# Timing the Application of Security Patches for Optimal Uptime

*Steve Beattie, Seth Arnold, Crispin Cowan, Perry Wagle, and Chris Wright*[†] – WireX Communications, Inc.
*Adam Shostack* – Zero Knowledge Systems, Inc.

## ABSTRACT

Security vulnerabilities are discovered, become publicly known, get exploited by attackers, and patches come out. When should one apply security patches? Patch too soon, and you may suffer from instability induced by bugs in the patches. Patch too late, and you get hacked by attackers exploiting the vulnerability. We explore the factors affecting when it is best to apply security patches, providing both mathematical models of the factors affecting when to patch, and collecting empirical data to give the model practical value. We conclude with a model that we hope will help provide a formal foundation for when the practitioner should apply security updates.

## Introduction

"*To patch, or not to patch, – that is the question: –
Whether 'tis nobler in the mind to suffer
The slings and arrows of outrageous script kiddies,
Or to take up patches against a sea of troubles,
And by opposing, end them?*" [24]

"When to patch?" presents a serious problem to the security administrator because there are powerful competing forces that pressure the administrator to apply patches as soon as possible and also to delay patching the system until there is assurance that the patch is not more likely to cause damage than it proposes to prevent. Patch too early, and one might be applying a broken patch that will actually cripple the system's functionality. Patch too late, and one is at risk from penetration by an attacker exploiting a hole that is publicly known. Balancing these factors is problematic.

The pressure to immediately patch grows with time after the patch is released, as more and more script kiddies acquire scanning and attack scripts to facilitate massive attacks [4]. Conversely, the pressure to be cautious and delay patching decreases with time, as more and more users across the Internet apply the patch, providing either evidence that the patch is defective, or (through lack of evidence to the contrary) that the patch is likely okay to apply. Since these trends go in opposite directions, it should be possible to choose a time to patch that is optimal with respect to the risk of compromising system availability. Figure 1 conceptually illustrates this effect; where the lines cross is the optimal time to patch, because it minimizes the total risk of loss.

This paper presents a proposed model for finding the appropriate time to apply security patches. Our approach is to model the cost (risk and consequences) of penetration due to attack and of corruption due to a defective patch, with respect to time, and then solve for the intersection of these two functions.

These costs are functions of more than just time. We attempt to empirically inform the cost of failure due to defective patches with a survey of security advisories. Informing the cost of security penetration due to failure to patch is considerably more difficult, because it depends heavily on many local factors. While we present a model for penetration costs, it is up to the local administrator to determine this cost.

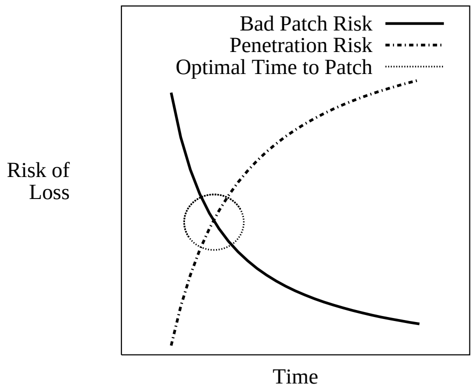

**Figure 1**: A hypothetical graph of risks of loss from penetration and from application of a bad patch. The optimal time to apply a patch is where the risk lines cross.

In particular, many security administrators feel that it is imperative to patch vulnerable systems *immediately*. This is just an end-point in our model, representing those sites that have very high risk of penetration and have ample resources to do local patch testing in aid of immediate deployment. Our intent in this study is to provide guidelines to those who do not have sufficient resources to immediately test and patch everything, and must choose where to allocate scarce

[†]This work supported by DARPA Contract F30602-01-C-0172.

security resources. We have used the empirical data to arrive at concrete recommendations for when patches should be applied, with respect to the apparent common cases in our sample data.

It should also be noted that we are *not* considering the issue of when to disable a service due to a vulnerability. Our model considers only the question of when to patch services that the site must continue to offer. In our view, if one can afford to disable a service when there is a security update available, then one probably should not be running that service at all, or should be running it in a context where intrusion is not critical.

Lastly, we do not believe that this work is the final say in the matter, but rather continues to open a new area for exploration, following on Browne, et al. [4]. As long as frequent patching consume a significant fraction of security resources, resource allocation decisions will have to be made concerning how to deal with these patches.

The rest of this paper is structured as follows. The next section presents motivations for the models we use to describe the factors that make patching urgent and that motivate caution. Then, the next section formally models these factors in mathematical terms and presents equations that express the optimal time to apply patches. The subsequent section presents methods and issues for acquiring data to model patch failure rates. The paper then presents the empirical data we have collected from the Common Vulnerabilities and Exposures (CVE) database and describes work related to this study. The paper ends with discussions the implications of this study for future work and our conclusions.

## Problem: When To Patch

The value of applying patches for known security issues is obvious. A security issue that will shortly be exploited by thousands of script-kiddies requires immediate attention, and security experts have long recommended patching all security problems. However, applying patches is not free: it takes time and carries a set of risks. Those risks include that the patch will not have been properly tested, leading to loss of stability; that the patch will have unexpected interaction with local configurations, leading to loss of functionality; that the patch will not fix the security problem at hand, wasting the system administrator's time. Issues of loss of stability and unexpected interaction have a direct and measurable cost in terms of time spent to address them. To date, those issues have not been a focus of security research. There is a related issue: finding a list of patches is a slow and labor-intensive process [7]. While this makes timely application of patches less likely because of the investment of time in finding them, it does not directly interact with the risk that applying the patch will break things. However, the ease of finding and applying patches has begun to get substantial public attention [20] and is not our focus here.

Most system administrators understand that these risks are present, either from personal experience or from contact with colleagues. However, we know of no objective assessment of how serious or prevalent these flaws are. Without such an assessment it is hard to judge when (or even if) to apply a patch. Systems administrators have thus had a tendency to delay the application of patches because the costs of applying patches are obvious, well known, and have been hard to balance against the cost of not applying patches. Other sources of delay in the application of patches can be rigorous testing and roll-out procedures and regulations by organizations such as the US Food and Drug Administration that require known configurations of systems when certified for certain medical purposes [1].

Some organizations have strong processes for triaging, testing, and rolling-out patches. Others have mandatory policies for patching immediately on the release of a patch. Those processes are very useful to them, and less obviously, to others, when they report bugs in patches. The suggestions that we make regarding delay should not be taken as a recommendation to abandon those practices.[1]

The practical delay is difficult to measure, but its existence can be inferred from the success of worms such as Code Red. This is illustrative of the issue created by delayed patching, which is that systems remain vulnerable to attack. Systems which remain vulnerable run a substantial risk of attacks against them succeeding. One research project found that systems containing months-old known vulnerabilities with available but unapplied patches exposed to the Internet have a "life expectancy" measured in days [14]. Once a break-in has occurred, it will need to be cleaned up. The cost of such clean-up can be enormous.

Having demonstrated that all costs relating to patch application can be examined in the "currency" of system administrator time, we proceed to examine the relationship more precisely.

## Solution: Optimize the Time to Patch

To determine the appropriate time to patch, we need to develop a mathematical model of the potential costs involved in patching and not patching at a given time. In this section we will develop cost functions that systems administrators can use to help determine an appropriate course of action.

First, we define some terms that we will need to take into account:

- $e_{patch}$ is the expense of fixing the problem (applying the patch), which is either an opportunity cost, or the cost of additional staff.
- $e_{p.recover}$ is the expense of recovering from a failed patch, including opportunity cost of work

[1]As a perhaps amusing aside, if everyone were to follow our suggested delay practice, it would become much less effective. Fortunately, we have no expectation that everyone will listen to us.

delayed. Both this and the next cost may include a cost of lost business.

- $e_{breach}$ is the expense of recovering from a security breach, including opportunity cost of work delayed and the cost of forensics work.
- $p_{fail}$ is the likelihood that applying a given patch will cause a failure.
- $p_{breach}$ is the likelihood that not applying a given patch will result in a security breach.

All of these costs and probabilities are parameterized. The costs $e_{patch}$, $e_{p.recover}$, and $e_{breach}$ are all particular to both the patch in question and the configuration of the machine being patched. However, because the factors affecting these costs are so specific to an organization, we treat the costs as constants. This is constant *within* an organization, not between organizations, which we believe is sensible for a given systems administrator making a decision.

The probabilities $p_{fail}$ and $p_{breach}$ vary with time. Whether a patch is bad or not is actually a fixed fact at the time the patch is issued, but that fact only becomes known as the Internet community gains experience applying and using the patch. So as a patch ages without issues arising, the probability of a patch *turning out to be* bad decreases.

The probability $p_{breach}$ is a true probability that increases with time in the near term. Browne, et al. [4] examined exploitation rates of vulnerabilities and determined influencing terms such as the release of a scripted attack tool in rates of breaches. However, the rate of breach is not a simple function $\frac{1}{|Internet\ Hosts|}$ or even $\frac{N}{|InternetHosts|}$ (where $N$ is the number of hosts or unprotected hosts that a systems administrator is responsible for and $|InternetHosts|$ is the number of hosts on the Internet). Not every host with a vulnerability will be attacked, although in the wake of real world events such as the spread of Code Red [6] and its variants, as well as work on Flash [26] and Warhol [28] worms, it seems that it may be fair to make that assumption.

Thus we will consider both probabilities $p_{fail}$ and $p_{breach}$ as functions of time ($t$), and write them $p_{fail}(t)$ and $p_{breach}(t)$.

Next, we want to to develop two cost functions:

- $e_{patch}(t)$: cost of patching at a given time $t$.
- $e_{nopatch}(t)$: cost of not patching at a given time $t$.

The probable cost of patching a system drops over time as the Internet community grows confidence in the patch through experience. Conversely, the probable cost of not patching follows a 'ballistic' trajectory, as the vulnerability becomes more widely known, exploitation tools become available, and then fall out of fashion [4]; but, for the part of the ballistic curve we are concerned with, we can just consider cost of not patching to be monotonically rising. Therefore, the administrator will want to patch vulnerable systems at the earliest point in time where $e_{patch}(t) \leq e_{nopatch}(t)$.

The cost of patching a system will have two terms: the expense of applying the patch, and the expense of recovering from a failed patch. Applying a patch will likely have a fixed cost that must be paid regardless of the quality of the patch. Recovery cost, however, will only exist if a given patch is bad, so we need to consider the expected risk in a patch. Since a systems administrator cannot easily know a priori whether a patch is bad or not, we multiply the probability that the patch induces failure by the expected recovery expense. This gives us the function

$$e_{patch}(t) = p_{fail}(t)e_{p.recover} + e_{patch} \qquad (1)$$

It is possible, although not inexpensive, to obtain much better estimations of the probability of failure through the use of various testing mechanisms, such as having a non-production mirror of the system, patching it, and running a set of tests to verify functionality. However, such systems are not the focus of our work.

The cost of not applying a patch we consider to be the expense of recovering from a security breach. Again, an administrator is not going to know a priori that a breach will occur, so we consider the cost of recovery in terms of the probability of a security breach occurring. Thus we have:

$$e_{nopatch}(t) = p_{breach}(t)\ e_{breach} \qquad (2)$$

Pulling both functions together, a systems administrator will want to patch vulnerable systems when the following is true:

$$p_{fail}(t)e_{p.recover} + e_{patch} \leq p_{breach}(t)\ e_{breach} \qquad (3)$$

In attempting to apply the functions derived above, a systems administrator may want to take more precise estimates of various terms.

## On the Cost Functions

Expenses for recovering from bad patches and security breaches are obviously site and incident specific, and we have simplified some of that out to ease our initial analysis and aid in its understanding.

We could argue with some confidence that the cost of penetration recovery often approximates the cost of bad patch recovery. In many instances, it probably amounts to "reinstall." This simplifying assumption may or may not be satisfactory. Recovery from a break-in is likely harder than recovery from a bad patch, because recovery from bad patch may simply be a reinstall, or at least does not involve the cost of dealing with malice, while recovery from getting hacked is identifying and saving critical state with tweezers, re-formatting, re-installation, applying patches, recovering state from backup, patching some more, ensuring that the recovered state carries no security risk, and performing forensics, a non-trivial expense [10]. However, it is possible that recovery from a bad patch could have a higher cost than penetration recovery – consider a patch that introduces subtle file system corruption that is not detected for a year.

Furthermore, we note that many vendors are working to make the application of security patches as simple as possible, thereby reducing the expense of applying a security patch [22, 25, 29]. As the fixed cost of applying a patch approaches zero, we can simply remove it from the equations:

$$p_{fail}(t)\ e_{p.recover} \leq p_{breach}(t)\ e_{breach} \tag{4}$$

Alternately, we can assume that recovery from being hacked is $C$ times harder than recovery from bad patch ($C$ may be less than one). While the math is still fairly simple, we are not aware of systemic research into the cost of recovering from security break-ins. However, a precise formulation of the time is less important to this paper than the idea that the time absorbed by script kiddies can be evaluated as a function of system administrator time. Expenses incurred in recovery are going to be related to installation size and number of affected machines, so an argument that there is some relationship between costs can be made. This allows us to state that:

$$e_{breach} = C\ e_{p.recover} \tag{5}$$

We can substitute this into equation 4:

$$p_{fail}(t)\ e_{p.recover} \leq p_{breach}(t)\ C\ e_{p.recover} \tag{6}$$

Dividing each side by $e_{p.recover}$, we arrive at the decision algorithm:

$$p_{fail}(t) \leq p_{breach}(t)\ C \tag{7}$$

Recall our assumptions that $p_{breach}(t)$ rises with time and $p_{fail}(t)$ drops with time. Therefore, the earliest time $t$ that equation 7 is satisfied is the optimal time to apply the patch.

### When to Start the Clock

While we discuss the value of the equations above at given times, there are actually numerous points from which time can be counted. There is the time from the discovery of a vulnerability, time from the public announcement of that vulnerability, and time since a patch has been released. Browne, et al. [4] work from the second, since the first may be unknown, but the spread of the vulnerability information may be better modeled from the first, especially if the vulnerability is discovered by a black hat. A systems administrator may only care from the time a patch is available, although some may choose to shut off services known to be vulnerable before that as a last resort, and work has been done on using tools such as chroot(2), Janus [12], and SubDomain [9, 13] to protect services that are under attack. In this paper, we have chosen to start counting time from when the patch is released.

## Methodology

The first thing to consider when deciding to experimentally test the equations derived previously is a source of data. We considered starting with specific vendors' advisories. Starting from vendor data has flaws: it is difficult to be sure that a vendor has produced advisories for all vulnerabilities, the advisories may not link to other information in useful ways, and different vendors provide very different levels of information in their advisories.

We decided instead to work from the Common Vulnerabilities and Exposures (CVE) [16], a MITRE-hosted project, to provide common naming and concordance among vulnerabilities. Since MITRE is an organization independent of vendors, using the CVE database reduces the chance of bias. Starting from CVE allows us to create generic numbers, which are useful because many vendors do not have a sufficient history of security fixes. However, there are also many vendors who do have such a history and sufficient process (or claims thereof) that it would be possible to examine their patches, and come up with numbers that apply specifically to them.

### Data Gathering

Starting from the latest CVE (version 20020625), we split the entries into digestible chunks. Each section was assigned to a person who examined each of the references. Some of the references were unavailable, in which case they were ignored or tracked down using a search engine. They were ignored if the issue was one with many references (e.g., CVE-2001-0414 has 22 references, and the two referring to SCO are not easily found.) If there was no apparent patch re-issue, we noted that. If there was, we noted how long it was until the patch was withdrawn and re-released. Some advisories did not make clear when or if a bad patch was withdrawn, and in that case, we treated it as if it was withdrawn by replacement on the day of re-issuance.

### Methodological Issues

"*There are more things in Heaven and Earth,*
*Horatio,*
*Then are dream't of in our Philosophy.*" [24]

Research into vulnerabilities has an unfortunate tendency to confound researchers with a plethora of data gathering issues. These issues will impact the assessment of how likely a patch is to fail. It is important to choose a method and follow it consistently for the results to have any meaning; unfortunately, any method chosen causes us to encounter issues which are difficult and troubling to resolve. Once we select a method and follow it, our estimates may be systematically wrong for several reasons. Cardinality issues are among the worst offenders:

- **Vendors rolling several issues into one patch:** An example of this is found in one vendor patch [19] which is referenced by seven candidates and entries in the CVE (CAN-2001-0349, CAN-2001-0350, CVE-2001-0345, CVE-2001-0346, CVE-2001-0348, CVE-2001-0351, and CVE-2001-0347).
- **Vendors rolling one patch into several advisories:** An example here is CVE-2001-0414 with a dozen vendors involved. The vulnerability is not independent because Linux and BSD vendors commonly share fixes, and so an

update from multiple vendors may be the same patch. If this patch is bad, then in producing a generic recommendation of when to patch, we could choose to count it as N bad patches, which would lead to a higher value for $p_{fail}$, and consequently, later patching. If the single patch is good, then counting it as N good patches could bias the probability of patch failure downward.

- **Vendors releasing advisories with workarounds, but no patches:** An example is CVE-2001-0221, where the FreeBSD team issued the statement "[this program] is scheduled for removal from the ports system if it has not been audited and fixed within one month of discovery." No one fixed it, so no patch was released. A related situation occurred when a third party, unrelated to Oracle, released an advisory relating to Oracle's product along with a workaround, and Oracle remained completely silent about the issue (CVE-2001-0326). We recorded these instances but treated them as non-events – our goal is to measure quality of patches; if no patch was released, there is nothing to measure.

There are several other potential sources of bias. We may not have accurate information on whether a vendor released an updated patch, because the CVE entry points to the original, and the vendor released a subsequent/different advisory. This potentially introduces a bias by reducing our computed probability of a harmful patch.

When patches are not independent, there is bias in a different direction; consider if one or more vendors released a revised update while others did not (for example, CVE-2001-0318). We considered each CVE entry as one patch, even if it involved multiple vendors. We chose to record the data for the vendor who issued the latest advisory revision (e.g., Debian over Mandrake and Conectiva in CVE-2001-0318). This potentially introduces a bias towards patches being less reliable than they actually are. Systems administrators tracking the advisories of one specific vendor would not have this potential source of bias.

It may be difficult to decide if a patch is bad or not. For example, the Microsoft patch for CVE-2001-0016 was updated six months after its release. There was a conflict between this patch and Service Pack 2 for Windows 2000. Installing the patch would disable many of the updates in Service Pack 2. Note that SP2 was issued four months after the patch, so there was four months where the patch was harmless, and two months where the patch and Service Pack 2 conflicted. We treated it as if was bad for the entire six months.

There is a potential for concern with the number of CVE entries we have examined. In the next section, we attempt to infer appropriate times to apply patches by observing the knees in the curves shown in Figures 7 and 9, and these inferences would be stronger if there were sufficient data points to be confident that the knees were not artifacts of our sample data.

More data points would be desirable, but obtaining it is problematic. We found the CVE repository to be limiting, in that it was difficult to determine whether any given security patch was defective. For future research, we recommend using security advisory information direct from vendors. In addition to providing more detail, such an approach would help facilitate computing patch failure rate with respect to each vendor.

We do not believe that these issues prevent a researcher from analyzing the best time to patch, or a systems administrator from making intelligent choices about when to patch. However, these methodological issues do need to be considered in further studies of security patch quality.

## Empirical Data

In this section, we examine the data we collected as discussed in the previous section. We examined 136 CVE entries, dating from 1999, 2000, and 2001. Of these, 92 patches never were revised leading us to believe they were safe to apply, 20 patches either were updated or pulled, and 24 CVE entries were non-patch events as discussed in 'Methodological Issues.' Table 2 summarizes this data. Of the 20 patches that were determined to be faulty, all but one (CVE-2001-0341) had an updated patch released. Of these, three were found to be faulty and had a second update released; one subsequently had a third revision released. Table 3 summarizes the data for the revised patches.

| | |
|---|---|
| Total CVE entries examined | 136 |
| Good patches | 92 |
| Revised or pulled patches | 20 |
| Non-patch entries | 24 |

**Table 2**: Quality of initial patches.

| | |
|---|---|
| Revised or pulled patches | 20 |
| Good revised patches | 16 |
| Re-revised patches | 3 |
| Pulled and never re-released patches | 1 |

**Table 3**: Quality of revised patches.

Table 4 analyzes the properties of the patch revisions. The middle column shows the number of days from the initial patch release until an announcement of some kind appeared indicating that the patch was bad, for the 20 patches that were revised. The right column shows the number of days from the revised patch release until notification that the revised patch was faulty, for the three issues that had subsequent revisions. Three data points is insufficient to draw

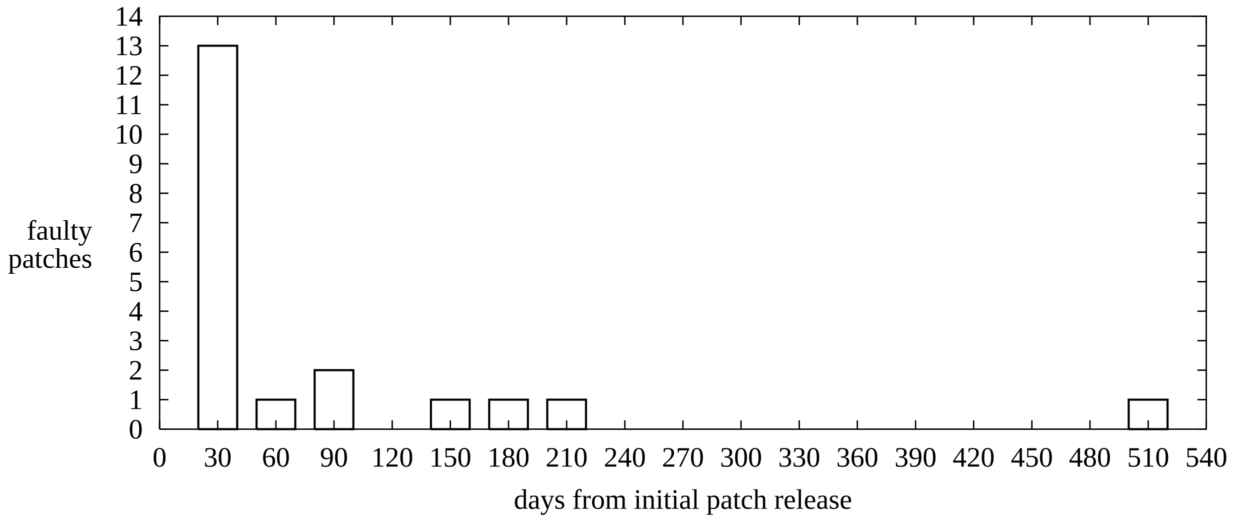

**Figure 5**: A histogram of the number of faulty initial patches.

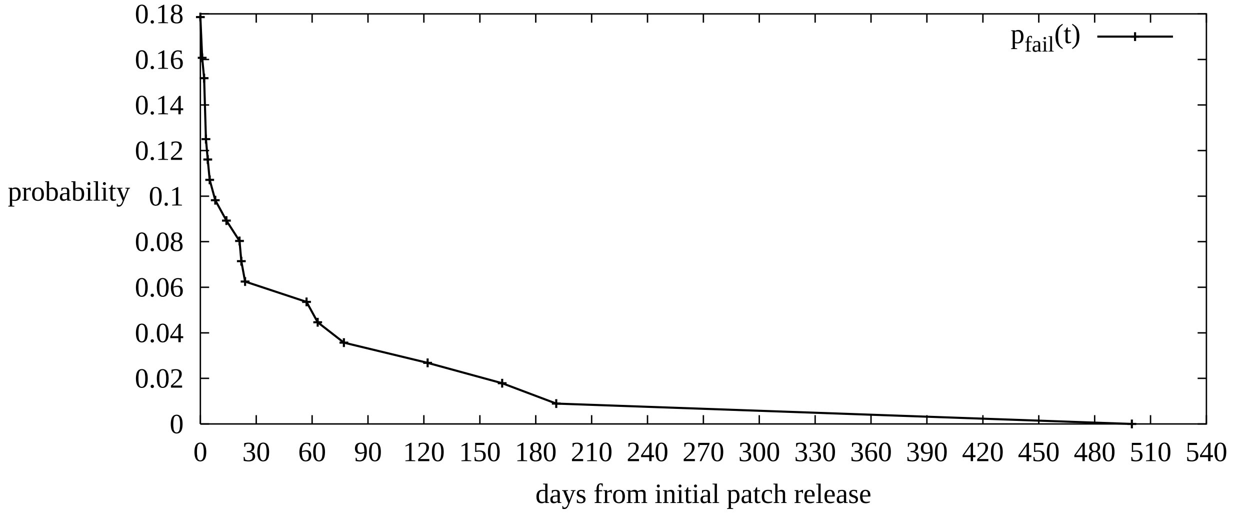

**Figure 6**: The probability $p_{fail}(t)$ that an initial patch has been incorrectly identified as safe to apply.

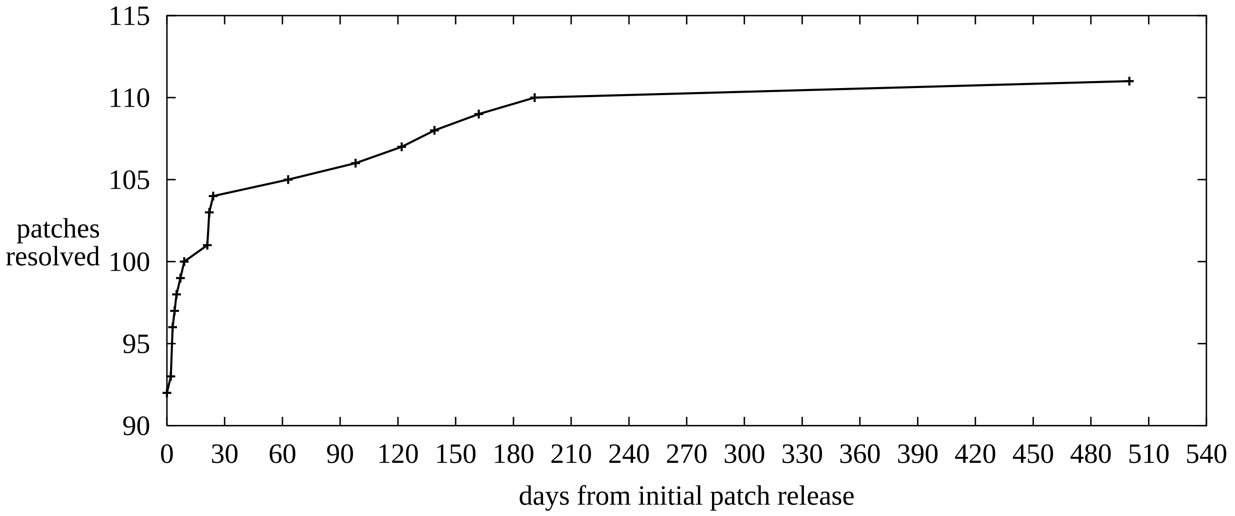

**Figure 7**: A cumulative graph showing the time to resolve all issues.

meaningful conclusions, so we will disregard doubly or more revised patches from here on. We found one triply revised patch, occurring seven days after the release of the second revision.

| Notification time in days | Initial revision (20 data points) | Subsequent revision (3 data points) |
|---|---|---|
| Maximum | 500 | 62 |
| Minimum | 1 | 1 |
| Average | 64.2 | 22.7 |
| Median | 17.5 | 5 |
| Std deviation | 117.0 | 34.1 |

**Table 4**: Analysis of revision data.

Figure 5 presents a histogram over the 20 revised patches of the time from the initial patch release to the time of the announcement of a problem with the patch, while Figure 8 examines the first 30-day period in detail. Figure 6 presents the same data set as a probability at a given time since initial patch release that a patch will be found to be bad, i.e., an empirical plot of $p_{fail}(t)$ from equation 7.

Figure 7 plots the days to resolve an accumulated number of security issues, while Figure 9 examines the first 30-day period more closely. These plots are subtly different from the previous data sets in two ways:

- **Time to resolution:** In the previous graphs, we counted time from when the security patch was announced to the time the patch was announced to be defective. Here, we are measuring to the time the defective patch is resolved. Of the 20 revised patches, 16 provided a revised patch concomitant with the announcement of the patch problem, two had a one-day delay to the release of a revised patch, one had a 97 day delay to the release of a revised patch, and one defective patch was never resolved.
- **No patch ever released:** Of the 136 CVE entries that we surveyed, 24 never had any patch associated with them, and so for these plots, will never be resolved.

Ideally, we would like to be able to overlay Figure 6 with a similar probability plot for "probability of getting hacked at time $t$ past initial disclosure," or $p_{breach}(t)$. Unfortunately, it is problematic to extract such a probability from Browne, et al.'s data [4] because the numerator (attack incidents) is missing many data points (people who did not bother to report an incident to CERT), and the denominator is huge (the set of all vulnerable nodes on the Internet).

From Honeynet [14] one may extract a $p_{breach}(t)$. Honeynet sought to investigate attacker behavior by placing "honeypot" (deliberately vulnerable) systems on the Internet, and observing the subsequent results. In particular, Honeynet noted that the lifespan of an older, unpatched Red Hat Linux system containing months-old known vulnerabilities could be as short as a small number of days, as attacker vulnerability scanning tools quickly located and exploited the vulnerable machine. However we note that this probability may not correlate to the system administrator's site. Site specific factors – site popularity, attention to security updates, vulnerability, etc. – affect the local $p_{breach}(t)$, and as such it must be measured locally.

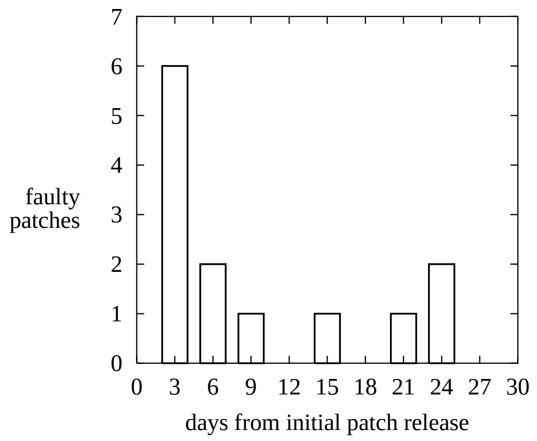

**Figure 8**: A close-up histogram of the first class interval in Figure 5. It shows the number of faulty initial patch notifications occurring within 30 days of initial patch release.

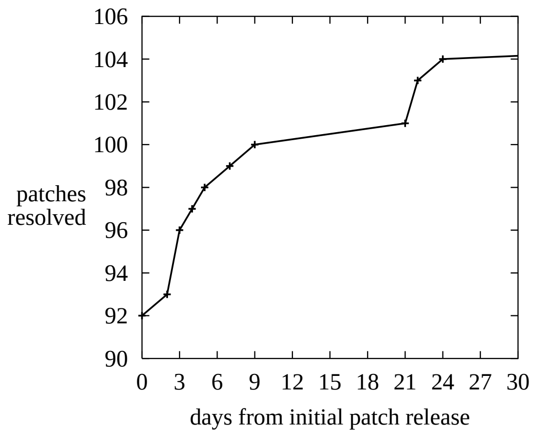

**Figure 9**: A close-up cumulative graph of the first 30 days in Figure 7. It shows the issue resolution time for those occurring within 30 days of initial patch release.

After determining local probability of a breach (i.e., $p_{breach}(t)$), the administrator should apply Figure 6 to equation 7 to determine the first time $t$ where equation 7 is satisfied. However, since $p_{breach}(t)$ is difficult to compute, the pragmatist may want to observe the knees in the curve depicted in Figures 7 and 9 and apply patches at either ten or thirty days.

## Related Work

This paper was inspired by the "Code Red" and "Nimda" worms, which were so virulent that some analysts conjectured that the security administrators of the Internet could not patch systems fast enough to stop them [20]. Even more virulent worm systems have been devised [28, 26] so the problems of "when to patch?" and "can we patch fast enough?" are very real.

The recent survey of rates of exploitation [4] was critical to our work. In seeking to optimize the trade-off between urgent and cautious patching, it is important to understand both forms of pressure, and Browne, et al. provided the critical baseline of the time-sensitive need to patch.

Schneier [23] also studied rates of exploitation versus time of disclosure of security vulnerabilities. However, Schneier conjectured that the release of a vendor patch would peak the rate of exploitation. The subsequent study by Browne, et al. of CERT incident data above belied this conjecture, showing that exploitation peaks long after the update is released, demonstrating that most site administrators do not apply patches quickly.

Reavis [21] studied the timeliness of vendor-supplied patches. Reavis computed the average "days of recess" (days when a vulnerability is known, but no patch is available) for each of Microsoft, Red Hat Linux, and Solaris. Our clock of "when to patch?" starts when Reavis' clock of "patch available" stops.

Howard [15] studied Internet security incident rates from 1989 to 1995. He found that, with respect to the size of the Internet, denial-of-service attacks were increasing, while other attacks were decreasing. The cause of these trends is difficult to establish without speculation, but it seems plausible that the exponential growth rate of the Internet exceeded the growth rate of attackers knowledgeable enough to perpetrate all but the easiest (DoS) attacks.

In 1996, Farmer [11] surveyed prominent web hosting sites and found that nearly two-thirds of such sites had significant vulnerabilities, well above the one-third average of randomly selected sites. Again, root causes involve speculation, but it is likely that this resulted from the complex active content that prominent web sites employ versus randomly selected sites. It is also likely that this trend has changed, as e-commerce sites experienced the pressures of security attacks.

In recent work, Anderson [2] presents the viewpoint that many security problems become simpler when viewed through an economic lens. In this paper, we suggest that the system administrator's failure to patch promptly is actually not a failure, but a rational choice. By analyzing that choice, we are able to suggest a modification to that behavior which addresses the concerns of the party, rather than simply exhorting administrators to patch.

Also worth mentioning is the ongoing study of perception of risk. In McNeil, et al. [17], the authors point out that people told that a medical treatment has a 10% risk of death react quite differently than people told that 90% of patients survive. It is possible that similar framing issues may influence administrators behavior with respect to security patches.

## Discussion

As we performed this study, we encountered several practical issues. Some were practical impediments to the execution of the study, while others were of larger concern to the community of vendors and users. Addressing these issues will both make future research in this area more consistent and valid, and also may improve the situation of the security practitioner.

The first issue is that of setting the values for the constants in our equations, e.g., the cost of breach recovery versus the cost of bad patch recovery, and the probability of a breach for a given site. These values are site-specific, so we cannot ascertain them with any validity:

- A web server that is just a juke box farm of CD-ROMs is not as susceptible to data corruption as an on-line gambling house or a credit bureau, affecting the relative costs of recovery.
- A private corporate server behind a firewall is less likely to be attacked than a public web server hosting a controversial political advocacy page.

We wish to comment that the administrator's quandary is made worse by vendors who do a poor job of quality assurance on their patches, validating the systems administrator's decision to not patch. Our ideas can be easily taken by a vendor as advice as to how to improve their patch production process and improve their customer's security. If the standard deviation of patch failure times is high, then administrators will rationally wait to patch, leaving themselves insecure. Extra work in assurance may pay great dividends. In future work, it would be interesting to examine vendor advance notice (where vendors are notified of security issues ahead of the public) and observe whether the reliability of subsequent patches are more reliable, i.e., do vendors make good use of the additional time.

In collecting data, we noticed but have not yet analyzed a number of trends: Cisco patches failed rarely, while other vendors often cryptically updated their advisories months after issue. The quality of advisories varies widely. We feel it is worth giving kudos to Caldera for the ease with which one can determine that they have issued a new patch [5]. However, they could learn a great deal from some Cisco advisories [8] in keeping detailed advisory revision histories. Red Hat's advisories included an "issue date," which we later discovered is actually the first date that they were notified of the issue, not the date they issued the advisory. There has not, to our knowledge, been a paper on "how to write an advisory," or on the various ways advisories are used.

If one assumes that all problems addressed in this paper relating to buggy patches have been solved, the administrator must still reliably ascertain the validity of an *alleged* patch. Various forms of cryptographic authentication, such as PGP signatures on Linux RPM packages [3] and digital signatures directly on binary executables [27] can be used. Such methods become essential if one employs automatic patching mechanisms, as proposed by Browne, et al. [4] and provided by services such as Debian apt-get [25], Ximian Red Carpet [29], the Red Hat Network [22], and the automatic update feature in Microsoft Windows XP [18].[2]

## Conclusions

"*Never do today what you can put off till tomorrow if tomorrow might improve the odds.*"
– Robert Heinlein

The diligent systems administrator faces a quandary: to rush to apply patches of unknown quality to critical systems, and risk resulting failure due to defects in the patch? Or to delay applying the patch, and risk compromise due to attack of a now well-known vulnerability? We have presented models for the pressures to patch early and to patch later, formally modeled these pressures mathematically, and populated the model with empirical data of failures in security patches and rates of exploitation of known flaws. Using these models and data, we have presented a notion of an optimal time to apply security updates. We observe that the risk of patches being defective with respect to time has two knees in the curve at 10 days and 30 days after the patch's release, making 10 days and 30 days ideal times to apply patches. It is our hope that this model and data will both help to inspire follow-on work and to form a best-practice for diligent administrators to follow.

## Author Information

Seth Arnold graduated from Willamette University in 2001 with a B.Sc. in Computer Science, Mathematics, and Religious Studies. He has been a system administrator, has played cryptographer, and is currently employed at WireX Communications in the research group. He can be reached via email at sarnold@wirex.com.

Steve Beattie is employed in the Research Group at WireX Communications and was involved in the development of the StackGuard, SubDomain, RaceGuard and FormatGuard security tools. He received a Masters Degree in Computer Science from the Oregon Graduate Institute, and was previously employed as a Systems Administrator for a Knight-Ridder newspaper. He can be reached via email at steve@wirex.com.

Dr. Crispin Cowan is co-founder and Chief Scientist of WireX, and previously was a Research Assistant Professor at the Oregon Graduate Institute. His research focuses on making existing systems more secure without breaking compatibility or compromising performance. Dr. Cowan has co-authored 34 refereed publications, including those describing the StackGuard compiler for defending against buffer overflow attacks. He can be reached via email at crispin@wirex.com.

Adam Shostack is currently on sabbatical from his role as Most Evil Genius for Zero-Knowledge systems. Prior to that, he was director of technology for Netect, Inc, where he built vulnerability scanners. He has published on topics including cryptography, privacy, and the economics of security and privacy. He can be reached via email at adam@homeport.org.

Perry Wagle received his M.Sc. in Computer Science at Indiana University in 1995, then dabbled in evolutionary biology until 1997, when he headed to the Oregon Graduate Institute to join the Immunix project's survivability research. For Immunix, he was, among a number of things, the primary programmer of the first released version of the StackGuard enhancement to GCC. When Immunix spun off into the WireX startup in 1999, he generated a second version of StackGuard, but stayed at OGI to work on the Infosphere project and is still participating in the Timber project there. He recently joined WireX to research various compiler enhancements to building secure Linux distributions, including a third and never-again version of StackGuard. He can be reached via email at wagle@wirex.com.

Chris Wright is one of the maintainers of the Linux Security Module project. He has been with the project since its inception and is helping guide LSM into the mainstream Linux kernel. He is employed by WireX where he gets to do security research and Linux kernel hacking. He can be reached via email at chris@wirex.com.

[2]We have not verified the cryptographic integrity of any of these automatic update mechanisms.

## References

[1] Ross Anderson, *Security Engineering: A Guide to Building Dependable Distributed Systems*, John Wiley & Sons, Inc., p. 372, New York, 2001.

[2] Anderson, Ross, Why Information Security is Hard – An Economic Perspective, *17th Annual Computer Security Applications Conference (ACSAC)*, New Orleans, LA, December 2001.

[3] Bailey, Ed, *Maximum RPM*, Red Hat Press, 1997.

[4] Browne, Hilary K., William A. Arbaugh, John McHugh, and William L. Fithen, "A Trend Analysis of Exploitations," In *Proceedings of the 2001 IEEE Security and Privacy Conference*, pages 214-229, Oakland, CA, http://www.cs.umd.edu/~waa/pubs/CS-TR-4200.pdf, May 2001.

[5] Caldera, Inc., Caldera Security Advisories, http://www.caldera.com/support/security/, 2001.

[6] CERT Coordination Center, CERT Advisory CA-2001-19 "Code Red" Worm Exploiting

Buffer Overflow In IIS Indexing Service DLL, http://www.cert.org/advisories/CA-2001-19.html, August 2001.

[7] Christey, Steven M., An Informal Analysis of Vendor Acknowledgement of Vulnerabilities, Bugtraq Mailing List, http://www.securityfocus.com/cgi-bin/archive.pl?id=1&mid=168287, March 2001.

[8] Cisco Systems, Inc., Security Advisory: Cisco Content Services Switch Vulnerability, http://www.cisco.com/warp/public/707/arrowpoint-cli-filesystem-pub.shtml, April 2001.

[9] Cowan, Crispin, Steve Beattie, Calton Pu, Perry Wagle, and Virgil Gligor, SubDomain: Parsimonious Server Security, *USENIX 14th Systems Administration Conference (LISA)*, New Orleans, LA, December 2000.

[10] Dittrich, Dave, *The Forensic Challenge*, http://project.honeynet.org/challenge/, January 2001.

[11] Farmer, Dan, *Shall We Dust Moscow? Security Survey of Key Internet Hosts & Various Semi-Relevant Reflections*, http://www.fish.com/survey/, December 1996.

[12] Goldberg, Ian, David Wagner, Randi Thomas, and Eric Brewer, "A Secure Environment for Untrusted Helper Applications," *6th USENIX Security Conference*, San Jose, CA, July 1996.

[13] Hinton, Heather M., Crispin Cowan, Lois Delcambre, and Shawn Bowers, "SAM: Security Adaptation Manager," *Annual Computer Security Applications Conference (ACSAC)*, Phoenix, AZ, December 1999.

[14] The Honeynet Project, *Know Your Enemy: Revealing the Security Tools, Tactics and Motives of the BlackHat Community*, Addison Wesley, Boston, 2002.

[15] Howard, John D., *An Analysis of Security Incidents on the Internet 1989 – 1995*, Ph.D. thesis, Carnegie Mellon University, http://www.cert.org/research/JHThesis/Start.html, October 1997.

[16] Martin, Robert A., "Managing Vulnerabilities in Networked Systems," *IEEE Computer Society COMPUTER Magazine*, pp. 32-38, http://cve.mitre.org/, November 2001.

[17] McNeil, B. J., S. G. Pauker, H. C. Sox, and A. Tversky, "On the Elicitation of Preferences for Alternative Therapies," *New England Journal of Medicine*, No. 306, pp. 1259-1262, 1982.

[18] Microsoft, *Automatic Update Feature in Windows XP*, http://support.microsoft.com/directory/article,asp?ID=KB;EN-US;Q294871, October 2001.

[19] Microsoft Corporation, *Microsoft Security Bulletin MS01-031 Predictable Name Pipes Could Enable Privilege Elevation via Telnet*, http://www.microsoft.com/technet/security/bulletin/MS01-031.asp?frame=true, June 2001.

[20] Pescatore, John, *Nimda Worm Shows You Can't Always Patch Fast Enough*, http://www.gartner.com/DisplayDocument?id=340962, September 2001.

[21] Reavis, Jim, *Linux vs. Microsoft: Who Solves Security Problems Faster?*, [no longer available on the web], January 2000.

[22] Red Hat, Inc., *Red Hat Update Agent*, https://rhn.redhat.com/help/sm/up2date.html, 2001.

[23] Schneier, Bruce, *Full Disclosure and the Window of Exposure,* http://www.counterpane.com/crypto-gram-0009.html#1, September 2000.

[24] Shakespeare, William, *Hamlet*, Act I, Scenes 3 and 4, F. S. Crofts & Co., New York, 1946.

[25] Silva, Gustavo Noronha, *Debian APT Howto*, http://www.debian.org/doc/manuals/apt-howto/index.en.html, September 2001.

[26] Staniford, Stuart, Gary Grim, and Roelof Jonkman, *Flash Worms: Thirty Seconds to Infect the Internet*, http://www.silicondefense.com/flash/, August 2001.

[27] van Doorn, Leendert, Gerco Ballintijn, and William A. Arbaugh, *Signed Executables for Linux*, Technical Report UMD CS-TR-4259, University of Maryland, 2001.

[28] Weaver, Nicholas C., *Warhol Worms: The Potential for Very Fast Internet Plagues*, http://www.cs.berkeley.edu/~nweaver/warhol.html, August 2001.

[29] Ximian Inc., *Ximian Red Carpet Automated Software Maintenance and Version Management*, 2001.