ECONOMICS OF SECURITY PATCH MANAGEMENT

Huseyin Cavusoglu†[*] Hasan Cavusoglu‡ Jun Zhang†
†A.B. Freeman School of Business, Tulane University
7 McAlister Drive, New Orleans, LA 70118, USA
‡Sauder School of Business, The University of British Columbia
2053 Main Mall, Vancouver, BC V6T1Z2, CANADA
huseyin@tulane.edu, cavusoglu@sauder.ubc.ca, jzhang4@tulane.edu

Abstract
Patch management is a crucial component of IT security programs. An important problem within this context is to determine how often to update the systems with necessary patches. Keeping the systems patched with more frequent patch updates increases operational costs while reducing security risks. On the other hand, leaving the systems unpatched with less frequent patch updates decreases operational costs while increasing security risks. In this paper we develop a game theoretic model to derive the optimal frequency of patch updates to balance the operational costs and damage costs associated with security vulnerabilities. We first analyze a centralized system in a benchmark case to find the socially optimal patch management policy and associated patch release cycle of the vendor and patch update cycle of the firm. Then we consider a noncentralized system in which the vendor determines its patch release policy and the firm selects its patch update policy in a Stackelberg framework. Given the results in centralized and noncentralized patch management, we next address how we can coordinate the patch release policy of the vendor and the patch update policy of the firm using cost sharing and/or liability to achieve the socially optimal patch management in a noncentralized setting.

Keywords: *Patch management, patch update cycle, patch release cycle, nested policies, coordination, cost sharing, liability*

1. Introduction

Today most security incidents are caused by flaws in software, called vulnerabilities. It is estimated that there are as many as 20 flaws per thousand lines of software code (Dacey, 2003). Computer Emergency Response Team/Coordination Center (CERT/CC) statistics reveal that the number of vulnerabilities reported has increased dramatically over the years, from only 171 in 1995 to 3784 in 2003 (CERT, 2004).

The ultimate solution to software vulnerabilities is application of patches. Software vendors generally release patches to fix vulnerabilities in their software products. These patches, if applied correctly, remove vulnerabilities from the systems. However, many systems are left unpatched for months, even years (Shostack, 2003). According to CERT/CC, around 95 percent of security breaches could be prevented by keeping systems up-to-date with appropriate patches (Dacey, 2003). The Slammer virus, which swept the Internet in January 2003, caused network outages all around the world, affecting 911 call centers, airlines, ATMs. But Microsoft had released the patch fixing the vulnerability that Slammer exploited six months before the incident. Similarly, Code Red and Nimda wreaked havoc on those companies that were not current with their software patch updates (McGhie, 2003).

So if patches are the panacea to software vulnerabilities, why don't firms apply them as soon as vendors make them available? To begin with, patch management is multifaceted, and there are several operational reasons for not applying patches immediately. First, there are too many vulnerabilities to patch. In an average week, vendors and security organizations announce around one hundred and fifty vulnerabilities

[*] corresponding author

along with information on how to fix them (McGhie, 2003). Sorting through all these vulnerabilities to find relevant ones is a tedious and labor-intensive job. Second, patches cannot be trusted without testing (Donner, 2003). Before being applied in production environments, each patch must be tested to make sure that it is working properly and does not conflict with other existing applications in the system. In some cases, firms need to reconfigure the system and/or recode some applications so that the patch can work without causing any new problem. Third, distribution of patches is not standard. Some patches are available on the vendor web site and some patches are not. Even if they are available, firms are too busy to continually check the vendor site. Fourth, every patch requires installation after testing. This mostly means taking the system down and restarting (rebooting) it. If the patch is applied to a critical system in a production environment, downtime can be very expensive. Therefore, updating systems with patches is costly. On the other hand, the consequences of not updating systems promptly with necessary patches can be severe too. The time window between identification of vulnerabilities and creation of exploits has shrunk dramatically over the years. Once an exploit is released, it may be too late to consider patching. For instance, Slammer exploited 90% of vulnerable systems within 10 minutes of its release (Dacey, 2003). Code Red infected a total of 359,000 computers within 14 hours of its release (McGhie, 2003), just to name a few. So firms must act fast to avoid falling victim to malicious acts.

Since firms can update their vulnerable systems only with available patches, the way software vendors release patches has a profound effect on how firms manage the patching process. Although the common practice in the past was to release patches as soon as they were ready, in an effort to ease the burden on system administrators struggling with frequency of updates, and to make the process more predictable, today many software vendors follow period patch release policies (Pruitt, 2003). For example, Microsoft changed its schedule and switched to a monthly patch release cycle in October 2003. Computer Associates and PeopleSoft, Inc. issue patches quarterly (Foley and Hulme, 2004).

Given the complexities surrounding patch management, quantitative models are needed to help firms determine the optimal patch management strategy (Donner, 2003). In this paper we specifically study this problem to understand how often systems should be updated with released patches. We first study a benchmark scenario to determine the optimal patch release and patch update policies for a centralized system consisting of a software vendor and a software user (firm). Next we analyze a noncentralized system in which patch release and patch update decisions are made by the vendor and the firm, individually in a Stackelberg game framework, where the vendor announces its patch release policy first, and then the firm reacts to it by choosing its patch update policy.

Given that patch management is costly and vulnerabilities are defects caused by software vendors, recently security experts started questioning the implications of cost sharing in patch management. Since firms currently bear the cost of patching, and firms cannot keep up with the sheer number of patches released by vendors every day, it may help firms if software vendors share this burden (Farber, 2003). The argument is very simple: Since you do not have to pay to repair your car when a manufacturer defect, such as faulty brakes, is found, why should firms pay for the cost of patching? What if the vendor's release policy prevents the patch from being released? Then the question changes to a liability issue in security. Using the previous analogy, if faulty breaks cause you to be in an accident, the car manufacturer can be held liable in court. What if your server is attacked because of a specific vulnerability for which a patch has not yet been released by the vendor? Therefore we examine cost sharing and liability as possible coordination schemes to achieve the socially optimal levels of patch release and update cycles.

2. Model Basics

We study a simple system consisting of one software vendor and one firm that uses the software. We assume that identification of vulnerabilities follows a Poisson process with an arrival rate of λ. The firm updates its system with new patches once in every T_f time units (i.e., the firm's patch update cycle is T_f).

The firm incurs two types of cost in patch management: damage cost as a result of exploitation of vulnerabilities not patched and patch update cost associated with identifying, testing, and installing patches. The objective of the firm is to minimize the sum of damage cost and patch update cost by choosing an appropriate patch update cycle. Damage cost is incurred because either (i) the vendor waits to release patches until its next release cycle for identified vulnerabilities or (ii) the firm does not update its system with released patches until its next update cycle. We call the period between the time at which a vulnerability is identified and the time at which a patch is released the pre-release period. Similarly, we call the period between the time at which a patch is released and the time at which the firm applies the patch the post-release period. We assume that the damage incurred is proportional to the duration of pre- or post-release periods. We define c_b and c_a as the damage cost per unit time in the pre- and post-release periods, respectively. Every time the firm updates its system with missing patches, it incurs (i) a fixed cost, K_f, which captures the cost associated with identification of missing patches and the cost of downtime during installation of tested patches and (ii) a variable cost, nc_f, due to testing and configuration changes and/or recoding, and installation, where n is the number of missing patches.

In our model the vendor releases patches to fix vulnerabilities in its software once in every T_v time units (i.e., the vendor's patch release cycle is T_v). The vendor incurs two types of cost: patch release cost associated with developing patches for its identified vulnerabilities and reputation cost for having vulnerable software. The vendor minimizes the sum of patch release cost and reputation cost by choosing an appropriate patch update cycle. In order to develop a patch for a vulnerability, the vendor must assign some resources. This is the variable component of patch release cost, nc_v. There is also a fixed cost of informing the public about patches, making patches available, and other PR costs, denoted by K_v. The vendor incurs a cost in term of reputation loss. This cost includes the loss in future sales because of reduction in perceived quality of software and trust in software by the firm. We define α_b and α_a as the reputation cost per unit time of exposure in pre- and post-release periods, respectively.

In this paper, we consider a nested policy. A patch management policy is nested if the patch update cycle of the firm is a multiple of the patch release cycle of the vendor; that is, one patch update cycle consists of k patch release cycles ($T_f = kT_v$).

3. The Integrated System

In the integrated system, a central planner decides on the patch release cycle and patch update cycle to minimize the total system cost. We can write the total expected system cost during one update cycle conditional on n patches released within this update cycle as

$$L^I(T_v,T_f \mid n) = K_f + c_f n + kK_v + c_v n + c_b n T_v/2 + c_a n(k-1)T_v/2 \tag{1}$$

Because identification process is Poisson, the total expected system cost during one update cycle is

$$L^I(T_v,T_f) = \sum_{n=0}^{\infty} L^I(T_v,T_f \mid n)\frac{(\lambda T_f)^n}{n!}e^{-\lambda T_f} = K_f + kK_v + (c_f + c_v + c_b T_v/2 + c_a(k-1)T_v/2)\lambda T_f \tag{2}$$

It follows that the expected average cost per unit time is

$$C^I(T_v,T_f) = L^I(T_v,T_f)/T_f = \frac{K_f}{T_f} + \frac{c_a}{2}\lambda T_f + \frac{K_v}{T_v} + (\frac{c_b - c_a}{2})\lambda T_v + (c_f + c_v)\lambda \tag{3}$$

Thus, the central planner's problem can be stated as

$$\min_{T_v \ge 0, T_f \ge 0} \{C^I(T_v,T_f) \mid T_f = kT_v \text{ for any } k\}.$$

The following result characterizes the optimal patch management policy when T_f and T_v are determined together by the central planner.

Proposition 1. Let T_v^* and T_f^* be the optimal patch release and update cycles for the integrated system, respectively. Then,

$$T_v^* = T_f^* = \sqrt{\frac{2(K_v + K_f)}{\lambda c_b}}\,, \tag{4}$$

and the minimum expected average system cost is $C^I(T_v^*, T_f^*) = \sqrt{2\lambda(K_v + K_f)c_b} + \lambda(c_f + c_v)$.

Proposition 1 states that at social optimality patch release and update cycles must be synchronized, that is, patches must be applied to the system as soon as they are released by the vendor. In practice, T_f and T_v are determined by the firm and the vendor individually. In the next section, we discuss how patch release and update cycles change when these are determined in a noncooperative setting.

4. The Decentralized System

To solve this game, we work backward and first focus on the firm's patch update problem, assuming a patch release cycle T_v is given.

4.1 The Firm's Problem

Given that the firm follows a nested policy, that is, $T_f = kT_v$, one patch update cycle of the firm is divided into k patch release cycles. Because the release times of n patches are uniformly distributed within *kTv* time units, (*n1, . . . , nk*) follows a multinomial distribution Hence, the firm's expected average cost per unit time is

$$C_f(T_v, k) = c_b T_v \lambda / 2 + c_a \lambda (k-1) T_v / 2 + K_f / (kT_v) + c_f \lambda \tag{5}$$

For a given T_v, the firm's problem can be formulated as

$$\min_k \quad \{c_b T_v \lambda / 2 + c_a \lambda (k-1) T_v / 2 + K_f / (kT_v) + c_f \lambda \mid \ k \ \text{ is } \ \text{integer}\}$$

Lemma 1 characterizes the optimal patch update policy for the firm.

Lemma 1. For a given patch release cycle T_v of the vendor, k satisfies

$$k^*(k^*-1) < \frac{2K_f}{\lambda c_a T_v^2} \qquad \text{and} \qquad k^*(k^*+1) \geq \frac{2K_f}{\lambda c_a T_v^2} \tag{6}$$

Given the firm's best response, we next study how the vendor chooses its patch release cycle.

4.2. The Vendor's Problem

Taking firm's response k^* into consideration, the vendor chooses its patch release cycle T_v to minimize its average cost. The average expected cost for vendor can be written as

$$C_v(T_v, k^*) = K_v / T_v + (\alpha_b - \alpha_a)\lambda T_v / 2 + \alpha_a \lambda k^* T_v / 2 + c_v \lambda \tag{7}$$

The vendor chooses T_v to minimize $C_v(T_v, k^*)$ for a given reaction k^*. Note that k^* is uniquely identified by constraints in (6); therefore, the vendor's problem can be formulated as

$$\min_{T_v, k} \quad K_v / T_v + (\alpha_b - \alpha_a)\lambda T_v / 2 + \alpha_a \lambda k T_v / 2 + c_v \lambda$$

$$s.t. \quad k(k-1) < \frac{2K_f}{\lambda c_a T_v^2} \qquad \text{and} \qquad k(k+1) \geq \frac{2K_f}{\lambda c_a T_v^2}$$

Unfortunately, there are no close-form solutions for the vendor's problem. Because our focus is on coordination schemes to achieve the socially optimal patch release and update cycles, and the patch release and update cycles are identical at social optimality, in the next proposition we characterize the region where the patch release and update cycles are identical without any coordination mechanism.

Proposition 2. If $K_f / c_a \le 2K_v / \alpha_b$, then the optimal patch release cycle is $T_v^{p*} = \sqrt{2K_v /(\lambda\alpha_b)}$ and the corresponding patch update cycle T_f^{p*} is equal to T_v^{p*}.

Note that K_f / c_a can be interpreted as the tolerance level of the firm to wait for update, and K_v / α_b as the tolerance level of the vendor to wait for release without any coordination. When the firm's tolerance level is smaller than twice the vendor's tolerance level, patch update will be synchronized with patch release when both decisions are made individually. This result is very intuitive. Unless the firm is tolerant to wait till every other release, the firm chooses to patch its system immediately with every new release. However, one should note that even for this case, the synchronized patch release/update cycle is not necessarily identical to the synchronized cycle that minimizes the social cost. Therefore, patch update and patch release cycles are, in general, not socially optimal when they are determined separately. In the next section, we study possible coordination schemes that can achieve the socially optimal patch management.

5. Coordination Schemes

Through cost sharing, every time a patch update occurs, ϕ^c $(0 \le \phi^c \le 1)$ portion of the fixed cost for patch update is charged to the vendor. Through liability, every time a vulnerability is exploited before the patch that fixes the vulnerability is released, the vendor has to pay ϕ^l $(0 \le \phi^l \le 1)$ portion of the damage cost incurred by the firm. For both coordination schemes, we first characterize the optimal patch release and update cycles and then develop sufficient conditions under which the socially optimal patch release/update can be achieved. We summarize results below.

Proposition 3 (*Cost Sharing Only*). Cost sharing by itself can achieve the socially optimal patch update and release if

$$\frac{K_v}{\alpha_b} \le \frac{K_v + K_f}{c_b} \le \frac{K_v}{\alpha_b}\frac{\alpha_a + \alpha_b}{\alpha_a - \alpha_b}.$$

Then, the required level of cost sharing is $\phi^c = (K_v + K_f)\alpha_b /(K_f c_b) - K_v / K_f$.

Proposition 4 (*Liability Only*). Liability by itself can achieve the socially optimal patch update and release if

$$\frac{K_v + K_f}{c_b} \le \frac{K_v}{\alpha_b}.$$

Then, the required level of liability is $\phi^l = K_v /(K_v + K_f) - \alpha_b / c_b$.

Proposition 5 (*Cost Sharing plus Liability*). Cost sharing together with liability can achieve the socially optimal patch release and update if

$$\frac{K_v + K_f}{c_b} \le \frac{K_v}{\alpha_b}\frac{\alpha_a + \alpha_b}{\alpha_a - \alpha_b}.$$

And the levels of cost sharing and liability satisfy $(K_v + \phi^c K_f)/(\alpha_b + \phi^l c_b) = (K_v + K_f)/c_b$.

Propositions 3, 4, and 5 characterize the conditions under which a central planner can use cost sharing or liability or both to achieve the socially optimal patch release and update. Our analysis shows that what coordination scheme to use depends on where the tolerance level of the system falls. When the system's tolerance level is lower than the vendor's tolerance level with maximum cost sharing, there is a coordination mechanism(s) that can achieve the socially optimal patch management. If the system's tolerance level is lower than the vendor's tolerance level without cost sharing or liability, then liability by itself results in the socially optimal patch release/update. On the other hand, if the system's tolerance level is higher than the vendor's tolerance level without cost sharing or liability, then cost sharing by itself may lead to the socially optimal patch release/update. If the system's tolerance level is very high, neither cost sharing nor liability can coordinate the vendor's patch release and the firm's patch update decisions. Interestingly, for these scenarios, the joint use of these two schemes cannot achieve the socially optimal patch release/update cycle either. Therefore, using both coordination schemes does not enlarge the feasible region to achieve social optimality compared to using cost sharing and liability individually. Proposition 5 implies that cost sharing and liability are not substitutes for each other; that is, using liability in addition to cost sharing does not reduce the level of cost sharing required to achieve optimality, or vice versa. Therefore the central planner should never use both coordination schemes at the same time to coordinate patch management decisions

6. Conclusions

In this paper we develop a game theoretic model to derive the optimal frequency of patch updates. Our analysis reveals that the socially optimal patch management requires synchronization of patch release and update cycles. When decisions for patch release and update policies are left to the vendor and the firm, individual decision making may result in asynchronized cycles for patch management. Even if patch release and update cycles are synchronized, the synchronized cycle is not the optimal patch release/update cycle. Further, our analysis demonstrates that cost sharing can always synchronize patch release and patch update cycles whereas liability can never be used to synchronize them. Since cost sharing is not a substitute for liability, and liability is not a substitute for cost sharing, using both cost sharing and liability together to achieve coordination does not add value other than causing the vendor to bear more cost. Therefore, depending on the incentive levels of the system and the vendors, vendors should share either the burden (cost sharing) or the damage (liability), but not both to reach the social optimality with minimum level of additional cost on the vendor side.

We assumed that there is only one vulnerable software program running on one computer. In reality, multiple computers with diverse risk profiles can be at risk from a specific vulnerability. To manage patch updates, firms can arrange these vulnerable machines into groups and apply different patch update policies. We also assumed that each vulnerability causes the same level of threat. This is a reasonable assumption given that there is no easy way for an administrator to tell whether a vulnerability is serious or very serious (Donner, 2003). Recently some vendors started attaching a severity rating to their vulnerabilities. Using this information, firms can devise patch update policies for vulnerabilities with different severity ratings. Future research should address above-mentioned problems.

References

CERT (2004). Cert/cc statistics 1988-2004. http://www.cert.org/stats/.

Dacey, R. F. (2003). *Effective patch management is critical to mitigating software vulnerabilities*. GAO-03-1138T.

Donner, M. (2003). Patch management - bits, bad guys, and bucks! *Secure Business Quarterly*, 3(2), 1-4.

Farber, D. (2003). *Should microsoft pay for your security patch costs*? TechUpdate, January 30.

Foley, J. and G. V. Hulme (2004). *Get ready to patch.* InformationWeek, August 30.

McGhie, L. (2003). Software patch management - the new frontier. *Secure Business Quarterly*, 3(2), 1-4.

Pruitt, S. (2003). *Software users hit a rough patch.* PC World, November 10.

Shostack, A. (2003). Quantifying patch management. *Secure Business Quarterly*, 3(2), 1-4.