.003)	(0.004)	(0.003)
β_{10}	Complexity: High	0.407^{***}	-0.115^{***}	0.411^{***}	-0.149^{***}
		(0.005)	(0.004)	(0.005)	(0.004)
β_{11}	Signature	0.843^{***}	-0.067^{***}	0.832^{***}	-0.001
		(0.004)	(0.004)	(0.004)	(0.004)
β_{12}	Age (ln) at Event		-0.176^{***}		-0.199^{***}
			(0.004)		(0.004)
	Monthly Fixed Effects	publish date	alert date	publish date	alert date
	Firm Fixed Effects		960 firms		960 firms
β_{13}	OpenSource	0.105^{***}		0.072^{***}	0.148^{***}
		(0.004)		(0.004)	(0.003)
	Log pseudo-likelihood		-1164115		-1163486

Heckman two stage regression; $n = 896,407$; 473,699 uncensored; 883 vulnerabilities; robust standard errors in parentheses; two-tailed significance levels: $^{*}p < 0.05$; $^{**}p < 0.01$; $^{***}p < 0.001$
Stage 1: uncensored if exploit attempt for the vulnerability is observed in the sample
Stage 2: natural log of the number of exploitation attempts

5. Conclusion

My theoretical development and empirical results indicate that, compared with closed source software, vulnerabilities in open source software: (a) have increased risk of exploitation, (b) diffuse sooner and with higher total penetration, and (c) increase the volume of exploitation attempts.

5.1. Exploitation of Vulnerabilities in Open Source Software

While the wide visibility of open source software code may be highly beneficial in eliminating vulnerabilities before release, this same visibility can be problematic in the exploitation stage. Once a vulnerability is discovered, attackers and defenders are in an innovation race. Attackers must develop exploits based on the vulnerability. Defenders, on the other hand, must notify vendors, correct code, create releases, and convince users to patch systems. The available of source code makes the attackers task easier by reducing the time to create exploits and enhancing the quality of those exploits.

From the empirical analysis, I conclude that the exploitation process is accelerated for open source products. From a diffusion of innovation perspective, the availability of source code reduces attacker exploitations costs and increases effectiveness. This amplified risk is evident in all three empirical models. However, it would be incorrect to conclude that open source is strictly worse for software security. The benefits from open source remain at the pre-release stage. Although these are difficult to quantify, it is likely that the benefits of open source outweigh the negative effects in the exploitation stage. Furthermore, security, while important, is not the only consideration in choice of software development models.

5.2. Reporting of Open Source Vulnerabilities

Given the trade-off between positive and negative aspects of open source software security, it would be ideal to combine the benefits of pre-release vulnerability discovery in open source products with the benefits of closed source exploitation reduction. Are there less public vulnerability disclosure mechanisms for open source products? Even if vulnerabilities could be privately handled initially, a source code correction would be necessary at some point; by definition, this is public. This situation is exacerbated by the reuse of open source products. Open source products are frequently used in other open source (or proprietary) products. Not only must the vulnerability be corrected in the initial source, it must be propagated through derivative products, released and installed. These steps give attackers more time, further increasing the expected benefits for the attacker.

5.3. Limitations and Future Research

This study has several limitations that future research may be able to address.

First, while I observe the exploitation of vulnerabilities, I cannot assess the discovery of vulnerabilities. While many believe that the increased number of viewers helps reduce the number of vulnerabilities in open source software, it is difficult to measure. Fundamentally, it is difficult to compare any two software products because many attributes (e.g. user base, complexity, feature set, etc.) are unknown or are difficult to measure. While these limitations continue to exist in the exploitation stage, they are more important in the discovery stage. In the exploitation stage, researchers know the total number of discovered vulnerabilities that could be exploited. In the discovery stage, the total number of undiscovered vulnerabilities is, by definition, unknown. Prior to release, it is difficult to quantify how many vulnerabilities would be typically expected in a software product. These inherent uncertainties make direct evaluation of the security benefits of open source software difficult.

Second, the descriptive statistics indicate that open and closed source products may have differences in the types of vulnerabilities found. It may be that some kinds of bugs are more shallow than others. While few differences have been found in vulnerabilities disclosed in open versus closed source (Schryen 2010), there may be differences in exploitation. Detailed analysis of vulnerability characteristics would be of theoretical and practical interest.

Third, while the IDS and NVD data used in this research is detailed, it has several limitations. IDS data contains a large number of false positives and false negatives. It is noisy and error-prone; IDS software and signatures are imperfect. Furthermore, the categorization of vulnerabilities in the NVD database is often subjective.

Fourth, the availability of source may also benefit defenders. For example, the developers of IDS signatures may find their task easier with the availability of source code. If so, the results of this study would be stronger since we observe the net effect of source code visibility. It would be insightful through future research to quantify and distinguish the benefits to defenders.

Fifth, while open source software is frequently reused as a component, closed source software is as well. The reuse and associated dependencies described in Section 5.2 may provide an increased incentive for attackers. Detailed analysis of exploitations based on embedded components would also be particularly interesting, both theoretically and practically.

Despite these limitations, managerial and policy implications can be gleaned from detailed empirical analysis. Unfortunately, there is little empirical analysis using real, cross-organizational data despite recognized need (Ransbotham and Mitra 2009, Schechter 2005). Most importantly, this study has only identified and quantified the effect of open source on the exploitation stage. Clearly, there are important disclosure policy implications. Future research is needed to propose and evaluate alternative disclosure mechanisms cognizant of the effects I find.

Acknowledgments

I thank SecureWorks, Inc., for making an anonymized subset of their historical data available to me. All errors and opinions in this research are entirely my responsibility.

References

Arora, A., J. Caulkins, R. Telang. 2006. Sell First, Fix Later: Impact of Patching on Software Quality. *Management Science* **52**(3) 465–471.

Arora, Ashish, Ramayya Krishnan, Anand Nandkumar, Rahul Telang, Yubao Yang. 2004a. Impact of Vulnerability Disclosure and Patch Availability— An Empirical Analysis. Workshop on the Economics of Information Security.

Arora, Ashish, Ramayya Krishnan, Rahul Telang, Yubao Yang. 2005. An Empirical Analysis of Vendor Response to Disclosure Policy. Workshop on the Economics of Information Security.

Arora, Ashish, Rahul Telang, Hao Xu. 2004b. Optimal Policy for Software Vulnerability Disclosure. Workshop on the Economics of Information Security.

August, Terrence, Tunay Tunca. 2008. Let the Pirates Patch? An Economic Analysis of Software Security Patch Restrictions. *Information Systems Research* **19**(1).

Baskerville, Richard. 1993. Information Systems Security Design Methods: Implications for Information Systems Development. *ACM Computing Surveys* **25**(4) 375–414.

Bloch, F., P. Markowitz. 1996. Optimal Disclosure Delay in Multistage R&D Competition. *International Journal of Industrial Organization* **14**(2) 159.

Bohme, Rainer. 2006. *Emerging Trends in Information and Communications Security*, *Lecture Notes in Computer Science*, vol. 3995/2006, chap. A Comparison of Market Approaches to Software Vulnerability Disclosure. Springer, 238–311.

Cavusoglu, H., B. Mishra, S. Raghunathan. 2005a. The Value of Intrusion Detection Systems in Information Technology Security Architecture. *Information Systems Research* **16**(1) 28–46.

Cavusoglu, Hasan, Huseyin Cavusoglu, Srinivasan Raghunathan. 2005b. Emerging Issues in Responsible Vulnerability Disclosure. Workshop on the Economics of Information Security.

Cavusoglu, Hasan, Huseyin Cavusoglu, Srinivasan Raghunathan. 2007. Efficiency of Vulnerability Disclosure Mechanisms to Disseminate Vulnerability Knowledge. *IEEE Transactions on Software Engineering* **33**(3) 171–185.

Cavusoglu, Hasan, Huseyin Cavusoglu, J. Zhang. 2008. Security Patch Management: Share the Burden or Share the Damage? *Management Science* **54**(4) 657–670.

Choi, Jay Pil, Chaim Fershtman, Neil Gandal. 2005. Internet Security, Vulnerability Disclosure, and Software Provision. Workshop on the Economics of Information Security.

Christey, S., R.A. Martin. 2007. Vulnerability Type Distributions in CVE. Http://cwe.mitre.org/documents/vuln-trends/, Accessed 22 February 2010.

Constant, D., L. Sproull, S. Kiesler. 1996. The Kindness of Strangers: The Usefulness of Electronic Weak Ties for Technical Advice. *Organization Science* **7**(2) 119–135.

Cox, D. R. 1972. Regression models and life tables. *Journal of the Royal Statistical Society, Series B* **34**(2) 187–202.

Dhillon, Gurpreet, James Backhouse. 2001. Current Directions in IS Security Research: Towards Socio-Organizational Perspectives. *Information Systems Journal* **11**(2) 127–153.

Frei, S., M. May, U. Fiedler, B. Plattner. 2006. Large-scale Vulnerability Analysis. *Proceedings of the 2006 SIGCOMM workshop on Large-scale attack defense*. ACM Press, 131–138.

Fudenberg, D., R. Gilbert, J. Stiglitz, J. Tirole. 1983. Preemption, Leapfrogging and Competition in Patent Races. *European Economic Review* **22**(1) 3–31.

Gordon, L.A., M.P. Loeb. 2006. Economic Aspects of Information Security: An Emerging Field of Research. *Information Systems Frontiers* **8**(5) 335–337.

Gordon, Lawrence A., Martin P. Loeb. 2002. The Economics of Information Security Investment. *ACM Transactions on Information and System Security* **5**(4) 438 – 457.

Gordon, Lawrence A., Martin P. Loeb, Tashfeen Sohail. 2010. Market Value of Voluntary Disclosures Concerning Information Security. *MIS Quarterly* forthcoming.

Jones, J.R. 2007. Estimating Software Vulnerabilities. *IEEE Security and Privacy* **5**(4) 28–32.

Kalbfleisch, John D., Ross L. Prentice. 2002. *The Statistical Analysis of Failure Time Data*. 2nd ed. Wiley-Interscience.

Kane, Gerald C., Sam Ransbotham. 2009. The Influence of Network Structure on the Quality of Peer Produced Medical Information. *Workshop on Information in Networks*. New York.

Kannan, K., R. Telang. 2005. Market for Software Vulnerabilities? Think Again. *Management Science* **51**(5) 726–740.

Kotenko, Igor, Mikhail Stepashkin. 2006. *Attack Graph Based Evaluation of Network Security*, *Lecture Notes in Computer Science*, vol. 4237. Springer, Berlin.

Lawton, G. 2002. Open Source Security: Opportunity or Oxymoron? *IEEE Computer* **35**(3) 21.

Li, P., H.R. Rao. 2007. An Examination of Private Intermediaries Roles in Software Vulnerabilities Disclosure. *Information Systems Frontiers* **9**(5) 531–539.

Lohmeyer, D. F., J. McCrory, S. Pogreb. 2002. Managing Information Security. *McKinsey Quarterly* **2002**(Special Edition 2) 12–16.

Mell, Peter, Sasha Romanosky. 2008. A Complete Guide to the Common Vulnerability Scoring System Version 2.0. Http://www.first.org/cvss/cvss-guide.html, Accessed 8 March 2008.

Mell, Peter, Karen Scarfone, Sasha Romanosky. 2006. Common Vulnerability Scoring System. *IEEE Security and Privacy* **4**(6) 85–89.

Meneely, A., L. Williams. 2009. Secure Open Source Collaboration: An Empirical Study of Linus' Law. *Proceedings of the 16th ACM conference on Computer and Communications Security*. 453–462.

Meneely, A., L. Williams, W. Snipes, J. Osborne. 2008. Predicting Failures with Developer Networks and

Social Network Analysis. *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. 13–23.

NVD. 2008. National Vulnerability Database. Http://nvd.nist.gov/, Accessed 8 March 2008.

Ozment, Andy. 2004. Bug Auctions: Vulnerability Markets Reconsidered. *Third Workshop on the Economics of Information Security* .

Ransbotham, Sam, Sabyasachi Mitra. 2009. Choice and Chance: A Conceptual Model of Paths to Information Security Compromise. *Information Systems Research* **20**(1) 121–139.

Ransbotham, Sam, Sabyasachi Mitra, Jon Ramsey. 2010. Are Markets for Vulnerabilities Effective? Working paper.

Raymond, Eric S. 1997. The Cathedral and the Bazaar. Http://www.catb.org/ esr/writings/cathedral-bazaar/cathedral-bazaar/ar01s04.html, Accessed 21 February 2010.

Rescorla, E. 2005. Is Finding Security Holes a Good Idea? *IEEE Security & Privacy* 14–19.

Rogers, Everett M. 2003. *Diffusion of Innovations*. 5th ed. Free Press, New York.

Schechter, Stuart. 2004. Toward Econometric Models of the Security Risk from Remote Attacks. Workshop on the Economics of Information Security.

Schechter, Stuart. 2005. Toward Econometric Models of the Security Risk from Remote Attack. *IEEE Security & Privacy* **3**(1) 40–44.

Schryen, G. 2009. Security of Open Source and Closed Source Software: An Empirical Comparison of Published Vulnerabilities. *Proceedings of Americas Conference on Information Systems, San Francisco, California*.

Schryen, G., E. Rich. 2010. Increasing Software Security through Open Source or Closed Source Development? Empirics Suggest that We have Asked the Wrong Question. *Proceedings of the 43rd Hawaii International Conference on System Sciences*. 1–10.

Schryen, Guido. 2010. Is Open Source Security a Myth? What do Vulnerability and Patch Data Say? *Communications of the ACM* Forthcoming.

Straub, Detmar W., Richard J. Welke. 1998. Coping with Systems Risk: Security Planning Models for Management Decision Making. *MIS Quarterly* **22**(4) 441–469.