Limiting Vulnerability Exposure through effective Patch Management: threat mitigation through vulnerability remediation

Submitted in fulfilment
of the requirements of the degree
MASTER OF SCIENCE
in the Department of Computer Science
of Rhodes University

Dominic Stjohn Dolin White
<project@singe.rucus.net>

January 2006

Abstract

This document aims to provide a complete discussion on vulnerability and patch management. The first chapters look at the trends relating to vulnerabilities, exploits, attacks and patches. These trends describe the drivers of patch and vulnerability management and situate the discussion in the current security climate. The following chapters then aim to present both policy and technical solutions to the problem. The policies described lay out a comprehensive set of steps that can be followed by any organisation to implement their own patch management policy, including practical advice on integration with other policies, managing risk, identifying vulnerability, strategies for reducing downtime and generating metrics to measure progress. Having covered the steps that can be taken by users, a strategy describing how best a vendor should implement a related patch release policy is provided. An argument is made that current monthly patch release schedules are inadequate to allow users to most effectively and timeously mitigate vulnerabilities. The final chapters discuss the technical aspect of automating parts of the policies described. In particular the concept of 'defense in depth' is used to discuss additional strategies for 'buying time' during the patch process. The document then goes on to conclude that in the face of increasing malicious activity and more complex patching, solid frameworks such as those provided in this document are required to ensure an organisation can fully manage the patching process. However, more research is required to fully understand vulnerabilities and exploits. In particular more attention must be paid to threats, as little work as been done to fully understand threat-agent capabilities and activities from a day to day basis.

Contents

List of Figures

List of Tables

Acknowledgements

For-most thanks to the Father, Son and Holy Spirit.

umuntu ngumuntu ngabantu - a person is a person through other people

Thanks to Barry Irwin, my supervisor, for his support. Thank you to my mother, father and brother for their care. Thank you to my friends who provided a sounding board; in particular Jason van Niekerk, Chantelle Morkel, (the KiDDiEs) Jonathan Hitchcock, Yusuf Motara, Ingrid Brandt, Bradley Whittington, Russell Cloran, and David Mackie. Thank you to members of the international security community, particularly Adam Shostack, Susan Bradley and the volunteers at the Internet Storm Center for their help. Particular thanks to Daniela Faris for her conditionless love; Bradley Whittington for lending me a house and Chantelle Morkel for the food and laughter.

A few people gave up their valuable time to help proof read this; thank you Johnathan Hitchcock, Thamsanqua Moyo, Fred Otten and Barry Irwin. Finally, thank you to Rhodes University Computer Science department, Professor Peter Clayton, John Gillam, the NRF and DAAD for providing me with the opportunity and resources to study.

Chapter 1

Introduction

1.1 Background

"At the moment computer security is rather basic and mostly reactive. Systems fail absolutely rather than degrade. We are still in a world where an attack like the slammer worm combined with a PC BIOS eraser or disk locking tool could wipe out half the PCs exposed to the Internet in a few hours. In a sense we are fortunate that most attackers want to control and use systems they attack rather than destroy them."

– Alan Cox, Linux Kernel Developer in an Interview with Edd Dumbill [1]

Alan Cox's quotation provides a concise introduction into the current state of information security, the field in which this research is conducted. The tone of the quotation sets the tone of the field: there are a significant number of evolving threats, and without effective research and defences we are in danger of being overwhelmed. He first refers to the binary nature of system failures, an all-or-nothing world in which nuanced risk mitigation strategies that allow for the reality of some intrusion without resulting in a complete system breach are often unavailable. He references one of the most effective worms we have seen in recent times, which managed to compromise 90% of its hosts within 10 minutes. It was the first example of a theorised Warhol worm, able to disable every host on the internet in 15 minutes, a reference to Andy Warhol when he said "everyone will have 15 minutes of fame" [2]. Cox points out the unsophisticated nature of the Slammer worm, it contained no destructive payload, in fact, all of its damage was caused by the excessive load it put on infrastructure in searching for and infecting hosts. Slammer's record has since been topped by more dangerous worms such as the Witty worm. Over the last few years some of the least destructive worms have resulted in a range of Hollywood-style consequences: ATMs infected with malicious code [3], planes grounded [4], waste-water plants disgorging sludge [5] and a nuclear power plant compromised [6].

Cox's next reference is to the changing nature of malicious entities on the Internet. Where previously malicious attackers were hypothesised to be curious geeks with questionable ethics, increasingly threats appear to be coming from criminal entities with a profit motive [7], who seem to be collaborating to use multiple simultaneous attack vectors [7]. Two neologisms have been added as sub-types of malware[1]; spyware and adware, a reference to the increasing financial motivation of malicious software that seeks to steal private information for a profit [8]. This malicious software is employing sophisticated attack and control techniques often utilising similar techniques to those employed by anti-virus vendors and infrastructure teams. For example, the hacker defender root-kit uses the same signature-based approach virus scanners use, to detect anti-virus software and disable it [9]. Cox's reference to controlling and using systems encompasses many examples, including wide scale identity theft [10], massive botnet farms, wide-spread phishing scams and an out of control SPAM problem. Examples of extortion and ensuing DDoS attacks at non-payment abound [11, 12].

As market places and businesses start building their services on top of the Internet, it is becoming increasingly attractive for criminals to follow suit [13]. This has resulted in an increase in malicious software (malware) and successful intrusions, many of which pass undetected. Recent activity has indicated a shift from large scale mass-mailer and worm attacks to rapidly evolving, targeted malware attacks, in an effort to make detection harder [7]. As more systems become networked and private networks are attached to public ones the attack surface of an organisation is increased, allowing an attacker to take advantages of both complex systems and complex interactions between multiple systems. In response there has been an increase in security activity to counter such threats. Much of the work is dealing with problems that have existed for a long time, but have been exacerbated by the increase in malicious activity. In particular the automated exploitation and propagation of malware in the form of worms has meant that an administrator has to deal with every vulnerability, and deal with it quickly.

Exploiting weaknesses and vulnerabilities requires an attacker to think outside of what is considered normal operating procedures, to discover what unusual behaviour will result in a higher

[1] A shorthand for malicious software.

level of access to the system. This attacker needs to find only one hole, but often many exist. Conversely, a security professional needs to apply the same level of creative thinking into defending against every possible hole. This tips the scales in the attacker's favour. However, there is an ongoing and concerted effort to provide workable defence strategies by the "white hat"[2] security community. If organisations develop and implement rigorous security policies many of the threats can be mitigated to a manageable level.

This work is part of such an effort and hopes to provide some guidance and understanding to the field of patch management.

1.2 Patch Management

This work's specific field of study is patch management, an intersection of two related fields, namely, vulnerability management and change management. A patch is used to mitigate a vulnerability permanently, and as such is a mandatory part of any vulnerability management program. When many patches are regularly installed, change is regularly introduced into systems which could potentially cause failures, and these changes need to be managed. This describes the patch paradox, where, without a patch an asset is vulnerable to attack, and with a patch the asset is vulnerable to failure.

While patch management has recently become a regular topic of discussion, the first recorded mention of the phrase 'patch management' on USENET is in 1992 [14, 15, 16], although the concept of patching was introduced before then. Larry Wall (of Perl fame) wrote the Unix *patch* utility in 1985 [17]. In 1997 a project to create a platform non-specific automated patching solution was funded by the US Department of Energy [2] while at the same time Eugene Spafford's COAST Laboratory Secure Patch Distribution Group investigated how best to distribute patches [18]. This may leave an observer wondering why patch management is receiving so much recent attention. The common perception is that the onslaught of several effective worms - Code Red, Nimda, Slammer, Blaster and Sasser - for which patches were available (often months or weeks in advance) highlighted the need for effective patch management. However, these worms were nothing new - the Morris worm [19] had done the same thing in 1988. The growth in the number of inter-networked users and devices on the Internet and their increasingly large bandwidth, the

[2]A "white hat" security professional is one dedicated to the protection of assets, as opposed to malicious "black hats". The terms are a reference to the colour of the hats traditionally worn by good and bad guys in cowboy movies.

increase in the number of software vulnerabilities, the increase in sophistication, number and speed of malicious attacks, and the difficulties in deploying patches have all contributed to a re-invigoration of the discussion.

1.2.1 Definitions

To aid further discussion, some definitions need to be provided. This is particularly important given the wide range of definitions for terms in the relatively young field of information security. Specifically, there is some argument over the use of the term ’threat’. Bejtlich claims that security professionals are “mixing and matching the terms *threat* and *vulnerability* and *risk* to suit their fancy. [20]” It makes sense to side with Bejtlich on this point, primarily because few seem to disagree with him on the subject. In addition there are several high quality resources that agree with his definitions, most notably the US military Information Assurance division [21] and the Office of Cyber Security & Critical Infrastructure Coordination [22], the National Institute of Standards (NIST) Special Document 800-30 [23] and Microsoft’s Security Risk Management Guide [24]. Lastly, Bejtlich’s use of the term allows for a more granular use of the others’ terms, particularly ’exploit’. It is unfortunate that documents such as ISO/IEC 17799 do not have a formal definition of such terms, while other high quality sources such as the National Institute of Standards Special Publication 800-40 on patch management actually define the term incorrectly, using threat as a synonym for malware [25]. Thus, the definitions for terms used in this document are:

Vulnerability

A vulnerability is a weakness in an asset which could be exploited by a threat. In the context of this discussion the asset is usually an electronic system. Other fields may define the asset differently, for example in the field of social engineering the asset usually refers to a person. A vulnerability usually refers to “flaws or misconfigurations that cause a weakness in the security of a system” [23].

Threat

ISO/IEC 13335-1 [26] defines a threat generally as “a potential cause of an unwanted impact to a system or organisation.“ More specifically a threat is an entity with both the capability and the intention to exploit a vulnerability in an asset. Some sources define a threat source as the actual entity and the threat as “capabilities, intentions, and attack methods of adversaries to exploit, damage, or alter information or an information system .” [21] This document finds little use for the distinction and groups both this definition of threat and threat-source under the same term.

Exploit

An exploit is either; a process or tool that will attack a vulnerability in an asset; or it is the action of attacking a vulnerability (exploiting a vulnerability) thereby realising the threat against that asset. Malware in the form of viruses, Trojans, root-kits and most often worms frequently (but not always) use exploits. For example, while phishing is an example of exploiting human trust, in this document exploits refer to tools or processes specifically aimed at exploiting vulnerabilities in software and electronic systems.

Patch

A patch is a piece of data used to update a software product [27]. A security patch is a change applied to an asset to correct the weakness described by the vulnerability. This corrective action will prevent successful exploitation and remove or mitigate a threat’s capability to exploit a specific vulnerability in an asset. In a broader sense a patch can be used to correct a flaw that might not be security related, such as performance issues, or could add new functionality. These are non-security patches and are usually called functionality or stability patches. A patch usually consists of packaged pieces of electronic systems code used to replace existing flawed code. A patch is distributed in one of three ways:

1. as a patch to the source code of a program
2. a patch to the compiled binary code
3. a complete file(s) replacement.

Typically a patch contains a small change and patches with large changes are usually given different names such as a service pack or cumulative update. Vendors such as Sun Microsystems, Microsoft, Oracle Red Hat etc. often have a defined nomenclature for their updates [28, 29, 30]. In this document the primary discussion will focus on security patches unless otherwise specified, as security patches are the the most critical patch and the most difficult to manage. This is for two reasons: failure to deploy a security patch may result in an intrusion; and security patches are released more often, with functionality patches usually rolled into product release cycles. A fuller discussion on this is provided in section 3.2.2.

Remediation

Remediation will refer to the super-set of possible ways of mitigating a vulnerability of which patches are just one method. Configuration changes, complete removal of the software, anti-virus signatures and other workarounds could all possibly mitigate a vulnerability, and will be referred to in general as *remediation* [23].

1.3 The Need for Patch Management

Correct patching is not simply a matter of installing every patch released by a vendor. Currently there are over a hundred new vulnerabilities announced each week, and this number appears to be growing (see 2.3.1). Each of these vulnerabilities usually has a corresponding patch or workaround. Sometimes these vulnerabilities remain unpatched for a period of time. An administrator needs to know which of these vulnerabilities is relevant to her organisation and what their implications are.

The window between the release of a vulnerability and the release of an exploit is decreasing [31] and, with some worms appearing hours after the release of a vulnerability [32], this window is often smaller than the average organisation's patch deployment window. This is partly because patches come with their own set of problems, and sometimes do more damage than than the exploitation of the vulnerability [33]. Thus, an administrator needs to perform a risk analysis on each one, often with incomplete information.

The Morris worm of 1988 lead Bill Cheswick to bemoan firewalling practises with the now famous description "a sort of crunchy shell around a soft, chewy centre." [34] With the advent of mobile computing, multiple service multiplexing over HTTP, ubiquitous e-mail and instant messaging, the phrase has only become more applicable. A firewall never was, and never will be a suitable defence by itself. End-user desktops are now the most commonly targeted, as threats exploit end-user trust with confidence tricks over the web, instant messaging, e-mail and more. Decision making is not, therefore, the only bottle neck, as a patch often needs to be deployed to hundreds or thousands of machines and not just internet-facing servers.

Each machine or group of machines has a different configuration or circumstances that need to be taken into account, which makes patching non-trivial. Different operating systems often have different methods of patching, thus if an organisation has followed the (often sensible) route of platform differentiation, they will need multiple patching mechanisms. Even if an organisation has a homogeneous computing platform, different software products may require their own patching mechanism, particularly in a Microsoft environment, as no third-party patches are currently handled by Microsoft's patching system.

These complexities all contribute to the quagmire many administrators and home users find themselves in when it comes to patching. There are too many vulnerabilities, requiring too many patches, with too many deployment mechanisms, to be deployed to too many machines. A more in-depth discussion of these problems is provided in chapter 2.

1.4 Objectives

The intention of this dissertation is to bring some sense into the patch management discussion. It aims to provide a discussion of all aspects of patch management that will hopefully provide guidance to managers, system administrators and software vendors. The dissertation provides an analysis and definition of the theory of patch and vulnerability management which is then distilled to provide practical advice.

Specifically, there are seven formal primary objectives. They start as investigations into the state and causes of patch management, and move toward providing solutions for some of the problems thus discovered.

The first objective is to provide an analysis of the vulnerability life-cycle. This will place patches in their correct context providing discussions on vulnerability disclosure, exploits and patches. The second objective is to provide an analysis of what causes vulnerabilities and the trends surrounding the vulnerability life-cycle. The third objective is to provide a discussion on patches and the problems which lead to such difficulty managing them. Together these three objectives describe the problem any solutions will need to address.

The fourth objective is to provide a method for implementing a patch management policy capable of effectively addressing the problems discovered. This method will be practically applicable, allowing implementation without recourse to multi-volume risk management strategies and expensive consultants. The fifth objective is to provide a discussion on how vendors can best implement a scheduled patch release strategy given the increasing trend towards releasing patches on a predictable schedule. Together these objectives provide a discussion and policies which can be used to solve many of the problems discovered under the previous headings.

The sixth objective is to provide a discussion on where the described patch management policy can be automated and benefit from software tools. This will also include a discussion on currently available tools, with a view to separating out the marketing hype present in this young growth industry. The seventh objective is the attempt to create or integrate some of these tools, to support the policy developed in the previous objectives.

A summary of these objectives is that the research conducted hopes to provide:

1. An analysis of vulnerabilities, exploits and patches by discussing the vulnerability lifecycle.
2. An analysis of vulnerability, exploit and attack trends.
3. An analysis of patches and their problems.
4. A discussion on how to implement a patch management policy.
5. A discussion on how vendors can implement a scheduled patch release policy.
6. A discussion on patch management tools and automating parts of the policy.
7. Tools to help automate and integrate parts of the policy.

The first is to discuss the cause of patching; vulnerabilities. Vulnerabilities are the root problem and, as such, a thorough understanding of them is required. The trends, causes and influences of vulnerabilities and related research will provide an understanding of the need for patches and what specific problems patches are being deployed to fix.

1.5 Methodology

In reaching the objectives discussed in the previous section, four primary methods will be used, namely:

1. a literature survey
2. argumentative analysis
3. case studies
4. best practice models

Each of these will be used to support or refute hypotheses, where best practice models are often the results of a hypothesis that holds true. A literature survey helps to situate the discussion in the current context of the patch management field. Given that the intention of the thesis is to provide theories around the *management,* human actionable processes, of patches, argumentative analysis is more appropriate than empirical testing. Additionally, much of the leading discussion in the field is very opinion based and requires synthesis and analysis. Additionally, vulnerabilities, exploits and patches are connected as an event, these events are used as the focus for most discussion. For example, a significant majority of references use the Slammer or Blaster worms as examples. Thus, case studies are a useful tool in this document. Finally, the intention of this dissertation is, as stated in the previous section "to provide guidance to managers, system administrators and software vendors", thus best practice models will be created.

The original intention of this work in its early incarnation was to provide an elegant software solution to solve the patch management problems. However, it was soon discovered that the problem is too complex to be solved by software alone. A quotation by Bruce Schneier aptly describes this, "If you think technology can solve your security problems, then you don't understand the problems and you don't understand the technology." [35] While time is spent discussing the plethora of software written to perform patch management tasks, it does not form the bulk of this document. Rather a thorough identification of the problems around vulnerabilities and patches is followed by solid policies and recommendations.

The large amount of writing on patch management and its related fields is drawn upon in each context. Often one author's work can be used with another's to form a derivative work that adds to the common wealth of security knowledge. This synthetic work is a vital tool on many levels, and will hopefully enable consensus to coalesce. The synthesis of tools will bolster the interaction necessary for a successful multi-layered approach to security. While this may seem like an obvious point, the advent of security companies and their related profit motive often results in reduced collaboration in attempt to become the sole product vendor of a product range.

New models and arguments can be derived or created from the existing literature. Thus, the method of research is largely phenomenologically oriented and the results will be interpretive in nature.

Finally, it should be noted that the majority of the references are electronic and not from formal academic references. This was done for two reasons. The first was to try to ensure that URLs for all work available on-line were included, allowing a reader to quickly locate them. However, the many purely electronic references are due to the immediacy (at the time of writing) of many of the issues discussed. As there has been little published research dealing with several of the points and events discussed in this thesis, strong reliance on electronic references was unavoidable. Additionally, there are still a significant number of peer-reviewed papers and other traditional references to legitimise many of the points. The author has taken into account the potential inaccuracy of on-line sources and care has been taken to differentiate between conclusions drawn from trusted and untrusted references. Additionally, traditional references have been used to back up core arguments.

1.6 Conclusion

Information, computer and network security is in a poor state. Specifically, our current methodologies for responding to malware are insufficient. Patch management provides a final solution to the holes that malware exploits. However, it has its own set of problems that must be dealt with. This dissertation will analyse the issues around patch management and plot a way forward.

This will be achieved in four parts. First, the patch paradox will be discussed, analysing the difficulties in remediating vulnerabilities in contrast with the difficulty in managing and deploying patches. Here a meta-policy framework is provided with an in-depth discussion of how an organisation can best implement a policy to realistically and effectively remediate vulnerabilities with patches while minimising the extra risk patches include. Following this, an argumentative analysis of the current trend of scheduled patching is used to provide advice as to how vendors could best implement a patch release policy. Finally, the technological aspects of patching are discussed - specifically, how a patch management policy can benefit from automation, and where current solutions fit in.

Chapter 2

Vulnerability and Patch Management

2.1 Introduction

"This impossible reality has sent patching and the newly minted discipline associated with it "patch management" into the realm of the absurd. More than a necessary evil, it has become a mandatory fool's errand."
–Scott Berinato, *"Patch and Pray"* CIO Magazine *[33]*

Software vulnerabilities have always existed, and probably always will. They result from the mistakes of human programmers. This section begins by providing an analysis of and discussion around the trends and statistics of vulnerabilities. Patching is the final response to a vulnerability, and patching trends will thus follow the cycles of vulnerability trends closely. An understanding of vulnerabilities will allow better decisions about when and how patches should be deployed. Hence, vulnerability trends or a case-study on a recent worm such as Blaster or Slammer usually form the introduction to most papers on security, and particularly on patch management. Successful worm runs have the effect of motivating the security community to action, for example the CERT/CC was formed in response to the Morris worm of 1988 and much of the recent work into patch management came after the worm outbreaks of 2001 [36].

Vulnerability management is the process of identifying, monitoring, and responding to vulnerabilities. Vulnerabilities in a released product are not managed risks that the product manufacturer has an understanding of. They are unknowns, and the liability for these risks often falls to the customer. It is thus the customers' responsibility to identify the vulnerabilities affecting them, in order that better risk management decisions can be made. This chapter provides a description of vulnerabilities and the trends they are facing.

- They are increasing.
- They are being exploited more often.
- The time until an exploit is released is shrinking.

Patch management is the process of correctly and timeously applying software patches to minimise downtime and the attack surface of a system. As patches are release in response to a vulnerability, they too cannot be predicted. Quantifying patch management is thus a difficult task. The complexities of vulnerabilities become apparent when attempting to fix them. These complexities prevent patches' being deployed timeously to vulnerable systems - in some cases patches are only deployed months later. For the few administrators diligently applying patches, the task is still non-trivial. The problems with patches and patching will be explored. These problems will provide the guidance necessary in formulating solutions in later chapters.

2.2 The Vulnerability Life-Cycle

As discussed in the previous chapter, a vulnerability is a weakness in an asset which could be exploited by an entity. The asset could be anything from a computer system to an employee. In the context of this chapter we will be discussing software and hardware vulnerabilities affecting computerised systems. There are several classes of vulnerability, each of which could allow a variety of activities, the worst of which being remote code execution leading to a full system compromise. For a discussion of trends to occur, an understanding of the vulnerability life-cycle is required. The life cycle of a vulnerability has several stages; Arbaugh *et al.* [37] suggest there are seven stages with an additional stage mentioned by Browne *et al.* [38] of the vulnerability becoming *passe*. Schneier [39] has a similar description of stages but does not differentiate between the release of a scripted exploit and the popularisation of the vulnerability. The stages are as follows:

1. The *creation* of the vulnerability. This is when the vulnerability is created during the implementation of the vulnerable product.

2. The *discovery* of a vulnerability. The vulnerability in the product is found. Several people could discover the vulnerability at different times. Little is ever publicly known about this step.

3. The discovered vulnerability is *disclosed*. The disclosure could come from a variety of sources, in a variety of ways. It could be announced by the vendor or an independent researcher, or secreted away in a product's Change Log[1].

4. The vulnerability is *corrected*. This is usually done by the vendor releasing a patch or workaround. This should lead to an overall reduction in successful intrusions.

5. The vulnerability is *publicised*. This can happen in a variety of ways; for example news reporting, publishing an advisory, worm activity; but the end effect is that many people know about the vulnerability.

6. The exploit is *scripted*. This can mean that workable exploit code was released, or instructions on how to produce one are released. In either case, the result is that the number of attackers is greatly increased as those with less skill (script kiddies) can now perform the attack.

7. The vulnerability becomes *passe*. Attackers become disinterested in exploiting this vulnerability. This is not guaranteed to happen with every vulnerability, and some vulnerabilities (and exploits) are shown to have cyclical popularity [37].

8. The vulnerability *dies*. This happens when the number of possible targets vulnerable to exploitation drops to an insignificant level.

The steps follow this rough order, but there can be significant variation. For example the vulnerability could be first corrected with the disclosure following after the correction is reverse engineered; or the disclosure, correction and publicity could all happen at once. Arbaugh *et al.* [37] note that in the past the vulnerability life cycle was theorised to look like something like Figure 2.1, which is a replica of Schneier's life cycle [39]. However, current research has shown some of these assumptions to be incorrect and has provided empirical data to better understand

[1]A register of changes made in a product from one version to the next.

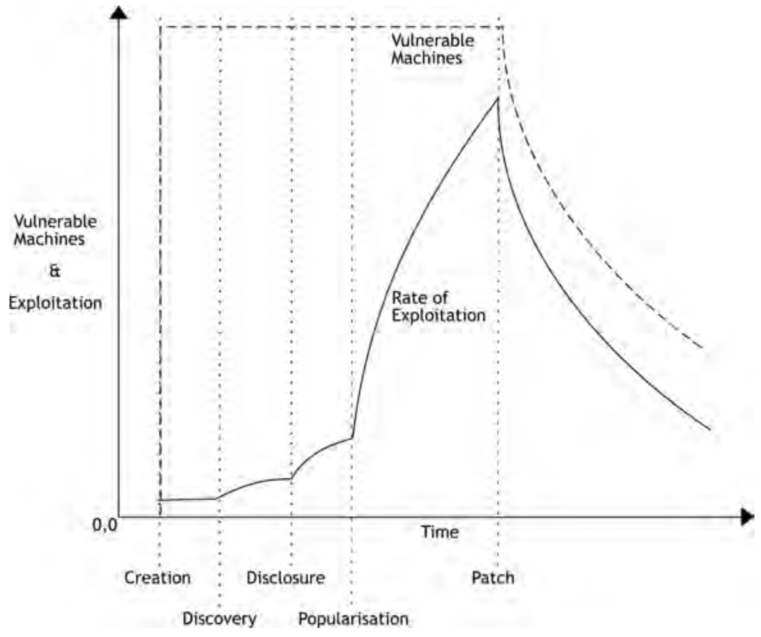


Figure 2.1: Theorised Vulnerability Life-Cycle [39]

some parts of the curve. The corrected life-cycle can be found in figure 2.2. There are several important differences.

- Arbaugh *et al.* [37] found that the significant factor which triggers an increase in the number of reported intrusions was the scripting of the exploit. This caused a dramatic increase in the number of attempted intrusions even if the correction (patch or workaround) had been released previously, thus rendering false the assumptions of the original theorised model, particularly the presumed immediate effectiveness of releasing a patch. In the resulting figure 2.2, the public disclosure of the vulnerability and patch are released at the same time. However the vulnerability could be disclosed immediately before a vendor can release a patch, but according to Arbaugh *et al.* this would make little difference as the scripting of the exploit is the significant factor. For a detailed discussion about the different types of disclosure refer to sections 4.1 and 4.2.
- Browne *et al.* [38] found that the number of reported intrusions can be modelled with the formula $C = I + S \times \sqrt{M}$ where C is the cumulative count of incidents, M is the time from the beginning of the exploit cycle and $I + S$ are the regression coefficients to fit the

curve to the specific incident. Thus, we know that the spike in exploitations will level off and tend towards a constant over time.

- Eschelbeck's [40] empirical data showed that the number of vulnerable machines had a half-life, which was 19 days in 2005, i.e. after 19 days the number of vulnerable machines halved. This data could explain why the increase in intrusions discovered by Arbaugh *et al.* levels off in the curve discovered by Browne *et al.*

- Eschelbeck [40] also discovered that most exploits are available before the end of the first half-life period of vulnerable machines. This is represented in the diagram by the scripted exploit being released before the first half-life.

- Browne *et al.* [38] discuss the drop-off in the number of intrusions when the vulnerability becomes *passe*. The number of intrusions would not drop off like that if the vulnerability had died (i.e. there were an insignificant number of vulnerable machines). However, both Eschelbeck's [40] and Browne *et al.*'s empirical data show that there may be repeated spikes in intrusion activity at a later date. Eschelbeck hypothesises that this is because of new unpatched machines being deployed, effectively extending some vulnerabilities' life span to the nearly infinite. In addition, if another event were to occur which would publicise the vulnerability (most notably a worm), another spike may occur. Thus, the death of a vulnerability is rarely observed, and the drop in intrusions will most likely be due to the vulnerability becoming *passe*. However, there is little empirical data for this drop-off, which is drawn as a steep curve, though this is not backed up by empirical findings.

- The small increase in intrusions between discovery and disclosure follows an exponential increase as a select group of Black Hats exploit the vulnerability, either because they discovered the vulnerability on their own, or because the vulnerability was being exploited in the wild. There is no empirical evidence to support this, however if a small group of black hats is slowly disseminating the information in a controlled manner to prevent mass proliferation and possible detection, it would make sense for this to grow exponentially, increasing faster as more people discover the exploit and tell their small group of associates.

This appears to be as complete an image as possible of the most common vulnerability life cycle. The most disturbing part of this life-cycle is the large number of intrusions that appear to occur well after the release of a patch. The trends discussed below will further discuss this life-cycle,

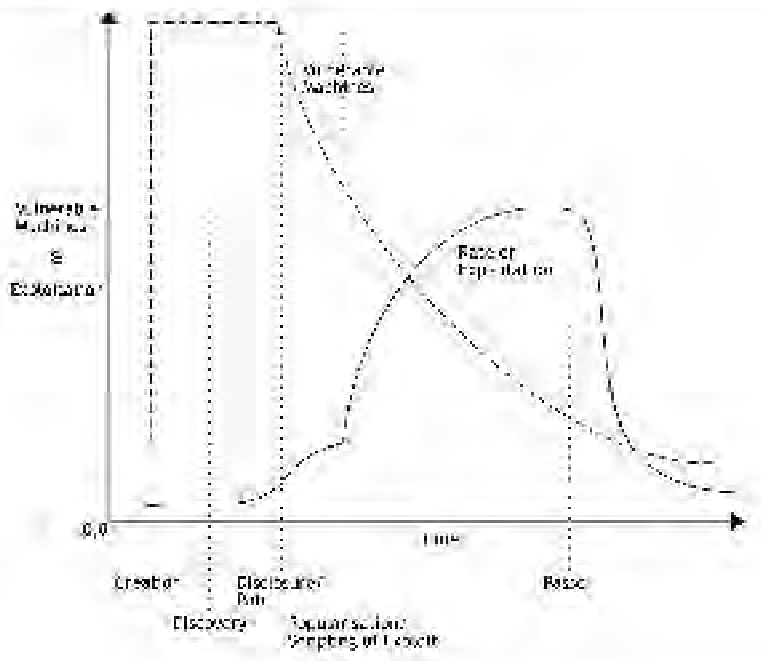


Figure 2.2: Generalised Model of Empirical Findings

highlighting in particular those aspects of vulnerability, exploit- and patch discovery and creation that are becoming more difficult to manage, and justify the solutions laid out in section 3.2.

2.3 Vulnerabilities, Malware and Exploitation Trends

In security, the variables are a moving target. This section will explain the direction in which that target is moving. A discussion of the trends and their likely causes is presented. These trends useful in situating any patch management discussion in the reality of the security landscape. In the context of this discussion, malware exploiting holes due to vulnerabilities in software are discussed. Other attack vectors such as e-mail, instant messaging and other confidence tricks fall outside of the scope of this discussion.

2.3.1 Increasing number of vulnerabilities

The general consensus is that there is an increase in the number of vulnerabilities. This is most often due to the increasing complexity of software and the increase in the number of software projects [41]. According to the National Institute of Science and Technology (NIST) [42], it is estimated that Microsoft Windows 2000 contains 35 million lines of code as compared with Windows 95's 15 million estimated lines and Windows 3.1's 3 million. Similarly RedHat Linux 7.1 had 30 million lines of code in 2001 up from 170 000 lines in Linux distributions of 1992. It is estimated that that the number of software bugs ranges from 5-20 per 1000 lines of code [43]. Thus on these estimates it can be seen that the number of potential bugs has grown immensely. Not all of these bugs will result in security flaws however, and it is difficult to make these extrapolations. For example the Qmail mail server written by D.J. Bernstein offers $500 to anyone who can find a security vulnerability in the code[2]. This has been unclaimed in 10 years. Sendmail, on the other hand, has had a plethora of vulnerabilities in its 20 years [44]. The 'rush to market' attitude of many software vendors is resulting in code with a higher number of vulnerabilities per line, often with poor architectures that make them difficult to secure post-completion [45]. In addition, this increasingly complex software is increasingly interacting with other complex software. The low cost of communication over the internet and its ubiquitous nature is replacing other means of electronic communication, opening systems not designed for the internet up to new vulnerabilities and creating unforeseen situations between system interactions [40]. Even if vendors do provide the ability to lock down their software, it is often not distributed in a secured state; couple this with a lack of security knowledge among system administrators, and a security industry that is woefully understaffed, and the reason for many of the preventable configuration errors becomes clear.

The most commonly quoted statistics of the increasing number of vulnerabilities come from the Computer Emergency Response Team/Coordination Centre (CERT/CC), who compile statistics for each quarter [46]. These statistics are taken from the Common Vulnerabilities Exposure list which assigns a common name to every discovered vulnerability. These statistics show that the number of vulnerabilities are increasing each year. The growth in the number of vulnerabilities each year follows an almost exponential upward trend, except for 2003 where the number dropped to 2000's levels. This could possibly be because of the dot.com crash and the resulting decrease in technology related work which lead to less vulnerability research, although this is unconfirmed. In the first two quarters of 2005 the upsurge is dramatic with the number of vul-

[2]Barring Denial of Service or unreasonable exploitation requirements.

nerabilities averaging 15 a day compared to the previous high in 2002 of 11 a day . However, the number of vulnerabilities reported by different vulnerability databases is not consistent. The Open Source Vulnerability Database in particular is a high-quality resource which claims that there were almost 6200 vulnerabilities in 2005 [47]. This number is almost double CERT's. This appears to be because OSVDB makes a point of creating and entry for every vulnerability, whereas CVE tends to group vulnerabilities into one issue [48]. In previous years the contrast is not as large, this is most likely because OSVDB only started breaking apart issues into separate vulnerabilities in 2004. This shift in reporting standards prevents 2004 and 2005's numbers from being viewed in context of years previous to them. There is still a jump from 4628 to 6161 number of reported vulnerabilities in OSVDB's stats indicating a significant increase. However, CVE itself did not claim to provide comprehensive coverage of all vulnerabilities and is only made full completeness a goal in 2004 [48]. This still shows a jump from 2283 to 3888 reported vulnerabilities [49]. In addition the Secunia vulnerability database [50] has shown a rise from approximately 3190 vulnerabilities in 2004 to 4120 in 2005, which is consistent with CVE's rise.

As long as the preconditions mentioned above hold true, there is little reason for these numbers to stop their upward trend. Discussing the seriousness of these vulnerabilities, Eschelbeck [31] posits two hypotheses: there is a constant discovery of new critical vulnerabilities, this leads to a situation where half of the most common and critical vulnerabilities are replaced every year and; these vulnerabilities often have an infinite lifespan due to the continual deployment and re-deployment of machines with unpatched software. Thus vulnerabilities have a cumulative effect, where the marginal discovery of vulnerabilities is increasing and the total number of critical vulnerabilities are increasing as previous vulnerabilities are not being successfully mitigated.

2.3.2 Increasing number of attacks

The number of increasing vulnerabilities has predictably lead to an increase in both the number of attacks and the number of successful attacks. However, this is a complicated statistic to measure, for several reasons. To monitor attack trends, some statistics indicating the number of attacks need to be collected. However, many attacks are not detected, and others are detected but not reported. Gathering statistics is a non-trivial task; first, if an attack goes undetected, then quite obviously it cannot be counted, and second, reporting is driven by the victim and many organisations who are attacked are either reluctant to report them [51] for publicity reasons, or administrators have dealt with the vulnerabilities and lose interest [38]. In addition, it is difficult to get organisations to allow outside entities to monitor their network [51]. An alternative would be to conduct penetration tests and note the number of reported intrusions. However, this provides an difficult dichotomy where on one hand it is illegal to attack sites without prior consent but, prior warning would influence the site's reporting rate. Worse still, there is evidence that attackers are moving away from using mass compromises and focusing on more targeted Trojan and rootkit installs, which provide more manageable results and help to evade detection [7]. The Hacker Defender anti-detection service provide a service where a semi-unique version of their Hacker Defender rootkit can be bought and used in a "pointed attack" specifically designed to avoid detection by anti-virus software by reducing the chances of the anti-virus researchers crafting a general detection signature for the rootkit and providing many unique versions [9].

This leaves two possibilities for estimating attack activity. In the first approach, taken by the CERT/CC, sites were asked to confidentially report incidents[3]. However, this approach was initially difficult with estimates for the number of incidents in 1995 ranging from 1200 to 22800 [52], and was eventually discontinued in 2004 as incidents were so widespread that they "provide little information with regard to assessing the scope and impact of attacks [46]." The alternative approach taken by the SANS Internet Storm Center and their DSHIELD [53] project is to receive submissions on network activity from distributed sites and perform central analysis of the data. This allows the number of attacks to be better modelled, however it does not provide information as to how many of those attacks are successful, unless a successful attack displays some obvious behaviour (this is often true of worm activity, but not of human exploitation). Some of the resulting noise from attacks can be used to perform a back-scatter analysis, which is particularly effective for Denial of Service attacks [54]. However difficult it is to model attack trends, research tends to agree that the number of attacks and incidents is increasing every year [52, 55]. Given the increasing number of vulnerabilities it is hypothesised that this will lead to a higher number of successful attacks. Indeed, this hypothesis is borne out by the continuing success of automated self-propagating malware (worms) and their continued activity even after a patch has been available for several months [40]. This is further corroborated by DSHIELD which has seen the average time between attacks drop below five minutes in both August 2005 and September 2005. This *time to live* or *survivability* statistic gives an unpatched machine less time until it is compromised than it would take to download and deploy the necessary patches [56]. It is clear, however that not enough public research is being conducted in threat analysis. If the security community had more information on what was occurring in 'the underground', less coarse assumptions of worse case scenarios would be possible.

[3]an incident could be made up of several attacks

There are several reasons for the increasing number of attacks. CERT/CC identifies an additional six trends, three of which are relevant [57]:

1. The increased automation in attack tools has lead to faster and more widespread exploitation due to several advancements. Advanced scanning techniques are regularly employed, for example *scanrand*[4] can portscan a network in record time *[58],* while *nmap*[5] can deploy a variety of stealthy scanning techniques. The release of exploit code has historically heralded the advance of script kiddies[6], however running and managing these exploits is becoming even easier with tools such as *metasploit*[7], which allow for point and click exploitation and provide a toolkit with which future exploits can be rapidly developed. When the advanced scanning is coupled with automated exploitation (in tools such as *AutoScan*[8]) an entire network block can be stealthily scanned and trivially exploited if the discovered systems are vulnerable. This exploitation often has the ability to propagate, allowing a malware creator to compromise several hosts without much involvement, and has proved particularly successful among the most vulnerable, home users. This has provided an increase in the coordination of distributed attack tools, allowing large bot nets to be used in massive distributed malicious activity.

2. Attack tools are becoming increasingly sophisticated and complex. This is making attack detection and prevention increasingly difficult. This sophistication is being packaged in modular code and redistributed. This makes it possible for a relatively inexperienced user to launch a highly sophisticated attack utilising a range of difficult-to-detect payloads - ranging from reverse shells to DLL-uploaded ssh servers - with the metasploit framework. In addition, the modularity of these tools allows different methods to be recombined and reused, which makes detecting a defined set of steps more difficult. Thus malware authors can rapidly create several different iterations of one piece of malware in an attempt to avoid detection by anti-virus software [7].

3. Increased permeability of firewalls. The advent of HTTP as the dominant protocol has caused a shift, whereby services are no longer differentiated by port, but are multiplexed over one port with protocols built on top of HTTP. For example corporate e-mail filtering

[4]`http://FINDOUT/`

[5]`http://insecure.org/`

[6]Less experienced cracker who use tools provided by more experienced authors to break into systems.

[7]`http://metasploit.org/`

[8]`http://autoscan.free.fr/`

policies become meaningless to users utilising web based e-mail services such as Gmail or Hotmail. The threats these services can introduce lead the US military into blocking access to web-mail products on their unclassified networks [59]. The rise of services such as instant messaging and e-mail move much of the content control decision-making from the firewall to the end-user. Attackers have recognised this and now employ a variety of attacks which exploit trust in the end-user using confidence tricks [7]. Phishing, pharming, mistyped domain squatting, and promulgation via e-mail or instant message are all examples that 'trick' the user. There is no need to look for a vulnerability in a firewall when you can instant message a trojan to several users. Moreover, mobile devices and portable storage now allow malware to piggyback its way through a firewall via the sneaker-net[9]. It is very difficult to control every laptop, USB flash-stick, digital camera, MP3 player and memory card which interacts with a network. This can render mobile devices one of the most common infection vectors for an organisation. This has allowed attackers many more targets; no longer are only internet-facing servers vulnerable, an attacker could potentially compromise any machine in the organisation, particularly end-user desktops.

2.3.3 Exploit window shrinking

The time between the release or announcement of a vulnerability and the release of public exploit code is known as the exploit window. This window is widely reported and agreed upon by many researchers [38, 60, 31, 57, 8, 7, 61]. Indeed, the evidence seems to agree; the Nimda worm appeared a year after the vulnerability had been announced, the SQL Slammer worm appeared after six months, Slapper took six weeks, Blaster halved that to three weeks, Sasser took two weeks, Zotob appeared after five days, and the fastest vulnerability-to-worm cycle to date has been the Witty worm, which appeared 36 hours after the vulnerability was announced [62, 63].

The time from the disclosure of the vulnerability until the release of a scripted exploit, the window of exploitation, is the most significant indicator of when a vulnerability has progressed from a theoretical discussion to both a "likely to occur" and "likely to be successful" attack. A quick look at the exploit window for previous worms shows that the exploit window appears to be shrinking. This hypothesis is confirmed by several sources [61, 40, 51]. In addition, the exploit window appears to be shrinking faster than the remediation window. A powerful example of this

[9]The sneaker net refers to the manual networking brought about by people physically walking devices from one place to another.

reduction is the emergence and growth of 0-Day(Zero Day) exploits. The term 'Zero Day' traditionally refers to an exploit for an undisclosed vulnerability, but is increasingly used to refer to scripted exploits released on the same day as the vulnerability was disclosed. In both situations the exploit window is but a few hours.

CERT/CC hypothesises [51] that underground groups could be privately hoarding exploit tools which could be made public immediately when a vulnerability is released, thus skewing the time from public disclosure of a vulnerability until public disclosure of an exploit. However, the most likely reason for the shrinking exploit window is an increase in the sophistication of exploit development tools [57, 51]. Some of these advances are:

1. The metasploit framework provides templates for many combinations of exploit types, payloads and target operating systems. An exploit can be rapidly created by utilising the metasploit framework which reduces the amount of effort required in development and provides access to a far greater range of sophisticated payloads.

2. The increase in abuse of web applications makes exploit development quite easy [64]. Vulnerable software can be easily found through search engines [65], the source code is easily available (allowing an attacker to find vulnerabilities faster), and often exploitation only requires a simple request. The requests can be rapidly developed with tool such as the Perl LWP module or Metasploit.

3. It is becoming increasingly easy to reverse-engineer patches to find and exploit the vulnerability they are supposed to repair. In a recent demonstration the MS05-025 patch from Microsoft was reverse-engineered in twenty minutes [66]. This rapid turnaround means that it should be assumed that an exploit exists a few hours after a patch is released.

4. The re-use and modularity of existing malware. This is particularly true of worms and bots. There are several propagation methods from mass-mailing to exploitation, from which a worm author can pick and choose. In addition modifying existing worms to utilise new exploitation techniques or incorporate new payloads is far easier than writing a new one. The number of variants of the more popular worms, such as Sober and Bagle, testify to this [63].

All of these advances contribute to making exploit development easier and faster, and future advances will only reduce this window.

2.4 Problems with Patches

The vulnerability cycle described in figure 2.1 above assumes that the number of intrusions would start to decrease after the release of a correction or patch. This decrease should continue until the number of vulnerable machines reaches some negligible value and the vulnerability reaches the last stage of its cycle. However, both Eschelbeck [40] and Browne *et al.*'s [38] research shows that this is not the case. A large number of notable worms within the last few years have exploited vulnerabilities for which a patch already exists. In addition, there are cyclical re-infections in the long-term, resulting in an infinite vulnerability life-cycle [40, 38]. This demonstrates that patches are not being deployed to a large number of machines, and for the few that are, with half the most prevalent vulnerabilities being replaced every year [40], the patch treadmill is here to stay. In some cases the notification of vendors has been poor and patches have gone uninstalled because administrators either didn't realise there was a patch or didn't realise its importance. However, even when there is ample notification, research by Rescorla [67] showed that in the case of a critical vulnerability in software more likely to be patched (OpenSSL), after two weeks 60% of vulnerable servers were still unpatched. Anecdotal evidence points to a variety of problems with patches that prevent them from being rapidly and regularly deployed [33]. Solutions to these problems and others are provided in the next three chapters. Chapter 3 describes how users of software can best manage patches coming from vendors, chapter 4 describes how vendors can best prevent the sorts of problems described below and chapter 5 describes technical solutions that can be used to ease the process.

2.4.1 Unpredictable Patches

A security patch should remove or mitigate a vulnerability, no more and no less. However, this does not always happen. Patches sometimes break the service they are supposed to repair, introduce changes that break compatibility and interoperability, add new and unwanted features, introduce new vulnerabilities, re-introduce old vulnerabilities or, in some cases, fail to repair the original vulnerability [60]. When there is a problem with a patch the vendor usually re-releases it. This brings its own set of problems such as removing and replacing the faulty patch and duplicating patch downtime and effort. There are numerous examples of faulty patches, and plenty of anecdotal evidence available on various support forums to make administrators wary of faulty patches. The cost of applying a patch is increasingly better understood, however the costs of potential patch failures weighed against the costs of not applying a patch is not a risk trade-off many are equipped to make. According to Beattie *et al.* this skews the risk analysis towards not applying a patch [60], establishing a situation where administrators are reluctant to apply patches for fear of creating a problem worse than that presented by the original vulnerability.

Some vendors choose to provide specific patches, allowing an administrator to limit the change introduced by a patch, by providing a patch for a specific issue. The Debian security team even goes so far as back-porting security fixes to the older 'stable' software version. This can create problems with multiple patches over time. Some patches deprecate or depend on other patches, and without careful planning and an intelligent patch tracking scheme these inter-dependencies can result in undefined results. To avoid this it is also useful to provide cumulative patches which contain multiple past patches, with the inter-dependencies pre-computed and tested, to ease bringing a newly deployed software instance up-to-date with its patches. However, some vendors find implementing proper patch tracking difficult and opt for cumulative patches only, thus maximising the change administrators make to their systems and increasing the chance of a patch breaking something. The lesson to vendors here is simple: keep patches specific and effective.

Section 2.4.6.1 provides a good example of how a patch can cause unwanted results.

2.4.2 Too Many Patches

As the number of vulnerabilities announced each year grows, so too do the number of corresponding patches. Each patch requires a significant amount of work before it can be deployed and forgotten about. The full process is discussed in chapter 3. The amount of time required to discover patches, research their related vulnerability, test the patches and then make risk management decisions far exceeds the time provided by the shrinking exploit window. Worse still, the exploit window is shrinking faster than the remediation cycle. There are several inefficiencies in the remediation cycle which exacerbate the problem.

Often patches are released by multiple vendors via different mechanisms which can make monitoring for and installing patches involve a large duplication of effort, thwarting organisational centralised patch distribution programmes. For example, users of Microsoft Windows and Adobe Acrobat will need to integrate both Microsoft and Adobe's patch distribution infrastructures.

The unstable nature of patches requires that an organisation perform thorough testing of each patch. However, particularly for large organisations with many machine and software configurations, duplicating every relevant configuration and interaction between critical applications can prove arduous. When this process is applied across several patches it can quickly become untenable.

Vulnerabilities in libraries on which many applications depend can require that each version of the vulnerable library is patched. This can create a situation where sometimes one vulnerability requires several patches from several different software vendors. This combines the above two problems to create a situation where both the problem of multiple distribution methods and complex testing lead to a deployment cycle which far exceeds the window in which attackers are most active. This is demonstrated below in the GDI+ JPEG vulnerability discussed in section 2.4.6.2.

2.4.3 Window to Patch is Shrinking

As the window from vulnerability to exploit deceases (described in section 2.3.3) so too does the window of time available for patching. The vulnerability life-cycle described in section 2.2 showed that the scripting of the exploit was the significant factor in any increase of intrusions. Thus, for an administrator to avoid a significant level of attacks the patch or mitigation should be deployed and working before the release of the exploit. In the case of the Witty worm [32], the exploit was released just thirty six hours after the announcement of the vulnerability. This is not enough time to perform even basic vulnerability assessments and patch deployments, let alone provide significant testing on the patch, a crucial step to avoid the problems with unstable patches. In some cases this may not be enough time to notify users and have them download the patch.

The shrinking of the vulnerability to exploit window is not the only factor in the decreasing patch window. While the scripting of the exploit leads to a significant increase in attacks, this is not to say there are not attacks before this time. When an exploit is scripted it becomes available to a large group of people, often termed script kiddies, who do not have the skill, or money to buy the skill, to write an exploit themselves. Following this logic, the largest threat of potential attack before the scripting of an exploit are the group of people with skills or money. So, while an organisation may not yet be at threat from automated worms wreaking havoc, it may be vulnerable to other activities such as corporate espionage. As security threats on the internet become increasingly criminalised, the threat from such attacks increases. Unfortunately there is very little public research into the activities of skilled attackers and the trends surrounding their activities. However, when a serious vulnerability is announced there is usually a large increase in activity on the vulnerable port as detected by organisations such as DSHIELD [53]. Ideally a patch or mitigation should be deployed before this time to prevent possible attacks in the future.

The vulnerability and exploitation trends discussed in the previous section show that the administrator does not have the luxury of time. Thus, an uncomfortable trade-off exists, with two conflicting pressures when timing the application of a patch; a pressure to wait for the patch to be tested by the community to prevent the problems of unstable patches, and a pressure to patch immediately to prevent exploitation.

2.4.4 Complex Patches

Not every patch is simple to deploy. While research is being put into creating easy to install and distribute patch packages [68, 69], the complexities of software interconnectedness often manifests itself. Programmers often use functionality provided by shared libraries to prevent having to reinvent the wheel and minimise the size of their applications. The result of this is that applications often have several dependencies. Thus, if one of the core dependencies is patched, this could potentially affect every application depending on it. Additionally, some applications may depend on different versions of another application, requiring several versions of a library to be installed.

However, dependencies don't apply only to applications, patches too, often have their own dependency hierarchy. Sometimes one patch may be required to be installed before another. This is not always a strict dependency. For example, in the case of a recent patch against a vulnerability in Windows Meta Files [70] an unofficial patch was provided until Microsoft could release an official patch. There was no specific patch dependency tree (one patch could be deployed without affecting the other), however if the unofficial patch was removed before the official patch was installed, the machine would be left vulnerable for the period of time in between. Thus it was necessary for an administrator to first install the official patch then remove the unofficial one, a rather counter-intuitive process.

Both of the examples provided in section 2.4.6 demonstrate this 'dependency hell' quite well.

2.4.5 Hard to obtain patches

The problems with patches are not always in the deployment, as getting ahold of them in the first place can sometimes be problematic. There are many possible reasons for this, although it is becoming rarer as software vendors become aware of the importance of patching. Therefore, many of the problematic patch deliveries are occurring within smaller software products, among companies who do not have a defined patch release and management policy. Some of the problems faced are:

1. Poor notification. After the bitterly fought full-disclosure debate, the notification of vulnerabilities and their corresponding patches has greatly improved. However, not everyone has cottoned on. For example, a flaw in Google's on-line mail client, Gmail, disclosed on Oct 14th 2005 and patched four days later, was never publicly acknowledged by Google [71]. While Google did not need to distribute a patch, it is still disturbing that such a large software vendor believes it does not need to notify anyone of the flaw and its fix. Other examples often include open-source software products which have a brief entry of an undisclosed security vulnerability in the Change Log of the latest release [72]. Without obvious disclosure of the vulnerability and corresponding fix, users are likely to stick to older versions for longer if there is no other significant reason to upgrade.

2. Unregistered software. Some vendors will only issue patches to software holders with a valid and verifiable license. This is problematic for two reasons. The first is that organisations with legitimate licenses may have too many machines or a unique configuration which makes registering each machine difficult. The second is that users of pirated software (which in the case of Microsoft products is no small minority), while they shouldn't benefit from their unethical behaviour, can impact legitimate users of the software if their software were to become infected with self-propagating malware that affected shared networking resources because they could not patch their software. Microsoft flirted with this idea with their 'Genuine Advantage' program [73] but soon relented and have made security patches available. However, other proprietary vendors such as Solaris and Oracle still require the purchase of a support contract or some other form of verification [74, 75].

3. Limited bandwidth. Some users and organisations either due to ineffective telecoms regulations, limited network infrastructure or limited funds with which to purchase bandwidth, or a combination of these may not have enough bandwidth at their disposal to rapidly

download patches. Much of the patch management effort has assumed access to broadband connectivity. In bandwidth starved countries such as South Africa and other emerging information economies (Brazil, India etc.) where many small organisations have only an expensive dial-up or ISDN line, spending several hours downloading security patches is not a suitable solution and often results in patches just not being applied. The corollary of this is that it is unlikely their machines would participate in any large scale worm propagation outside of their organisation, a bitter-sweet consolation for the administrators performing the local mop-up operation.

2.4.6 Problem Patch Examples

Some of the potential problems patches can create are best illustrated by examples. Two such 'problem patches' are discussed below. The first demonstrates how a patch can fail to effectively fix a vulnerability, re-introduce a vulnerability, or conflict with existing software. The second demonstrates how difficult it can be to discover which applications are vulnerable and ensure that all traces of a vulnerability are patched.

2.4.6.1 SQL Slammer/Sapphire Worm

On July 24, 2002 Microsoft released the MS02-039 [76] patch for SQL 2000 Server and Microsoft Desktop Engine 2000 (MSDE), which patched critical buffer overflows. The overflows could be triggered by sending trivially small UDP packets to port 1434. One day over 6 months later, the SQL Slammer or Sapphire worm was released in a 376-byte UDP packet. According to an analysis by the Cooperative Association for Internet Data Analysis (CAIDA) [77] the worm infected at least 75 000 hosts. Moore *et al.* [77] had this to say about its spread:

> In the first minute, the infected population doubled in size every 8.5 (± 1) seconds. The worm achieved its full scanning rate (over 55 million scans per second) after approximately three minutes, after which the rate of growth slowed down somewhat because significant portions of the network did not have enough bandwidth to allow it to operate unhindered. Most vulnerable machines were infected within 10-minutes of the worm's release. Although worms with this rapid propagation had been predicted on theoretical grounds, the spread of Sapphire provides the first real

> incident demonstrating the capabilities of a high-speed worm. By comparison, it was two orders magnitude faster than the Code Red worm, which infected over 359,000 hosts on July 19th, 2001. In comparison, the Code Red worm population had a leisurely doubling time of about 37 minutes.

The only hindrance to the worm appeared to be its own effectiveness. It managed to infect a huge number of machines, even though the patch had been released a significant amount of time earlier. The time-line of the patch possibly shows why so few hosts had been patched 6 months later.

The vulnerable library patched by MS02-039 was ssnetlib.dll, and Microsoft later released further patches for SQL Server. On August 14 they released MS02-043 [78] which contained the same version of ssnetlib.dll as MS02-039. On October 2 they released MS02-056 [79] which included a newer version of ssnetlib.dll. On October 16, Microsoft released a cumulative patch, MS02-061 [80] which contained all changes applied by MS02-{039,043,056}. However, on October 30 Microsoft released a security hotfix Q317748 [81] to fix a handle leak in SQL Server 2000 Service Pack 2. The hotfix contained a version of ssnetlib.dll released prior to MS02-039's version, thus reverting the fixes made in MS02-{039,043,056} and MS02-061, and reintroducing the vulnerability [82]. Thus, a fully patched system was now vulnerable to several previously fixed vulnerabilities. In addition, the precedence of patches was unclear, and users were unsure as to whether the patch should be installed and then the hotfix or vice-versa. Microsoft re-released the hotfix with a corrected version of ssnetlib.dll and re-released MS02-061 [80] to include the hotfix on October 30. However, the worm only hit 3 months later, which should be enough time for a significant number of machines to be patched. The lack of patching could have been because notification of the problems Microsoft repaired was not widely disseminated, leaving many machines vulnerable. Particularly, given the wide inclusion of the MSDE in third-party applications, resulting in many non-security conscious user's machines being infected. Russ Cooper, editor of NTBugTraq only posted his understanding of the changes on January 28 after the worm had hit [82].

On January 20 Microsoft released SQL Server Service Pack 3 (SP3) [83] which contained a significant number of changes, including an up-to-date version of ssnetlib.dll. Given the large number of changes, regression testing on Service Packs can often take significantly longer than the testing required for smaller patches. Of the few who did deploy SP3, it was found that there was a conflict with Best Software's MAS 500 accounting package which required users to reformat their machines. Currently Best Software only certifies their software to work with patches up until MS04-021 [84], indicating the difficulty third-party vendors and consumers face in avoiding conflicts from patches. Thus, when the worm hit five days later at 5:30 on a Saturday morning (UTC) [77] many administrators either thought they were patched or were still testing SP3. Of the many organisations crippled by the worm, Microsoft was one [85], indicating that even they had difficulty managing the patch soup surrounding the worm.

It should be noted that it is unlikely that all patches will provide this many problems - Slammer is a particularly bad example which allowed for a concise demonstration of some of the problems related to patching. The fiasco surrounding Slammer demonstrates that patching is not always straightforward. Often several versions of a patch exist and must be applied in a specific manner. Sometimes those patches fail to remediate the vulnerability or expose the organisation to new vulnerabilities (or re-open old ones). Also, patches are not guaranteed to be compatible with every specific configuration and application, and a significant amount of testing is required before they can be deployed.

2.4.6.2 GDI+ JPEG Vulnerability

On September 14 2004 Microsoft released MS04-028 [86] which described a vulnerability in the way the GDI+ library processed JPEG files. An attacker could thus craft a malicious JPEG file capable of executing code on a victim's machine. This type of vulnerability is particularly dangerous as many users and applications don't treat pictures as potentially malicious and often view or process them without confirmation. For example, Google's Desktop Search application automatically indexes images, which could trigger the vulnerability if a malicious JPEG is indexed, without user interaction [87]. In addition, JPEG viewing is supported by a large number of applications, creating a very large attack vector. Thus the risks were such that administrators should have expedited deploying the patch.

Deploying the patch was, however, a non-trivial task. The GDI+ library (gdiplus.dll) can be run side by side with other versions of the library [88]. The ubiquitous nature of JPEGs and the resulting number of effected applications, both Microsoft and third-party, had their own versions of gdiplus.dll installed. Thus, deploying the operating system patch alone was not sufficient to mitigate the vulnerability. For example, even though the version of gdiplus.dll bundled with Windows XP Service Pack 2 was not vulnerable, an installation of Microsoft Office 2003 would make it vulnerable. To help with this, on October 12 Microsoft released the MS04-028 Enterprise Update Scanning Tool [86] which would scan for Microsoft applications which contained a vulnerable version of gdiplus.dll and update it. A necessary task, considering that Microsoft's advisory on the issue [86] lists over 50 Microsoft applications which are vulnerable with links to over 30 additional updates for individual software.

Even if an administrator managed to find and patch all vulnerable Microsoft software, there were still many third-party applications vulnerable. To help with this, a third-party tool was created by Tom Liston of the Internet Storm Centre (ISC) [89] which scanned for potentially vulnerable versions of the DLL. This tool only helped in discovering vulnerable applications though - a user would still have to obvtain a specific patch from that application's vendor.

Thus, to repair one vulnerability, an administrator would likely have to run two separate scanning applications and deploy a significant number of patches, demonstrating how it is not always a simple point-and-click case of one-vulnerability-one-patch. The complexity of software and its inter-dependencies is carried through into patching.

2.5 Conclusion

This chapter serves to provide an examination of the current problems and trends contributing to the difficulty of, and need for patch and vulnerability management. First, the life-cycle of vulnerabilities was introduced and discussed. There has been much research into several aspects of the vulnerability life-cycle, and the resulting life-cycle provides significantly more insight into the process than previous life-cycles have assumed. The solid understanding of the way in which vulnerabilities are introduced, disclosed and remediated provided a platform from which the trends and issues surrounding this cycle were discussed. Several problems and trends in vulnerabilities, malicious software and attacks were discussed. It was shown that:

- The number of discovered vulnerabilities is increasing every year.
- The number of attacks on those vulnerabilities and on still older unpatched vulnerabilities is increasing.
- The release of a scripted exploit results in the largest increase in attack rate, and the time between disclosure of a vulnerability and the release of a scripted exploit is decreasing.

Patching provides an effective method of finally remediating a vulnerability and can provide a powerful defence against these trends. However, these trends impact the creation, release, and deployment of patches. In addition, patches have several pitfalls of their own. These were discussed, and it was shown that:

- Patches do not always behave as expected and can sometimes cause errors instead of fixing them
- The increasing number of vulnerabilities is resulting in an increase in the number of patches, which can often be overwhelming
- The decreasing vulnerability to exploit window results in a smaller window in which patches need to be applied, however other problems with patches are resulting in a deployment time frame which exceeds this window
- Patches are not always straightforward to deploy. Often the complexity of the underlying software results in complicated installation procedures
- While vendors are improving, it is not always easy to obtain patches or notification of their release.

These findings constitute a problem statement for which a patch management programme must provide solutions. Understanding these problems and their causes allows a security professional to design and implement policies, procedures, and technologies capable of responding to these threats. The next chapter discusses how an organisation can do this by implementing an internal policy for remediating vulnerabilities by patching. This discussion is then expanded to include how a vendor can best respond to these threats in the chapter following that. Finally, in chapter 5, advice on the technical solutions available to improve patching and vulnerability mitigation is discussed.

Chapter 3

Policy Solutions

3.1 Introduction

The previous chapter featured a discussion on the difficulties presented by managing vulnerabilities while dealing with patches. This provides the basis from which this chapter will discuss how best to respond to this situation.

As the vulnerability landscape shifts and the threats evolve, so too must patch management. The trends described in Chapter 2 show how patching is an increasingly necessary and non-trivial task, bringing its own problems with it. These trends showed the problems getting bigger, while there is less time in which to solve them. At the start of this project we believed that a software solution to the intricacies of patch management would be practical. Resclora [90], for example, believes that if automatic patching were more widely deployed the costs incurred due to intrusions would decrease. However, later in the same paragraph Resclora states "any measures which improve the rate of patching [...] are likely to pay off." We soon realised that *automated patching* is not sufficient on its own to improve the *rate of patching,* due to the complex and inherently fuzzy nature of patch management. Given the trends discussed in the previous chapter, the speed at which patches can be deployed is one of the most important issues a patch management solution must provide. While the burden of patching can be eased with effective tools, it cannot be completely managed by them. Patch management is a risk trade-off. It requires information from many sources (asset management, network monitoring, vulnerability lists, patch lists), and integrates with existing processes (risk management, change management, vulnerability management). A software tool may help to integrate these process and centralise the information-gathering efforts, but in the end someone is required to evaluate this information and make a decision. Chan [91] and Schneier [35] both believe that patch management is inherently a technology problem, but that a sole focus on technology is insufficient. A quotation from the ISO/IEC 17799 document states "The security that can be achieved through technical means is limited, and should be supported by appropriate management and procedures." [92] This chapter follows this line of thought and focuses on the management and procedures that can be implemented to ensure that patch management, and not just patch deployment, can effectively occur. To this end, a comprehensive patch management policy is required. An organisation must have a process for managing patches and gathering relevant information to make an accurate decision. To quote MacLeod [93]:

> "Any organisation implementing a well thought out patch management process is on the right track to reducing its exposure and risk to published security vulnerabilities."

This chapter details an organisational patch management policy framework. It describes a process for effectively managing security patches within an organisation, with practical advice on implementing such a policy.

3.2 Patch Management Policy Framework

The rise in recent patch management research has resulted in a large number of best practice policy documents being released. The problem with existing policies is that their focus is too often only on a few elements of patch management. For example, one might focus on asset management and decision making, while another might focus on deploying patches and the software necessary to do so. What is lacking is an intelligent synthesis of the available information into one body of knowledge. The recommendations and steps enumerated below achieve this synthesis, based on four such existing policies. The first is an early paper on patch management by Chan entitled *Essentials of Patch Management Policy and Practice* [91]. The second is a paper by Voldal [94] entitled *A Practical Methodology for Implementing a Patch Management Process*. These two papers were the subject of an early iteration of this work [95]. The third paper is NIST's *Patch and Vulnerability Management Program*, special publication 800-40 version

2.0 [42], by far the most comprehensive of the documents. Lastly, Sun Microsystem's *Solaris Patch Management: Recommended Strategy [30]* contained valuable insight. These policies were specifically chosen because of the depth and breadth of strategy and ideas they presented. NIST's work is highly regarded and used to maintain federal systems with strict requirements and regulatory conditions. Sun's document represents the thoughts of a large vendor, one of the first to start discussing patch management. Voldal's paper is, at the time of writing, the only best practices paper on implementing a patch management policy available from the SANS reading room[1], an excellent and well regarded resource for technology professionals. Chan's paper is both well regarded in its own right and published by an influential security organisation, @stake. While these policies form the basis of the work to follow, the result is greater than the sum of its parts. Other documents are referenced when necessary and this researcher's own insights are added. Specific technologies will not be discussed, as a process-oriented, technology agnostic look at what needs to be done, whether automated or manual, seems more appropriate.

The main aim of this *policy* is, as defined by Chan "to create a consistently configured environment that is secure against known vulnerabilities in operating system and application software" [91]. However, the end goal of this *discussion* is to provide a reference for an organisation looking to design and implement their own patch management policy from the ground up, or to enhance an existing policy. This discussion falls under the umbrella of best practice and provides a utopian framework. As such, it is not necessary for every organisation to implement every step - smaller organisations may combine several steps into one, for example. While insight into all aspects of managing patches in available, this section is geared towards an organisation looking to manage its computing assets, rather than the home user.

3.2.1 Patch and Vulnerability Group

A policy is of little use without stakeholders assigned the responsibility of managing and controlling its running and implementation. NIST recommends that a patch and vulnerability group be established [42, pg VII]. The size, make-up and operation of this group can vary widely across organisations. In smaller companies, and depending on budget, it may be the additional responsibility of a systems administrator, whereas in larger companies it could be a cross-section of relevant people from various departments, divisions or branches. The patch and vulnerability

[1]`http://rr.sans.org/`

group will be responsible for gathering information, implementing the policy, reporting to management and disseminating relevant information across the organisation. The group should be differentiated from a general security group which, if it exists, should remain a separate group. Patching is only as useful as the security of the organisation. There is no point expending a large amount of effort on managing patches if all the machines and services have been left with poor default configurations and no hardening effort has been expended.

It is recommended that the group be made up of system administrators, network administrators, security staff and IT support staff. The wide impact of patch management and need for integration with other security and policy systems requires a diverse group. This group should have the support of top management, and the authority to perform their functions [94]. The position of the group within the organisational structure is specific to each organisation, although a group cutting across many departments would provide a better understanding of the organisation's configuration and provide someone to drive patching in each department. The group could also have subgroups operating in different parts of the company. Whatever the operation of the group, having a centralised resource and a primary patch and vulnerability management group overseeing the whole organisation's patching is essential.

3.2.2 Security, Stability, Functionality Patches and Workarounds

The policy outlined will deal primarily with security patches and workarounds for security issues. This is justified, given that the current difficulty in dealing with security patches is the reason for implementing such a policy, as other types of patches can be handled by and fall within normal maintenance and upgrade change control cycles. In more detail, security patches need to be installed more often, at least once a month and often more regularly than that. Regular maintenance cycles are usually designed to be run less often, usually annually. Security patches are also more important, as the risks of not patching (suffering an intrusion) are far higher that those faced by non-security patches. Additionally, the installation of security patches needs to be expedited to deal with the unique risks security vulnerabilities introduce (see section 2.3). The need for deployment speed is in contrast to the nature of the steps required during maintenance such as adequate and thorough testing, which in the case of non-security patches does not need to be rushed, but may require certain trade-offs when deploying security patches. Security patches are also easier to not install, while other defences are getting better, without an effective workaround it is likely that most security patches will need to be installed. Whereas non-security patches can often be ignored if the problem has not manifested itself, there is no guarantee that an attacker will not try to exploit any available vulnerability. Security workarounds fall into the same category as security patches here - the changes introduced will still need to be tested and the effectiveness and risks of the change must be looked examined in the same manner that a security patch would. Both security workarounds and patches have more urgency attached to their deployment. The only real difference is that the distribution methods of workarounds and patches may sometimes differ. A breakdown of the different types of patches and remediation is provided in table 3.1.

- Security Patches
  - installed more frequently
  - higher risks of not patching
  - unique requirements
  - fewer alternatives
  - need to be deployed even for non-critical services
- Security Workarounds
  - same requirements as a security patch
  - usually easier to implement
- Stability Patches
  - lower risks, threat already exists
  - only required if corresponding stability problem exists
  - is accommodated within normal maintenance cycles
- New Functionality/Features Patch
  - low risk, threat does not exist
  - only required if business needs dictate it
  - is accommodated within normal upgrade cycles

Table 3.1: Types of Patch and Remediation Summary

3.2.3 Policy

This meta-policy framework contains eight steps, each of which should occur within the patch management policy implemented by the organisation. Each step has a wide range of variables that will need to be tweaked and set to reasonable standards within the organisation, and relevant to the organisational and operational business context. These are considered best practice guidelines for implementing a policy for managing security patches within an organisation. A breakdown of each step is provided in table 3.2.

1. Information Gathering
   - Host and asset inventory
   - Patch and vulnerability research
   - Exploit and threat research
2. Risk Assessment
   - Patch and security threats
   - Patch and security impact
   - Assessment
3. Scheduling and Patching Strategy
   - Define patch schedules
   - Minimise change
4. Testing
   - Mirror production environment in test lab
   - Check: patch authenticity, dependencies and requirements, whether vulnerability is remediated, conflicts with other applications
   - Create repeatable steps to verify patch installation
   - Test back-out and undo steps
5. Planning and Change Management
   - Proposed change
   - Contingency and back-out plans
   - Risk mitigation
   - Patch monitoring and acceptance
6. Patch Deployment and Installation
   - Automate where possible
   - Secure patch distribution mechanism
   - Utilise technologies to speed patch distribution
7. Verification and Reporting
   - Verify patches were installed to all relevant machines
   - Follow contingency plans if patch is faulty
   - Generate Metrics
   - Report and document progress
8. Maintenance
   - Analyse policy for improvement
   - Train staff

Table 3.2: Patch Management Policy Summary

3.2.3.1 Information Gathering

The information gathering phase is a required input for making informed and accurate decisions and providing proper risk analysis. This is an ongoing phase that will have inputs from many of the other steps, and is the primary input into the risk management phase.

Asset & Host Management For a patch management process to be effective you need to know which machines are utilising your organisation's network, what software they are running, and which previous patches (both security and functional) or modifications to the software have been applied. This is necessary at the very least to establish which machines are affected by which vulnerabilities and hence require patching. Further information is particularly useful in making risk management decisions; the ability to anticipate the possible effects of a vulnerability or patch is invaluable for minimising disruption and cost while maximising availability. This process should be continuous rather than once-off, ensuring detection of new machines entering the network. This requires a passive method which does not require software to be installed onto machines, allowing the discovery of all new machines.

Typically the kind of information collected about each machine should include the system's hardware, operating system, application software, location (both physical and logical), the person responsible for its administration and its function (end-user machine, print server etc.). For both the operating system and application software (and possibly firmware) details of the current software version, all applied updates and patches should be collected [94, 30]. While each policy provides examples of various details that can be recorded, NIST's provides the most comprehensive list of possibly useful information [42, pg 2-4]. For servers, additional information, such as the services they have been authorised to run, can be recorded [94].

Systems should then be grouped and assigned a criticality or priority level. Grouping should be appropriate for the organisation and can be grouped along a variety of differentiators. While NIST defines stringent criteria for grouping [96] American federal resources, the other policies leave it entirely up to the organisation. Example groupings are:

- according to departments
- according to user base

- primary function
- managerial control.

These groups should then be given a priority rating. Voldal's [94] policy document recommends classifying these groups into one of three priority levels, although they can be further broken down:

- Mission critical. Machines providing services critical to the business operation. e.g. Amazon's Web Servers
- Business critical. Machines providing important services that can tolerate short breaks in service. e.g. E-mail servers or machines that aren't used after hours
- Operational critical. Machines providing non-critical services. e.g print servers

Factors such as the importance of the server's data and the consequences of downtime should be used when determining priority levels. The ease with which the machine can be rebuilt in the event of an intrusion is also a factor. In addition, the server's vulnerability to attack should be noted and used to modify it's priority. For example, publicly accessible machines or high profile targets should be given a higher priority [94, 42, pg 2-6]. It is also important to take note of system interconnects, as a lower priority system may provide an attack route to a higher priority one - thus, an internet-facing non-critical print server (a silly thing to do), while not mission critical, should have a higher priority due to the increased risk of attack and possibility for further compromise. This classification will be useful in determining the seriousness of a vulnerability, guiding other actions such as whether or not a patch should be installed immediately. Without this classification, an organisation may embark on costly and unnecessary patching and mitigation projects, or cause further avoidable problems [42]. The information collected and determined here will be a direct input into the risk management decisions described later. The broad groups and the factors influencing priority are summarised in table 3.3.

Broad Priority Groups	**Description**
Mission Critical	Services and machines the organisation could not continue without.
Business Critical	Important services tolerant of some short downtime
Operations Critical	Useful machines where downtime would be an inconvenience

Differentiators

- Data sensitivity and criticality
- Consequences of downtime
- Difficulty of repair and restoration
- System exposure - accessibility and vulnerability to attack
- System's interconnects - what access can this machine provide to other services if compromised

Table 3.3: Factors influencing priority rating

Vulnerability and Patch Research An understanding of the effects of a patch and the problem the patch is trying to address is required. In the case of security patches this requires an understanding of the vulnerability in the software being run and how the patch remediates this.

If the trends described in sections 2.3 and 2.4 are taken into account, then it is clear that an administrator needs to know of relevant security patches and vulnerabilities as soon as they are released to minimise the effects of the shrinking time to exploit cycle. In addition, given the large number of vulnerabilities and patches, some intelligent filtering is required to limit the list to relevant patches only, with patches for software not present in the organisation weeded out. This can be achieved with software which provides filtering based on criteria such as the vulnerability criticality and affected products. While it is not a dichotomy, manual filtering is still preferable to imperfect automatic filtering, and can be more easily managed via escalation procedures. The vulnerability and patch notification should include which versions of the software are affected, a criticality rating of the vulnerability's seriousness, and what steps can be taken as workarounds or stop-gap measures. There are two factors which contribute to the criticality. The first is how easy it is to perform an exploit (if it requires obscure user interaction it would be less critical that a vulnerability which can be exploited remotely without valid credentials). The second is the impact of the vulnerability (if the vulnerability results in a simple DoS that can be easily circumvented then it would be less critical than one which provides arbitrary code execution in kernel space). In addition, the effectiveness, stability and maturity of a patch should be determined. In particular any conflicts with other software or configurations should be noted, with conflicts specific to the organisation made clear. Other indications include the number of times previous versions of the patch have been recalled and for what reasons, any testing notes provided by the vendor or other users, and any other special considerations the patch may require. These are important when determining whether or not to deploy a patch under the *risk assessment* step.

Notification can be minimally achieved by subscribing to the security notification and announcement services of operating system and application vendors. In addition public disclosure lists such as BugTraq[2] or Full Disclosure[3] can provide more comprehensive notification, though at a poorer signal-to-noise ratio. Vulnerability databases, on the other hand, can provide targeted vulnerability announcements and comprehensive vulnerability coverage including links to exploits, further discussion, and threat analysis. Mailing lists and discussion groups are also a useful resource, particularly for monitoring the effectiveness of a patch or better understanding the implications of a vulnerability. NIST provides a comprehensive discussion and listing of notification services [42]. A summary of the research goals related to patches and vulnerabilities can be found in table 3.4.

[2] `http://www.securityfocus.com/archive/1`
[3] `http://lists.grok.org.uk/mailman/listinfo/full-disclosure`

Vulnerability

1. Affected Software and Version
2. Vulnerability criticality/seriousness
   (a) Ease of Exploitation
   (b) Impact if Successfully Exploited
3. Workarounds and Stop-Gap measures

Patch

1. Software or Configuration Conflicts.
2. Number of, and Reason for, Patch Reissue
3. Vendor and End-User testing Notes
4. Special Considerations

Table 3.4: Patch and Vulnerability Detail Summary

Threat and Exploit Research While asset, patch and vulnerability notification helps provide an overview of the organisation, exploit and threat notification can help provide an overview of the security landscape. To use a crude analogy, if asset, vulnerability and patch notification provide an understanding of the vehicle, then threat and exploit notification provide an understanding of the terrain. The threats from patches themselves, including non-security patches, come from possible faults with the patch which should be determined while performing the patch maturity and stability analysis described above.

Threat and exploit notification should provide knowledge of what tools (exploits) are available to aid an attacker in exploiting the vulnerabilities, and which threats are currently known or likely to allow an exploit. While knowledge of vulnerabilities, patches and workarounds is mandatory for any patch management policy, knowledge of exploits and threats is not. However, this information allows for better risk management decisions to be made, and in high risk situations can be critical.

Unlike vulnerability and patch notification, threat and exploit notification is more difficult, with few resources on the matter. Attackers are controlled by human whim and attack methods develop fairly rapidly, and usually in secret. For example, NIST's document was the only one to discuss threat notification, but does so briefly without providing any direct threat notification resources [42, pg 2-8]. Even if a comprehensive threat notification service existed, it would be inherently crippled due to the difficulty in predicting threat actions, particularly human based threats. This is not to say threat and exploit notification is non-existent. Quite the contrary - there are several public exploit clearing houses, honeypot projects examining attacks to discover attacker's methods, internet telescopes analysing malicious traffic looking for attacks and, commercial vendors (particularly anti-virus vendors) receiving feedback from their software installed on customer's machines. However, the security community's knowledge of the exploits and attack methods traded in underground communities is still limited, and it should be assumed that an exploit exists for every vulnerability. This is especially true if a patch has been released; reverse engineering of patches can allow for rapid exploit creation [66]. Public exploit clearing houses such as FrSIRT[4], milw0rm[5] and PacketStorm[6] should also be monitored, and are particularly useful for exploits released after the announcement of the vulnerability or patch. Exploits released with a vulnerability advisory are usually referenced in the original advisory or by vulnerability databases. Information about how effective and easy to use the exploit is should also be noted, as sometimes crippled exploits are released (though these can be improved rapidly).

The public release of a scripted exploit has been shown to lead to an upsurge in attack activity[38] (see figure 2.2). Knowing when this happens can change the parameters of the risk management equation and indicate when patch deployment should be sped up. Monitoring services such as DSHIELD's *Top 10 Targeted Ports* and *Port Reports* can provide insight into the size of potential threats, and also provide an early warning system of current or impending attacks. For example, after a recent spike in activity on port 1025 led the Internet Storm Centre to issue a warning [97], it was discovered that the increase was due to the release of the Dasher worm [98] exploiting the Microsoft Distributed Transaction Coordinator service described in MS05-051 [99]. While the reason for the increased attacks was only evident later, administrators could still have taken steps to mitigate the potential for attack, or to speed up patching of the MS05-051 related patches. This process should be integrated with the organisation's own monitoring from a variety of relevant local sources such as firewalls, web-servers, anti-virus and intrusion detection system logs, par-

[4]`http://frsirt.org/FINDOUT`
[5]`http://milw0rm.com/`
[6]`http://packetstormsecurity.org/`

ticularly if signatures exist with which organisations can detect known attacks. These signatures can be anti-virus signatures with statistics gathered at the mail server or the intrusion detection system's logs, whichever are the most relevant sources. Work by research groups such as the Internet Storm Centre (ISC), CERT and malware research laboratories (F-Secure, LURHQ etc.) should also be monitored to discover the source of potential threats. While these generally only provide an overview of threats exploiting on a mass scale, it is still useful to know whether most attacks are coming from an automated worm, diverse group of script kiddies, or coordinated criminal organisations. This should be augmented by an organisation's own threat source analysis which, for large organisations, often includes competitors. With these tools and resources, information such as available attack and exploit tools, the frequency and scale of observed attacks, and the entities most likely to attempt an attack should be researched. Complacency due to a lack of obvious mass attack activity is dangerous, as a quiet network does not preclude the possibility of an intrusion. The risk from threats to high-profile and high-exposure targets such as a large company's web server should be considered higher. Unfortunately none of the four policies provide any discussion on threat management beyond discovering available exploits. A summary of the research goals related to threats and exploits is provided in table 3.5.

The information gathered at this stage should be compiled into useful forms (such as an internal advisory document) and distributed to relevant stakeholders. This distribution can vary and should be determined by each organisation. A detailed version should be compiled for use within the patch and vulnerability group, as this will be used throughout the rest of the process. In addition, some action can be taken at this early pre-patch stage to reduce the risks of a threat exploiting the vulnerability. Intrusion detection and anti-virus signatures can be updated to detect and prevent possible attacks and exploits, this is discussed further under defence in-depth in section 5.3.

Exploit

1. Availability (is it publicly available)
2. Effectiveness
3. Ease of Use

Threats

1. Observed Attacks Frequency and Scale
2. Entities Most Likely to Attempt an Attack
3. Profile of Vulnerable Machines

Table 3.5: Exploit and Threat Detail Summary

Automated tools can help when monitoring resources for information about vulnerabilities, patches, threats and exploits, particularly XML feed aggregators. Many of the resources discussed provide their topical information in an XML feed for easy syndication. A feed aggregator can monitor these feeds and provide alerts when there is new content. These feeds can even be used to syndicate the content from a mailing list. Many of the resources discussed above such as vulnerability databases and exploit clearing houses provide feeds updated with their latest content. While other resources such as the ISC and AV vendors provide a regularly updated web log (blog) detailing and discussing new threats.

3.2.3.2 Risk Assessment

This step's primary concern is deciding on the risks presented by not patching and comparing these to the risk of applying a patch. Risk Management is a large field that could fill several theses by itself, and this section does not aim to provide a complete description of how to implement a risk management process. Rather, the focus is on the specific aspects of managing the risks of patching. Minimally, this step should allow a patching policy to answer the question: "To patch or not to patch?" However, given that in the face of a security threat the decision will usually be to patch, a mature risk management policy should also allow better judgements about *when* and to *which* systems a patch should be applied. Retrospective judgements should also allow for improved risk mitigation in the long term.

This step provides a high level risk assesment approach, however any risk methodology can be used in its stead. There is a serious lack of discussion on patching risk management within the reference documents. While each document does discuss the decision of whether to patch or not, it is sometimes inaccurate and often incomplete. For example, Chan [91] never mentions the possibility of choosing in favour of not installing a patch or the factors which could lead to such a decision. Voldal [94] touches briefly on risk management, but does not include it as a step in the patch management policy. NIST [42] spends more time discussing the problem and provides three relevant factors (the description is paraphrased):

1. Threat Level - public and high profile servers are more likely to be attacked.
2. Risk of Compromise - the likelihood of a compromise occurring.
3. Consequence of Compromise - the end result of a successful intrusion.

This is a strange list; they do not indicate that the *threat level* should be one of the primary inputs into *risk of compromise*, whereas the *consequence of compromise* would not, they are either being inconsistent or were not aware of specifics. In addition, NIST seems to be breaking from their earlier definition of the word *threat* (see 1.2.1). Sun's document provides the best discussion of the four, but still only deals with a limited subset of issues, namely *cost* and *availability* which, while necessary, are not a sufficient enumeration of possible consequences. The decrease in the time available to patch, the increase in patches and the problems some patches have caused have only recently highlighted the need for solid alternatives to applying a patch. This, coupled with the high risks of not applying security patches may have lead to the insufficient attention to risk management in the referenced policy documents. However, as patching and patch management matures, vendors have increasingly been providing alternative workarounds. In addition, third-party technologies such as anti-virus, firewall and intrusion detection systems have helped to provide additional protections which can stave off patch installation until the patch is considered stable and tested. In searching for further work into the to-patch-or-not trade-off, the patching related risk management work by MacLeod [93] was found a useful reference and is discussed further.

This discussion will revolve around three important factors which constitute risk; threat, vulnerability and, impact. These three factors are relatively independent of one another, making it possible for one of the threat, vulnerability or impact levels to change without influencing the other two factors directly[7]. This is in contrast to both the NIST and Sun policy. The terms threat and vulnerability are used in a wider scope here than the rest of the document. Their use above is specific to a security vulnerability and their related security threats. In this context the risk of both applying and not applying a patch is determined, thus the threats and vulnerability of applying and not applying a patch must be determined.

Risk Risk is defined by the equation: $risk = threat \times vulnerability \times impact$ [93, 100] . in more detail, risk is the probability of a threat successfully exploiting a vulnerability and bringing about the ensuing consequences [24]. Thus, as McLeod [93] states "you need to experience a level of threat to a vulnerability and a significant impact (Cost) for the vulnerability to present a significant risk." For example a high profile target (e.g Citibank) would always have high risk levels because of both the increased probability that it will be attacked, and the increased probability that a successful attack will have a large impact. However, if it can reduce its vulnerability surface it can reduce its risk. This definition appears targeted at discussing the risks of not applying a patch. If we discuss the risks of applying a patch, then concepts such as the threat-source become less obvious. When the threat source is the actual patch, the vulnerability would be the vulnerability to a system failure due to a faulty patch and the impact would be the resulting cost and downtime. Thus, this definition of risk can cover both situations.

The following three factors are discussed below; patch and security threats, patch and security vulnerability, and consequences and impact. These are inputs into the risk equation. Once they have been determined, the risk of a decision can be determined and compared.

Patch and Security Threats Threat has already been defined in section 1.2.1. For the sake of ease it is paraphrased as: an entity or adversary with the capabilities, intentions, and attack methods to exploit a vulnerability in an asset. In the context of security patches, this entity is usually malicious and could be anything from a curious teenager, professional cracker, enemy government or automated worm. At this point the threat research from step one should be compared

[7]Although there is some influence. For example the impact of intruding into a large financial organisation would be large, which may contribute to the higher profile, and hence increased threat level of the organisation.

with information such as the public profile of the vulnerable systems. For example a vulnerability in the FBI's web servers is likely to attract a higher threat level. This will allow the threat level to be gauged while the patch remains undeployed.

The threats to systems and assets when deploying a patch are different, and stem from possible faults in the patch itself. These can manifest in a variety of ways, from affecting other systems and software negatively, to re-introducing further problems or failing to perform the function for which the patch was issued. This type of threat level is gauged from the research performed into the maturity and stability of a patch during step one. For security patches the possible threats faced by deploying a patch usually pale in significance relative to the alternative, namely an intrusion. With proper testing, regular backups and careful monitoring, the threats from patches can be discovered, planned for, and mitigated. An intrusion by an attacker on the other hand, depending on skill and motivation, could be far more difficult to detect, cause far more damage and be more difficult to repair. For example, if an attacker were to steal sensitive customer account details, the option of a quick restore from backup is not available.

However, whether choosing to patch or not, the threats faced by not patching will always apply as, even if the decision is made to patch, the organisation will still be vulnerable from the time of public vulnerability disclosure until the patch is deployed internally. This will be discussed further under the scheduling step (section 3.2.3.3).

Patch and Security Vulnerability The level of vulnerability of an organisation to a particular threat can be calculated as the seriousness of the vulnerability multiplied by the number of vulnerable machines multiplied by the exposure of the vulnerable machines. The asset information gathered in step one should allow every machine running the vulnerable service, software or system to be determined. The seriousness of the vulnerability should not include the impacts of successfully exploiting the vulnerability, but rather the level of access required for the vulnerability to be exploited and the ease with which it can be exploited. These should be determined when performing the vulnerability research discussed above. Not all systems are likely to be exploited, as machines whose vulnerable service is publicly available are more vulnerable to attack than services which are only internally available. However, internally available systems are still more vulnerable than systems which disallow direct user access, because the vulnerability of internal machines must be considered if a threat penetrates the organisation's border. Thus, a grade of vulnerability should be applied to each vulnerable machine. The grade would be tempered by the ease of exploitation. The level of vulnerability of an organisation will then be a summation of the vulnerability grade of the vulnerable machines.

The vulnerability of not applying a patch is different, as it is easier to calculate and easier to minimise. As the patch is being installed by the organisation, the likelihood that the patch will be installed to the relevant systems is near 100 percent. Thus, estimations of the machines' exposure are unnecessary and the vulnerability will be based on the number of machines which are to receive the patch. As mentioned above, there is a window of vulnerability before a patch is deployed. This measurement can help to determine when the patch should be deployed and is discussed further under the scheduling step.

Consequences & Impact Consequences are the result of a threat being successfully realised against a vulnerability. These are the both the direct and indirect consequences. Direct consequences include; cost of recovery, cleanup and re-deployment, downtime and loss of availability etc., which are usually only marginally influenced by the specifics of the business. Indirect consequences include; lost revenue, damaged reputation etc. and are business specific effects of the direct consequences. Together these consequences impact on the realisation of a threat. For example, lost availability on a mission critical server will have a larger impact than the same consequence on a less important server.

Calculating consequence consists in working out what effects the realisation of a threat against the vulnerability would have. The three most obvious consequences are the costs of a particular action, the lost availability and the interruption to operations. It is difficult to determine the consequences of an intrusion, as the extent to which the intruder can compromise vulnerable machines cannot be predicted. For example, an intruder could compromise a machine and use the access to snoop on an organisation's activities, launch other attacks, or just disable the machine. It is equally difficult to predict the behaviour of a faulty patch, which could disable a server, re-open an old vulnerability or cause miscalculations in critical billing information. However, more time is available to prevent and plan for patch failures. In either of these situations (if the attack/fault is detected) the server will have to be rebuilt resulting in associated downtime and varying costs to the organisation, both direct and indirect.

To avoid the complications of enumerating all possible consequences, the criticality and priority of the vulnerable machines determined in step one should be used to gauge the impact. If the vulnerable machines are of higher criticality then the impact will be higher. This allows the impact to be usefully abstracted.

The Australian Department of Commerce's *Information Security Risk Management [101]* document recommends assigning an impact level to one of the measures described in table 3.6 *[101]*. This is a basic example, and any risk methodology in use at the organisation can be used to augment or replace this measurement. The impact levels are deliberately vague, as the specifics of what differentiates a catastrophic from a major risk need to be specific to each organisation.

Qualitative Measure	**Description**
Catastrophic	Critical services and core business operations would be threatened.
Major	Effective service provision would be threatened and require top management intervention.
Moderate	Core services would function, but an organisational review or procedure change may occur.
Minor	Some services would suffer, but not fail. It could be dealt with internally.
Insignificant	Routine operations and maintenance could repair it.

Table 3.6: Impact Level *[101]*

Assessment Once the threat, vulnerability, and impact levels have been determined, a decision on the risk posed by an action can be taken. The Australian Department of Commerce's *Information Security Risk Management [101]* provides a good methodology, where the threat and vulnerability level is used to determine the likelihood (see table 3.7 *[101]*) of the threats exploiting the vulnerability. This is then used along with the impact level to determine the level of risk (see table 3.8 *[101]*). Finally, this risk assessment does not stop here, but will be constantly modified and used in the next steps.

Qualitative Measure	**Description**
Nearly Certain	The threat level and vulnerability level are both high making this almost certain to occur.
Likely	The threat and vulnerability level are high, but it is not certain this will occur.
Moderate	It is likely this event will occur, but it probably won't happen immediately.
Unlikely	It is doubtful this event will occur, but there is still a possibility.
Rare	The threat and vulnerability levels are so low this would only occur in an exceptional circumstance.

Table 3.7: Likelihood *[101]*

	Impact				
Likelihood	Insignificant	Minor	Moderate	Major	Catastrophic
Nearly Certain	H	H	E	E	E
Likely	M	H	H	E	E
Moderate	L	M	H	E	E
Unlikely	L	L	M	H	E
Rare	L	L	M	H	H

E = Extreme Risk
H = High Risk
M = Moderate Risk
L = Low Risk

Table 3.8: Risk Level *[101]*

3.2.3.3 Scheduling and Patching Strategy

Too often in the past patching was done in an ad-hoc, 'as the patch arrived' manner. Vendor release policies have helped this somewhat (see section 4.3). To ensure that patching is done regularly in a controlled and predictable manner a patch schedule should be created. This schedule will primarily be informed by the initial risk assessments performed in the previous step. Chan [91] recommends the creation of two patch cycles.

- Regular, defined and predictable cycle for non-critical standard patches. Usually with time-based triggers.
- Expedited, when necessary cycle for critical patches. Usually with event-based triggers.

The first is a regular cycle whose purpose is to ensure the application of normal, non-critical, standard patches and updates, these are often non-security updates, updates for which an effective workaround/mitigation exists, or an update for a vulnerability the organisation's security infrastructure already mitigates. The cycle can be either time- or event-based e.g. monthly or after the release of several such patches. This can be split to form a separate longer cycle for large cumulative updates such as service packs or operating system upgrades. Given the large number of changes such upgrades introduce, they usually require more testing and integration, e.g. training support staff, upgrading related applications, integrating software. Thus, a longer, more carefully planned cycle can be split from the first. The second cycle's purpose is to ensure the installation of critical security patches and updates, and should be completed whenever a critical patch is announced.

An initial assessment of the patch is required to determine which of the two cycles a patch should be placed in. In addition, within each of these cycles a hierarchy of patch priority should be developed to determine what order patches and machines are worked on. These decisions will primarily be informed by the risk level associated with a patch and it's related vulnerability. When determining when to patch, there are two conflicting risks, as shown above. The first is the risk of applying a patch, the second the risk of compromise (or the risk of not applying a patch). In their seminal work on the subject Beattie *et al.* [60] describe how the optimal time to patch can be solved. Over time the risk from compromise will increase, as exploit and attack tools are published and improved, while becoming more widely known, and the risk from a patch will decrease as bugs are reported and the vendor re-issues the patch or provides advice. Thus, a hypothetical graph of the risks will look like figure 3.1 [60], where the optimal time to patch is at the intersection of the two risk curves. Beattie *et al.* provide research analysing a cross-section of patches and vulnerabilities and showed that the optimal time to patch was at either *ten* or *thirty* days after the release of a patch. This was based on comparing the number of times patches were re-released due to problems to the number of intrusions. The risk drops off at the ten day

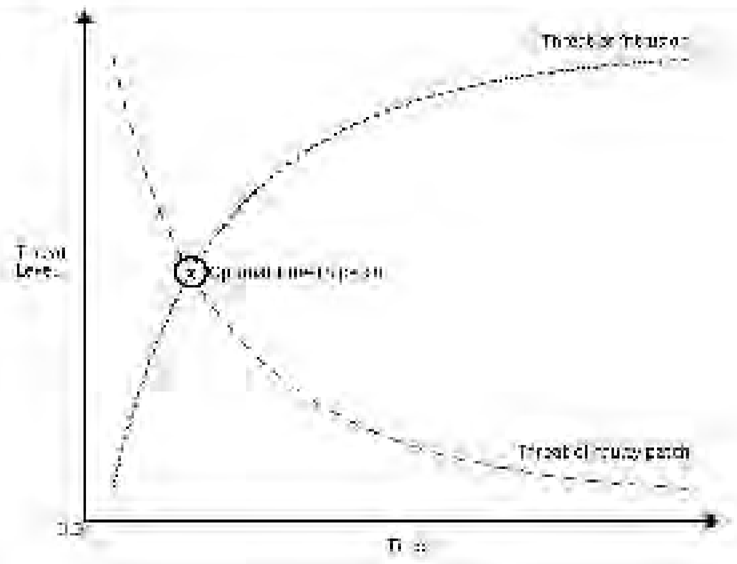


Figure 3.1: Hypothetical graph of the risk of compromise and patching [60].

and thirty day markers. TheyBeattie *et al.* also provided their methodology, encouraging further research on the optimal patch application time of specific vendors. Ideally, solving the optimal time to patch for the specific subset of software vendors used within each organisation would provide an optimised estimate, and could be shared with the wider community. It is important to note that if an organisation has sufficient resources to thoroughly test a patch before deployment, then the risks a patch presents to that organisation can be reduced at a sharper rate, speeding up patch deployment and reducing the risks posed by a compromise. The ten and thirty day deployment suggestion is primarily for smaller organisations with limited resources that cannot afford to deploy a large patch testing regime. The specific risks, and risk thresholds should be worked out in the previous *risk assessment* step.

Sun Microsystem's document *Solaris Patch Management: Recommended Strategy [30]* recommends a strategy not mentioned in other documentation: minimising change. The argument supplied for this supports the risk assessment conducted above, and should be taken into consideration when crafting a patch schedule. The argument claims that "overall downtime, planned and unplanned combined, goes up with more frequent application of patches. *[30]* " This theory is demonstrated in figure 3.2 *[30]*. Up-time increases at a constant rate over time, which in an ideal world (with no downtime out) looks like a straight light from the point of origin. However, if there is an outage up-time stops increasing and a plateau is reached. The current patching

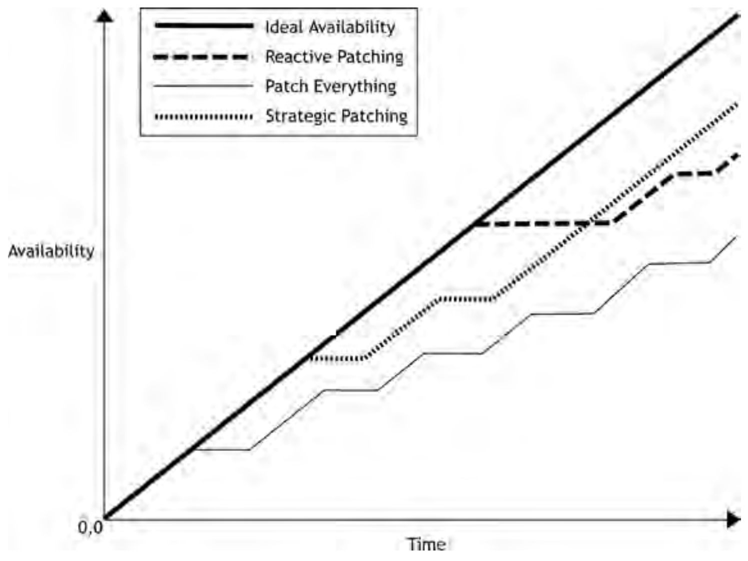


Figure 3.2: Patch application and its impact on Availability *[30]*

strategy, 'apply every patch,' thus looks like a regular set of plateaus that results in the lowest total up-time. The other extreme is to apply patches only after a failure such as an intrusion. A failure would result in unplanned downtime which would be longer than planned downtime, due to the extra time required for problem diagnosis, to divert resources, and due to being less prepared. Once the failure resulting in unplanned downtime is corrected, an additional planned downtime is necessary to apply the fixes that could have prevented the intrusion. This is represented by the reactive line. It is interesting to note that it is possible for reactive patching to result in less downtime than applying every patch. This graph does not include downtime due to faulty patches, which would presumably reduce the up-time of the current strategy even more. However, the reactive strategy isn't acceptable, and intrusion can have far more negative results that just downtime. Therefore, a strategic patching schedule should seek to optimise between these two extremes. By only applying necessary patches, the planned downtime from patch installation, downtime from patch failure, and downtime from an intrusion can be minimised. Given that downtime also has a cost element, strategic patching can also help to reduce the costs of patching.

Minimising change recognises that not every patch that is released is applicable to an organisation. There are two primary considerations to minimise change *[30]*.

1. Address only known issues for which no acceptable workaround exists.

The patch and vulnerability group should analyse the patch and identify whether the organisation suffers from the problem it purportedly fixes. If it does, then research into alternative 'cheaper' methods of remediating the problem should be conducted.

2. Keep current according to business needs.

   The version of software used should be the lowest, still maintained, version appropriate to the specifics of an organisation, unless new software is being deployed. In addition, new features should only be deployed if necessary to business needs.

For example if there is a vulnerability in a mail client that only affects people using the IMAP mail protocol, then users of the software who do not use IMAP (and have it disabled) but rather POP3 can ignore the patch. Alternatively, if an acceptable workaround such as disabling a non-critical service exists, it can be used instead of the patch. This will ensure that the software is kept as up to date as your organisation requires, instead of as up to date as the vendor has allowed. An intelligent choice of which patches should be installed can reduce the number of patches installed. However, it is important to ensure patches are distributed to all vulnerable machines [42, pg 2-11]. Minimising change by limiting distribution to high risk groups only is an ineffective measure due to the nature of an intrusion, where often low criticality and low risk machines are compromised first. Thus, providing an attacker with internal access to the organisation [93] from which further attacks attempting to achieve a higher level of compromise can be performed.

A patch schedule should be created from this information. The triggers and timing of this schedule should be specific to the organisation. The optimal time to patch for the software used within the organisation should be determined and used to determine the lengths of, or triggers for, the schedules. In addition a strategy for deciding into which schedule a patch should be placed should be determined. This will take as its primary input the risk assessment from the previous step. This should be used first to decide into which cycle a patch should be placed, and second to determine when a trigger has been reached.

3.2.3.4 Testing

Testing is a critical part of any patch management process. The primary goal of testing is to reduce the threat of faulty patches discussed in the *risk management* section. Given the amount of regression testing that can be required, this goal can be the primary delay in patch deployment, and stands in competition with the need for rapid patch deployment in the face of a shrinking exploit and patch window (discussed in section 2.3). Worse still, the hasty application of a faulty patch can cause a wide range of damage. For example it could fail to remediate the vulnerability, undo fixes from past patches, introduce new vulnerabilities, impair the functioning of the software being patched, or impair the functioning of other software. This could be either malicious or accidental. Testing is especially important if an automated patch deployment solution will be used. A common worry about automatic patching is that faulty patches will be deployed automatically [33]. This is usually due to inadequate patch testing, both from the vendor and the organisation. The deployment of a patch to production machines should not be considered testing, whether manual or automatic. To minimise these risks of patching it is critical that an organisation thoroughly test a patch before it is deployed. Testing is primarily a technical step to determine whether the patch and resulting updated software will actually correct the vulnerability, and that the affected components continue to function correctly. Particularly in the context of your organisation's specific configuration. However, additional information such as the likely disruption to business during patch deployment and any changes to business processes can be observed and documented.

The steps performed when testing should be determined within the organisation and documented for each relevant system and piece of software. These checks can vary from a simple check that the patch installed and the system rebooted correctly, to a series of automated scripts checking the critical functionality of the officially supported software. Standardised configurations set to a common baseline should be created. This helps to reduce the number of different configurations which need to be managed, as it does in the production environment. A document for each baseline should be created with the expected behaviour of the system described and verifiable tests provided to check this [102].

Ideally, the testing should be done in an environment which exactly mirrors the production environment, however this is very often not possible. At a minimum, the test environment should represent all mission critical servers [91]. It is not always possible to re-create the exact production environment, particularly in organisations with limited budgets. Standardised baselines allow the configuration of machines to be more easily defined and re-created. With standardised baselines, a testing environment would need only one example of each configuration. Virtual machine technology can be used to reduce costs and re-create particular environments. Several virtual machines can be run on one actual machine, creating a 'lab-in-a-box' which can dramatically reduce the hardware costs of setting up a test lab, and improving ease of maintenance [94]. The downside is that specific physical hardware interactions can be difficult to model with a virtual machine, particularly for hardware-specific software such as drivers. Three excellent products which can be used here are VMWare [103], Xen [104] and Microsoft Virtual PC [105] and Virtual Server [106]. After testing patches in the lab, they can be deployed in a waterfall style roll-out, where patches are deployed to the lowest criticality, easily recoverable machines first, then continue up the criticality hierarchy. This can help to discover any bugs missed in the lab while helping to minimise risk, but may still result in some unwanted downtime and should not be used as the primary testing method.

While the tests will mostly be organisation specific, some tests are common to all patches. Some patches rely on other patches or supersede previous patches. All required patches and their dependencies should be tested and deployed in the correct order. For example: Oracle's *AD Merge Patch* [107] can merge several patches into one install path and can help to reduce the complexity of installing multiple patches; Sun and other vendors accumulate their patches into one package before hand [30]; while Debian [69] and Microsoft [108] build dependency checking into their deployment tools. In addition, vendors often release cumulative or roll-up patches. Most vendors provide a method for checking the authenticity of a patch. The most basic versions involve checking a hash[8] of the file with a fingerprint available at the vendors website. More advanced authentication mechanisms involve an automated check for an authoritative digital signature. This authenticity should be re-checked as the patch is moved around the organisation, to prevent tampering [91]. Some organisations may choose to put their own signature on the patch. After verifying the authenticity, the patch should be scanned for any malware by an up-to-date anti-virus software package. If possible, the patched software should also be scanned in case the patch contained malware that only became obvious once deployed - one such example can be found in Ken Thompson's *Reflections on Trusting Trust* [109]. None of these methods are guaranteed to protect against all malicious patches - for example, if the creator of the patch had an, as yet unseen, trojan stowed away in their final patch release, it would appear signed, and most anti-virus packages would not detect the trojan [42]. After deploying the patch to the test environment, it must be ensured that the vulnerability has been correctly patched and that no new vulnerabilities have been re-opened. This is mostly easily, but not completely, checked by a vulnerability scan. Repeatable tests that can be used to ensure that a patch has been correctly installed should be devised and documented. Most often the verification of a successful deployment is provided for

[8]Given the recent cryptanalysis attacks against MD5 and SHA-1, verifying with one of these hashes is not sufficient. Other hashing algorithms such as SHA-256 should be employed.

by automated patching software. However, the range of applications and functions which require patches through their life-cycle mean that this is not always straightforward. Some vendors provide information on how to verify that the patch was installed by providing repeatable checks that can be performed. These checks can include looking at file versions and hashes, checking configuration settings, observing different behaviour etc. and will have to be determined for each patch. Some patches also provide a method to undo the changes in the event that a patch needs to be rolled-back [42]. Debian, Red Hat, Solaris and Microsoft all provide some patches which can be easily removed, but not all patches have this functionality, and without exact copies of the previous files they often revert to a default state which is not always desirable. These should be tested and appropriate backups taken to restore the system if a patch needs to be removed and the undo functionality does not work or is not present.

Given the need for an expedited testing process, several methods can be employed. On a procedural level, noting the interactions already tested by the vendor can save time while subscribing to the vendors patch notification service can provide early warnings and reduce redundant checks. Community lists should also be monitored. For example the patch management mailing list[9] often has discussions on faulty patches and their solutions. If possible, automated tests for the core business process should be implemented, for example comparing the accounting information produced from the same input sent to two versions of the software, one patched the other unpatched, can catch subtle data corruption bugs and can be trivially implemented. Performing as much of the testing overhead as possible before the announcement of a patch will help to reduce the time taken when the patch is released. For example, having a regular automated back-up system in place can reduce the time required to make back-ups before a patch is deployed, allowing removing the patch to be tested more rapidly. Lastly, checking the dependencies of the patched software can allow for certain tests to be prioritised over others. For example, if a patch updates a dynamic library then all programs depending on that library should be tested. As a further time saver, scoping the testing to the changed functions can result in fewer tests without significant danger. For example, if only one cryptographic algorithm is patched in a dynamic library, then testing all of the algorithms is somewhat less useful. These tests can also help in determining which services should be checked for new or old vulnerabilities that were inadvertently created by the patch. However, scoping the tests too much can result in an incomplete study and missed bugs.

Due to the difficulty in working out program dependencies and preventing too narrow a scope,

[9]`http://patchmanagement.org/`

some tools have been developed in an attempt to partly automate this procedure. The assumption is that if the patch can be analysed for all possible dependencies, then the scope can be narrowed to only testing dependent programs with no danger. In addition, reverse walking the dependency tree allows fro better troubleshooting of faults. Two such tools attempt to do just this, Microsoft's *Strider* product [110] or Solaris' *sowhat* [111].

If there are several possible methods of remediating a vulnerability, then the assessments must be carried out on each one. The risk management step provides the tools which can ease making a decision between the competing threat from patches and from attackers. Thus, any additional information as to the threats a patch poses to the organisation must be used as an additional input into the ongoing risk assessment. At this point an assessment as to whether the risks are such that the patch should or should not be deployed must be made. If the decision is to not deploy the patch then alternative layers of defence must be tweaked, this is discussed further in section 5.3.

3.2.3.5 Planning & Change Management

Much of the purpose of a patch management policy is to manage the change introduced by a patch. As such, integration with existing change management structures is critical to its success [91]. As with risk management, change management is a large field that will not be discussed in detail here. The primary goal of change management will be to provide documented procedures for various aspects of the patch application to keep changes consistent and avoid surprises. Having a clear change management policy will help when troubleshooting problems, as specific changes which caused the problem can be pinpointed, relevant personnel summoned, and future problems avoided. During this step a plan for how the patch will be deployed and the changes logged will be developed. It should seek to minimise the risks of patching by fully utilising the advanced warning afforded by knowing when a patch will be deployed. The benefit of advanced warning is that contingency and back-out plans can be developed. The end result should be a documented process specifying explicit steps when planning for and applying change and ensuring accountability for applying changes.

Such a policy requires four important functions [91]:

1. Proposed Change

2. Contingency and back-out plans
3. Risk mitigation
4. Patch monitoring and acceptance

The proposed changes, namely the patch or workaround that will be deployed, should be documented. The details of what change is introduced would have been discovered during testing. These changes should then be approved and signed off by the people responsible for the systems which will be modified. This will help to provide a clear authenticated audit trail of changes introduced. To prevent inconsistent deployment, access controls should be used to disallow users or other programs installing their own patches, unless it is preferable to do so. To enforce this consistency policy, guidelines should be drawn up as to what level of drift from the baseline is acceptable, and how users should behave and respond to patch deployment notifications. These guidelines should be coupled with regular checks to ensure that they are effective. These changes must be distributed to relevant stakeholders, which can be achieved with an organisational patch and remediation database. The advantage of this is that it provides a central resource than can be referenced at a later stage, when information on the patching process is required, such as when creating new baselines or calculating metrics. Change management allows dependencies to be created between groups and systems, allowing a change in one group to trigger an alert to another group that might find the change relevant. This is important when maintaining operational baselines, as build images and documentation must be updated to include the deployed patch. To quote Chan [91] "These modifications are most ideally and suitably handled via an enterprise-wide change management system."

Contingency and back-out plans should be prepared for a worst case scenario. Documentation describing what is being installed, its intended outcome and how to remove it should be drawn up. The procedures to restore system state from back-ups created during the testing step should be documented and made repeatable. Ideally these should be worked into a regular schedule that doesn't wait for a patch release, to save time during deployment. The inventory of system assets drawn up in step one can be used to inform the direct end-users and notify or request help from the people marked as responsible for the relevant assets. This will allow personnel to be notified and on standby in the event of a failure, with support staff notified of the upgrade and briefed on the relevant information with which to respond.

Risk mitigation requires performing the roll-out in a way that will limit possible complications in an attempt to reduce the likelihood of a threat being realised. To achieve this, both the technical and procedural aspects of deploying the patch should be analysed for possible failure. Any failure points should then either be removed, mitigated or minimised. On a technical level this may require that the infrastructure can handle the patch deployment. For example, ensuring that the file server distributing the patch has enough bandwidth, and if not, staggering the times at which machines update or providing more bandwidth, are possible solutions that should be implemented before a deployment. At a procedural level this requires ensuring that the necessary non-technical components for both the changes and contingency plan are available (for example, ensuring relevant personnel are available or staggering updates to ensure that personnel are not swamped with troubleshooting). This is often a difficult task to perform, as it is not always easy to see the pitfalls. Previous experiences with faulty patch installations should always be documented, and can provide a useful resource when looking for possible points of failure. A common component of risk mitigation will include details of patching machines that the automated deployment methods failed or are unable to patch, such as machines that were powered off, mobile devices that were outside the organisation network, and unsupported software and hardware devices (e.g. router firmware). These must be planned for. Common solutions include:

- Using pull-based patching, where the device pulls its own patches as soon as it can
- Quarantining unpatched devices in a limited access network sub-net
- Enlisting the help of users
- Plain manual patching

A deployment schedule should be drawn up, detailing which systems and groups of systems will be patched and in what order, and taking into account the business needs and risks associated with each group.

Plans relating to the monitoring and acceptance of patches detail which criteria must be met for the patch to be considered successful and how these criteria will be monitored. This will provide a specific and measurable milestone for the completion of the upgrade [91]. It is naive to think that all patches will install smoothly and working in emergency mode until all patches are installed can be a waste of resources, and divert attention from more important vulnerabilities and threats. This should provide specific and measurable criteria based on the level of risk the organisation is facing and find acceptable. In addition, these points can be used when developing patching metrics discussed below.

3.2.3.6 Deployment, Installation and Remediation

Many system administrators have the most experience with this stage of the process [91]. Often when referring to patch management or patching, many are actually referring to the physical act of installation or deployment of patches. Due to the focus on deployment this is one of the better understood steps, and the area into which the most work has been performed, particularly into automated patch deployment tools. It is important to view this step as part of a larger patch management process, and it is the snag many patch management products fail to realise.

This step is primarily concerned with creating a method for effectively deploying patches with minimal manual intervention. Unattended automated deployment is not always desirable however, and it may be more appropriate to patch mission critical systems manually, during off-peak hours [94]. Automated solutions do help aspects such as reducing costs of large-scale deployments and automating repetitive stages of the patch management process, providing both a benefit to speed and reduced chance of human error [30]. More on current patch management solutions can be found in section 5.2 with further discussion of the technical aspects of an automated solutions discussed in section 5.2.1.6. However, not every piece of software and device will be supported by the automatic deployment methods used. The plans drafted in the previous *change management* step (see section 3.2.3.5) should be followed and should handle predictable problems. This control will help to prevent drift from the consistency correct change management procedures seek to create. The patches should be deployed in a controlled and predictable manner that limits disruption to the business' processes.

Several technologies can be used to improve the speed and accuracy of patch deployment. Compression can help to speed the transfer of the patches to end-user machines. Distributing patches as binary differentials can dramatically reduce patch size [112]. Encryption can help to reduce the chance of tampering and hide the often sensitive information, such as details of operating system, hardware, installed applications and patch levels, being sent between client and server machines. Digital signatures, particularly if they have already been implemented within an organisation-wide public key infrastructure, are a mostly mandatory method of preventing tampering with patches and ensuring only approved patches are installed. Unfortunately, many of these features need to be implemented by vendors, and patch deployment tools are not always developed in-house. These technical features are discussed further in section 5.2.1.6.

It is important to remember that, since patch deployment tools usually install software at a higher privilege level to many machines in the organisation, the severity of a compromise of the patch deployment tool would be high, allowing it to be used as a malware infection vector. Unfortunately, this security implication of correcting security vulnerabilities is sometimes ignored. For example, when Microsoft released the MS05-038 patch [113] with corrupted digital signatures, neither Microsoft nor end-users mentioned the possibility that this was the same symptom a compromised patch would demonstrate [114]. Thus, the security of the patches and patch distribution mechanism should not just be a function of testing, but rather a constant pressure, with every stage of the patch's life-cycle authorised and authenticated, from first obtaining it from the vendor, right through to its successful deployment [91].

In a larger organisation, multiple patch deployment methods may be used, as determined by relevant business units. In this case, it is appropriate for the patch and vulnerability group to provide the relevant information to the various parties. Once again, the organisational patch and remediation database mentioned earlier can provide this. This can also be useful in allowing end-users to apply their own patches for organisations which give the user more control over their desktop machines, such as universities or other research institutions.

3.2.3.7 Verification & Reporting

Not every patch deployment will be successful. Some machines will be unavailable during the roll-out while others will fail mysteriously. The goal of this step is to verify the successful installation of the patch, and discover which patches failed to deploy to which machines, and why.

The deployment plans drawn up will have detailed which machines and groups of machines the patch should have been deployed to. In addition, during testing, specific repeatable tests which can be used to verify the successful installation of the patch should have been provided. The documentation and resources provided by the asset and host inventory created during information gathering, the patch verification steps drawn up in testing, and the deployment plans created in the change management step, should provide an easier way to verify that the machines and services to which patches were deployed had the patches successfully installed. It is interesting to note that Chan [91] argues that this step should contain the asset and host management inventorying, which the present policy recommends performing during information gathering. Given that system discovery is critical for more security aspects, it is believed to be more appropriately placed where it is currently.

Verification that the patch has been installed and that the vulnerability has, in fact, been remediated needs to be conducted. It should have been ascertained during testing whether the patch does remediate the vulnerability, thus verifying that it can be minimised at this point. However, if the vulnerability has been remediated then it can be assumed that the patch was successfully installed (but not vice-versa). Thus, if the choice is between verifying the patch install or verifying that the vulnerability was remediated, the latter should be opted for. Verification can be either direct or indirect. Direct methods would include actions that require local machine access[10], for example checking patch logs and file hashes or configuration options (e.g. registry settings), indirect methods are performed remotely and would include methods such as observing port connection strings or remote vulnerability scanning. Some vulnerability testing should occur by performing a vulnerability scan on a representative sample of patched machines. More on vulnerability scanners can be found in section 5.2. Vulnerability scans sometimes include actual exploit techniques and may cause harm to the system, so the specifics of the scan should be noted to prevent a harmful scan [42, pg 2-14] from wreaking the kind of damage the patch was supposed to prevent. In the time since the initial creation of the host inventory, new machines may have become active on the network or mobile devices may have returned. It is important to include them in the patch deployment. A good automated asset inventory system should update the inventory as the new machines become active, but this does not necessarily mean they have had patches deployed.

At this point some problems due to a fault in the patch should become evident. These need to be identified and remediated as soon as possible. The risk mitigation steps put together during the change management process could help to minimise the impact, by ensuring that problems are planned for and the relevant staff are ready to respond with the contingency plans. Staff should be aware that a change has been implemented and cautioned to be on the lookout for subtle inconsistencies, such as minor miscalculations, as a small fault in the patch, if unnoticed, could potentially be very harmful. At this point a decision should be made as to whether the changes should be rolled-back. This decision should be made if the patch is causing more problems than the related exploit, or if there is more chance of a bigger problem (higher risk) manifesting itself than an intrusion presents. If the previous steps have been conducted thoroughly, it is rare that this decision should be made. If it is, as in chess, it should be ensured that every system which has the patch defence removed is covered by an alternate defence.

[10]Local access is not the same as physical access, but it has similar requirements, usually valid user credentials. However, local methods can often be performed remotely. Physical access can provide direct manipulation of the machine allowing root or administrator access.

This phase should also generate reports, record relevant statistics and document any problems that occurred, to prevent repeat mistakes. These reports should be summarised and regularly forwarded to upper-management to ensure they know the patch management process is functioning correctly [94]. These reports can also be used to tweak other steps, particularly the risk assessment step, for future patches. To properly report on how well the implemented patch management process is meeting its targets, the targets need to be defined using metrics. To quote MacLeod [93]:

> Without having available metrics to measure specific aspects of your patch management programme, it is difficult to establish or set appropriate patching targets and objectives. [Which] makes it impossible to measure deviation from targets and [define] acceptable tolerance limits. Metrics help to demonstrate that your patching efforts are effective and offer the security management team solid information that allow them to communicate security posture to the business stakeholders in a meaningful way.

The metrics measured here are not limited to the most recently deployed patch - they should also be used to provide a summary for relevant groupings of patches and machines. These grouping can be time-based or be made up of a relevant basket of patches. Some example groupings are; all patches across all machines, all critical patches deployed to mission critical servers, all patches deployed in the last year to desktop machines, all critical patches pending during the last three successful intrusions. By measuring the relevant statistics, it is possible to generate new reports rapidly. Very little extra work is necessary, as the required information is gathered in this and other steps of the policy. Automated tools will help to gather these data and easily scope them to the group desired.

A particularly useful metric is that of *patch coverage*, which is the percentage of machines that have a patch or group of patches installed. The data required for its calculation are:

- Nm - Number of machines in grouping
- Np - Number of patches being analysed
- Np_i - Number of the specified patches installed
- Np_u - Number of the specified patches not installed (unpatched machines)

The equation then required to calculate patch coverage of an organisation (PC) is a simple percentage [93] listed in equation 3.1:

$$PC = (Np_i \div (Nm \times Np)) \times 100 \tag{3.1}$$

For accuracy purposes, the number of patches analysed multiplied by the number of machines in the grouping should be the same as the addition of the number of patched machines and unpatched machines:

$$Np \times Nm = Np_i + Np_u$$

This is important to ensure that the measured result ($Np_i + Np_u$) result is consistent with the predetermined result ($Np \times Nm$) and hence the patch coverage result is accurate for the grouping. For example if the metric is calculating the patch coverage of mobile devices, and only half the mobile devices are included, the metric cannot be said to be accurate. This is less important for groups of machines that are stable on the network. The opposite of patch coverage is the organisation's vulnerability coverage (VC) which provides an indication of how vulnerable an organisation is:

$$VC = 100 - PC$$

For example, if the patch coverage for all critical patches on mission critical servers were to be calculated, Nm would be the number of critical servers in the group (e.g. 100), Np would be the number of critical patches deployed so far (e.g 10), Np_i would be the number of machines found to have the patch successfully installed during verification (e.g. 800) and Np_u would be the number of unpatched machines discovered during verification (e.g. 200). Therefore, the result is:

$= (\frac{800}{(10 \times 1000)}) \times 100$

$= (\frac{800}{1000}) \times 100$

$= 0.8 \times 100$

$=80\%$ patch coverage

Scoping these metrics by time can also be useful. Providing common time intervals, such as 5 or 10 day intervals, will allow the patch coverage at the same interval to be compared across patches. Calculating patch coverage when events in the vulnerability life-cycle occur can be used as input to risk management decisions or to prove the effectiveness of the patch schedule, e.g. when the first exploit was released the total patch coverage was at 75%.

1. Time at which exploit code was publicly available for the vulnerability
2. Time at which an automated attack was released (worm)
3. Set patch coverage targets at a fixed time interval after the release of the patch.

A picture of the patch coverage at each point can be measured. Earlier it was shown that the release of exploit code is the primary trigger for an increase in attacks, so knowing the patch coverage at that point, and when a rapidly spreading worm is released, is a useful metric. The time at which the vulnerability was announced and the patch released should be included to provide a more accurate picture of the metric. For example, if the patch is released after the exploit, then a patch coverage of 0% is better explained as being caused by a patch not being available rather than ineffective patch deployment. Including the risk assessment in a summarised form along with the metric will further help to explain the patch coverage, as a coverage of 0% without any additional defences or steps taken is very different from a well defended vulnerability with no patches deployed. A larger example demonstrating how these metrics can be employed during a patch deployment is provided in table 3.2.3.7.

In-depth Patch Coverage Example A more detailed example will demonstrate the various metrics that can be determined with patch coverage.

- If we imagine an organisation where all known vulnerabilities have been patched then the initial patch coverage will be 100%.

- Later, a vulnerability is publicly disclosed and a patch is released at the same time. The patch coverage for that specific patch across the organisation will start at 0%. The vulnerability coverage is at 100%.
- A couple of days later an exploit for the vulnerability is publicly disclosed. At this point a calculation of the patch coverage for each priority group is made:

  Nm would be the number of machines in each group, Np would be 1 as we are only calculating for one patch and can be ignored, Np_i would be the number of machines found to have the patch successfully installed during verification and Np_u would be the number of unpatched machines discovered during verification. Therefore, using the patch coverage equation 3.1:

Mission Critical

$$\begin{aligned} Nm &= 230 \\ Np_i &= 191 \\ PC &= (Np_i \div Np) \times 100 \\ &= (\frac{191}{230}) \times 100 \\ &= 0.83 \times 100 \\ &= 83 \end{aligned}$$

Business Critical

$$\begin{aligned} Nm &= 654 \\ Np_i &= 196 \\ PC &= (Np_i \div Np) \times 100 \\ &= (\frac{196}{654}) \times 100 \\ &= 0.29 \times 100 \\ &= 29 \end{aligned}$$

Operation Critical

$$\begin{aligned} Nm &= 5015 \\ Np_i &= 492 \\ PC &= (Np_i \div Np) \times 100 \\ &= (\frac{402}{5015}) \times 100 \\ &= 0.08 \times 100 \\ &= 8 \end{aligned}$$

Total

$$\begin{aligned} Nm &= 230 + 654 + 5015 = 5899 \\ Np_i &= 191 + 196 + 402 = 789 \\ PC &= (Np_i \div Np) \times 100 \\ &= (\frac{789}{5899}) \times 100 \\ &= 0.13 \times 100 \\ &= 13 \end{aligned}$$

This shows that while the patch coverage of the organisation is poor at only 13%, the mission critical systems are well patched. The nature of the vulnerability could be that the business and operation-critical priority groups are well protected by adequate edge defences, and less vulnerable than the mission critical services, resulting in the focus on mission critical machines. Alternatively, if the vulnerability was more likely to affect user desktops these metrics should set off warning bells.

- Ten days after the release of a patch the organisation has defined a target patch coverage of 50%. The calculations above are re-calculated as 97% mission critical, 82% business critical, 50% operation critical resulting in a total of 56% organisational patch coverage. If the low patch coverage in operational priority machines is unusual, an investigation could help to identify problems such as a deployment fault or many out of range mobile devices.
- Several days later an automated worm is released exploiting the vulnerability. The metrics are again re-calculated after responding to any problem, and it is found that the total patch coverage is now at 98% putting the threat from the worm at a very low level.
- Another target at thirty days states that patch coverage should be 95% or higher.

As the metrics are calculated they can provide information on improving the current deployment or help identify deployment problems. In addition, they can serve as input to the risk assessment. Maintaining a database of these metrics for past patches will allow the patch coverage at the fixed points (10 and 30 days) to be compared between patch deployments.

3.2.3.8 Maintenance

The maintenance phase is initiated when patch deployment completion is reached (as defined by the change management plan). This is a meta-policy step that should allow the lessons learned to be converted into feedback with which the policy can be improved. It is a reflective step allowing aspects of the implemented policy which are not working effectively to be modified.

Each step of the policy should be examined for errors or problems that can be improved. Information gathering may require better research methodologies and resources, or its host discovery methods may need to be improved. Risk assessment may require that the risk thresholds which determine action be modified, or that the methods by which risk is measured changed. Scheduling may require a different schedule that better fits the organisations needs, or a modification of the agreed upon trigger events to ensure a faster response to patch announcements. Testing may be incomplete and require that additional documented testing procedures be added. Deployment may be consistently missing several mobile devices, and require improved methods for doing so. The verification step provides redundancy for other steps, and may help turn up inconsistencies in the way certain steps are implemented. If so, the cause of these inconsistencies should be investigated and corrected. For example, during verification it may be discovered that the host database does not identify mobile devices correctly, or that patch testing did not identify certain potential errors or whether the patch really did remediate the vulnerability correctly.

During this step other activities of the organisation should be examined to see if appropriate interactions between them and patching can be established. Two important activities that will most certainly impact on patching are staff training and software acquisition. However, the broad range of activities within an organisation may present much wider opportunities for the patch management policy to be matured.

Training Shortages of the skills required for patch management should be identified and training provided. Usually, much of the skill and expertise required to implement the organisational patch policy will reside in the patch and vulnerability groups and any subgroups they utilise. However, these skills are not always present or at an acceptable level and some training may be required. Additionally, some departments may require software that isn't officially supported by the organisation, or mobile end-users may be required to perform some of the steps from the policy themselves (although this should be limited). To meet these needs, patch, remediation and vulnerability management training should be integrated into the organisational training regime where appropriate.

3.2.3.9 Summary

A summarised view of the policy is provided in table 3.2 and the figure 3.3.

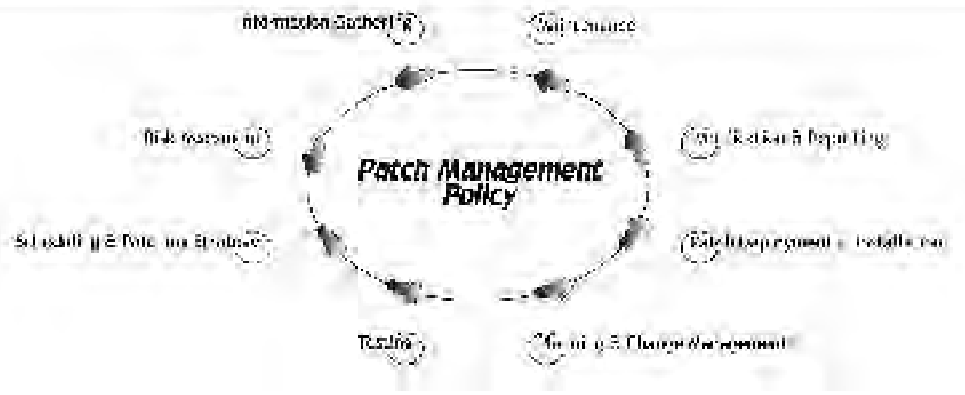


Figure 3.3: Diagram of the proposed Patch Management policy

3.3 Conclusion

Effectively managing vulnerabilities requires more than a method to deploy patches effectively. Many factors are relevant in making decisions about how best to limit the vulnerability of an organisation. The primary and final remediation of a vulnerability brings with it its own problems. Managing all of this crosses multiple disciplines, including vulnerability, configuration, change and risk management. This complexity can soon get out of hand and patching can become a chaotic affair performed in a panic and informed by incomplete and inaccurate information, chosen because it was the only information available. Implementing a comprehensive patch management policy is vital for ensuring the ongoing security of an organisation. The steps described in this chapter provide a description of how such a policy can be implemented. Each step draws from the work of several high quality sources and a thorough understanding of the current patch management field. Unfortunately, each organisation is unique, and the steps outlined describe merely how a process can be implemented, not what process should be implemented. A discussion on how to asses risk, for example, cannot judge the acceptable risk thresholds and levels appropriate for individual organisation. Specifics should be tweaked and augmented with internal policies, practices, and (most importantly) the experience of existing personnel. The trends described in section 2.3 appear to be worsening, not improving. Implementing a policy such as this takes time, since patch testing and risk assessment will initially be slow as an organisation learns how best to perform those activities in their context. Malicious attackers, on the other hand, have a head start and appear to be learning and collaborating. This impetus makes implementing an effective patch management policy critical.

This chapter has provided a discussion on what steps are required for the effective management of patches and vulnerabilities. It has focused on organisations as users of software. In the next chapter the actions of vendors when they release patches are examined, and guidance on how to best implement a patch release program in light of the complexities of vulnerability disclosure is discussed. This should complete the picture of how to manage the complete vulnerability life-cycle.

Chapter 4

Vendor Patch Release Policy

4.1 Introduction

The previous chapter discussed how users of software could best implement a policy for managing the patches released by creators of the software in response to discovered vulnerabilities. This chapter reverses these roles, discussing how creators of software can best implement a policy for releasing patches. For the purposes of simplicity, users of software will be called end-users and creators or maintainers of software will be called vendors.

Effective policies are not only the responsibility of the users of software (end-users) - software vendors must have a clear understanding of how they manage their patches, and how best to release them. Historically, vulnerability disclosure and responding to vulnerabilities has proved difficult to standardise, with a high level of confusion and antagonism between security researchers and vendors. To combat this and ensure meaningful and useful interaction between researchers and vendors, several disclosure policies have been suggested. A resource dedicated to collecting publications related to disclosure lists a total of twenty two different disclosure policies published between 1999 and 2004 [115] by vendors, security researchers and third parties. This confusion makes it difficult for vendors to standardise on a release policy, and instead the responsibility for formulating an effective patch management policy is passed onto the end-user. As will be demonstrated in this chapter, this is because the type of disclosure has an impact on the effectiveness of a vendor's patch release policy.

In an effort to ease the end-users' patching burden, some vendors have decided to move to a predictable patch release schedule. The first vendor to announce such a move was Microsoft. Soon afterwards Oracle and Adobe announced they would also move to a predictable cycle. John Pescatore of Gartner believes predictable patch release schedules are on their way to becoming an industry standard [116]. However, simplifying a patch release cycle ignores the complexities that the full disclosure debate has introduced, and risks oversimplifying the matter, as will be demonstrated below. In both Microsoft and Oracle's case, the reactions to the announcements were varied. Some security experts were for the move [117, 118], others against it [119], and the majority were silent. The lack of consensus indicated a shortage of research and understanding as to the possible effects. Since then both Microsoft and Oracle have come under heavy criticism, and received praise, for their patch schedule implementations by security professionals commenting on the same events. Propagating this policy to other vendors without a thorough analysis and with little understanding of the effects would not be desirable.

Surface observations of the implemented schedule have revealed both successes and failures. This chapter provides a detailed argumentative analysis of patch release schedules and their effectiveness. By examining examples of how various types of disclosure affect the risks faced by end-users, recommendations on how patch schedules should be implemented and when they are effective, or not, are formulated. In addition, lessons learned from recent public security incidents are used to suggest additional improvements to the process. The resulting observations are used to describe a method for other vendors to implement such a cycle that will both minimise risk and help ease the burden of patching on administrators.

4.2 State of the Art

In the past, vendors operated without an obvious patch release schedule. When a vendor was notified of a vulnerability, either through delayed disclosure or otherwise, the general approach was to create a patch and distribute it as soon as possible[1]. The problem with the "release when ready" approach is that it requires end-users to continually monitor patch and vulnerability announcements. The average systems administrator has to check for new security patches, usually daily or weekly depending on the available resources. This creates a situation in which, combined with worsening number of vulnerabilities described in section 2.3 and additional problems

[1]Some vendors had a more nuanced approach, however this is not currently relevant and is discussed later

created by patches described in section 2.4, many administrators, either due to a lack of resources or will, simply are not installing patches effectively. Eschelbeck [31, 40, 120] is the only researcher at the time of writing to have provided empirical data demonstrating the impact of patch release schedules. Eschelbeck's data [120] shows that in 2004 it took 21 days to patch half the vulnerable machines on the internet after a patch was release (i.e. at 21 days 50% of vulnerable machines are patched), and 62 days for internal systems. Internal systems are increasingly vulnerable (as shown in section 2.3.2), due to the increased multiplexing of protocols over fewer ports, and content control decisions moving from the organisational network boundary to the end-user. Thus, it has become necessary to protect internal systems as one would external systems, and the window from patch release to patch deployment (62 days) allows ample time for intrusions. Several notable examples of this have been large scale worm attacks such as the Code Red, Nimda, Sadmind, SQL Slammer, Blaster, Sasser, Witty and Zotob worms, which all showed significant numbers of internal 'desktop' machine infections. To combat this two high profile vendors, first Microsoft [121] and then Oracle [122], and more recently Adobe [116] chose to move to a monthly patch release schedule. The caveat was that critical patches could be released out of schedule, similar to the internal policy of some organisations where critical patches are given an expedited install plan (see section 3.2.3.3). Microsoft chose to release patches on the second Tuesday of each month (a monthly release), while initially Oracle chose to follow suit, then changed to a quarterly release cycle [123]. However, Oracle have come under heavy criticism, with some released patches containing flaws up to three years after the vulnerability was announced [124]. Adobe, while planning to implement a monthly schedule, had not done so at the time of writing. Oracle's response to published vulnerabilities and quality of released patches has been poor. Most recently, Gartner came out severely criticising Oracle's patch practices [125]. Thus, given the lack of alternatives, Microsoft provides the best implementation of a patch release schedule, and will be the focus of the examples used. However, this discussion is intended to be relevant to any vendor implementing a patch release schedule. In particular, this discussion applies to both open source and proprietary vendors.

The next iteration of Eschelbeck's research [40] showed that the scheduling appears to have improved things somewhat. In 2005 it took 19 days (down from 21) to patch half of the vulnerable machines on the internet, and 48 days (down from 62) to do the same for internal machines. This improvement in patching speed is provided in table 4.1. However, the improvement in patching is likely due to many other factors such as the renewed hype around patching, better patch and vulnerability notification, and better automated patching tools, and cannot all be credited to patch schedules, especially since many vendors do not implement schedules as yet. The specific impact of scheduled patches was measured by Eschelbeck as being installed 18% faster. Additional statistics from Microsoft [126] indicate that the number of people applying Microsoft patches has improved dramatically (sometimes as high as 400%) since the change to a regular patch schedule. At first glance, the release schedule appears to be vindicated and proved successful - however, this research hypothesises that there are other intrinsic flaws in a patch release cycle that cannot be discounted.

	2003	**2004**	**2005**
External System's Half-Life	30 days	21 days	19 days
Internal System's Half-Life	N/A	62 days	48 days

Table 4.1: Half-Life of Vulnerabilities [31, 40, 120]

4.3 An analysis of patch schedules

This section provides an argumentative analysis of patch schedules. An analysis of the specific effects schedules have when vulnerabilities are disclosed differently is provided. Some background is necessary for the discussion, namely the arguments provided by instigators of patch schedules and background on the types of disclosure.

Specifically, a patch schedule provides a predictable routine describing how often and when patches are to be released, with a constant time between patch releases. This is supposed to provide two primary benefits:

- Higher Quality Patches
- Better Patch Deployment Planning by End-Users

These improvements are advanced by vendors in the various press releases and discussions on implementing schedules [116, 121, 122]. There are other indirect benefits sometimes cited, such as faster deployment and greater patch deployment. However, these are knock-on effects of the improvement in quality and planning listed above, and are not solely influenced by quality and end-user planning alone. For example, more detailed advisories, advertised to a wider audience, could also result in faster deployment due to more readily available information for decision-making, and greater deployment due to a wider demographic being aware of the patches. Thus, the focus will be limited to the direct benefits claimed by vendors. The analysis below discusses what trade-offs occur in gaining these benefits, and whether such trade-offs are acceptable. Most importantly, these benefits will provide ample justification for a patch release schedule if and only if they;

1. Are actually achieved
2. They are not achieved at the cost of a large increase in risk
3. They cannot be achieved through better means.

4.3.1 The Disclosure Debate

Before a discussion of the differences created in a schedule by different types of disclosure can be had, it is necessary to provide some background on the types of disclosure and the disclosure debate.

There are two primary types of disclosure: delayed, and instantaneous. Delayed disclosure is often referred to as 'responsible disclosure'. Unfortunately, this is an emotionally-laden term which is not always accurate, and will be avoided in this discussion. There has been much debate in the internet community about the socially optimal method of disclosure. The full disclosure movement of the late 90's argued that by providing as much detail about a security vulnerability, the information was brought into the open and provided administrators with information with which to make their own security decisions. The introduction of the BugTraq[2] and Full Disclosure[3] mailing lists was an important part of this. Where previously vulnerabilities had been discussed in private between security professionals, the information was now freely available [127]. Arora *et al.* [128] state that proponents of full disclosure argue that it "increases public awareness, makes as much information public as needed for users to protect themselves against attacks, puts pressure on the vendors to issue high quality patches quickly, and improves the quality of software over time." The problem with full disclosure is that without an effective defence for the vulnerability, usually in the form of a patch, the information is of more use to malicious entities than to users [129]. Thus, the concept of delayed, or responsible, disclosure was introduced. Here the information is first released privately to a vendor, and then disclosed publicly when

[2]`http://www.securityfocus.com/archive/1`
[3]`http://lists.grok.org.uk/mailman/listinfo/full-disclosure`

the vendor releases a patch [127]. However, many vendors adopted an attitude of 'shooting the messenger', wherein researchers who disclosed the vulnerability were publicly slammed [130] for reporting on vulnerabilities that would exist in the product whether they were reported or not. Most recently, Michael Lynn had his presentation at the Black Hat 2005 conference literally torn from conference proceedings and threats of legal action from Cisco systems for elaborating on previously disclosed memory corruption vulnerabilities [131]. At the same time, vendors would sometimes excessively delay the release of a patch [128], leading to much antagonism between vendors and security researchers. As a result, third-party trusted disclosure intermediaries such as CERT/CC were used to intervene in vulnerability disclosures, providing reasonable deadlines for vendors and ensuring security researchers disclosed 'responsibly' [129]. This also resulted in several recommended disclosure policies, with the more noteworthy being Rain Forest Puppy's *RFPolicy 2* [132], the Organisation for Internet Safety's policy [133], Russ Cooper's NTBugTraq policy [134] and CERT/CC's policy [135]. Several papers have been written discussing the pros and cons of non-disclosure, full disclosure, partial disclosure and 'socially planned' disclosure *[67, 90, 136, 128, 129, 127, 137].* A discussion on the various types of disclosure is beyond the scope of this section, but a simple summary is that the debate has fallen to the side of delayed disclosure. It is sufficient to understand that there are two types of vulnerability disclosure, one in which the public becomes aware of the vulnerability when a patch is released and the other in which the public and the vendor become aware when the vulnerability is released.

4.3.1.1 Delayed Disclosure

Figure 4.1 provides a visual depiction of a simplified vulnerability life-cycle based on the model presented in section 2.2, in which the disclosure is delayed.[4]. The vulnerability is created when the software is first developed. At some point the vulnerability is discovered, this can happen multiple times and by different parties. The vulnerability is then privately reported to the relevant vendor and a patch is developed. At this time the only exploitation of the vulnerability occurs by the original discoverer and is of a limited scope. When the patch is ready, the vulnerability is publicly disclosed and corrected at the same time. At this point, the number of vulnerable machines starts to decrease as patches are installed. At the same time the disclosure of the vulnerability details and the ease in which patches can be reverse engineered results in a rise in public exploitation of vulnerable machines. As the vulnerability and patch are publicised,

[4]The vulnerability life-cycle used here is simplified to highlight the differences between the types of disclosure, without muddying the waters with additional details.

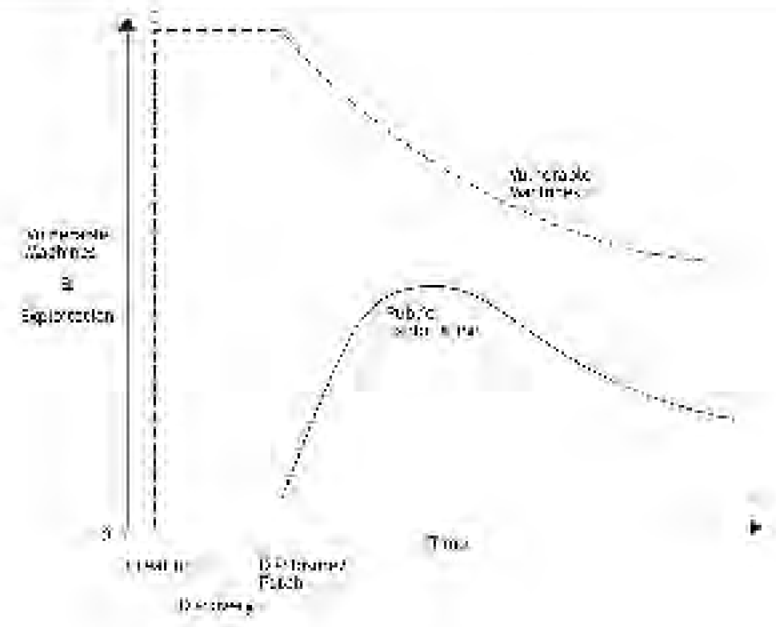


Figure 4.1: Delayed Disclosure and its effects on vulnerable machines and exploitation
Source: Modified from Rescorla [90]

the number of vulnerable machines continues to decrease while the number of intrusions of still vulnerable machines continues to increase. A scripted exploit could be released soon after the release of the patch, or longer. This will result cause a rise in the rate of exploitation, but is not relevant for the purposes of discussing the type of disclosure. It is sufficient to know that active exploitation is occurring, and is not included in the figure.

4.3.1.2 Instantaneous Disclosure

The process of instantaneous disclosure is similar to delayed disclosure, but with some pertinent differences. Figure 4.2 details the relevant events. Once again, the vulnerability is created and at some point discovered. However, instead of reporting the vulnerability to the vendor, the exploit is circulated within a community of black hats and private exploitation occurs. Sometime after this, the private exploitation is discovered 'in the wild' by a member of the public community and is reported to the vendor. At this point the process described in delayed disclosure occurs but with the difference that public and private exploitation occurs until a fix is released. The rate of exploitation will increase as the vulnerability is publicised and the exploit is possibly scripted.

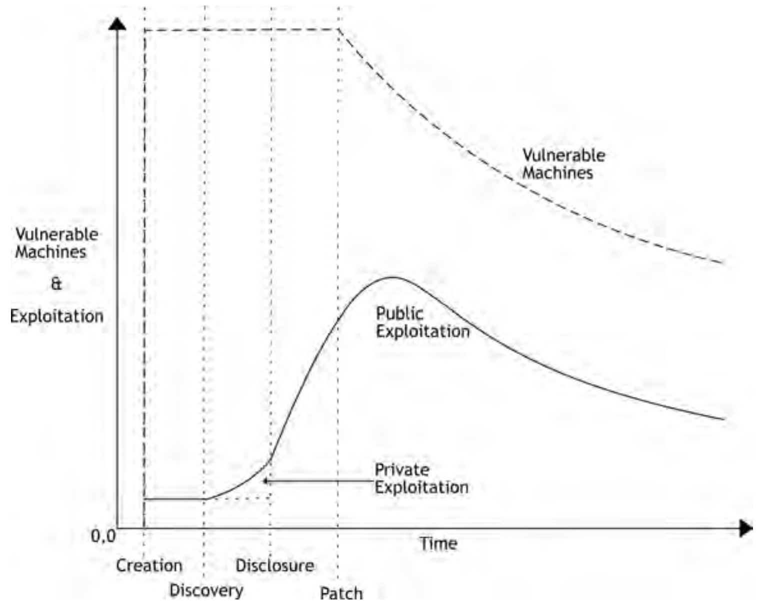


Figure 4.2: Instantaneous Disclosure and its effects on vulnerable machines and exploitation
Source: Modified from Rescorla [90]

Once again, the increase in exploitation caused by the scripting of the exploit is not displayed. The number of vulnerable machines will only start to decrease once a patch has been released.

4.3.2 Patch Schedules and Delayed Disclosure

The benefits of withholding the information until a patch is released are most obvious when the vendor has a choice as to when a vulnerability is publicly disclosed: The problem is acknowledged, but a fix is available. It is important to remind the reader that open source projects also withhold vulnerability information from the general public until a patch can be developed. For example, the Mozilla foundation frequently fixes 'Security-Sensitive' bugs which had not previously been disclosed [138]. A slight modification to ensure that these patches are released per a defined schedule brings more benefits. Administrators can avoid surprises and make plans ahead of time. Resources can be allocated, time scheduled, and deployment planned. In addition, the vendor can thoroughly test a patch to reduce the likelihood of a faulty patch being released without the pressures of attacks in the wild that need to be mitigated. With both the details of a vulnerability available and a patch which can be reverse engineered, a scripted exploit, whether released publicly or not, can be rapidly created [66]. This forces the vulnerability life-cycle to be synchronised with the patch release schedule. The only potential problem is that knowledge of the vulnerability may already exist within private and malicious groups or people[5]. This brings us to the original justification for full disclosure; by publicly announcing a vulnerability and encouraging people to patch, the number of attack vectors available to such groups is reduced. Were there no existing threat, the vendor could silently fix the vulnerability in the next upgrade. The only defence from attacks against unknown vulnerabilities is a comprehensive defence-in-depth strategy which will hopefully mitigate, or at least detect, such an attack. Organisations currently face these threats, and releasing the patch per a schedule which results in the patch being delayed longer than if the vendor released it when ready, will not significantly increase the threat to an organisation from malicious attackers. This assumes a limited exploit distribution within these 'underground' groups, a safe assumption in this case. Thus, the reduced threat from faulty patches and the increased efficiency of an organisation's patch management policy appear to more than justify this marginal increase in risk.

An important assumption is that the vendor develops the patch within a reasonable time frame. While the threats from an undisclosed vulnerability are limited, they are usually not zero. There is a potential for a separate discovery of the same vulnerability to occur by a malicious agent, or for the vulnerability to be 'leaked' by either the original researcher or agents within the vendor. The possibility of these events occurring increases over time and provides an incentive for a patch to be developed quickly. Thus, patch schedules with too long a wait between releases are likely to provide more than a marginal increase in risk and should be avoided. This is partly why Oracle is invalidated as providing a good implementation of a patch release cycle, as their quarterly release is too long. Unfortunately, there is little research into the probability of a leak occurring or a black hat discovering the same vulnerability, and this claim is based on an informed guess.

4.3.3 Patch Schedules and Instantaneous Disclosure

When vulnerabilities are disclosed irresponsibly the vendor no longer has control over when details of the vulnerability and a related exploit are released to the public. In the case of zero-day exploits, a working exploit is made publicly available without providing the vendor with advanced warning. Similarly, if no proof of concept exploit was released with the vulnerability, the

[5]It is possible that the number of publicly disclosed vulnerabilities and the poor patching record of many organisations provides malicious groups with enough attack vectors without needing to research their own.

existence of a vulnerability for which there is no patch provides an attractive target and it can be assumed an exploit is not far off. Current research indicates that the release of a scripted exploit triggers the largest increase in attack activity [37]. Given the large increase in the threat level, minimising vulnerable organisations' exposure is a priority for minimising risk. The critical factor thus becomes how soon the vulnerability can be effectively remediated. If a patch schedule will allow the patch to be released as soon as possible, then it is vindicated. If, however, the patch is delayed until the next release date instead of being released as soon as possible, this action is only justified if significant other benefits occur that cannot be achieved by any other means. The two benefits most commonly cited, as mentioned in the previous section, are that the delay due to the patch schedule allows more testing and allows administrators to plan for patch deployment. Both of these will be examined.

4.3.3.1 Quality

The argument for improved patch quality through more patch testing can be a persuasive one. The effort required by an organisation to minimise the risk of a patch causing problems are substantial and, as shown in section 3.2.3.4, represent the single largest bottleneck in patch deployment. Improving the quality of patches to a point where they could be deployed with little testing would substantially speed up patching and reduce risk. The argument is that by delaying the release of a patch, the vendor can engage in a thorough testing process. For example when a vulnerability in WMF files was discovered in the wild (a type of instantaneous disclosure exploit) [70], Microsoft's Security Response Centre had this to say about the patch [139]:

> We have finished development of a security update to fix the vulnerability and are testing it to ensure quality and application compatibility. Our goal is to release the update [...] as part of the regular, monthly security update release cycle, although quality is the gating factor.

However, the question must be asked: why must this testing be conducted in isolation? Surely collaboration with the wider community of end-users utilising the vendor's products would result in an increase in testing and wider test bed? If the reader will bear with us, the benefits of community collaboration are well demonstrated by the activities of Lawrence Lessig, a Stanford professor of Law, who has been pioneering the *Creative Commons*[6] movement. This movement

[6] `http://creativecommons.org/`

seeks to encourage collaboration and remove the systems of control that seek to monopolise creativity. Lessig and his proponents advocate a *remix culture* in which the works of others can be freely used and built upon. One example of the benefits of such a culture were demonstrated when Lessig released his book *Free Culture* for free over the internet, something that until now would have been ludicrous to suggest to a publisher.

> Last year Penguin Press made an unprecedented move to release Lessig's 'Free Culture' under a [...] license that enabled people to freely download the book from the internet, and make derivatives for non-commercial purposes. After 24 hours, the book had been made available under 9 separate formats (txt, pdf etc.), after 36 hours, an audio version of the book had been announced, after 48 hours, a wiki had been launched [...] for others to build on and add to, and after one week, 200 000 copies of the book had been downloaded. Today, non-commercial translation projects have started in Chinese, Catalan, Danish, French, German, Italian, Polish, Portuguese (2) and Spanish (2). There are 3 audio versions of the book as well as versions for the Palm, MobiPocket and Newton. [140]

Since then several other derivatives have been created, including more ebook versions and several easy to use hyperlinked versions. However, such creativity and collaboration is not unique to publishing and and provides a highly appropriate analogy to a recent event in the world of patching. The WMF vulnerability, for which no patch was available, was discovered on the 27th of December 2005 [141, 142]. One day later, initial anti-virus [143] and snort intrusion detection signatures [144] were available for the first variant. Two days later, a partial workaround for the vulnerability was posted [145], a movie of an exploit occurring was provided [145], and malicious sites exploiting the vulnerability were being shut down [146, 147]. Five days later, a third-party patch was provided by Ilfak Guilfanov [148], and later that day the patch had been disassembled and verified by the Internet Storm Centre (ISC), which offered a digitally signed version [149], A block-list of malicious sites and net-blocks utilising the exploit was created [150], and CERT provided a detailed vulnerability note on the issue [141]. Six days later, a version of the unofficial patch was made available that allowed for an unattended install [151], and was distributed along with scripts for deploying the patch enterprise-wide [152]. On the same day, 'safe' versions of the exploit were provided for vulnerability testing [153] along with an executable vulnerability checker for vulnerability testing and patch verification [154]. The next day a comprehensive FAQ on the vulnerability was made available by the ISC [155], and within a few hours this had been translated into 12 different languages, which had increased to 17 by the day after that [156], with presentations available in several different formats [155]. Eight days later, the unofficial patch was made available as a Microsoft Installer Package (MSI) by Evan Anderson [157], for easier deployment, and this too was verified and signed by the ISC. Later that day the site hosting Guilfanov's patch experienced difficulty due to high load, but returned a few hours later with 9 additional mirrors serving the files [158]. During this time, Microsoft maintained that an official patch would only be released on the 10th of January 2006, during the normal patch scheduled release [159]. Following massive consumer pressure Microsoft eventually capitulated, releasing the patch on the 5th [160].

Why then did Microsoft not cooperate with this community in developing a patch? If knowledge of the vulnerability already existed, then the benefits of keeping the patch confidential are lost, particularly when beta patches could be improved on and tested by such a wide and active community. Ironically, Microsoft possibly acknowledges this argument with their Security Update Validation Program (SUVP), which allows for patches to be beta tested within a chosen group of organisations, such as the US Air Force [161]. Microsoft benefits by the additional testing provided by an organisation with enough resources and interest to thoroughly test patches, and in return the Air Force benefits from the early protection afforded by getting a jump start on their patch deployment process[7]. Although members of the SUVP are not allowed to use these beta patches in a production environment, they can benefit from early testing and ensuring their configuration is supported. There is no reason to assume these benefits would not scale upward if such a beta program was extended to include the public. A possible counter-argument to this is that a vendor can implement a better planned testing process, whereas testing within a community will involve a lot of redundancy and cannot be guaranteed to perform all necessary tests. However, this is simply a false dichotomy, as all the benefits of a well planned vendor test schedule can be accrued in addition to testing input from a community. Tools and mechanisms allowing members of the community to interact and share their testing experiences already exist in the form of public mailing lists such as BugTraq and PatchManagement[8]. The only modification required to take advantage of this testing community is to release the patches early and clearly mark them as unsupported beta's. By providing obvious warnings of the dangers inherent to deploying a beta patch (on the patch download site for example, and in the actual patch's installer) or taking further steps such as providing a registration system, users who do not know

[7]Given the ease with which exploits can be reverse engineered from patches, it is worrying to contemplate the American military being given such offensive capabilities before the rest of the world.

[8]`http://patchmanagement.org/`

better can be prevented from installing these beta patches.

The level of community involvement in response to the WMF vulnerability, particularly related to the unofficial patch, is unusual. While IDS and AV signatures, and cooperation to shut down malicious sites, are (thankfully) fairly standard, the community does not always get as involved as it did for the WMF vulnerability. The situation may have resulted from the combination of significant threat level and confirmed inaction from Microsoft. However, while arguments claiming that one cannot always expect this level of community involvement are correct, this does not invalidate the point. If the community were to provide no additional help or guidance (an unlikely case), the vendor would still not lose anything by releasing beta patches. The community would, at worst, fail to benefit from the early release, but would not lose anything either. In addition, if the vendor were to release details of which configurations the patch had been successfully tested on, the few which fulfilled those criteria could benefit from early patching without having to wait for all testing to be completed, ensuring that even if the community were of no help with testing, some organisations' exposure could be minimised sooner.

4.3.3.2 Planned Deployment

> "Having a predictable schedule makes it easier for customers to plan and when you can plan, it puts less stress on the customers' infrastructure and their people and the results are better."
>
> – Mike Nash, Corporate Vice President responsible for Security, Microsoft [162]

Providing a predictable patch release schedule can endear end-users to their vendor. The capability to plan and allocate resources ahead of time results in a much smoother deployment, with a smaller likelyhood of errors. It moves patching from an emergency-mode procedure to an understood business process. Unfortunately, these benefits are, once again, only available if the vulnerability disclosure was delayed. Threat and vulnerability monitoring are a separate process from patch deployment. A patch schedule helps to synchronise the release of the patch, vulnerability, and exploits so that threat, vulnerability, and patch monitoring can likewise be synchronised. However, if the vulnerability was instantaneously disclosed, the vendor is not able to maintain this synchronisation. Thus, an end-users need to be constantly monitoring their networks for attacks, and understand and respond to potential threats. If a significant threat and vulnerability are discovered, a risk assessment must be conducted and steps taken to mitigate the risk. This must be conducted whether the patch exists or not. Thus, the exact emergency mode scheduling patching seeks to avoid persists. The best way to “put less stress on the customers’ infrastructure and people” is to provide an effective remediation as soon as possible. Placating end-users and playing down the threat to maintain the patch schedule instead of releasing a beta patch and encouraging community support to develop quality remedies is counter-intuitive. Even if the benefits of planning did apply in this situation, the corresponding increase in exposure is an unacceptable trade-off. This increase in exposure makes it more likely that an intrusion may occur. Intrusions are usually unscheduled and costly to recover from, which would provide a greater inconvenience than deploying an unscheduled patch. The emphasis within the patch management community and this document is for an organisation to perform their own risk assessment and choose a course of action relevant to their needs. However, without the option of an effective remediation, a vendor would be severely limiting the organisations’ options for dealing with this risk.

4.3.3.3 Examples

The critical flaw in a patch release schedule is that it assumes all patches are responsibly disclosed. While the WMF vulnerability has provided the primary example used in the discussion above, there are other examples of instantaneously disclosed patches that have remained unpatched for a significant amount of time and resulted in a needless increase in an organisation’s exposure to threats. Once again, the focus on Microsoft is unavoidable, given the lack of any other vendor having effectively implemented a patch release cycle. The WMF vulnerability is unique in its level of community support and discussion, particularly from Microsoft, who have been reluctant to discuss their motives in the past. Thus, the examples below are of vulnerabilities which could have been patched sooner, but were not, for the sake of the patch schedule. However, they do not demonstrate the same level of community involvement as the WMF example above, and contained no serious flaws, indicating that the testing within Microsoft is effective. Unfortunately, they do illustrate both the unacceptable increase in exposure and an inordinately large amount of time from vulnerability disclosure to patch release. It should be noted that these examples are illustrative of the failings of a patch schedule for instantaneously disclosed vulnerabilities only; Microsoft’s patch schedule has proved quite effective for delayed disclosure vulnerabilities.

Krebs [163] researched the time it took Microsoft to release a patch from either the time of disclosure or the time it was reported to the vendor for 2003, 2004 and 2005. The dates and times were gathered by contacting the original researcher who discovered the vulnerability, and Microsoft. Unfortunately, according to our investigations, Krebs' calculations appear to be wrong [164] with inconsistencies and errors in the number of days from first disclosure until patch release and the number of patches counted. However, the dates he gathered appear correct, and once the calculations were fixed, because some days were too high and others too low, his conclusions based on the averages remain true. The results appear in table 4.2, and show that when Microsoft moved to a scheduled deployment in 2004, the average time it took for a patch to be released increased for all vulnerabilities. They also show that for instantaneously disclosed vulnerabilities, Microsoft has been getting faster at patching. Both these results make sense. The average time to produce a patch has increased due to the additional testing and quality assurance that occurs, and the average time to produce a patch for instantaneously disclosed vulnerabilities has decreased due to an increased security effort and an increase in threats. However, even at the lowest average of 46 days, this is still far too long. This provides plenty of time for scripted exploits to be circulated and used by anyone, including unskilled attackers. To reiterate, even if the patch quality is increased, the high exposure time buys this quality at too high a cost. By involving the community in the testing effort high quality patches can be produced sooner than in this situation.

	2003	2004	2005
Number of Critical Patches	34	28	37
Average Days from Report to Patch	90.7	136	134
Average Days from Full Disclosure to Patch	73.6	55	46

Table 4.2: (Corrected) Microsoft Time to Patch Summary

To support the claim that 46 days is too long a wait for users, two examples of the type of damage that can occur during these long exposure times can be found in MS04-040 and MS05-054.

MS04-040 This Internet Explorer patch took 38 from the date of public disclosure days to produce. This vulnerability was not disclosed to the vendor before hand. The average time taken to release such a patch in 2004 was 55 days (38 days is therefore well below the average). However, during this time a variant of the MyDoom virus used the exploit as a propagation mechanism, resulting in mass compromises. In addition, a banner-ad service was compromised, and the exploit placed into the advertisements. These were then distributed across many high profile sites such as The Register and BBC leading to a substantial number of compromised machines [165]. As a final blow, the Bofra/MyDoom mass mailing worm was developed, and used the MS04-040 vulnerability to infect a machines [166]. These three large scale incidents all occurred within the 38 day window.

MS05-054 The original vulnerability related to this patch was publicly disclosed on May 28th 2005. However, the vulnerability was described as a DoS attack and did not carry a high criticality. Microsoft still had not provided a patch after five months, at which point it was publicly disclosed, on November 21th, that the vulnerability could allow remote code execution, raising its criticality. Proof of concept code was provided and soon afterwards the attack was detected in the wild [167]. A patch to repair the vulnerability was only released on December 13th as part of the normal patch release. This means that, despite having 177 days to develop a patch, the vendor still took 22 days to produce the patch once it had been re-evaluated as critical.

4.3.4 Conclusion

The conclusion is quite simply that the arguments for a patch release schedule assume that all vulnerability disclosure is delayed. The benefits claimed for a patch schedule are that a higher quality patch can be released and that end-users can better plan and schedule their deployments. However, when a vulnerability is disclosed instantaneously, these benefits are either lost, moot, or could be better achieved by other means. Patch quality could be achieved faster by utilising a community testing approach, and scheduled patch deployments are not useful if they are likely to result in unscheduled post-incident recovery.

4.4 Advice for implementing a Patch Release Schedule

The prescribed policy is to have two release programs, one scheduled and predictable for delayed disclosure vulnerabilities, and one immediate and collaborative for instantaneously disclosed vulnerabilities. This simple solution is similar to what is already supposedly implemented by vendors with their possibility of ’out of band’ patches. However, there are problems with the criteria used to differentiate between when a patch should be released per schedule or not. In addition, specific guidance is required as to how vendors can best help end-users and involve the community to improve patch quality at a faster rate. The policy discussed below provides a simple and effective method for releasing high quality patches and helping end-users to minimise their risk. It first provides a clear criterion for discerning which patch release mechanisms should be used. It then details how each mechanism can be implemented, with reference to several currently effective vendor practices.

4.4.1 Dual Schedules and Separation Criteria

As mentioned above, a vendor should utilise two release mechanisms. The first is a predictable and regular schedule, the second an unpredictable 'when ready' release. One of the current criteria for distinguishing when to use which mechanisms appears to be risk. If a sufficiently large risk exists, in the form of a significant threat, then a patch will be released out of band. Threat is the deciding factor in the incomplete risk assessment conducted, as vulnerability appears to make little difference. When a worm is released or significant exploitation is detected, there is more pressure to release a patch out of band, often in the form of customer complaints and bad press reports. However, if an instantaneously disclosed vulnerability indicates that a significant proportion of end-users will be vulnerable, then the pressure to patch only appears to come after a large threat is detected. For example, Microsoft's justification for releasing the WMF patch as per scheduled indicated that their 'intelligence sources' did not perceive a large threat [139], and the patch released out of band only once significant customer pressure had been brought to bear. Thus, the current criteria can be extended to be one of either threat or external pressure. There are problems with these criteria. The problem with responding to threats is that a widespread and recognised threat does not negate the possibility of targeted and specific attacks. Vendors should be seeking to minimise all vulnerability, not to minimise significant threats only. The problem with responding to external pressure is a similar one - once people are detecting attacks, it is often too late. Vendors should be seeking to prevent an attack in the first place. In addition, the size of the threat and external pressure are not easy to measure or objective criteria. A vendor's view of threats abstracted across all end-users is naturally a generalised one, so that, while certain organisations may be facing significant threats and others none, the view to the vendor is only a medium threat. As for external pressure, the amount of 'noise' one group makes is only weakly linked to the actual problem. Thus a specific, objective, and measurable criterion is needed to differentiate between release mechanisms, and determine which should be used. This document proposes that the form of disclosure be that criterion, and proposes the following maxim:

> If a vulnerability is disclosed responsibly then release the patch at the earliest possible scheduled release date. Alternatively, if a vulnerability has not been disclosed responsibly then release at the earliest possible date, ignoring the schedule.

This is the most relevant criterion if the arguments given above, which conclude that the benefits of patch scheduling only apply in cases of delayed disclosure, are taken into account. In addition, this criterion is trivially easy to determine and can be objectively judged by both the vendor and end-users. Vendors should therefore adopt this as the discerning factor between a scheduled release and a critical release, and clearly communicate as much to their end-users to prevent misunderstandings.

4.4.2 Predictable Patch Release Schedule

Taking cognisance of the above, the vendor should develop a regular release schedule for those patches covering vulnerabilities which had their public disclosure delayed. To reiterate, a delayed disclosure vulnerability is one which has been privately disclosed to the vendor. Most often researchers, who disclose vulnerabilities privately, will synchronise the release of their advisory with the time at which the vendor releases the patch. For example, eEye security maintains a list of vulnerabilities [168] they have reported to vendors, for which a patch has not been released and they have been waiting to disclose their advisory. However, on occasion a researcher will specify a fixed date at which they will disclose their research. If negotiations fail and the fixed date is out of the schedule then the customers should be informed of the out of band release. This is a rare occurrence however, and is an example of why vendors should attempt to maintain good relationships with the security research community.

An important part of creating such a schedule is deciding on the length between patch releases. The difficulty in setting this length is twofold. The first is in choosing a length that reduces the time available for the vulnerability to be either discovered independently or leaked. The possibility of a vulnerability being discovered independently is only a concern for schedules that extend over several months. It is unlikely that such an extended schedule is necessary, as the majority of patches should not take long to develop and test, particularly since the critical release will require rapid patch development and testing. In addition, there is the possibility of delaying the release of a patch for a number of schedule iterations. For the same reasons that the schedule shouldn't leave too mcuh time between iterations - there should be a maximum cap on the number of releases for which a patch can be delayed without very good reason. The second difficulty is in ensuring that the release cycle is optimised for all end-users. The deciding factor in this optimisation will be how often end-users can realistically afford to engage in patch management activities. Customer feedback and surveys should be conducted to gauge the optimal length. Bear in mind that customers will have a bias towards patching less often as it translates to a smaller workload. This bias should be offset by the desire to minimise the potential of a leak or separate discovery, and to keep the number of patches deployed per release to a reasonable minimum, as offloading too many patches at once makes end-user risk assessments too complex, can impair the efficiency of monitoring efforts, and exposes an organisation to too many threats at once. The current trend is towards a monthly patch cycle. A charitable assumption is that Microsoft, Oracle and Adobe engaged in comprehensive end-user discussion and the resulting choice of a month is optimised for the above values. However, the needs of customers, the frequency at which vulnerabilities are discovered and the speed at which patches can be developed are all dependant on the vendor, and as such this value cannot be generalised across all vendors.

One potential concern raised by an 'industry standard one month patch release' is that administrators may be flooded with several patches from separate vendors on the same day creating the same problems a vendor was trying to avoid. Alternatively, if the patches are released on different schedules at different times of the month, the problem of constantly applying patches (which schedules try to minimise) is re-created. This is a difficult problem that will affect end-users with multiple vendors for which vulnerabilities are regularly released. While automated patch deployment solutions will help with the deployment and installation of these patches, they provide little support for the larger and more time consuming problem of testing them. Ideally, end-users will standardise on manageable baselines. It will be in the vendor's interest to forge connections between vendors whose software is commonly used in conjunction with other vendors' to ensure that the number of patches released at one time are kept to a minimum, and to interact correctly. In addition, planning for patches to be released within short gaps of each other would allow end-users to plan deployment and manage threats more effectively than if all patches were released on the same day. While this 'multiple vendor' problem is quite limited at the moment, as more vulnerability research occurs and consequently the number of patches released grows, this problem may worsen. Once such example of the multiple vendors problem was on July 12th 2005 when patches from Microsoft, Oracle, Mozilla and Apple were all released on the same day [169]. Granted, only two vendors engaged in a predictable release, but even if end-users had been aware off all the patches released, some end-users requiring all the patches would be forced into an awkward triage.

As privately disclosed vulnerabilities must remain private until a patch is available, a discreet, secure and confidential group of developers should be tasked with managing security patches and vulnerabilities. This is particularly true of open source vendors, where the development is, by its nature, open. The majority of vendors already have such a group implemented, and it is only mentioned as a requirement here in passing. The members of this group should be held accountable for any leaks, and given the required access to ensure that they can develop patches quickly. Given that patch development cannot be a task assigned to a small and constant group and by its nature spans all development and developers, mechanisms for temporarily bringing in other groups of developers, testers etc. with the same levels of confidentiality and accountability need to be developed.

4.4.3 Critical Patch Release

The critical patch release mechanism will seek to release a patch as soon as possible after the disclosure of an instantaneously disclosed vulnerability, where the vulnerability was not privately disclosed to the vendor beforehand. In this situation the vendor would be informed of the vulnerability at the same time as the general public. This does not always occur through the release of a vulnerability advisory - a zero-day exploit could be provided, or a vulnerability advisory could be accompanied by proof-of-concept code. In all of these situations, a vulnerability has been instantaneously disclosed. Some vendors already claim to have implemented such a critical release strategy. However, as discussed above, this release mechanism is only invoked at a subjective point determined by the vendor. In this version, the disclosure type of the vulnerability is the only appropriate discerning criterion. If a vulnerability has been privately disclosed and, before the chosen patch release date the vulnerability is either leaked or discovered independently and publicly disclosed, the patch should be shifted from a scheduled release to a critical release.

Once it has been determined that a patch should be fast-tracked and released as part of the critical patch release mechanism, a vendor should seek to engage the community of end-users to help ready a patch. The arguments discussed in section 4.3.3.1 described the benefits a community can provide, and how keeping the details of a patch secret until release are counterproductive. The possible help a user community could provide is as limited as human imagination. Whether it is documentation, vulnerability scanners, workarounds, third-party patches or vital testing, with the right motivation the skills of technical administrators can be leveraged. The work required in developing and delivering high quality patches has a high level of commonality across patches.

This is not to say that the vulnerability and related fix are the same, but rather that all patches require, for example, testing and documentation. A vendor should enumerate the required tasks and highlight those where community support could provide a benefit. On-line collaboration tools should be provided to enable the community to engage in the required tasks. Most often these simply consist of an on-line forum; either a mailing list, forum software, wiki or bug tracking program such as bugzilla[9] can be employed. Peripheral benefits aside, the most specific and beneficial area of community involvement is in testing. By providing alpha or beta quality patches for early download, and sharing information on what has been successfully tested, a community can get involved. If multiple beta versions of a patch are to be released, enhancing or providing a patch roll-back mechanism would be one area where tools could be developed to aid testing.

A possible concern is that end-users would not be interested in deploying patches that are not at final release quality. However, end-users would not be applying beta patches directly to their systems. An effective patch management policy should always include a comprehensive testing strategy as discussed in section 3.2.3.4. In such a set-up no patch should be deployed without any testing, and the same would apply here. However, there are benefits to end-users in getting involved in testing. By testing the patch on their specific configuration an end-user can ensure that the patch finally released works correctly for them. In addition, if a patch appears to function correctly it could be deployed early to machines that warrant it. Since testing has a 'long tail' (where the initial work is in testing common configurations which apply to many users, whereas the later tests usually only apply to a few users but require as much work), once testing is completed on the common configurations, many users could deploy the patch sooner or at least get a head start on testing. For example, if a vulnerability primarily affects the Chinese version of a vendor's product, releasing the patch once the Chinese documentation is ready would allow the majority of users to start their deployment without having to wait for all translations of the documentation to be completed. The testing provided by the end-user community would allow the vendor to test different configurations faster, and the 'release-when-ready' approach would allow more end-users to deploy patches, and hence decrease their vulnerability, sooner. The only cost is a slight increase in the amount of testing performed by some end-users. However, the size of the community will usually help to ensure no one end-user's testing time increases dramatically, as the work is distributed and testing performed by one group can benefit many more with similar configurations. Thus, many end-users could continue as they do now and wait until the final release of the patch.

[9]`http://bugzilla.org/`

This release when ready approach can only help security by speeding the availability of vulnerability remedies. Only faulty patches being deployed on production machines would invalidate this. Thus, the vendor must emphasise that only the final production release of the patch should be deployed to production machines and all beta releases should be tested in a sand-boxed testing lab. Two advantages then become possible. The first is that testing feedback provided by the community will speed up the vendor's testing process, resulting in a patch being available sooner. The second is that the patch, if it passed some configuration's testing, could be deployed sooner to some end-users without having to wait for every configuration to be tested.

The vendor should work hard to ensure all feedback is consolidated into a quality patch as soon as possible. With proper encouragement, embracing the community prototyping approach will help to cut down on the window of exposure from disclosure until a patch is available.

4.4.4 Encouraging Delayed Disclosure

Given the benefits evident when a patch release schedule is used for vulnerabilities which have had their disclosure delayed, it is in vendors' interest to encourage delayed disclosure of vulnerabilities. Much discussion is available in each of the disclosure policies discussed earlier on how to maintain an amicable relationship between the vendor and security researcher. Vendors should make an effort to maintain positive relationships with the security community and vulnerability researchers in an effort to reduce the instances of instantaneous disclosure. Researchers too should consider how to best minimise risk to end-users when disclosing vulnerabilities, however that is outside the scope of this discussion. Two vendors contrast quite differently in their approach to this. Microsoft has done quite well in building its relationship with researchers over the last few years. There are few examples of recent public outcries by researchers who feel the vendor is not providing the patch within a reasonable time-frame. In addition, throwing parties for security researchers at conferences such as BlackHat [170] and outreach events such as BlueHat [171] have further helped to build a positive relationship. Oracle, on the other hand, has created controversy by taking too long to fix some bugs [172], and providing poor fixes even after these extended periods of time [173]. This has resulted in a negative perception of Oracle's patch release process and may decrease the chances of researchers working with the firm.

Another approach which has proved quite successful is the bug bounty program run by the Mozilla foundation [174], wherein $500 is awarded for each previously unknown security bug discovered in Mozilla software that is privately reported to the foundation. The foundation claims that the bounty program is working well. As of December 2005 they had awarded $2 500 in bounties since its inception earlier that year [174].

Additionally, relationships with security researchers can be smoothed by providing a clear and accessible description of how the vendor's organisation will respond when vulnerabilities are reported. Defining time frames in which contact will occur can help to manage the expectations of the researchers.

4.5 Conclusion

This chapter has provided a discussion around the benefits and disadvantages of implementing a patch schedule. This discussion has provided *a priori* arguments on how patch schedules influence risk and are influenced by disclosure. These arguments have shown that patch schedules provide two benefits to end-users; the first is a higher quality patch with less chance of a fault, and the second is a predictable schedule which allows end-users to plan their resources and patch deployment, reducing the surprise factor and helping to integrate patching as a normal business process. However, the argumentation also showed that these benefits do not accrue or come at too high a cost when the vulnerability has been instantaneously disclosed. The patch quality could be better achieved by releasing patches early as betas and gaining community support (although this cannot eliminate the ”surprise factor” in these instances, due to the unpredictable nature of threats). To remedy this situation it is proposed that vendors maintain their patch schedule only for delayed disclosure. The type of disclosure forms a clear and objective differentiator for which patches should be scheduled and which shouldn't. In the past the differentiating factor has been a subjective threat assessment. In the situation of instantaneously disclosed vulnerabilities vendors should implement a critical release strategy that releases a beta of a patch to a community as soon as possible, allowing more testing to occur and providing benefits to end-users and the vendor.

This chapter has provided a discussion on how vendors can better manage the risks patch release cycles expose end-users to. In the next chapter, practical and available tools, and solutions that can be used to ease the burden of the patch management policy discussed in chapter 3 will be discussed.

Chapter 5

Practical Solutions

5.1 Introduction

"I need automation to deploy patches, I do not want automated patch management."

– Tim Rice, Network Systems Analyst, Duke University School of Medicine [33]

This chapter provides some technical discussion as to how aspects of patch management can be improved with technology. The focus is initially on the packaging and distribution of patches, and secondly on additional measures that can be used to limit the vulnerability of systems until a patch is deployed.

5.2 Patch Management Software

Patches have existed for as long as software has existed, and they have always been tedious and difficult to manage (The timeless nature of patch management was introduced in section 1.2). In 1985 Larry Wall made distributing and merging patches to source code easier with the introduction of his *patch* utility [17]. In 1993, operating system vendors introduced tools to help automate the process of keeping software up to date [175, 176]. In 1997 software that allowed for the management and deployment of standardised patches across multiple operating systems was proposed [177]. Advancing to the present, we see that tools to manage application updates have become a mandatory part of an operating system, and automated patch deployment software has become a growth industry with a flood of new patch management products and tools. A report on the patch management industry showed that sales reached $80 million in 2003 and the number of direct competitors topped the 20 mark[178]. With this many 'solutions' available, it is tempting to believe the problem of managing patches has been solved. This section exists to critically classify and analyse the many types of patch management software, and demonstrate which parts of the patch management process described in section 3.2, have been and can be automated. This chapter discusses the premise which introduces the necessity of a patch management policy and demonstrates the failure of 'software only' approaches. The premise is that while patch management products fulfill a necessary role, they can only help by automating tasks necessary to patch management, but never automate tasks sufficient for patch management. The thorough understanding of the patch management process provided by section 3.2 demonstrates the difficulty of complete automation. In short this section concludes that patch management is too complex, with too many variables requiring experience and human decision-making skills for it to be completely automated. To quote Bruce Schneier [35]:

> “If you think technology can solve your problems then you don’t understand the technology and you don’t understand the problems.”

This is not to say that nothing can be automated. Patch management is made necessary by the nature of the technology, and is an apt example of the productivity paradox [179], wherein technology introduced to save time has resulted in a new set of time-consuming problems. Thus, patch management is dependant on technology and can benefit from automation.

5.2.1 Functionality and Classification of Patching Tools

There are, at the risk of understatement, a large number of patch management tools. A reference collecting reviews of some such products [180] lists 22 companies, some with multiple products, offering various forms of patch management solution. Each product implements varying levels of functionality. Most often, patch management products are differentiated simply by whether or not they utilise agents which must be installed on client machines. This is a simplistic differentiator of functionality, but is mentioned in several places while discussing patch management software[42, 91, 94, 181]. A discussion on the use of agents is available in section 5.2.2. The Gartner Group has described nine characteristics that an automated patch management solution should contain [182]:

- The ability to create and maintain an inventory of systems including information about installed software and running services. It should be able to discover new systems without the need to distribute an agent.
- Information on the software and patch revision level of each software component on each system.
- Automatic evaluation of patch dependencies and tracking of which patches are out-of-date or superseded.
- A dynamically refreshed patch inventory and ability to classify the patch according to severity.
- Reports on what patches are needed on which systems by correlating information from the various inventories. This should take the system's role into account.
- The solution should provide for system groupings, allowing many machines to be abstracted into one group. The system should also support role-based administration, allowing different parts of the work-flow to be executed by different roles (e.g. Quality Assurance).
- A scalable patch distribution and installation method, providing for patch roll-back if necessary.
- The system should be cross-platform, especially given number of devices including servers, network devices such as routers, handhelds, cell phones etc. that need to be supported.
- The solution should leverage existing software for patch management and only introduce a software agent where necessary.

While these criteria are fairly comprehensive, in an earlier version of this work [95] it was noted that they are not exhaustive, and several additional ideal characteristics were defined and on which this list builds.

- The system should be secure. Being able to automatically deploy malicious content to an entire organisation by compromising one distribution source is a tempting target for attackers [183]. While all applications should attempt to be secure, patch management tools are security critical applications and, hence have a higher likelihood of being attacked.
- Patches are being issued from multiple vendors, and a patch management solution should support this to prevent the need for multiple redundant patch management systems.
- The purposes of patches is to remediate vulnerabilities, but there is not always a one-to-one mapping between vulnerabilities and patches. The system should therefore maintain as complete as possible an inventory of vulnerabilities, with the ability to test whether specific vulnerabilities are applicable. This is particularly useful for confirming whether a patch is effective in mitigating a vulnerability.
- Provide detailed and powerful reporting mechanisms that allow information for risk management decisions to be gathered easily.
- Integrate with other security mechanisms to minimise vulnerability, particularly during the window of exposure between vulnerability disclosure and patch deployment.

The words *'functionality'* and *'capability'* are often used as synonyms. However, for the purpose of this discussion they shall be used to represent two distinct concepts. What we notice about the list resulting from the combination of the Gartner's list and our own, is that some points discuss *functionality* - for example "an inventory of available patches should be implemented" - while others discuss the *capabilities* of that functionality - for example, "the system should be secure". *'Functionality'* will be used to discuss core features that directly support and enable one of the policy elements described in section 3.2. *'Capability'* will describe a general feature intended to modify one or more of the functional components. These capabilities imply extra functionality, but the scope of the discussion precludes examining them. In the introduction to the policy framework described in chapter 3, it was noted that patch management integrates with several other management fields; asset management, vulnerability management, change management, configuration management, and risk management. Patch management tools demonstrate this integration. A summary of the policy discussed in chapter 3 is available in both figure 3.3 and table 3.2. If we distill from this policy the functionality which can be supported by an automated system, seven distinct areas of functionality are found. These seven primary functional areas are based on Gartner's ideal characteristics, automated functionality discussed in chapter 3, the functionality available in some existing tools, and the author's own observations. They are:

1. Notification
2. Inventory Management
3. Vulnerability Scanning
4. Patch Testing
5. Patch Packaging
6. Patch Distribution
7. Reporting

These functionality areas are groupings of similar functions. For example, inventory management would involve asset inventories, a patch database, an inventory of vulnerabilities etc. With the functionality isolated from the original list, we can do the same with capability. The resulting capabilities are:

1. Allow arbitrary grouping and classification of inventories
2. Support patches from multiple vendors
3. Provide a portable cross-platform implementation
4. Focus on security, with regular authentication and authorisation of patches
5. Integration with other security applications

The following sections provide a brief discussion on each functional area. This is intended as a technical discussion focusing on how automation can support a patch management policy. The concepts are only briefly introduced, as it is hoped a fuller investigation and implementation of such a patch management product will be a part of future research.

5.2.1.1 Notification

To ensure that accurate risk assessments can be made (as described in section 3.2.3.2), it is necessary to receive regular notification of three aspects of the risk equation[1]:

- Vulnerabilities
- Patches
- Threats

Vulnerability A method of discovering new vulnerabilities and notifying a the patch and vulnerability group is required. There are many vulnerability databases, each of which provide some sort of notification service. The rise of XML-based RSS and ATOM feeds for easy syndication means that these notifications could easily be integrated into an application, something vendors should be encouraged to provide. Thus an 'interrupt' approach can be used, with notification arriving when relevant, as opposed to requiring an administrator to engage in 'polling' by searching through busy mailing lists such as BugTraq. For example, the National Vulnerability Database provides two feeds, one of all CVE vulnerabilities and one with deeper analyses [184]. The Open Source Vulnerability Database (OSVDB) [47], Secunia [50], ISS X-Force [185] and Security-Focus [186] vulnerability databases all provide syndicated XML feeds. OSVDB goes one step further and provides an XML-RPC interface for dynamic real-time queries of the vulnerability database. There is a large amount of redundancy between databases, and it is preferable to select one vulnerability database as a source, with recourse to others for further research. There may be some lag in databases' adding information on instantaneously disclosed vulnerabilities, but the automated notification will save a considerable amount of time in comparison to manually trawling mailing lists.

Patch Similarly to vulnerability notification, many vendors provide XML feeds in order to improve notification of released patches. For example, Microsoft [187], Debian [188] and FreeBSD [189] all provide XML feeds of their latest patches. The flexibility of XML can easily allow feeds from relevant vendors to be aggregated and filtered for vulnerabilities affecting an organisation's

[1]These aspects are enough to determine risk based on affected system's criticality, but such an assessment should still be done by a human agent.

deployed software. For unscheduled patch releases, the automatic notification can allow an administrator to be notified immediately and react quickly.

Threat Threat notification is more complicated, due to the intrinsic complexity of the threats. Some threat notification can be automated, particularly with tools that support correlating and aggregating information from multiple network sensors. Tools such as Squil [190] or DeepSight Analyser [191] allow for the information from multiple network monitoring devices to be correlated at one monitoring console. Threat management services such as DSHIELD [53] or, once again, DeepSight Analyser can be used to detect wide-scale attacks. Specific attacks can be discovered through the use of Intrusion Prevention Systems and honey pots, where the first uses signatures to detect an attack, and the second can provide insight into an attacker's methods, or distract an attacker from real systems. A discussion of defence-in-depth tools is available in section 5.3. Whatever monitoring devices are used, the information must be correlated to provide effective notification. Too many false-positives will result in the sensor being ignored.

5.2.1.2 Inventory Management

This is a broad functionality group, and one of the most critical. Based on the discussions in section 3.2.3.1 we can see that the three primary inventories required are:

- Asset inventory
- Patch inventory
- Vulnerability inventory

Asset Inventory Section 1.3 highlighted the difficulty of managing many patches for many vulnerabilities in many software products and deploying them to many machines. Section 3.2.3.1 discussed the need for proper asset management. This is a function that can benefit greatly from automation. Without automation, the process of discovering and enumerating all the hosts on a network, all the software on each host, and the patches (both available and installed) can be time consuming and tedious. Most patch management tools contain some combination of these inventory management tools, allowing an administrator both to automatically populate the inventory, and to better organise and track the large amount of information this will create. Advanced inventory management systems function as reporting tools, allowing ad-hoc queries of the state of the inventory. These queries can provide valuable information when performing the kind of verification and reporting described in section 3.2.3.7.

Patch Inventory An inventory of all available patches is a basic requirement of a patch management system, and will be populated with patches discovered during the notification process. The primary benefit of this inventory is in helping to minimise the number of patches which need be reviewed in the patch management process by resolving any internal patch dependencies - for example, by excluding patches which have been superseded, or automatically resolving the order in which patches should be installed. Further optimisation will be provided by a cross-correlation with the asset inventory to exclude patches for software not installed or patches already installed. Providing user modifiable areas so that testing notes and other discussions about the patch can be added is useful, particularly if the patch and vulnerability group wants to create a centralised organisational patch database.

Vulnerability Inventory The purpose of patch management is to resolve known vulnerabilities. Thus, a database of known vulnerabilities is required. This database will be populated with the high quality vulnerability information available from vulnerability databases. Beyond listing a CVE number and the affected software, vulnerability entries could include information from the OVAL project [192] which provides a standardised XML schema [193] for describing how a vulnerability can be verified. This could be integrated into a vulnerability scanner discussed in the next section. This database too should correlate information with the other databases to display how which vulnerabilities affect software actually deployed in the organisation.

5.2.1.3 Vulnerability Scanner

The difficulty with managing vulnerabilities is that they are an unknown risk. It is difficult to quantify the expected number of vulnerabilities in a product before they are announced. When a vulnerability is disclosed the possibility of a risk is created. However, whether that vulnerability is applicable to the specific configuration of an organisation is not clear. Determining this is not an easy task - which is particularly true in cases when little information is provided with the vulnerability, or it requires a complex set of pre-conditions to be true. To help an administrator in this task, vulnerability scanners can be used. Vulnerability scanners do not mitigate the vulnerability (that is the job of the patch) - rather, their purpose is to discover the existence of a vulnerability.

Given that not all vulnerability disclosures provide enough information with which to generate a verification mechanism, sometimes the only verification that can be performed is to ensure that the patch has been correctly installed. This verification should not be a part of the vulnerability scanner. The vulnerability scanner should be able to independently test for vulnerabilities, and hence to independently verify whether they have been successfully remediated by a patch.

Network-based vulnerability scanners attempt to interrogate the machine remotely, and should be used on all machines that are being patched. Local vulnerability scanners usually require the installation of software on a machine, and can be more time-consuming to set up. Local scanners should be used on critical servers and machines that provide local access accounts. Local scanners can usually perform a more in-depth scan, involving issues such as configuration vulnerabilities, while network-based vulnerability scanners are limited to what the machine presents to the network, which in the case of some machines may be very little. Vulnerability scanning can quickly become quite complex, and scanners usually only focus on a subset of functionality. For example, web applications have specific vulnerability scanning requirements that are different from interrogating open ports for vulnerable services.

5.2.1.4 Patch Testing

Software to support patch testing is notable only in its absence. Virtual machines were discussed in section 3.2.3.4, and can provide a cheap method for multiplexing several different machine configurations on one physical machine, saving hardware costs. However, they are limited in that testing hardware-specific interactions (for example hardware drivers [110]) is poor.

In observing the policy, testing patches is the step likely to require the greatest amount of time, and is the only defence against threats from faulty patches. Methods which allow regression testing of patched applications and software to be narrowed in scope could potentially provide a dramatic speed increase in patch deployment. By tracking the dependencies of the software being patched, and particularly the dependencies of the patched component, a list of components most affected by the change introduced by a patch can be generated. Much of this scoping is currently done manually. For example, if there is a patch to Mozilla Firefox's handling of JavaScript, then web applications that rely on JavaScript should be the focus of testing, rather than print functionality. Applications such as Microsoft's *Strider*, or Sun Microsystem's *sowhat* [111] can provide this insight. In addition, a well maintained patch database that includes testing notes could allow patches with similar dependencies, that are re-issued or linked via a dependency can help to prioritise tests that previously displayed problems.

5.2.1.5 Patch Packaging

Due to the proliferation of package managers and their related patch formats such as Debian's *.deb*, RedHat's *.rpm*, FreeBSD's ports and Microsoft's *.msi* there is a lot of functionality that has been placed in the patch distribution format, or package. Several aspects of patch packaging are discussed below.

Dependency Tracking Many of the complexities of patch dependencies can be automatically resolved by providing enough information in the packaging of the patch. There are several different types of dependencies that could occur. The dependency types used by Debian's .deb package format are used as an example [69], as the dependencies between thousands of open-source projects are difficult to maintain, and Debian's APT has a history of performing this task well.

- Depends - package A depends on package B if package A cannot run without without package B. In the case of source packages this is further decomposed into packages required to compile package A (build dependency) and packages required to run package A (run-time dependency). This is a hard dependency.
- Recommends - package A recommends package B if the package maintainer decides that most users would only want package A with the functionality of package B.
- Suggests - package A suggests package B if package B is related to or enhances the functionality of package A.
- Conflicts - package A conflicts with package B when package A cannot run with package B installed. This is often combined with 'Replaces' as conflicts usually occur between packages providing the same functionality.

- Replaces - package A replaces package B when package A contains similar files to package B that would result in the files from package B being replaced or overwritten if package A were installed at the same time.
- Provides - package A provides package B when package A has the same files and functionality as package B. This is an abstraction of functionality from a package, as often several packages exist to fulfil one purpose.

This dependency tracking needs to be implemented in a package management solution, and should not be implemented in the specific patch package. However, the quality of dependency tracking is directly related to how much information is provided by the actual patch package. An alternative would be to provide the dependency information through another channel, and minimise the patch package. Either way, detailed dependency information will ensure that patches are installed smoothly and in the correct order with conflicts minimised.

Binary Patching Once the patches are fetched, the dilemma of whether to replace the entire binary or use a binary patch is presented. Microsoft used binary patching techniques in the past, but decided to stop due to the unpredictable behaviour created by differing configurations. Investigation into binary patching algorithms will be conducted and an option to either patch the binary or replace it in its entirety will be given to the administrator. The advantage of binary patching is a significantly reduced distribution time, especially for the often small changes that a patch performs. The created patch and relevant documentation will then be stored in a patch database. This is separate from the systems database as it could be beneficial to have this database available to the Internet as a whole. This would allow organisations to learn from one another's patching techniques and reduce effort. This is best summarised in a quotation from Mykolas Rambus, CIO of WP Carey, "It would take an industry body - a nonprofit consortium-type setup-to create standard naming conventions, to production test an insane number of these things, and to keep a database of knowledge on the patches so I could look up what other companies like mine did with their patching and what happened." [33] It is hoped that instead of a consortium, a community could be created to share their experiences.

Traditionally, patches are distributed by packaging files to be replaced, instead of packaging the differences between the two versions. The advantage of the traditional method is that the same package can be used to upgrade from any (or many) previous versions or for new users to perform a fresh install. Thus, the software maintainer's job is made easier. However, if the difference

	Patch Tool					
	bzip2 compression		**xdelta**		**bsdiff**	
Binary	*bytes*	*percent*	*bytes*	*percent*	*bytes*	*percent*
gaim	317 699	100%	3 877	1.22%	782	0.25%
gaim-remote	4 979	100%	157	3.15%	140	2.81%
lsusb	20 673	100%	17 837	86.28%	15 731	76.09%
usbmodules	5 040	100%	3 815	75.69%	2 944	58.41%
BSD ls -> GNU ls	36 026	100%	36 919	102.48%	37 604	104.38%

Table 5.1: Table comparing file sizes of different methods of distributing the same file.

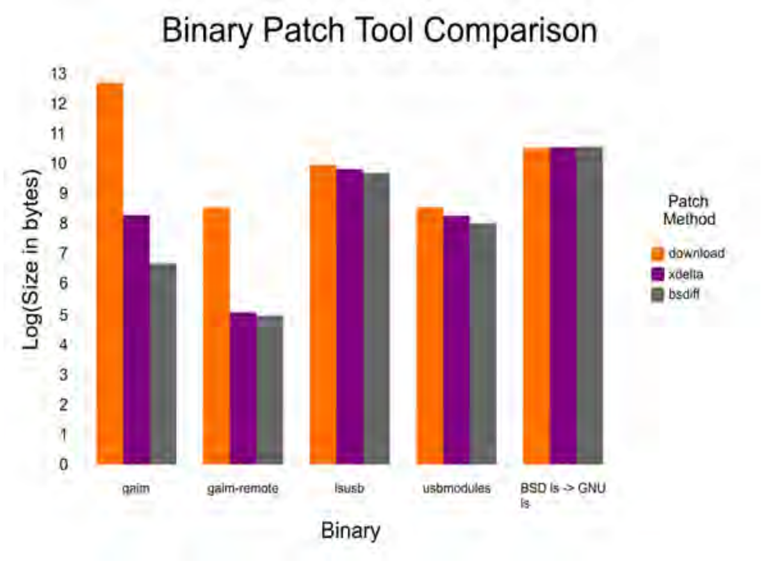


Figure 5.1: Graph of the effectiveness of binary patch tools

from one version to the next is only a small change, the user will still be forced to download a full copy of the new software. An alternative is to package the incremental difference between the two files: this would result in smaller patches, particularly when only a small change has been made, as is often the case with security patches. Below is a comparison of two binary patching tools, Xdelta [194] and bsdiff [195]. As can be seen in the table 5.1 and figure 5.1 drawn from an earlier work [112], the binary patches provide anywhere between a 90% to 25% reduction in size compared to a full binary download. The last example was a test case in which two completely different files were used (i.e. there were no similarities between the two files).

However, there are some disadvantages to binary patches. A binary patch can only patch from one specific version to another. Thus if the end user is likely to have several different versions of a vulnerable software package, multiple binary patches may have to be distributed. This can sometimes make a binary patch larger than a traditional patch (this is certainly the case with Microsoft's binary patching [196]). With careful package management, this risk can often be mitigated by tailoring the delivered patches to the systems requesting them (i.e. a semi-intelligent patch tool) or by attempting to keep software versions in lock-step. The last disadvantage is that it is harder for a software maintainer to manage binary patches with the release of one new version requiring several binary patches to handle users who are not running the immediately previous version, as patches will be required for each version up to the current. This process that can be fairly well automated.

Patch Authentication A vendor-provided patch provides a central point of failure for every application deploying it. Once it is distributed to a client's own centralised patch deployment system, a central point of failure persists for all machines within that organisation. Thus, the patch needs to be authenticated every time it is distributed. This is quite easily solved using public key cryptography. Providing a public key with which users can verify communications and patches signed by the vendor's private key can help ensure that patches are not tampered with. A model of how this can practically be achieved in open-source projects is provided in the *Strong Distribution HOW-TO* [197], and is expanded on by Sohn *et al.* [198]. However, practical problems often occur when users do not update their stored copy of the vendor's public key, or when vendors do not correctly sign patches [199]. If we assume that the vendor is behaving correctly, many of the tasks for authenticating packages can be automated - and should be at every point possible, particularly once it has been downloaded from the vendor to a client machine.

Patch Back-Out Providing an effective back-out mechanism to allow changes introduced by patches to be undone would go a long way to minimising the potential threat of a faulty patch. For some patches this can be quite a trivial process, in which the updated files are merely reverted to their original form. However, in some cases changes introduce features that are not backwards-compatible. For example, if a database schema is changed, any new data added to the database requires considerable effort to converted to the previous schema. Several vendors provide roll-back mechanisms, though these are not always used.

5.2.1.6 Patch Distribution

Most current solutions distribute their patches via either a single server or several servers, depending on the size of the organisation. This method is very inefficient and subject to dangerous denial-of-service (DoS) attacks. The advances in peer-to-peer distribution should not be ignored, and protocols such as Bittorrent [200] or other rapid distribution methods could provide benefits in mitigating DoS attacks against central distribution centres. This will have the advantage of a reduced bandwidth load on the distributing server [201], and provide greater security as many more machines will need to be compromised to distribute a malicious binary (assuming the initial upload is correctly authenticated). The public key infrastructure discussed in section 5.2.1.5 can be implemented. The server component can be given a server root key whose public component is published to the network. This would allow for each patch to be signed by the root server's key and the agent to verify this by checking against the published root key. Additionally, deployment can easily be scheduled to occur at certain times appropriate to the organisation, even when there is no one present, allowing unattended installs of less critical patches to occur with minimal interference.

5.2.1.7 Reporting

Before a patch can be applied to mission critical servers, the patch needs to be tested with the current system configuration, and processes for removing the patch are usually drawn up. This can take a large amount of time to troubleshoot, which often leaves the system administrator with a dilemma: to deploy the patch and risk losing critical services, or not deploy and risk a security breach. To resolve this, a system administrator requires more information on the possible effect an exploit could have on his organisation. Reporting is thus a major advantage of such a project, due to the decision-making benefits.

Reporting should involve extensive correlation of information between then vulnerability, patch, and hosts' inventories. This should be extended with information gleaned during the deployment process, such as patch and vulnerability verification information. The ability to create ad-hoc queries into this data would allow an administrator to rapidly and accurately obtain data relevant to the risk management decision-making. Additionally, other tasks such as metrics and trends can be extensively supported by a well implemented reporting function.

5.2.1.8 Summary

Table 5.2 provides a summary of the functional areas and the tasks performed in each.

5.2.2 Architecture

"The entire agent vs. agentless debate [is] a red herring."
– Mark Shavlik, CEO of Shavlik Technologies [181]

A brief discussion on the nature of agent-based versus agentless patch management solution is included here only because it is discussed in nearly every paper on the subject [42, 91, 94]. However, we believe that this debate is essentially a propaganda war between various vendors attempting to sell their product. The quote at the beginning of this section has been deliberately taken out of context. It is taken from a paper entitled *Security Patch Management: Breaking New Ground* [181] published by Shavlik, vendor of the HFNetChkPro patch management solution. The title does not reveal that the paper is actually a discussion on the agent versus agentless debate, and sides strongly with agentless technology. It is not surprising to learn that HFNetChkPro is an agentless solution. The paper contained a number of unsubstantiated and demonstrably untrue claims, and of its meagre five references, one refers to semantic web intelligent agents, and appears to be quoted incorrectly. The author mistakenly compares deploying *patches* with agentless technology to deploying *agents and patches* with agent-based technology - even though the agent would only need to be deployed once. We believe that papers such as this, and the resulting marketing hype as vendors attempt to advertise their selectively agent or agentless solution as the best architecture, have contributed to the amount of time that has been devoted to this debate. Agent and agentless solutions are both necessary for a patch management solution, and many tasks required for patch management can be done using either.

5.2.2.1 Agentless

And agentless, or non-agent, architecture should technically be able to operate without utilising any software installed on the client machine, thus limiting the server to things such as blind vulnerability scans. However, in reality "agentless" usually refers to the fact that no additional software is required to be installed on the client, and standard remote administration tools are

1. Notification
   - Vulnerability
   - Patch
   - Threat
2. Inventory Management
   - Network hosts inventory
   - Host software inventory, including patch level
   - Available patch inventory with dependency tracking
   - Vulnerability inventory
3. Vulnerability Scanner
   - Remote network scanner
   - Local host scanner
4. Patch Testing
   - Virtual Machines
   - Test scoping
5. Patch Packaging
   - Authentication & Authorisation
   - Compression
   - Back-out
6. Patch Distribution
   - Scheduler
   - Distribution
7. Reporting
   - Correlate information sources (hosts, software, patches, vulnerabilities, verification, time)

Table 5.2: Patch Management Automation

used. The use of these administration tools amounts to the same essential functionality as an agent-based architecture. The upshot of this is that it encourages the use of standards, as default remote administration tools are used, instead of proprietary communication protocols. However, given that the agentless solutions often only use the remote administration capability to deploy executable content (an agent), this is limited, and the waters dividing agent from agentless software become murky indeed. Given the difficulty in drawing a clear distinction between agent and agentless software, and the inadequacy of the “any additional software required” definition, we will provide a slightly different, but functionally useful definition.

Agentless technology is limited to pushing patches to clients - an ’interrupt’ approach wherein patches are pushed when they arrive, rather than a pull-based ’polling’ approach, in which clients regularly query the server for new patches. Pushing patches is limited in situations where machines are not connected to the network during the patch deployment, requiring the server to perform the same ’polling’ as an agent, albeit in reverse, to detect when disconnected machines re-join the network. Conversely, this approach is quite useful in the case of new machines joining the network that might not have had an agent deployed to them yet, or in situations where an agent has failed, possibly due to a conflict caused by a new patch. Thus, an agentless approach is both a necessary and sufficient part of an effective patch management solution.

5.2.2.2 Agent

With agent-based architecture there is a central server which can serve patch files and an agent that is installed onto the client machines to perform local tasks. The amount of work performed by the server and the clients varies greatly depending on the product’s feature set. Agent-based patching can use either push (interrupt) or pull (polling) type patching. With agent based patching, when patches are pushed to clients the server initiates a connection to the client machine’s agent, instructing it to deploy the patch. When patches are pulled, the client machine’s agent will initiate the connection to the server, copy the patch, deploy it and report back to the server. Pushing patches allows a server to push patches to clients as soon as the patch is available. This can help in reducing the time to patch. However, if a machine does not receive the push instruction, the patch might not get installed. This is particularly pertinent with mobile devices which often pose an increased risk as they allow malware to piggyback its way past a firewall. With pull-based patching, the mobile device can ’check-in’ when it is back within the organisation instead of having to wait until the next patch release cycle. Ideally, an agent based solution should utilise both methods to minimise patch deployment time. Patches can be pushed as soon as they are available for deployment, and agents can check-in to pull patches at regular intervals, or during a client-side event such as a reboot or rejoining a home network.

Thus, agent based technology gives you more options and more control. It also prevents credentials from being transferred around the network and reduces the amount of bandwidth required. Difficulty in installing agents can be avoided by including the agent in standard baseline images, or using remote administration tools to deploy them. However, agent-only architectures cannot protect new machines on the network, or in situations when the agent fails, and therefore cannot be sufficient as a patch management solution.

Thus, a combination of agentless and agent-based architectures is ideal for a patch management solution.

5.2.3 Available Tools

Given the large number of tools claiming to be capable of managing the many aspects of patch management, an in-depth review of each one is outside the scope of this research. In some earlier work, an in-depth review of Microsoft's WSUS patch management product was conducted, and this is included as appendix B. This section provides a brief description of the evolution of various forms of patch management tools, and a classification of a handful of popular tools based on the functionality and capabilities described above.

5.2.3.1 Evolution

Generally, patch management software fits into one of five categories. These categories appear to have arisen as software that provided an aspect of the patch management process bolted on the ability to deploy patches, with software actually developed to manage the entire patch management process (4) the only noticeable exception.

1. Vulnerability Scanner
2. Configuration Managers

3. Package Manager
4. Original Patch Managers
5. Defence in-depth tool

The last category is usually consists of additional defences that can be used, and does not include tools directly related to patch management. These tools will be discussed in section 5.3.

Vulnerability Scanners These tools started off simply as vulnerability scanners and realised the need to provide the option of remediating discovered vulnerabilities. Thus, a method for pushing patches to machines was added. These tools are usually agentless solutions that started off as remote network vulnerability scanners. An example of this type of component is Shavlik's HFChkNet which started off as the engine used for vulnerability scans in Microsoft's Baseline Security Analyser (MBSA) scanner [202]. However, Shavlik has since split its products into many separate products, each implementing some specific functionality, and thus GFI Languard [203] may be a more appropriate example.

Configuration Managers These tools attempt to centralise the administration of all aspects of machines on a network or in a domain. Examples of these include Microsoft's Systems Management Server (SMS) [204], IBM's Tivoli [205] or Configuresoft [206]. These are usually expensive agent-based solutions that already performed many of the tasks necessary for patch management, such as asset and change management, that have since expended to include patch management. The advantage they provide is that one agent can be used for a variety of tasks instead of managing several agents.

Package Managers Package management has traditionally been driven by the unix operating systems which have needed to develop systems to manage the large number of third-party software they require. Each operating system has its own package management system. Microsoft's SUS and later incantation, WSUS [207] is essentially a package manager with some extra functionality. Package managers have traditionally focused almost exclusivley on the patch inventory, packaging, and distribution, but become patch managers when they branch out to include some of the other functionality described in section 5.2.1. Examples of this include Debian's APT [69], and RedHat's RPM [175] systems.

Original Patch Managers This class of software describes tools that have been recently developed with the original intention of fulfilling the needs of patch management. It is not surprising that this is the largest category of patch management tools. Some examples include UpdateExpert, Patchlink Update [208], BigFix [209] and Ecora Patch Manager [210].

5.2.3.2 Examples

Some examples of each type of product are given below.

- Vulnerability Scanner
  - GFI Languard [203]
- Configuration Managers
  - IBM Tivoli [205]
- Package Manager
  - Debian APT [69]
  - FreeBSD Ports [211]
  - Microsoft WSUS [207]
- Original Patch Managers
  - BigFix [209]
  - Patchlink Update [208]

Table 5.3 indicates which of the broad functionality areas each product fulfils, though the quality and depth of the implementation is not represented. The feature set of each category is made somewhat clearer, but in general many of the patch management tools automate similar functionality. All of the tools implemented patch notification, but Tivoli was the only one to correlate information from threat sensors. Tivoli provided almost all of the functionality of the other products, nicely demonstrating the scope of configuration management tools. All of the tools apart from the package managers provide vulnerability scanning, however BigFix and Patchlink do this with third-party tools such as Nessus that they integrate with their product, while Tivoli and GFI appear to provide their own scanners. Vulnerability scanners are available for WSUS-, Apt-, and Ports-based systems, but they are not integrated into the tool. Similarly reporting tools and host databases are available for Apt and Ports, but are not integrated. BigFix, Patchlink, APT and Ports provide their own patches after testing patches released by other vendors, while WSUS is a vendor-specific tool. GFI and Tivoli use the vendor patches as they are released. BigFix's and Patchlink's patches are available through a paid subscription service. It is interesting to note that none of the tools provide support for automating testing. Some allow for a 'test' group to be created and patches deployed to them, but this does not provide any functionality or actually help with the testing.

We can see that there are still opportunities to develop the functionality of patch management tools.

	Notification	**Inventory Management**	**Vulnerability Scanner**	**Testing**	**Packaging**	**Distribution**	**Reporting**
Possible Values	(P, V, T)	(H, P, V)	(Y | N | TP)	(Y | N)	(Y | N)	(Y | N)	(Y | N)
Tivoli	P, T	H, P, V	Y	N	N	Y	Y
BigFix	P	H, P	TP	N	Y	Y	Y
Patchlink	P	H, P	TP	N	Y	Y	Y
GFI	P, V	H, P, V	Y	N	N	Y	Y
WSUS	P	H, P	N	N	Y	Y	Y
Apt	P	P	N	N	Y	Y	N
Ports	P	P	N	N	Y	Y	N

Key

- H - Host
- P - Patch
- T - Threat
- V - Vulnerability
- TP - Third Party

- Y - Yes
- N - No
- (a, b) - a and b
- (a | b) - a or b

Table 5.3: Comparison of Patch Management Tool Functionality

5.3 Defence in-depth

Defence in-depth is a security strategy pioneered by the military, wherein multiple layers of security are used to minimise the amount of damage caused by an intrusion [21]. In a broader context it refers to every aspect of information security where a combination of people and technology are used to form the multiple layers. In the context of patching it will be used to refer to additional techniques that can be used to mitigate the threats faced by machines with unpatched vulnerabilities. While testing occurs the organisation is left vulnerable, and is often in a situation where it cannot turn off a critical service. Here additional technologies designed to reduce the effectiveness of an attack, or at least to allow for the attack to be discovered, can be utilised to minimise the consequences of a successful exploit.

5.3.1 Firewalls and Anti-Virus

Firewalls and anti-virus solutions are a well understood solution to some attacks. However, they are often fairly inadequate alone. The failings of firewalls were described in section 1.3. The rise in use of web-based applications and the multiplexing of several services over the HTTP port means that a firewall is only useful in certain select circumstances. A firewall should be deployed however, and ports for commonly attacked services such as Windows RPC or OpenSSH should be firewalled off if possible. Additionally, services only required by a select group of people, for example firewall management interfaces, should limit the machines allowed to connect to the port. If a port can be completely firewalled off, then questions as to whether the service is needed at all should be raised.

Anti-virus solutions can help to mitigate attacks. Given their near ubiquitous deployment on all end-user machines, they can help to combat the rise in malicious software that relies on confidence tricks and minimal user interaction to spread. Keeping signatures up-to-date can help to prevent against known malicious software attacks. However, anti-virus solutions that rely solely on signature-based detection are becoming less effective. Statistics provided by the malware submission service, VirusTotal consistently show significantly more failures in detection than successes [212]. For example, statistics for seven days in December 2005 showed 261 successful detections compared to 14 285 failures. Malware variants can often be rapidly created and discreetly spread, making it difficult for malware analysts to discover and analyse each piece of malicious software rapidly. Additional techniques, such as heuristics and policy controls, can help to catch new forms of malicious behaviour [213] by detecting many of the results used by malicious software, instead of the specific technique. For example detecting a browser trying to execute code stored in a data segment assigned to a picture could pick up on any new malware that attempts to use this technique. Thus, when a patch is being tested, signatures for potential attacks can be distributed to client machines to provide short term protection against the threat.

In some cases the anti-virus solution can be effectively coupled with a proxy to provide some protection from the crunchy firewall problem [34] in the form of content filters. An example of such a tool is WebMarshal [214]. With the rise of processing power it is likely that many more application-specific proxies will become available for services multiplexed over HTTP. Both the use of firewalls and anti-virus software is strongly recommended. While neither will provide absolute protection, they can sometimes completely block a threat and or minimise it.

5.3.2 Intrusion Detection/Prevention Systems

There are both host and network Intrusion Detection Systems (IDS) solutions. The focus here is on network IDS's to mitigate attacks conducted over the internet. Intrusion detection systems IDS's operate in a similar manner to anti-virus solutions. Usually, a set of signatures are used to detect signs of malicious activity or heuristics, and policy controls are used to detect new attacks. Signatures are currently the most commonly used method of detection, given the difficulty in determining 'regular' use of diverse network protocols. These signatures will be used to look for patterns in network traffic and alert when they are discovered. However, IDS's have a notoriously high rate of false positives [215], and can require extensive tuning to provide an accurate reporting rate. IDS signatures are easy to create, usually only consisting of a few lines of information, and are often available very soon after the detection of a malicious application. The simplicity of the signatures means that they can be rapidly tuned to lower the false positive rates. Similarly to anti-virus solutions, when a patch is being tested the signature can be used to monitor for any attacks.

An IDS can be turned into an Intrusion Prevention System (IPS) by integrating it with a firewall to block traffic detected as malicious. This could be extended to drop all traffic from hosts detected to have sent malicious traffic. This should be used cautiously, as malicious traffic could be sent from a spoofed address, potentially causing traffic from a legitimate host to be dropped, effectively causing a Denial of Service attack [42]. However, in the short term (with careful monitoring) this can provide an effective tool to minimise the chance of a threat successfully exploiting an unpatched vulnerability.

An extension of IDS systems are block lists [216]. These are lists of addresses known to be involved in malicious activity. They can be used to limit the number of attacks from known bad sources. However, block lists are easily circumvented by moving an attack to a new host. Given the large number of zombie machines theorised to be compromised by attackers, this is often fairly trivial. However, they can be of use in the case of web-based threats in which specific sites are distributing malicious content.

5.3.2.1 Virtual Patching

Some vendors are marketing a defence in-depth tool providing 'virtual patching,' most notably BlueLane Technology's PatchPoint solution [217]. The basic working of the technology appears to be that of an IPS with the additional feature of being able to 'correct' traffic. This amounts to stripping out the known bad part of malicious network activity and forwarding it down the wire. While the marketing hype promises this as a final solution to the woes of stop-gap defences during patch testing, we believe these claims to be false. Correcting traffic is a two step process. First malicious traffic must be detected, then it must be corrected. This is the exact process used by an IPS, where first malicious traffic is detected and the 'correction' is a total block of the request. Thus, the only difference between 'virtual patching' and an IPS is the additional step of trying to correct the traffic. This is arguably a bad approach. The fallibility of signatures has already been mentioned. A signature only provides detection for known bad activity. Even in these cases this can be a difficult task. For example, during the recent WMF vulnerability, malicious WMF files were made difficult to detect through the use of gzip compression and header-padding tricks [218]. Thus, a malicious request may have more to it than a signature can detect, and a good strategy is to block the entire request if part of it has been discovered as malicious. Trying to correct the request and forwarding it is similar to going to the effort of detecting know criminals, then removing their visible weapons and letting them into your jewelry store. Ptacek [219] provides a nice comment on the matter:

> If your in-line network security device claims to provide "virtual patching", the box must use the actual binary patch from [the vendor] to do it.

5.3.3 Other Hardening

There are a plethora of other defence in-depth steps that can be taken to harden the configuration of a machine and its services. Some examples include host-based IDS, cryptographically signed executables, router white lists to limit worm infections, and configuration hardening. This is a broad area with the potential for much innovation that can be leveraged to extend the defence in-depth concept to buy an administrator more time to test patches.

5.3.4 Software Selection

Given that patch management will become a significant and regular activity if properly implemented, minimising the number of patches required by deployed software will be of both a security and a cost benefit. By making good choices when software is first being deployed, high maintenance and patch costs can be avoided later when the cost of migrating away from the software is too high (some software is patched more than others). Unfortunately, it is not as simple as figuring out which software has fewer patches, as this is no indication of actual security. Older, more mature software, will often have a larger user base and a correspondingly large support community, which often results in more people finding vulnerabilities, and due to its popularity, more people looking for vulnerabilities. This may result in more patches being released, making it appear poorly coded, while it is objectively more secure than a new software project fulfilling the same functionality without the same level of security review.

An organisation then has two choices. The first is which software package should be used, given multiple products. The organisation should conduct a security review of each product, this review should be more in-depth than counting the number of patches and vulnerabilities announced for each product. If possible, the types of vulnerabilities, frequency of serious vulnerabilities, and patch response time should be included. Once this decision is made there is a choice between which version of the software should be used. This choice is less frequently made, as most organisations deploy the latest version. However, this is not always the best choice. It is hypothesised that older software that fulfils all necessary business and technical requirements and is still actively maintained[2] will have fewer announced vulnerabilities *and* will, in fact, be

[2]Actively maintained within acceptable limits. For example Microsoft Windows 98 was until June 2006 still being unofficially maintained, and patches were released slowly, placing it outside the definition of 'actively maintained'.

more secure. To test this, vulnerability data for the Linux kernel was collected from the Common Vulnerabilities and Exposures List [220] and analysed. The results seen in table 5.4 and figure 5.2 demonstrate that older kernel versions have fewer vulnerabilities over time, and hence fewer patches to fix those vulnerabilities, than their newer counterparts. This is due to three primary factors:

- There is less functionality and code with potential security holes.
- Older software has been subject to more and longer security review
- There is less interest in discovering vulnerabilities in older software

	Year						
Kernel Version	**1999**	**2000**	**2001**	**2002**	**2003**	**2004**	**2005**
2.2	3	4	17	2	1	0	3
2.4	n/a	1	6	5	12	30	11
2.6	n/a	n/a	n/a	n/a	2	33	35
Total	4	6	19	7	15	50	40

Table 5.4: Table depicting vulnerabilities in the different Linux kernel versions over time
Source: CVE [220]

Note: The total columns do not add up correctly as some vulnerabilities affect multiple kernel versions or non-standard kernel patches. For example in 2004 there were 13 vulnerabilities which overlapped and in 2000 one vulnerability was in the trustees kernel patch and in 1999 one vulnerability was in the 2.0 kernel version which isn't included. These are included in the total to provide an idea of the general reporting trends in the linux kernel.

Thus, if the security of an older software version is still being maintained (within acceptable limits) and provides all required functionality, it is often better to use the older version over the newer version, as this will reduce the number of patches required without adversely affecting security. This analysis is specific to a well known project with a large user and developer base, such as the popular operating systems and server software (e.g. Linux, Windows, Apache). However, this behaviour is not intrinsic and summaries of vulnerability numbers are no substitute for thorough analysis.

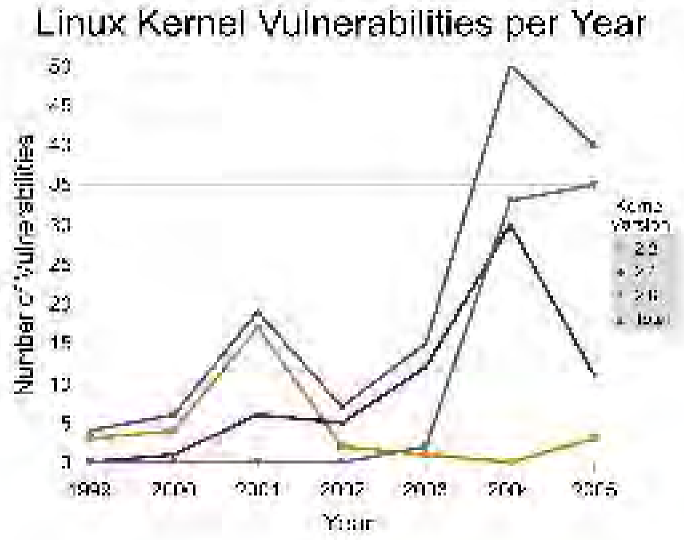


Figure 5.2: Graph of the number of vulnerabilities in different Linux kernel versions per year.
Source: CVE [220]

5.4 Conclusion

This chapter has focused on the technical aspects of patch management. First, a description of the functional areas where a patch management policy can benefit from automation was provided. A brief discussion on each of these aspects was then provided, pointing out where specific technologies could be used to improve on current patching tools. After this, a brief analysis of some existing patch management tools was conducted. With the functionality and capabilities model developed in the beginning of the chapter, it was possible to asses existing patch management tools, and it was found that while some provided invaluable automation, a variety of tools is still required, with no solution providing the 'silver bullet'. This chapter further concluded that only a subset of a patch management policy can be automated, but that this automation is necessary to the task. In addition, there is still room for much technical improvement in automated patching tools, particularly in providing tools to make patch-testing easier. The chapter then turned to some additional technical discussion, focusing on other activities and technologies that can be used to improve the patching process. Some defence in-depth techniques were discussed that could allow an administrator to deploy stop-gap defences while a patch was being tested. These defences are not always completely effective, but can help minimise some threats.

This chapter provided a description of actual tools that can be used in assisting with patch management, thus bringing us to the end of the analysis of this thesis. In the next chapter brief summaries and conclusions are provided for each previous chapter.

Chapter 6

Conclusion

6.1 Introduction

In the introduction to this thesis the objectives of this research were put forward. This researcher believes that these objectives have been largely been met. This chapter provides a summary of the work that has been presented, with a focus on how the described objectives were achieved. Hindsight allows for a clearer perception of the activities undertaken during the period of research, and some of the problems encountered are discussed in this context. Finally, further work that has been identified as useful is discussed.

6.2 Objectives

Several objectives were discussed in chapter 1. It was hoped that some sense could be brought to the patch management debate. This sense is sorely needed given the growth of security as an industry, and patch management in particular, where vendors have a commercial interest in hyping threats and products. To this end, seven objectives were proposed, and are repeated here:

1. An analysis of vulnerabilities, exploits and patches by discussing the vulnerability life-cycle.

2. An analysis of vulnerability, exploit and attack trends.
3. An analysis of patches and their problems.
4. A discussion on how to implement a patch management policy.
5. A discussion on how vendors can implement a scheduled patch release policy.
6. A discussion on patch management tools and automating parts of the policy.
7. Tools to help automate and integrate parts of the policy.

Barring the last, it is believed that these objectives have been achieved. The first objective was dealt with in section 2.2, where the conclusions of several sources based on their analysis of empirical evidence were synthesised to produce the most complete understanding of the current vulnerability life-cycle that this researcher is aware of. The second objective utilised this new understanding to discuss the trends that are currently modifying this life-cycle. In section 2.3 several trends indicating that the risks related to vulnerabilities in software were demonstrated. The number of vulnerabilities is increasing, the number of attacks is increasing, while the amount of time available to an administrator to remediate these vulnerabilities is decreasing. This analysis fulfilled the second objective and provided a justification as to why *patches* are necessary and need to be expediently deployed.

The third objective was to analyse why patches appeared to be difficult to manage and install. Section 2.4 discussed several specific problems administrators face when deploying patches were. Two examples were then provided in section 2.4.6, where several of these problems were demonstrated. This objective provided a justification as to why *patch management* is necessary.

The fourth objective was to derive a solution from the understanding now developed. Originally, this objective consisted in a stronger form of objective seven, and it was naively believed that a software tool could effectively manage the problems relating to patch management. However, it was modified to its current state in chapter 3, where an in-depth discussion is provided on how an organisation can develop a patch management policy. In a drastic shift from the original objective, this discussion remained technology-agnostic, and focused on the procedures that could be employed. Section 3.2.3.2 provided an introduction to risk assessment. This was found to be the single most useful part of the derived policy. It was discovered that the largest problem facing patch management was a lack of information with which to assess risks. The risk assessment discussed influenced the rest of the research greatly.

The fifth objective came about due to theome vendors' shift to a scheduled patch release cycle, and the threat of this cycle becoming an industry standard (Microsoft in particular drove much of the inquiry into patch schedules). Within the context of the vulnerability life-cycle described in the earlier chapters, it was clear that an end-user policy dealt with only part of the group relevant to patch management, and that vendors could make a significant difference to any efforts. Chapter 4 thus provided an argumentative analysis of scheduled patch release policies. It was concluded that a patch schedule only works in a situation of delayed disclosure. In the cases of instantaneous disclosure described in section 4.3.3.1, vendors should release beta-patches and benefit from community collaboration and testing, which will make effective vulnerability remediation available sooner.

The sixth objective went through several iterations before being met in its final state. Some confusion as to how to relate the functionality of an ideal patch management tool to the policy proposed earlier existed. This confusion was overcome, and section 5.2.1 provided a discussion on how parts of the patch management policy could be automated. This was then used to classify a subset of current patch management tools in section 5.2.3, demonstrating that patch management tools don't support every necessary step in a patch management policy. In the next section (5.3) cast the net a bit wider, discussing defence in-depth tools that could be used to defend the organisation while patches were being tested. This completed the solution objectives to the problems discovered in objectives one to three.

The need for integration of several management areas and information sources in a patch management policy is critical. It was hoped that tools could be developed to help provide the automation described in the previous objective. However, a lack of time and scope precluded this. This will be further discussed in the 'future work' section below.

6.2.1 Summary

Thus, it is believed that the objectives were met. The problem and its nuances were discerned, and solutions that responded to and mitigated these problems were developed.

6.3 Problems and Solutions

The specific problem this research tried to address was to find workable solutions to the problems presented by patch management. Specifically, the problem was that of vulnerability management: in the face of increasing threats and vulnerabilities, how can patches be used to effectively remediate these vulnerabilities and render the threats moot. The related problem was that of the patch paradox - that without a patch an asset is vulnerable to attack, and with a patch the asset is vulnerable to failure.

The developed solution presented in chapter 3 was to produce a realistically implementable policy guide with which organisations could develop their own comprehensive patch management policies. This policy took into account the many variables present both in vulnerability/attack scenarios and those in the average organisation. Often, administrators have to deploy patches to many machines with diverse requirements that can have complex effects on business processes. The policy focused on risk management in section 3.2.3.2 as a method for directing decision making. The second part of the solution presented in chapter 4 was less direct, but involved a discussion of how vendors could implement a scheduled patch release policy that responded to the threat trends discovered, better integrated with organisation patch management policies and, most importantly, reduced the likelihood to the end-user of exploitation.

6.4 Future Work

There were many aspects of this work that could be branched off into a thesis of their own. For example, several anti-virus companies make their money by focusing solely on virus-threat notification and mitigation, which were but small sub-components of this discussion. Some of the work related to this subject which could be undertaken in the future is discussed below.

6.4.1 Threat Management

Chapter 2 found it very difficult to gain an accurate picture of current threats, or provide comprehensive threat management resources. It is very difficult to discern current threat activity. Most threats mentioned are those attacking on a very large scale, where the scale of the attack is directly related to its amount of coverage. However, a small scale targeted attack could potentially do more damage to an organisation. Tools to provide better real-time threat reporting and correlation from public threat monitors such as dark-nets, internet telescopes, honeypots and vendor sensor networks are thus very important in gaining an accurate picture of current threats. Extending these to include data from local sensors such as IDS and Firewall logs to provide an organisation-scale view of threats would also help to get an accurate picture of attack activity. For example, one current project is Symantec's DeepSight analyser [191], which provides both an internet-wide and organisation-wide view of threat activity.

6.4.2 Vulnerability Detail and Trend Tracking

Current vulnerability information is targeted at providing information on individual vulnerabilities. Trend data can provide some valuable insight. For example, information on the dates on which vulnerabilities were first reported to vendors could provide an understanding of how fast vendors are at providing patches, and provide more empirical evidence for some of the claims in chapter 4. It would also allow vendors to be compared, and possibly motivated to patch faster. More advanced information - such as code-level granularity, allowing description of the vulnerable function or the type of vulnerability, instead of just the vulnerable version of the software product - can be used to look for consistent security errors, providing insight for developers looking to secure their products.

6.4.3 Optimal Time to Patch for Large Vendors

Section 3.2.3.3 worked out the optimal time to patch for a group of vulnerabilities that involved many vendors. Beattie *et al. [60]* called for further research into the optimal time to patch for individual vendors. This would give organisations a better idea of how they could minimise risks from patches. This would integrate well with the vulnerability detail research described above, as calculating the optimal time to patch requires information on when and how often a patch was recalled. This research could help in patch scheduling decisions, choosing between different vendor's products and motivating vendors to improve their patches.

6.4.4 Patch Standards

Patches are currently implemented in several forms, usually specific to an operating system or deployment platform. One of the problems highlighted throughout this work was the existence of multiple patch deployment mechanisms. This is less of a problem on open source platforms, as most package management tools were developed to include a wide range of software - however, it is problematic on Microsoft's platforms. Currently, there exist XML schemas to describe vulnerabilities, most notably VuXML [189] and OVAL [192]. Providing a similar CVE [49] compatible standard description for patches would allow for standard patch deployment applications to be built and used. This would need to coupled with a standard patch packaging format. Given that many of the tasks of a package manager (dependency tracking, reverse dependencies, roll-back etc.) are well understood, a cross-platform deployment mechanism could be developed. This could decouple patch deployment from specific patch deployment tools. This would ease patch testing for multiple platforms and distributions, reducing the redundant testing performed by many groups, particularly in the open-source world. A patch could be developed by Red-Hat and rolled out on a Debian machine with minimal modification.

6.5 Final Word

Patching is a problem that will be with us for a while. However, the current discussions around patching generally revolve around the simple tasks of patch management. It is hoped that this thesis has managed to broaden this discussion, but not unnecessarily so. Some of the issues raised are problems that run right through the information security field, such as threat reporting and change management. The broadest conclusion which can be drawn from this research is that there is no simple solution to the problems of patch management; to realistically implement a comprehensive and effective patch management policy would take some larger organisations several years. However, some dependencies of such an implementation have not been effectively fulfilled either. For example, our knowledge of threats is still very poor. Additionally, the profit motive of many security vendors still has them bowing to the wrong pressures, and it is unlikely that they would change overnight, especially while some still call themselves 'Unbreakable'. Solutions to the problems of patch management will take time before they can be easily implemented.

However, this is the work of a small group, and much more discussion, argument and debate is required to find workable solutions to the problems that face each unique instantiation of a patch administrator. The bar of information security debate must be raised above the noise of vendor marketing and threat hype, so that meaningful discussion can occur at every level of security management. It is our hope that this thesis has nudged the bar a little higher, however it is the continued efforts of the many dedicated security professionals tirelessly analysing and responding to events that has been most notably impressive. The power of a community cannot be denied, and there are many ways in which the community can contribute. Open-source security projects and communities such as Snort [216]or ClamAV [221], open database projects such as OVAL [192] or OSVDB [47] or volunteer organisations such as the ISC [53] all contribute to improving the security response.

References

[1] Dumbill, Edd. *The Next 50 Years of Computer Security: An Interview with Alan Cox*. O'Reilly Network, Interview (September 12, 2005).
Available at: `http://www.oreillynet.com/pub/a/network/2005/09/12/alan-cox.html`

[2] Weaver, Nicholas C. *Warhol Worms: The Potential for Very Fast Internet Plagues*. In (2001).
Available at: `http://www.iwar.org.uk/comsec/resources/worms/warhol-worm.html`

[3] Poulsen, Kevin. *Nachi worm infected Diebold ATMs*. Security Focus - Columnist (November 24, 2003).
Available at: `http://www.securityfocus.com/news/7517`

[4] Harding, Luke. *Court hears how teenage introvert created devastating computer virus in his bedroom*. The Guardian Newspaper (July 6, 2005).
Available at: `http://www.guardian.co.uk/germany/article/0,2763,1522192,00.html`

[5] Thomas, Daniel. *Are our critical systems safe from cyber attack?* vunet.com News (April 21, 2005).
Available at: `http://www.vnunet.com/computing/analysis/2142496/critical-systems-safe-cyber`

[6] Glave, James. *Crackers: We Stole Nuke Data*. Wired News (June 6, 1998).
Available at: `http://www.wired.com/news/technology/0,1282,12717,00.html`

[7] *Sophos Security Threat Management Report 2005*. *Technical report*, SOPHOS Inc. (December 6, 2005).
Available at: `http://www.sophos.com/virusinfo/whitepapers/SophosSecurity2005-mmuk.pdf`

[8] Danchev, Dancho. *Malware - future trends*. In (January 9, 2006).
Available at: `http://www.packetstormsecurity.org/papers/general/malware-trends.pdf`

[9] holy_father@phreaker.net. *Hacker Defender Antidetection Service*. Product Description (December 2005).
Available at: `http://hxdef.czweb.org/about.php`

[10] Eckelberry, Alex. *Massive identity theft ring*. Sunbelt Software Blog (August 4, 2005).
Available at: `http://sunbeltblog.blogspot.com/2005/08/massive-identity-theft-ring.html`

[11] Salusky, William. *Mitgleider Hell*. SANS Internet Storm Center Handler's Diary (October 3, 2005).
Available at: `http://isc.sans.org/diary.php?storyid=722`

[12] *'Mafiaboy' hacker jailed*. BBC News (September 13, 2001).
Available at: `http://news.bbc.co.uk/1/hi/sci/tech/1541252.stm`

[13] *Current Malware Threats and Mitigation Strategies*. *Technical report*, US-CERT (May 16, 2005).
Available at: `http://www.cscic.state.ny.us/msisac/webcasts/05\_05/info/mal\_%20thrt\_mit\_strat.pdf`

[14] *386BSD + LINIX + GNU + X11R5 on CDROM - let us know what you want!* USENET (December 1, 1992).
Available at: `http://groups.google.com/group/comp.unix.bsd/browse\_thread/thread/134942a64ef36f5e/8d03067120d4f2bf`

[15] *When will HP supply PATCHES before they are Required?* USENET (November 17, 1992).
Available at: `http://groups.google.com/group/comp.sys.hp/browse\_thread/thread/e065debcf70b5ec0/5cd814ab642863ce`

[16] *Top 10 Admin problems on Suns?* USENET (February 27, 1992).
Available at: `http://groups.google.com/group/comp.sys.sun.admin/browse\_thread/thread/921af6e2129df23c/5f95293a20a34f19`

[17] Wall, Larry. *Patch version 1.3*. USENET (May 24, 1985).
Available at: `http://groups.google.com/group/mod.sources/browse\_thread/thread/c5240ceb77b7f586/488b0929254d936a`

[18] Mohd A. Bashar, Markus G. Kuhn E. H. Spafford S. S. Wagstaff Jr, Ganesh Krishnan. *Low Threat Security Patches and Tools*. In *IEEE Computer Society* (1997). CSD-TR-96-075; COAST TR 97-10.

Available at: `https://www.cerias.purdue.edu/tools_and_resources/bibtex_archive/archive/97-10.pdf`

[19] Eichin, Marchk W. and Rochlis, Jon A. *An Analysis of the Internet Virus of Novemberember 1988*. In *IEEE Symposium on Research in Security and Privacy* (1989).

Available at: `http://web.mit.edu/eichin/www/virus/main.html`

[20] Bejtlich, Richard. *Miscategorizes Threats*. Blog Entry (July 8, 2005).

Available at: `http://taosecurity.blogspot.com/2005/07/cool-site-unfortunately-miscategorizes.html`

[21] US Army Information Assurance Division. *Army Regulation 25-2*. Glossary (November 14, 2003).

Available at: `http://ia.gordon.army.mil/iaso/Army/AR25-2/main.htm\#term`

[22] Office of Cyber Security & Critical Infrastructure Coordination. *National Webcast Initiative, Cyber Security Risk Assessment Webcast, Glossary of Terms*. Glossary (August 26, 2004).

Available at: `http://www.cscic.state.ny.us/msisac/webcasts/8\_04/info/804\_webcast\_glossary.htm`

[23] Stoneburner, Gary; Goguen, Alice and Feringa, Alexis. *Risk Management Guide for Information Technology Systems*. *Technical report*, National Institute of Standards (NIST), Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology, Gaithersburg, MD 20899-8930 (July 2002). Special Publication 800-30.

Available at: `http://csrc.nist.gov/publications/nistpubs/800-30/sp800-30.pdf`

[24] *The Security Risk Management Guide*. *Technical report*, Microsoft (October 15, 2004).

Available at: `http://www.microsoft.com/technet/security/topics/policiesandprocedures/secrisk/default.mspx`

[25] Bejtlich, Richard. Personal Communication (December 11, 2005).

[26] *Guideline for Management of IT Security-Part 1: Concepts and Models for IT security*. *Technical report*, ISO/IEC (1996).

[27] *Definition: Patch*. The Jargon File.

Available at: http://www.catb.org/~esr/jargon/html/P/patch.html

[28] *KB 824684*. Microsoft Knowledge Base (September 15, 2005).

Available at: http://support.microsoft.com/kb/824684

[29] *Oracle9i Database Administrator's Guide*. Product Guide (April 23, 2002).

Available at: http://www.lc.leidenuniv.nl/awcourse/oracle/server.920/a96521/dba.htm\#13284

[30] Whitaker, Steve; Fish, Barry and Sands, Carl. *Solaris Patch Management: Recommended Strategy*. *Technical report*, Sun Microsystems (February 2005).

Available at: http://www.sun.com/blueprints/0205/819-1002.pdf

[31] Eschelbeck, Gerhard. *The Laws of Vulnerabilities*. In *Black Hat Briefings* (edited by Jeff Moss). Black Hat, Inc, 2606 Second Avenue, 406, Seattle, WA 98121 USA (July 2003).

[32] *Analysis of the Witty Worm*. *Technical report*, LURHQ (March 20, 2004).

Available at: http://www.lurhq.com/witty.html

[33] Berinato, Scott. *Patch and Pray*. In *CSO Online* (August 2003).

Available at: http://www.csoonline.com/read/080103/patch.html

[34] Cheswick, Bill. *The Design of a Secure Internet Gateway*. In *Proceedings of the USENIX Summer 1990 Conference*, pages 233–237. Anaheim, CA (June 11-15, 1990).

Available at: http://research.lumeta.com/ches/papers/gateway.ps

[35] Mann, Charles. *Interview with Bruce Schneier*. The Atlantic News, Interview (September 2002).

Available at: http://www.theatlantic.com/doc/prem/200209/mann

[36] Dekker, Marcel. *The Froehlich/Kent Encyclopedia of Telecommunications*, volume 15. New York (1997).

Available at: http://www.cert.org/encyc_article/tocencyc.html\#History

[37] Arbaugh, William A.; Fithen, William L. and McHugh, John. *Windows of Vulnerability: A Case Study Analysis*. In *Computer*, volume 33, no. 12: pages 52–59 (2000). ISSN 0018-9162. doi:http://dx.doi.org/10.1109/2.889093.

[38] Browne, Hilary K.; Arbaugh, William A.; McHugh, John and Fithen, William L. *A Trend Analysis of Exploitations*. In *SP '01: Proceedings of the 2001 IEEE Symposium on Security and Privacy*, page 214. IEEE Computer Society, Washington, DC, USA (2001).

Available at: `http://www.securityfocus.com/data/library/CS-TR-4200.pdf`

[39] Schneier, Bruce. *Full Disclosure and the Window of Exposure*. Crypto-Gram Newsletter (September 15, 2000).

Available at: `http://www.schneier.com/crypto-gram-0009.html\#1`

[40] Eschelbeck, Gerhard. *The Laws of Vulnerabilities 2005*. Qualys Research & Development (2005).

Available at: `http://www.qualys.com/research/rnd/vulnlaws/`

[41] Dacey, Robert F. *GAO-03-1138T: Effective Patch Management is Critical to Mitigating Software Vulnerabilities. Technical report*, United States General Accounting Office (Septmember 10, 2003). Testimony Before the Subcommittee on Technology Information Policy, Intergovernmental Relations, and the Census, House Committee on Government Reform.

Available at: `http://www.gao.gov/cgi-bin/getrpt?GAO-03-1138T`

[42] Mell, Peter; Bergeron, Tiffany and Henning, David. *Creating a Patch and Vulnerability Management Program. Technical report*, National Institute of Standards (NIST), Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology, Gaithersburg, MD 20899-8930 (November 2005). Special Publication 800-40 ver. 2.

Available at: `http://csrc.nist.gov/publications/nistpubs/800-40/sp800-40.pdf`

[43] Panko, Ray. *Human Error Website*. Research Website (April 1, 2005).

Available at: `http://panko.cba.hawaii.edu/HumanErr/`

[44] Bernstein, D. J. *The qmail security guarantee*. Website (May 29, 2005).

Available at: `http://cr.yp.to/qmail/guarantee.html`

[45] Ellis, James; Fisher, David; Longstaff, Thomas; Pesante, Linda and Pethia, Richard. *Report to the President's Commission on Critical Infrastructure Protection. Technical report*, CERT® Coordination Center, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania (January 1997).

Available at: `http://www.cert.org/pres\_comm/cert.rpcci.body.html`

[46] *CERT/CC Statistics 1988-2005*. CERT/CC Website (January 2005).
Available at: `http://www.cert.org/stats/cert\_stats.html`

[47] *Open Source Vulnerability Database Search*. OSVDB Website (December 2005).
Available at: `http://osvdb.org/search.php`

[48] Martin, Brian; Christey, Steve and White, Dominic. *SANS Top 20 Report Value*. Online Discussion (November 2005).
Available at: `http://www.osvdb.org/blog/?p=67\#comments`

[49] *Statistics Query Page*. National Vulnerability Database Website (December 2005).
Available at: `http://nvd.nist.gov/statistics.cfm`

[50] *All Secunia Security Advisories 2003-2005*. Secunia Website (December 2005).
Available at: `http://secunia.com/graph/?type=all\&graph=adv`

[51] Houle, Kevin and Weaver, George. *Trends in Denial of Service Attack Technology*. In (October 2001).
Available at: `http://www.cert.org/archive/pdf/DoS\_trends.pdf`

[52] Howard, John D. *An Analysis Of Security Incidents On The Internet, 1989 - 1995*. Ph.D. thesis, Carnegie Mellon University, Pittsburgh, Pennsylvania 15213 USA (April 7, 1997).
Available at: `http://www.cert.org/research/JHThesis/Chapter12.html`

[53] *DShield - Distributed Intrusion Detection System, The Internet's Early Warning System and Internet Security community site*. Product Website.
Available at: `http://www.dshield.org/`

[54] Moore, David; Voelker, Geoffrey M. and Savage, Stefan. *Inferring Internet Denial-of-Service Activity*. In *Proceedings of the 10th USENIX Security Symposium*. Washington, D.C., USA (August 2001).
Available at: `http://www.usenix.org/publications/library/proceedings/sec01/moore.html`

[55] Yegneswaran, Vinod; Barford, Paul and Ullrich, Johannes. *Internet intrusions: global characteristics and prevalence*. In *SIGMETRICS '03: Proceedings of the 2003 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 138–147. ACM Press, New York, NY, USA (2003). ISBN 1-58113-664-1. doi: http://doi.acm.org/10.1145/781027.781045.

[56] *Survival Time History*. SANS Website (December 2005).
Available at: http://isc.sans.org/survivalhistory.php

[57] *Overview of Attack Trends*. *Technical report*, CERT/CC (October, 11 2005).
Available at: http://www.cert.org/archive/pdf/attack_trends.pdf

[58] Kaminsky, Dan. *Scanrand Dissected: A New Breed of Network Scanner*. *Technical report*, LURHQ Threat Intelligence Group.
Available at: http://www.lurhq.com/scanrand.html

[59] Jontz, Sandra. *Navy, Marines Block Commercial Email Sites*. Military.com News (October 19, 2005).
Available at: http://www.military.com/NewsContent/0,13319,78905,00.html

[60] Beattie, Steve; Arnold, Seth; Cowan, Crispin; Wagle, Perry; Wright, Chris and Shostack, Adam. *Timing the Application of Security Patches for Optimal Uptime*. In *LISA '02: Proceedings of the 16th USENIX conference on System administration*, pages 233–242. USENIX Association, Berkeley, CA, USA (2002).
Available at: http://www.usenix.org/publications/library/proceedings/lisa02/tech/full_papers/beattie/beattie_html/

[61] Turner, Dean; Entwisle, Stephen; Friedrichs, Oliver; Ahmad, David; Hanson, Daniel; Fossi, Marc; Gordon, Sarah; Szor, Peter; Chien, Eric; Cowings, David; Morss, Dylan and Bradley, Brad. *Symantec Internet Security Threat Report: Trends for July 04-December 04*. *Technical report*, Symantec (March 2005). Volume VII.
Available at: http://ses.symantec.com/pdf/ThreatReportVII.pdf

[62] Eschelbeck, Gerhard. *Security Vulnerabilities, Exploits and Patches*. Creativematch Online Magazine (May 3, 2005).
Available at: http://www.creativematch.co.uk/viewnews/?90970

[63] Sancho, David. *The Future of Bot Worms: What we can expect from worm authors in the coming months*. *Technical report*, Trend Micro (2005).
Available at: http://www.trendmicro.com/NR/rdonlyres/B612D246-283C-444C-8A92-B0AC6782A2D1/17115/Future_of_Bots_FINAL.pdf

[64] Miller, Charles. *Expanding Exposure: The Decreasing Time Between Web Application Vulnerability and Exploitation*. OWASP Papers Program (November 11, 2005).
Available at: http://www.owasp.org/docroot/owasp/misc/webapp-oswap.doc

[65] Long, Johnny. *Goggledork Database*.

Available at: `http://johnny.ihackstuff.com/`

[66] Flake, Halvar. *SABRE BinDiff*. Product Website (June 26, 2005).

Available at: `http://www.sabre-security.com/products/bindiff.html`

[67] Rescorla, Eric. *Security holes... Who cares?* In *Proceedings of the 12th USENIX Security Symposium*, pages 75–90 (August 2003).

Available at: `http://www.rtfm.com/upgrade.pdf`

[68] *Software Installation and Maintenance*. Microsoft TechNet.

Available at: `http://www.microsoft.com/technet/prodtechnol/windows2000serv/maintain/featusability/inmnwp.mspx`

[69] *Basics of the Debian package management system*. The Debian GNU/Linux FAQ (September 14, 2005). Maintained by Javier Fernandez-Sanguino.

Available at: `http://www.debian.org/doc/FAQ/ch-pkg\_basics`

[70] *Vulnerability in Graphics Rendering Engine Could Allow Remote Code Execution (912919)*. Microsoft Security Bulletin (January 5, 2006).

Available at: `http://www.microsoft.com/technet/security/bulletin/ms06-001.mspx`

[71] ANELKAOS. *Gmail Bug*. Vulnerability Advisory (October 2005).

Available at: `http://www.elhacker.net/gmailbug/english\_version.htm`

[72] *All Vulnerabilities discovered through ChangeLog entries*. Open Source Vulnerability Database.

Available at: `http://www.osvdb.org/searchdb.php?text=ChangeLog`

[73] *Genuine Microsoft Software*. Vendor Website.

Available at: `http://www.microsoft.com/genuine/default.mspx?displaylang=en`

[74] *OracleMetaLink*. Vendor Website.

Available at: `https://metalink.oracle.com/`

[75] *SunSolve Online*. Vendor Website.

Available at: `http://sunsolve.sun.com/`

[76] *Buffer Overruns in SQL Server 2000 Resolution Service Could Enable Code Execution (Q323875)*. Microsoft Security Bulletin (July 24, 2002).

Available at: `http://www.microsoft.com/technet/security/bulletin/ms02-039.mspx`

[77] Moore, David; Paxson, Vern; Savage, Stefan; Shannon, Colleen; Staniford, Stuart and Weaver, Nicholas. *The spread of the Sapphire/Slammer worm*. *Technical report*, The Cooperative Association for Internet Data Analysis (CAIDA) (February 2003).

Available at: `http://www.caida.org/outreach/papers/2003/sapphire/sapphire.html`

[78] *Cumulative Patch for SQL Server (Q316333)*. Microsoft Security Bulletin (August 14, 2002).

Available at: `http://www.microsoft.com/technet/security/bulletin/ms02-043.mspx`

[79] *Cumulative Patch for SQL Server (Q316333)*. Microsoft Security Bulletin (October 2, 2002).

Available at: `http://www.microsoft.com/technet/security/bulletin/ms02-056.mspx`

[80] *Elevation of Privilege in SQL Server Web Tasks (Q316333)*. Microsoft Security Bulletin (October 16, 2002).

Available at: `http://www.microsoft.com/technet/security/bulletin/ms02-061.mspx`

[81] *FIX: Handle Leak Occurs in SQL Server When Service or Application Repeatedly Connects and Disconnects with Shared Memory Network Library*. Microsoft Security Bulletin (October 30, 2005).

Available at: `http://support.microsoft.com/default.aspx?scid=kb;en-us;317748`

[82] Cooper, Russ. *Confusion about versions*. NTBugTraq Mailinglist (January 28, 2003).

Available at: `http://archives.neohapsis.com/archives/ntbugtraq/2003-q1/0045.html`

[83] Thurrott, Paul. *Microsoft Releases SQL Server 2000 SP3*. WindowsITPro News (January 23, 2003).

Available at: `http://www.windowsitpro.com/Article/ArticleID/37800/37800.html`

[84] *Compatibility and Resource Guide*. *Technical report*, Best Software (July 7, 2004).

Available at: `http://www.blytheco.com/pdf/bes/misc/MAS500CompatibilityGuide63.doc`

[85] Roberts, Paul. *Microsoft Slammed by Its Own Vulnerability*. IDG News Service (January 28, 2003).

Available at: `http://www.pcworld.com/news/article/0,aid,109043,00.asp`

[86] *Buffer Overrun in JPEG Processing (GDI+) Could Allow Code Execution (833987)*. Microsoft Security Bulletin (September 14, 2004).

Available at: `http://www.microsoft.com/technet/security/bulletin/ms04-028.mspx`

[87] Hyppönen, Mikko. *Be careful with WMF files*. F-Secure Anti-Virus Weblog (December 28, 2005).

Available at: `http://www.f-secure.com/weblog/archives/archive-122005.html\#00000753`

[88] *All About GDI+*. *Technical report*, Microsoft.

Available at: `http://msdn.microsoft.com/security/gdiplus/default.aspx`

[89] Liston, Tom. *GDI Scan*. Internet Storm Centre (October 2, 2004).

Available at: `http://isc.sans.org/gdiscan.php`

[90] Rescorla, Eric. *Is Finding Security Holes a Good Idea?* In *IEEE Security and Privacy*, volume 3, no. 1: pages 14–19 (2005). ISSN 1540-7993. doi:http://dx.doi.org/10.1109/MSP.2005.17.

[91] Chan, Jason. *Essentials of Patch Management Policy and Practice*. @stake (January 2004).

Available at: `http://www.patchmanagement.org/pmessentials.asp`

[92] British Standard/International Standard Organization. *ISO/IEC 17799 Information technology – Code of practice for information security management* (2000).

[93] MacLeod, Kenneth J. *Patch Management and the Need for Metrics*. In (July 14, 2004). SANS Security Essentials GSEC Practical Assignment.

Available at: `http://www.sans.org/rr/whitepapers/bestprac/1461.php`

[94] Voldal, Daniel. *A Practical Methodology for Implementing a Patch management Process*. In (September 26, 2003).

Available at: `http://www.sans.org/rr/whitepapers/bestprac/1206.php`

[95] White, Dominic and Irwin, Barry. *A Unified Architecture for Automatic Software Updates*. In *Proceedings of Information Security South Africa 2004* (June 2004).

Available at: `http://www.cs.ru.ac.za/research/students/g00w1690/files/issa2004.pdf`

[96] Swanson, Marianne. *Guide for Developing Security Plans for Information Technology Systems*. *Technical report*, National Institute of Standards (NIST), Computer Security

Division, Information Technology Laboratory, National Institute of Standards and Technology, Gaithersburg, MD 20899-8930 (December 1998). Special Publication 800-18, Federal Computer Security Program Managers' Forum Working Group.

Available at: `http://csrc.nist.gov/publications/nistpubs/800-18/Planguide.PDF`

[97] Carothers, Tony. *Port 1025/6000 Action (Part III)*. Internet Storm Centre Handler's Diary (December 11, 2005).

Available at: `http://isc.sans.org/diary.php?storyid=926`

[98] Kohen, Javier and Rizzo, Juliano. *DCE RPC Vulnerabilities New Attack Vectors Analysis*. *Technical report*, Core Security Technologies (December 9, 2003).

Available at: `http://www.coresecurity.com/common/showdoc.php?idx=393\&idxseccion=10`

[99] *Vulnerabilities in MSDTC and COM+ Could Allow Remote Code Execution (902400)*. Microsoft Security Bulletin (October 11, 2005).

Available at: `http://www.microsoft.com/technet/security/Bulletin/MS05-051.mspx`

[100] Gregg, Michael. *CISSP Exam Cram 2*. Que (September 22, 2005). ISBN 078973446X.

[101] Office of Information and Communications Technology. *Information Security Guideline for NSW Government Part 1 Information Security Risk Management*. *Technical report*, New South Wales Department of Commerce (June 2003).

Available at: `http://www.oit.nsw.gov.au/pdf/4.4.16.IS1.pdf`

[102] Bradley, Tony. *Critical Elements For Patch Testing Policies*. In (June 17, 2005). Vol. 27, Issue 24.

Available at: `http://www.processor.com/editorial/article.asp?article=articles%2Fp2724%2F22p24%2F22p24%2Easp\&guid=8BF8F1B9C3044EDDB8172AF340C1667C\&searchtype=0\&WordList=`

[103] VMWare. *VMWare Virtualisation Software*. Vendor Website (June 26, 2006).

Available at: `http://www.vmware.com/`

[104] Pratt, Ian. *The Xen virtual machine monitor*. Project Website (April 13, 2006).

Available at: `http://www.cl.cam.ac.uk/Research/SRG/netos/xen/`

[105] Microsoft. *Microsoft Virtual PC 2004*. Vendor Website (June 26, 2006).

Available at: `http://www.microsoft.com/windows/virtualpc/default.mspx`

[106] Microsoft. *Microsoft Virtual Server 2005 R2*. Vendor Website (June 26, 2006).

Available at: `http://www.microsoft.com/windowsserversystem/virtualserver/default.mspx`

[107] Shaw, Yun. *Patch Management in Oracle Applications Release 11i*. *Technical report*, Oracle Corporation (May 2005).

Available at: http://whitepapers.zdnet.co.uk/0,39025945,60143559p-39000388q,00.htm

[108] *Windows Update Services Deployment White Paper*. *Technical report*, Microsoft (Novemberember 2004).

Available at: http://www.microsoft.com/windowsserversystem/wus/deployment.mspx

[109] Thompson, Ken. *Reflections on trusting trust*. In *Communications of the ACM*, volume 27, no. 8 (August 1984).

[110] Dunagan, John; Roussev, Roussi; Daniels, Brad; Johson, Aaron; Verbowski, Chad and Wang, Yi-Min. *Towards a Self-Managing Software Patching Process Using Black-Box Persistent-State Manifests*. In *Proceedings of IEEE International Conference on Autonomic Computing (ICAC)*. Institute of Electrical and Electronics Engineers, Inc. (March 2004).

Available at: http://research.microsoft.com/research/pubs/view.aspx?tr_id=726

[111] Sun, Yizhan and Couch, Alva. *Global Impact Analysis of Dynamic Library Dependencies*. In *Proceedings of the 2001 Large Installation System Administration Conference(LISA01) (USENIX Association: Berkeley, CA)*, page 145 (December 3, 2001).

Available at: http://www.usenix.org/publications/library/proceedings/lisa2001/tech/sun.html

[112] White, Dominic and Irwin, Barry. *Patching for Low Bandwidth Communities*. In *Proceedings of Southern African Telecommunication Networks & Applications Conference* (September 11, 2005).

Available at: http://www.cs.ru.ac.za/research/students/g00w1690/files/satnac2005/patchingbandwidth.pdf

[113] *Cumulative Security Update for Internet Explorer (896727)*. Microsoft Security Bulletin (August 9, 2005).

Available at: http://www.microsoft.com/technet/security/Bulletin/MS05-038.mspx

[114] Keizer, Gregg. *Microsoft Initially Released Corrupted IE Patch*. TechWeb News (August 10, 2005).

Available at: http://techweb.com/wire/security/168600527

[115] *Vulnerability disclosure publications and discussion tracking*. University of Oulu, Electrical and Information Engineering Department (May 10, 2005).

Available at: `http://www.ee.oulu.fi/research/ouspg/sage/disclosure-tracking/`

[116] McMillan, Robert. *Adobe Adopts Monthly Patch Cycle*. IDG News Service (December 15, 2005).

Available at: `http://www.theregister.co.uk/2005/12/15/adobe\_monthly\_patch\_plan/`

[117] Emigh, Jacqueline. *Users Weigh In on Oracle's Patch Plan*. eWeek.com News (August 23, 2004).

Available at: `http://www.eweek.com/article2/0,1895,1638797,00.asp`

[118] Livingston, Brian. *Microsoft's Patch-A-Month Club*. eWeek.com News (November 3, 2003).

Available at: `http://www.eweek.com/article2/0,1895,1490665,00.asp`

[119] Bott, Ed. *Patches: Once a month is not enough*. Ed Bott's Microsoft Report Blog (March 24, 2006).

Available at: `http://blogs.zdnet.com/Bott/?p=23`

[120] Eschelbeck, Gerhard. *The Laws of Vulnerabilities*. In *Black Hat Briefings* (edited by Jeff Moss). Black Hat, Inc, 2606 Second Avenue, 406, Seattle, WA 98121 USA (March 2004).

Available at: `http://www.blackhat.com/presentations/bh-asia-04/bh-jp-04-pdfs/bh-jp-04-eschelbeck.pdf`

[121] Lemos, Robert. *Microsoft releases monthly security fixes*. CNET News.com (October 15, 2003).

Available at: `http://news.com.com/Microsoft+releases+monthly+security+fixes/2100-7355\_3-5091835.html`

[122] Pruitt, Scarlet. *Oracle moves to monthly patching schedule*. IDG News Service (August 20, 2004).

Available at: `http://www.computerworld.com/securitytopics/security/story/0,10801,95388,00.html`

[123] Evers, Joris. *Oracle to deliver security patches on quarterly basis*. IDG News Service (November 18, 2004).

Available at: `http://www.infoworld.com/article/04/11/18/HNoraclepatchquarterly\_1.html`

[124] Litchfield, David. *Opinion: Complete failure of Oracle security response and utter neglect of their responsibility to their customers*. BugTraq Mailing list (January 6, 2005).
Available at: `http://seclists.org/lists/bugtraq/2005/Oct/0056.html`

[125] Mogull, Rich. *Flaws Show Need to Update Oracle Product Management Practices. Technical report*, Gartner (January 23, 2006).
Available at: `http://www.gartner.com/DisplayDocument?ref=g\_search\&id=488567`

[126] Fisher, Dennis. *Changing Patch Habits With Microsoft*. eWeek.com News (December 6, 2004).
Available at: `http://www.eweek.com/article2/0,1895,1735542,00.asp`

[127] Farrow, Rik. *The Pros and Cons of Posting Vulnerabilities*. IT Architect (May 10, 2005).
Available at: `http://www.itarchitect.com/shared/article/showArticle.jhtml?articleId=8702916`

[128] Arora, Ashish; Telang, Rahul and Xu, Hao. *Optimal Policy for Software Vulnerability Disclosure*. In *Workshop on Economics and Information Security* (May 2004).
Available at: `http://www.heinz.cmu.edu/~rtelang/disclosure\_finalMS\_IS.pdf`

[129] Arora, Ashish; Krishnan, Ramayya; Nandkumar, Anand; Telang, Rahul and Yang, Yubao. *Impact of Vulnerability Disclosure and Patch Availability - An Empirical Analysis*. In *Third Annual Workshop on Economics and Information Security WEIS04* (April 2004).
Available at: `http://www.heinz.cmu.edu/~rtelang/disclosure\_finalMS\_IS.pdf`

[130] Rauch, Jeremy. *The Future of Vulnerability Disclosure?* In *;login: the USENIX Association Newsletter*, volume 11 (December 8, 1999).
Available at: `http://www.usenix.org/publications/login/1999-11/features/disclosure.html`

[131] Schneier, Bruce. *Cisco Harasses Security Researcher*. CryptoGram Newsletter (July 29, 2005).
Available at: `http://www.schneier.com/blog/archives/2005/07/cisco\_harasses.html`

[132] Rain Forest Puppy. *Full Disclosure Policy (RFPolicy) v2.0*. Unofficial Policy (September 8, 2004).
Available at: `http://www.wiretrip.net/rfp/policy.html`

[133] *OIS Guidelines for Security Vulnerability Reporting and Response, V2.0. Technical report*, Organisation for Internet Safety (September 17, 2004).
Available at: `http://www.oisafety.org/guidelines/secresp.html`

[134] Cooper, Russ. *NTBugtraq Disclosure Policy*. *Technical report*, NTBugTraq (July 26, 1999).

Available at: http://www.ntbugtraq.com/default.aspx?sid=1\&pid=47\&aid=48

[135] *CERT/CC Vulnerability Disclosure Policy*. *Technical report*, CERT/CC (October 9, 2000).

Available at: http://www.cert.org/kb/vul_disclosure.html

[136] Laakso, Marko; Takanen, Ari and Roning, Juha. *Introducing constructive vulnerability disclosures*. In (2001).

Available at: http://www.ee.oulu.fi/research/ouspg/protos/sota/FIRST2001-disclosures/paper.pdf

[137] Arora, Ashish; Krishnan, Ramayya; Telang, Rahul and Yang, Yubao. *An Empirical Analysis of Vendor Response to Disclosure Policy*. In *The Fourth Annual Workshop on Economics and Information Security WEIS05* (March 2004).

Available at: http://infosecon.net/workshop/pdf/41.pdf

[138] *Handling Mozilla Security Bugs*. *Technical report*, Mozilla Foundation (February 11, 2003).

Available at: http://www.mozilla.org/projects/security/security-bugs-policy.html

[139] Kean, Kevin. *Updated Advisory: WMF Vulnerability*. Microsoft Security Response Centre (January 2006).

Available at: http://blogs.technet.com/msrc/archive/2006/01/03/416809.aspx

[140] Ford, Heather. *An open invitation to culture-jamming with Laugh It Off*. Creative Commons South Africa News (March 2005).

Available at: http://za.creativecommons.org/blog/archives/2005/03/18/an-open-invitation-to-culture-jamming-with-laugh-it-off/

[141] Havrilla, Jeffrey S. and Dormann, Will. *Vulnerability Note VU#181038 Microsoft Windows Metafile handler SETABORTPROC GDI Escape vulnerability*. *Technical report*, US-CERT (January 20, 2006).

Available at: http://www.kb.cert.org/vuls/id/181038

[142] Anonymous. *Is this a new exploit?* BugTraq Mailinglist (December 27, 2005).

Available at: http://archives.neohapsis.com/archives/bugtraq/2005-12/0305.html

[143] *Exploit-WMF*. McAfee Virus Information Library (January 5, 2006).

Available at: `http://vil.mcafeesecurity.com/vil/content/vi\_137760.htm`

[144] *Bleeding Snort Current Events WMF Exploit Signature*. Bleeding Snort Current Events CVS Signature Repository (February 7, 2006).

Available at: `http://www.bleedingsnort.com/cgi-bin/viewcvs.cgi/sigs/CURRENT\_EVENTS/CURRENT\_WMF\_Exploit`

[145] Carboni, Chris. *Update on Windows WMF 0-day*. SANS Internet Storm Centre Handler's Diary (December 29, 2005).

Available at: `http://isc.sans.org/diary.php?storyid=975`

[146] Wesemann, Daniel. *The most hated IP address of 2005 ?* SANS Internet Storm Centre Handler's Diary (December 28, 2005).

Available at: `http://isc.sans.org/diary.php?storyid=974`

[147] Serino, Jim. *RE: [Full-disclosure] Someone wasted a nice bug on spyware...* BugTraq Mailinglist (December 28, 2005).

Available at: `http://archives.neohapsis.com/archives/bugtraq/2005-12/0320.html`

[148] Guilfanov, Ilfak. *Windows WMF Metafile Vulnerability HotFix*. Hex Blog (December 31, 2005).

Available at: `http://www.hexblog.com/2005/12/wmf\_vuln.html`

[149] Frantzen, Swa. *New exploit released for the WMF vulnerability*. SANS Internet Storm Centre Handler's Diary (January 1, 2006).

Available at: `http://isc.sans.org/diary.php?storyid=992`

[150] Ullrich, Johannes. *Recommended Block List*. SANS Internet Storm Centre Handler's Diary (January 2, 2006).

Available at: `http://isc.sans.org/diary.php?storyid=997`

[151] Sachs, Marcus. *Installing a Patch Silently*. SANS Internet Storm Centre Handler's Diary (January 2, 2006).

Available at: `http://isc.sans.org/diary.php?storyid=1004`

[152] Sachs, Marcus. *Scripting the Unofficial .wmf Patch*. SANS Internet Storm Centre Handler's Diary (January 2, 2006).

Available at: `http://isc.sans.org/diary.php?storyid=1008`

[153] Sachs, Marcus. *Checking for .wmf vulnerabilities*. SANS Internet Storm Centre Handler's Diary (January 2, 2006).
Available at: `http://isc.sans.org/diary.php?storyid=1006`

[154] Guilfanov, Ilfak. *WMF Vulnerability Checker*. Hex Blog (January 1, 2006).
Available at: `http://www.hexblog.com/2006/01/wmf\_vulnerability\_checker.html`

[155] Frantzen, Swa. *WMF FAQ*. SANS Internet Storm Centre Handler's Diary (January 7, 2006).
Available at: `http://isc.sans.org/diary.php?storyid=994`

[156] Sachs, Marcus. *.wmf FAQ Translations*. SANS Internet Storm Centre Handler's Diary (January 3, 2006).
Available at: `http://isc.sans.org/diary.php?storyid=1005`

[157] Liston, Tom. *Updated version of Ilfak Guilfanov's patch / ,msi file*. SANS Internet Storm Centre Handler's Diary (January 1, 2006).
Available at: `http://isc.sans.org/diary.php?storyid=999`

[158] Hyppönen, Mikko. *Hexblog.com overloaded*. F-Secure Anti-Virus Weblog (January 4, 2006).
Available at: `http://www.f-secure.com/weblog/archives/archive-012006.html\#00000767`

[159] Reavey, Mike. *WMF Vulnerability Security Update*. Microsoft Security Response Centre Blog (January 4, 2006).
Available at: `http://blogs.technet.com/msrc/archive/2006/01/04/416847.aspx`

[160] Nash, Mike. *Mike Nash on the Security Update for the WMF Vulnerability*. Microsoft Security Response Centre Blog (January 5, 2006).
Available at: `http://blogs.technet.com/msrc/archive/2006/01/05/416980.aspx`

[161] Mook, Nate. *US Govt. to Test Windows Patches Early*. BetaNews (March 11, 2005).
Available at: `http://www.betanews.com/article/US\_Govt\_to\_Test\_Windows\_Patches\_Early/1110560071`

[162] Nash, Mike. *Mike Nash on the Security Update for the WMF Vulnerability*. Microsoft Security Response Centre (January 2006).
Available at: `http://blogs.technet.com/msrc/archive/2006/01/05/416980.aspx`

[163] Krebs, Brian. *A Time to Patch*. Washington Post's Security Fix (January 11, 2006).
Available at: http://blogs.washingtonpost.com/securityfix/2006/01/a_timeline_of_m.html

[164] White, Dominic. *Microsoft Patch Speed Inconsistencies*. .tHE pRODUCT Weblog (January 13, 2006).
Available at: http://singe.rucus.net/blog/archives/687-Microsoft-Patch-Speed-Inconsistencies.html

[165] Haugsness, Kyle. *Bofra/IFrame Exploits on More Web Sites (updated); IFRAME vulnerability summary; Two more IE Exploits*. SANS Internet Storm Center Handler's Diary (November 20, 2004).
Available at: http://isc.sans.org/diary.php?date=2004-11-20

[166] *Update for Microsoft Internet Explorer HTML Elements Vulnerability*. *Technical report*, US-CERT (December 3, 2004).
Available at: http://www.us-cert.gov/cas/techalerts/TA04-336A.html

[167] Frantzen, Swa. *Black tuesday - the day after*. SANS Internet Storm Center Handler's Diary (December 14, 2005).
Available at: http://isc.sans.org/diary.php?storyid=932

[168] *Upcoming Advisories*. eEye Digital Security (January 2006).
Available at: http://www.eeye.com/html/research/upcoming/

[169] de Beaupre, Adrien. *Handler's Diary*. SANS Internet Storm Centre Handler's Diary (July 12, 2005).
Available at: http://isc.sans.org/diary.php?date=2005-07-12

[170] Toulouse, Stephen. *Microsoft presenting at the Black Hat security conference in Las Vegas*. Microsoft Security Response Centre Blog (June 9, 2006).
Available at: http://blogs.technet.com/msrc/archive/2006/06/09/434600.aspx

[171] Microsoft. *Microsoft BlueHat Security Briefings*. TechNet Security (March 8, 2006).
Available at: http://www.microsoft.com/technet/security/bluehat/sessions/default.mspx

[172] Vaas, Lisa. *Oracle's Silence on Database Security Wearing Thin*. eWeek.com News (August 17, 2004).
Available at: http://www.eweek.com/article2/0,1895,1637079,00.asp

[173] Vaas, Lisa. *Security Firm: Oracle Opatch Leaves Firms Uncovered*. eWeek.com News (August 22, 2005).

Available at: `http://www.eweek.com/article2/0,1895,1850287,00.asp`

[174] *Mozilla Foundation Awards Bug Bounties*. Mozilla Foundation News (March 28, 2005).

Available at: `http://www.mozilla.org/press/mozilla-2005-03-28.html`

[175] *RPM Guide*. The Fedora Project (November 11, 2005).

Available at: `http://fedora.redhat.com/docs/drafts/rpm-guide-en/ch-intro-rpm.html`

[176] Debian Documentation Team. *A Brief History of Debian*. Debian Foundation (August 10, 2005).

Available at: `http://www.debian.org/doc/manuals/project-history/ch-releases.en.html`

[177] Bartoletti, Tony; Dobbs, Lauri A. and Kelley, Marcey. *Secure Software Distribution System*. *Technical report*, Computer Security Technology Center, Lawrence Livermore National Laboratory, PO Box 808 L-303 Livermore, CA 94551 (June 30, 1997).

Available at: `http://ciac.llnl.gov/cstc/ssds/ssdswp.pdf`

[178] Trusted Strategies. *Patch Management Sector Report*. *Technical report*, Trusted Strategies (May 2004).

Available at: `http://www.trustedstrategies.com/nl1/rnr.php`

[179] Brynjolfsson, Erik and Hitt, Lorin. *Computing Productivity: Firm-Level Evidence*. In *MIT Sloan Working Paper No 4210-01* (June 2003). doi:http://dx.doi.org/10.2139/ssrn.290325.

Available at: `http://ssrn.com/abstract=290325`

[180] *Patch Management Product Comparisons*. PatchManagement.org (November 1, 2004).

Available at: `http://www.patchmanagement.org/comparisons.asp`

[181] Landesman, Mary. *Security Patch Management: Breaking New Ground*. *Technical report*, Shavlik (2004).

Available at: `http://www.shavlik.com/whitepapers/security\_patch\_management.pdf`

[182] Nicolett, Mark and Colville, Ronni. *Robust Patch Management Requires Specific Capabilities*. Research Note T-19-4570 (March, 2003).

[183] Furrow, Chris and Manzuik, Steve. *Injecting Trojans via Patch Management Software and Other Evil Deeds*. In *Black Hat Europe*. Black Hat, Inc, 2606 Second Avenue, 406, Seattle, WA 98121 USA (August, 2005).

Available at: `http://www.blackhat.com/presentations/bh-europe-05/bh-eu-05-farrow.pdf`

[184] *NVD Download and Product Integration Page*. National Vulnerability Database (January 2006).

Available at: `http://nvd.nist.gov/download.cfm`

[185] *X-Force: Alerts and Advisories*. Internet Security Systems (January 2006).

Available at: `http://xforce.iss.net/xforce/alerts`

[186] *SecurityFocus: Vulnerabilities*. SecurityFocus Vulnerability Database (January 2006).

Available at: `http://www.securityfocus.com/vulnerabilities`

[187] *New Security Information*. Microsoft Website (January 24, 2006).

Available at: `http://www.microsoft.com/athome/security/rss/default.mspx`

[188] Debian Foundation. *Security Information*. Debian Security Team Website (February 13, 2006).

Available at: `http://www.debian.org/security/`

[189] *FreeBSD VuXML*. Website (February 7, 2006).

Available at: `http://www.vuxml.org/freebsd/`

[190] *SGUIL(tm) The Analyst Console for Network Security Monitoring*. Website (2006).

Available at: `http://sguil.sourceforge.net/`

[191] *DeepSight(tm) Analyser*. Symantec Website (February 13, 2006).

Available at: `http://analyzer.securityfocus.com/`

[192] *Open Vulnerability and Assesment Language OVAL* (May 19, 2006).

Available at: `http://oval.mitre.org/`

[193] *Open Vulnerability and Assesment Language OVAL - XML Schema* (June 9, 2006).

Available at: `http://oval.mitre.org/language/index.html`

[194] McDonald, Josh (2005).

Available at: `http://xdelta.blogspot.com/`

[195] Percival, Colin. *Naive differences of executable code* (September 2003).
Available at: http://www.daemonology.net/bsdiff/

[196] Microsoft. *Binary Delta Compression. Technical report* (March, 2004).
Available at: http://www.microsoft.com/downloads/details.aspx?FamilyID=4789196c-d60a-497c-ae89-101a3754bad6

[197] Brennen, V. Alex. *Strong Distribution HOWTO.* Online HOWTO (April 1, 2003).
Available at: http://www.cryptnet.net/fdp/crypto/strong_distro.html

[198] Sohn, Tae-Shik; Moon, Jong-Sub; Lee, Cheol-Won; Im, Eul-Gyu and Seo, Jung-Taek. *Safe Patch Distribution Architecture in Intranet Environments.* In *Security and Management*, pages 455–460 (2003).

[199] *Vulnerabilities in Operating System Patch Distribution. Technical report*, BindView, No Longer Available.
Available at: http://razor.bindview.com/publish/papers/os-patch.html

[200] Cohen, Bram. *BitTorrent.* Vendor Website (2004).
Available at: http://bitconjurer.org/BitTorrent/

[201] Cohen, Bram. *Incentives Build Robustness in Bittorrent. Technical report* (may 2003).
Available at: http://bitconjurer.org/BitTorrent/bittorrentecon.pdf

[202] Microsoft Corporation. *Microsoft Baseline Security Analyser (MBSA) version 1.2.1 is available.* Microsoft Website (July 7, 2005).
Available at: http://support.microsoft.com/default.aspx?kbid=320454

[203] GFI. *GFI Languard: Security scanning and patch management.* Vendor Website (June 26, 2006).
Available at: http://www.gfi.com/languard/

[204] Microsoft. *Microsoft Systems Management Serve.* Vendor Website (June 26, 2006).
Available at: http://www.microsoft.com/smserver/

[205] IBM. *Tivoli Software.* Vendor Website (June 26, 2006).
Available at: http://www.ibm.com/software/tivoli/

[206] Configuresoft. *Configuresoft: Configuration Management & Compliance*. Vendor Website (June 26, 2006).

Available at: http://www.configuresoft.com/

[207] *Microsoft Windows Update Service Home* (2005).

Available at: http://www.microsoft.com/wsus/

[208] Patchlink. *Patchlink Update: #1 Patch Management Software for Securing the Enterprise*. Vendor Website (June 26, 2006).

Available at: http://www.patchlink.com/

[209] BigFix, Inc. *BigFix Inc. Vulnerability Management*. Vendor Website (June 26, 2006).

Available at: http://www.bigfix.com/

[210] Ecora. *SECURITY COMPLIANCE & CONTROL MADE EASY*. Vendor Website (June 26, 2006).

Available at: http://www.ecora.com/ecora/

[211] FreeBSD. *About FreeBSD Ports*. Project Website (June 26, 2006).

Available at: http://www.freebsd.org/ports/

[212] *Failures in Detection (Last 7 Days)*. VirusTotal Website (February 13, 2006).

Available at: http://www.virustotal.com/flash/graficas/grafica4_en.html

[213] White, SR. *Open Problems in Computer Virus Research. Technical report* (1998).

Available at: http://www.research.ibm.com/antivirus/SciPapers/White/Problems/Problems.html

[214] Software, Marshal. *WebMarshal Product Information*. Vendor Website (June 26, 2006).

Available at: http://www.marshalsoftware.com/pages/webmarshal.asp

[215] Patton, S; Yurcik, W and Doss, D. *An Achilles Heel in Signature-Based IDS: Squealing False Positives in SNORT*. In *Proceedings of RAID 2001* (2001).

Available at: http://mel.icious.net/ids/raid01.pdf

[216] *Malware Prevention through black-hole DNS*. Bleeding Snort Projects (March 3, 2005).

Available at: http://www.bleedingsnort.com/blackhole-dns/

[217] *PatchPoint(tm) System*. BlueLane Website (February 2006).

Available at: http://www.bluelane.com/

[218] Sachs, Marcus. *More .wmf Woes*. SANS Internet Storm Centre Handler's Diary (January 2, 2006).

Available at: `http://isc.sans.org/diary.php?storyid=1002`

[219] Ptacek, Thomas. *Thomas Ptacek's Second Rule Of Security Marketing*. Matasano Security Weblog (November 21, 2005).

Available at: `http://www.sockpuppet.org/tqbf/log/2005/11/thomas-ptaceks-second-rule-of-security.html`

[220] *CERT/CC Common Vulnerabilities and Exposures*. Website (jun 2005).

Available at: `http://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=linux+kernel`

[221] Kojm, Tomasz. *ClamAV: Project News*. Project Website (June 26, 2006).

Available at: `http://www.clamav.net/`

[222] Foley, Mary Jo. *Microsoft Delays By a Year Delivery of Two New Patching Systems* (July 2004).

Available at: `http://www.microsoft-watch.com/article2/0,1995,1656785,00.asp`

[223] *Microsoft Windows Update* (2005).

Available at: `http://www.windowsupdate.com/`

[224] *Windows Update Services Deployment White Paper*. *Technical report*, Microsoft (November 2004).

Available at: `http://www.microsoft.com/windowsserversystem/wus/deployment.mspx`

[225] Zinman, Amit. *Windows Update Services Review* (November 2004).

Available at: `http://www.windowsecurity.com/articles/Windows-Update-Services-Review.html`

[226] *Microsoft .NET Framework Version 1.1 Redistributable Package* (March 2004).

Available at: `http://go.microsoft.com/fwlink/?LinkId=9104`

[227] *Microsoft .NET Framework 1.1 Service Pack 1 for Windows Server 2003* (August 2004).

Available at: `http://go.microsoft.com/fwlink/?LinkId=35326`

[228] *Microsoft Windows Update Services BITS 2.0 beta for Windows 2000 Server* (November 2004).

Available at: `http://www.microsoft.com/windowsserversystem/wus/betaeulaWin2k.mspx`

[229] *Microsoft Windows Update Services BITS 2.0 beta for Windows Server 2003* (November 2004).

Available at: `http://www.microsoft.com/windowsserversystem/wus/betaeulaWin2003.mspx`

[230] *Download Internet Explorer 6 Service Pack 1* (September 2002).

Available at: `http://go.microsoft.com/fwlink/?LinkId=22355`

[231] *Microsoft SQL Server 2000 Desktop Engine (MSDE 2000) Release A* (December 2004).

Available at: `http://go.microsoft.com/fwlink/?LinkId=35713`

[232] *Automatic Updates June 2002* (June 2002).

Available at: `http://go.microsoft.com/fwlink/?LinkId=22338`

[233] *Software Update Services Deployment White Paper*. *Technical report*, Microsoft.

Available at: `http://www.microsoft.com/windowsserversystem/sus/deployment.mspx`

[234] Semilof, Margie. *Microsoft taking steps to integrate WUS with Windows* (March 2004).

Available at: `http://searchwin2000.techtarget.com/qna/0,289202,sid1\_gci956193,00.html`

[235] Thurrott, Paul. *What You Need to Know About Windows Update Services* (April 2004).

Available at: `http://www.windowsitpro.com/Windows/Article/ArticleID/41969/41969.html`

[236] Microsoft. *Description of the new features in the package installer for Windows software updates. KB 832475* (March 2005).

[237] Hoover, Ken. *Ken's SUS Scripts* (June 2004).

Available at: `http://pantheon.yale.edu/~kjh27/sus-scripts.html`

[238] White, Dominic. *SUS Reporting Tools* (December 2004).

Available at: `http://singe.rucus.net/sus/`

Appendix A

Time-line of Notable Worms and Viruses

A.1 Introduction

While researching this document, much work was put into analysing past virii and worms. This resulted in the formation of a Wikipedia article which has since been added to. The below is a time-line of notable worms and viruses. This serves to place the oft-discussed incidents into a greater context.

A.2 Time-line

A.2.1 2006

- January 20th: The Nyxem worm is discovered. Spread by mass-mailing, its payload, which activates on the 3rd of every month, starting on February 3, attempts to disable security-related and file sharing software and destroy files of certain types, such as Microsoft Office.

A.2.2 2005

- August 16th: The Zotob Worm and several variations of malware exploiting the vulnerability described in MS05-039 are discovered. The effect is overblown because several United States media outlets are infected.

A.2.3 2004

- December 2004: Santy, the first known "webworm" is launched. It exploits a vulnerability in PhpBB described in BID10701, and uses Google in order to find new targets. It infects around 40000 sites before Google filters the search query used by the worm, preventing it from spreading.
- May 1st: The Sasser worm emerges by exploiting a vulnerability in LSASS described in MS04-011. It causes problems in networks, even interrupting business in some cases.
- March 19th: The Witty worm, record-breaking in many regards. It exploits holes in several Internet Security Systems (ISS) products. It is the fastest disclosure to worm, the first internet worm to carry a destructive payload, and spread rapidly using a pre-populated list of ground-zero hosts.
- Late January: MyDoom emerges, and still holds the record for the fastest-spreading mass mailer worm.

A.2.4 2003

- October 24th: The Sober worm is first seen and maintains its presence until 2005 with many new variants.

The simultaneous attack of the Blaster and Sobig worms causes a massive amount of damage.

- August 19th: The Sobig worm (technically, the Sobig.F worm) spreads rapidly via mail and network shares.
- August 18th: The Welchia (Nachi) worm is discovered. The worm tries to remove the blaster worm and patch Windows.
- August 12th: The Blaster worm, also know as the Lovesan worm, spreads rapidly by exploiting Microsoft Windows computers vulnerable to exploits first described in MS03-026 and later in MS03-039.
- January 24th: The SQL slammer worm, also known as the Sapphire worm, attacks vulnerabilities in Microsoft SQL Server and MSDE described in MS02-039 and MS02-061, and causes widespread problems on the Internet.

A.2.5 2001

- October 26th: The Klez worm is first identified.
- September 18th: The Nimda worm is discovered and spreads through a variety of means, including vulnerabilities described in MS01-044 and backdoors left by the Code Red II and Sadmind worms.
- August 4th: A complete re-write of the Code Red worm, Code Red II begins aggressively spreading, primarily in China.
- July 13th: The Code Red worm attacking the Index Server ISAPI Extension in Microsoft's Internet Information Services with a vulnerability described in MS01-033, is released.
- July: The Sircam worm is released, spreading through e-mails and unprotected network shares.
- May 8th: The Sadmind worm spreads by exploiting holes in both Sun Microsystem's Solaris (Security Bulletin 00191)and Microsoft's Internet Information Services (MS00-078).
- January: A worm strikingly similar to the Morris worm, named the Ramen worm infected only Red Hat Linux machines running version 6.2 and 7, using three vulnerabilities in wu-ftpd, rpc-statd and lpd respectively.
- May: The VBS/Loveletter worm, also known as the "I love you" virus, appears. As of 2004, this is the most costly virus to business, causing upwards of 10 billion dollars in damage.

A.2.6 1999

- March 26th: The Melissa worm is released, targeting Microsoft Word and Outlook-based systems, and creating considerable network traffic.

A.2.7 1998

- June 2nd: The first version of the CIH virus appears.

A.2.8 1995

- The "Concept virus," the first Macro virus, is created

A.2.9 1992

- Michelangelo predicted to create a digital armageddon on 6th of March, with millions of computers having their information wiped, according to a mass media hysteria surrounding the virus. Later assessments of the damage showed the aftermath to be minimal.

A.2.10 1989

- October 1989: The Ghostball virus is the first multipartite virus and was discovered by Fridrik Skulason.

A.2.11 1988

- November 2nd: The Morris worm, created by Robert Tappan Morris, infects DEC VAX and SUN machines running BSD UNIX connected to the Internet, and becomes the first worm to spread extensively "in the wild", and one of the first well-known programs exploiting buffer overrun vulnerabilities.

A.2.12 1987

- October: The Jerusalem virus is found in the city of Jerusalem, Israel. It is a destructive virus programmed to destroy executable files on every occurrence of Friday the 13th.
- November: The SCA virus, a boot sector virus for Amigas appears, immediately creating a pandemic virus-writer storm. A short time later, SCA releases another, considerably more destructive virus, the Byte Bandit.

A.2.13 1982

- A program called Elk Cloner, written for Apple II systems, is credited with being the first computer virus to appear "in the wild", i.e. outside the single computer or lab where it was created.

Appendix B

Analysis of WSUS

B.1 Introduction

On November 16th 2004, Microsoft announced the availability of the Windows Update Service (WUS) Beta. This date came half a year after Microsoft originally planned to release the service[222]. Then, four months later (March 22nd 2005), Microsoft announced a new release candidate (RC) and a name change, WUS was to be called WSUS (Windows Server Update Services), and on June 8th 2005, Microsoft released WSUS to manufacturing. Given the growth in focus and solutions for patch management over the last year, the delays and changes are hardly surprising. With the window from vulnerability announcement to exploit release rapidly diminishing, patching has become one of the essential tools on the front-line of security. Many organisations have been making do with Microsoft's Software Update Service (SUS) for the last year, with many more still looking for a patch management solution. SUS was seen as a quick-fix due to its limited nature, leaving more than one administrator with handfuls of hair. Microsoft has provided Systems Management Server (SMS) for enterprise patch management and other administration. However, the price is not always appropriate - particularly for small to medium enterprises. It is hoped WSUS can address these issues.

This review will detail the experience with WSUS from installation to use. It will start with an abridged description of WSUS's installation and configuration, and move on to a tour of its abilities. The narrative then takes a turn for the technical, with the basic workings of WSUS explained, based on information gleaned from a live packet capture. The review is concluded with a list of useful WSUS resources.

B.2 What's New

WSUS introduces a number of new features, mostly due to the new back-end that Microsoft has implemented, WSUS uses the same back-end technology as Microsoft's Windows Update service[223]. These new features are:

- Reporting Features. SUS provided no reporting at all, although the back-end implemented it. This lead to administrators having to rely on third-party tools to derive what was going on in their organisation. WSUS provides a host of reporting features, partially fulfilling this much needed customer request.
- More Updates. In addition to operating system updates, WSUS now provides updates for Office XP and 2003, Exchange and SQL server. It still only provides support for Microsoft products and patches however.
- Update Filters. The administrative interface provides useful filter options to navigate the thousands of updates.
- Target Grouping. Machines can now be placed into different groups, allowing different update approval options for each group. This is particularly useful for pushing patches to a test lab before large scale deployment.
- Improved Distribution. WSUS now allows express updates which use binary patching to push only those changes required (called deltas), rather than a whole file. It also supports an improved topology making distributing updates across the organisation easier. Finally, the download on demand feature can ensure that updates are only downloaded when necessary.
- Patch Options. WSUS now allows several new update approval options, which enable it to optionally check if an update is required, install the updaten or remove the update. WSUS also provides for automatic approval rules for specific updates and target groups.
- Improved Update Options. The new version of the BITS (Background Intelligent Transfer Service) includes new options which make installing updates on client machines less intrusive and disruptive.
- New Back-end. WSUS now sports a new back-end finished off with a SQL database. With the option of either SQL server or the free Microsoft Desktop Engine (MSDE).

- Secure Server Replication. Updates and configurations can be replicated between servers. In addition SSL connections can be used in server to server and server to client connections.

B.3 Installation

Installing WSUS is fairly straightforward. Microsoft have provided a good description of the process and its options in their WSUS deploy guide[224], along with a brief guide by WindowsSecurity.com[225]. Thus, this section will be fairly brief.

B.3.1 Topology

Before an installation, an administrator should be aware of the topologies WSUS affords. With the new grouping options and the ability to distribute WSUS servers, several different topologies are possible. There are four basic models which can then be combined to form fairly complex systems if necessary. There are three primary components used: Microsoft Update (MU), the WSUS server, and the WSUS clients, called Automatic Update(AU) clients.

B.3.1.1 Default

This is the 'normal' way of doing things, with the WSUS server receiving its updates and meta-data from Microsoft Update (in a process called synchronisation) and passing it on to its AU clients. The meta-data contains extended information about the update, such as the licensing and description, and can be separated from the actual update. See figure B.1.

B.3.1.2 Grouping

WSUS's new grouping feature allows AU clients to be grouped separately. Each group can then have its own patch approval options. This is useful for testing, allowing patches to be pushed to a test lab before being pushed to the AU clients in a production group. Grouping does not allow for a machine to be part of (often requested) multiple groups. However, this is not a disadvantage, as requiring this is usually indicative of a poor topology, with the additional complexities required to implement it being prohibitive. See figure B.2.

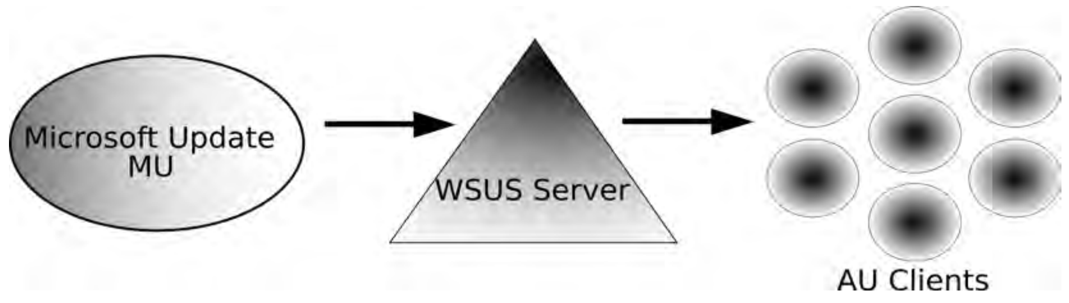


Figure B.1: Default Topology

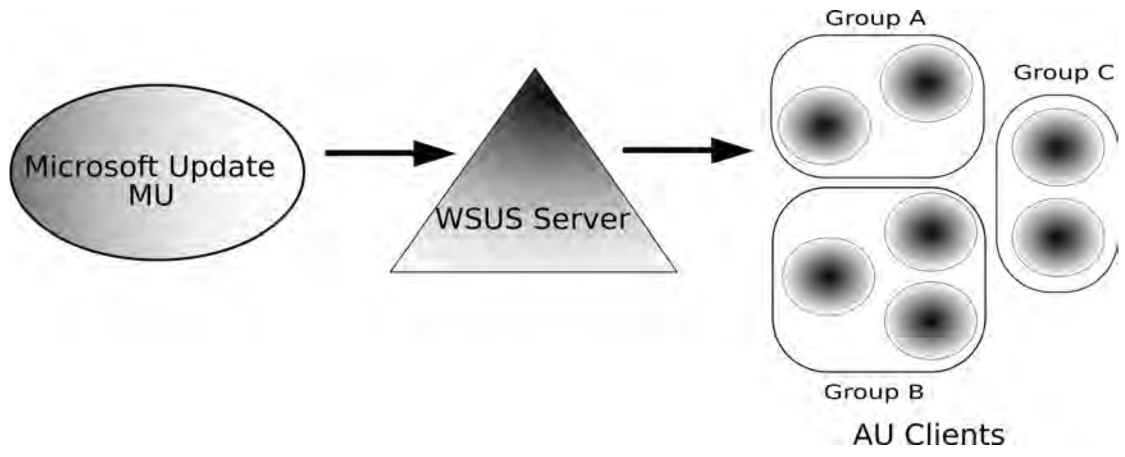


Figure B.2: Grouped Topology

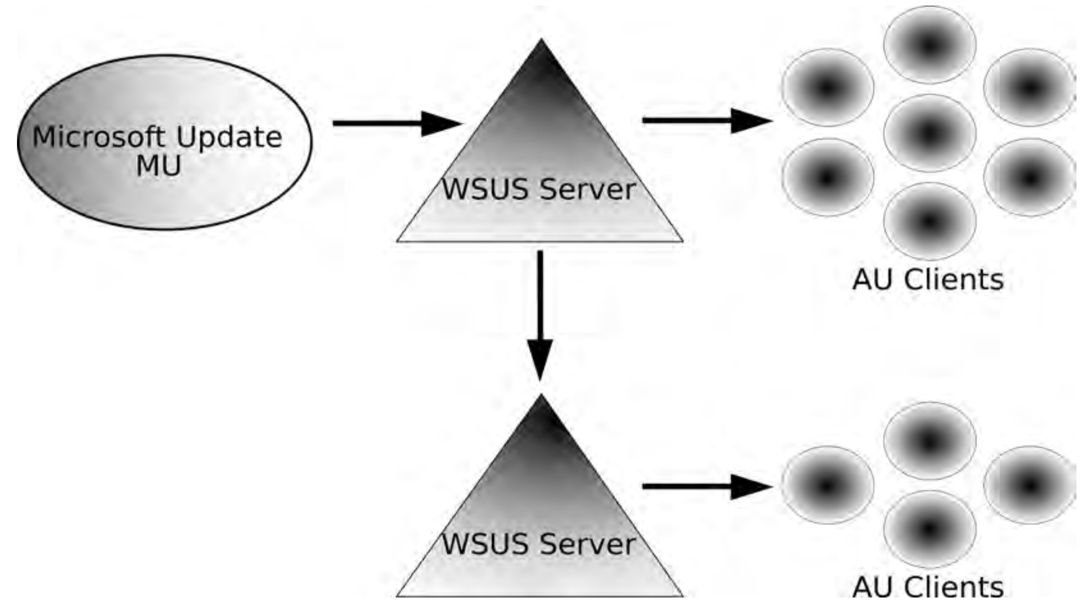


Figure B.3: Chained Topology

B.3.1.3 Chaining

As with SUS, WSUS allows for a WSUS server to synchronise from another WSUS server rather than Microsoft Update. This is useful for creating a distributed hierarchical environment. Microsoft recommends that the hierarchy be no more than three levels deep, though they have tested it with up to five levels[224]. With this model a downstream WSUS server will inherit the approval and transfer setting of the upstream WSUS server. Such a topology can also be used as a disconnected architecture where WSUS's import/export update feature allows for updates to be hand-carried via sneaker net[1] from a connected WSUS server to a disconnected one.See figure B.3.

B.3.1.4 Client Download

It is not always practical to download updates to the WSUS server for distribution. This is particularly true in mobile environments where the AU client's proximity to the WSUS server is unknown. In such situations, the WSUS server can be configured to store only update meta-data.

[1]Manually delivering patches to each machine without a network.

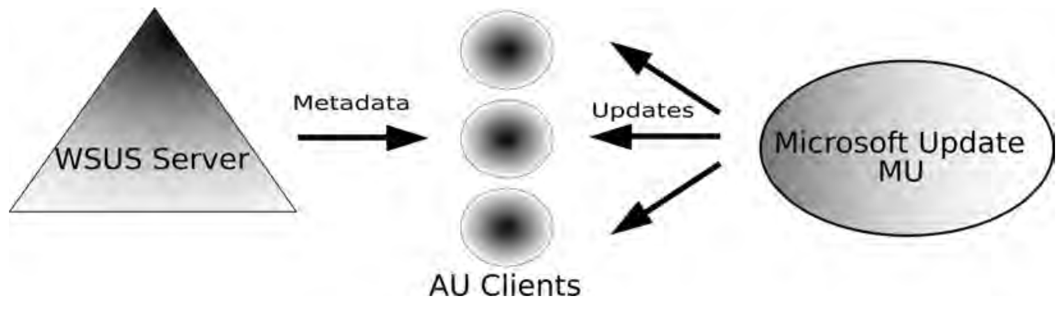


Figure B.4: Client Download Topology

This allows the WSUS server to retain control over update approval without needing to store or distribute the updates themselves. The AU clients can then download the approved updates from Microsoft Update. See figure B.4.

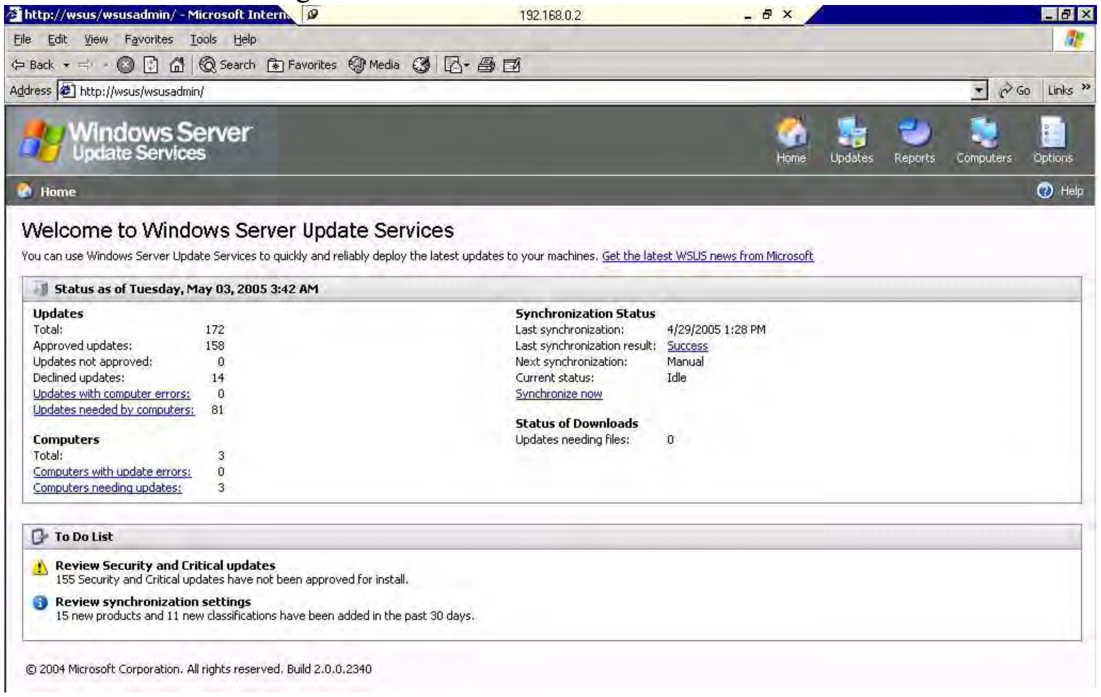
B.3.2 Requirements

WSUS requirements are fairly minimal, and typical of the average server. The requirements are described in more detail in the WSUS Deployment Guide[224].

Microsoft recommends a 1GHz machine for <500 AU clients and a 2GHz machine for >500 AU clients. It should also have at least 1GB of RAM. WSUS requires either Windows Server 2003 or Windows 2000 Server with both requiring the .NET framework ver 1.1 SP1[226, 227], BITS 2.0[228, 229] and IIS 6.0 and Windows 2000 Server requiring IE 6.0 SP 1[230]. Both should have 30GB of an NTFS file system free for updates and 2GB free for MSDE. Microsoft recommends using SQL Server over MSDE with >500 AU clients. MSDE for Windows Server 2003 (now named Windows SQL Server 2000 Desktop Engine or WMSDE) is distributed with the WSUS installer however Windows 2000 Server users will have to download it[231].

The automatic updates client has the same requirements as SUS and will only work on Windows 2000 with Service Pack 3 or later, Windows XP and Windows 2003. WSUS then uses SUS to update the client to work with WSUS. However, this won’t work on Windows XP machines without service packs installed, as it requires the SUS upgrade[232].

Figure B.5: WSUS Administrative Interface


http://wsus/wsusadmin/ - Microsoft Intern
192.168.0.2
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://wsus/wsusadmin/
Go Links
Windows Server Update Services
Home Updates Reports Computers Options
Home
Help

Welcome to Windows Server Update Services

You can use Windows Server Update Services to quickly and reliably deploy the latest updates to your machines. Get the latest WSUS news from Microsoft

Status as of Tuesday, May 03, 2005 3:42 AM

Updates

Total:	172
Approved updates:	158
Updates not approved:	0
Declined updates:	14
Updates with computer errors:	0
Updates needed by computers:	81

Computers

Total:	3
Computers with update errors:	0
Computers needing updates:	3

Synchronization Status

Last synchronization:	4/29/2005 1:28 PM
Last synchronization result:	Success
Next synchronization:	Manual
Current status:	Idle
Synchronize now	

Status of Downloads

Updates needing files:	0

To Do List

Review Security and Critical updates
155 Security and Critical updates have not been approved for install.

Review synchronization settings
15 new products and 11 new classifications have been added in the past 30 days.

© 2004 Microsoft Corporation. All rights reserved. Build 2.0.0.2340

B.3.3 Server

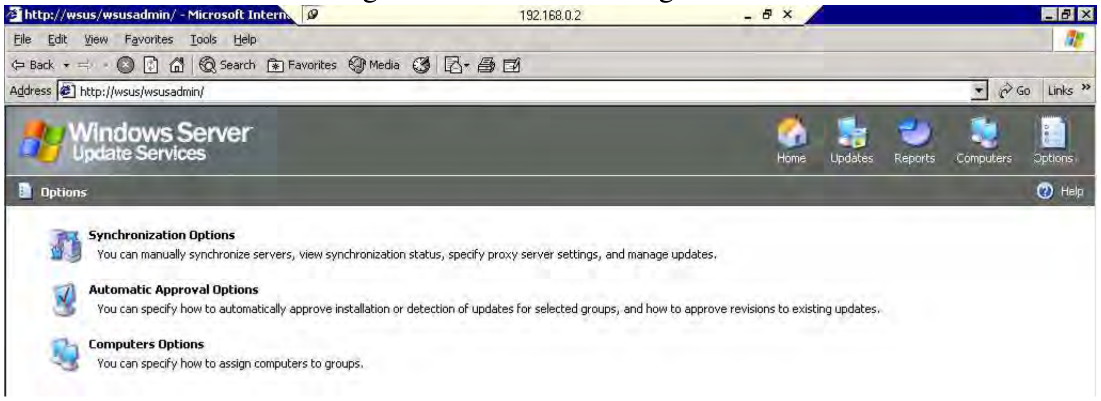
Server installation is facilitated by a wizard, which presents the user with three decisions: whether to store updates locally or have clients fetch them from Microsoft Update, whether to install MSDE or use an existing SQL database, and whether to use the default web site or create a new WSUS site. ASP .NET 1.1 will be installed at the same time. The wizard will also allow for an upstream WSUS server to be configured, instead of connecting to Microsoft Update.

After a successful installation the WSUS administrative interface (see figure B.5) can be found at `http://server[:port]/WSUSAdmin/`, where [port] will only be used if WSUS was not installed to the default site, in which case the port will be 8530.

B.3.4 Client

WSUS uses the automatic update (AU) client's self-update feature to install the new AU client on each machine. The client is first upgraded from the cab files found in \Selfupdate directory of the

Figure B.6: WSUS Configuration


http://wsus/wsusadmin/ - Microsoft Intern
192.168.0.2
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://wsus/wsusadmin/
Go Links
Windows Server Update Services
Home Updates Reports Computers Options
Options
Help

Synchronization Options
You can manually synchronize servers, view synchronization status, specify proxy server settings, and manage updates.

Automatic Approval Options
You can specify how to automatically approve installation or detection of updates for selected groups, and how to approve revisions to existing updates.

Computers Options
You can specify how to assign computers to groups.

web server. Once upgraded, it installs the new Windows Installer 3.1, BITS 2.0 and WinHTTP 5.1 which are needed to support the new configuration options WSUS affords. Windows XP SP2 already has an updated automatic updates client, but will still self update to the latest version. A more technical description of this process can be found later in this document.

B.4 Configuration

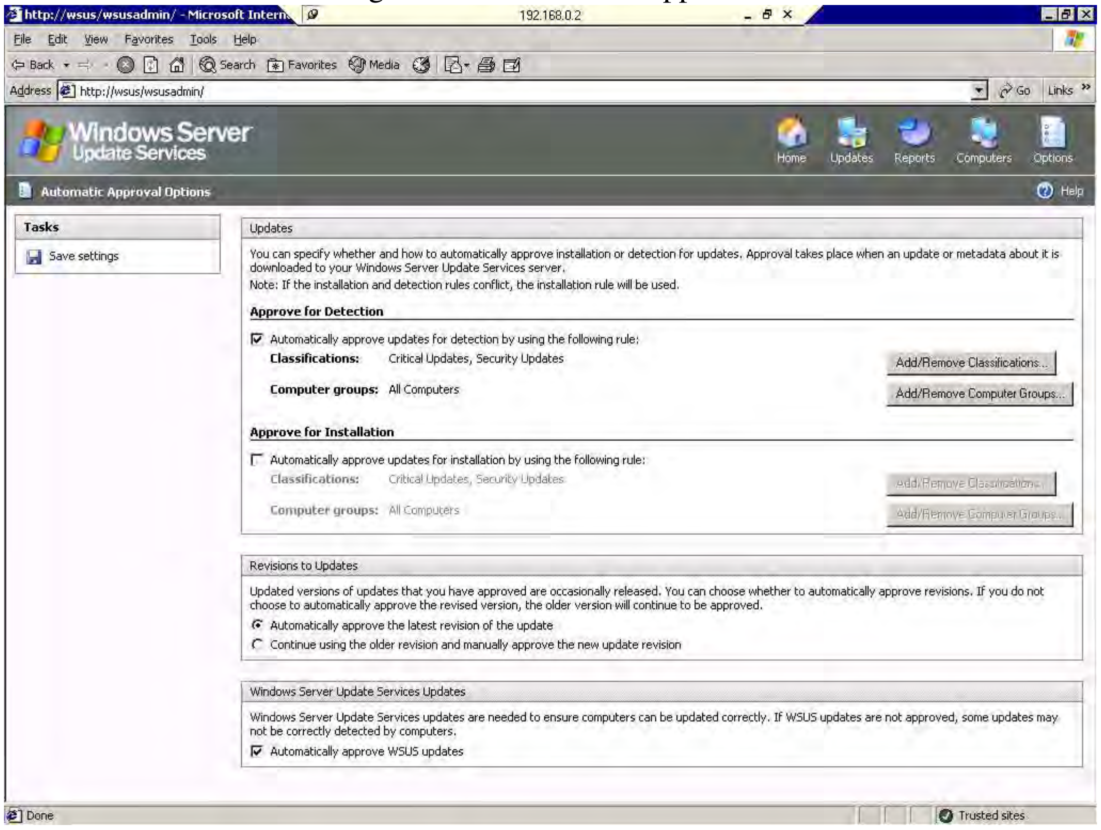
WSUS configuration is similar to SUS configuration. The behaviour of the WSUS server is controlled through the WSUS administrative interface (see figure B.5) while the behaviour of the AU clients is handled through group policy or the registry. This section provides a brief introduction to the various configuration settings available. Once again this is documented in greater detail in the WSUS deploy guide[224], with additional information available in the (currently) more complete SUS deploy guide[233].

B.4.1 Server

Server configuration is done via the WSUS administration page (http://server[:port]/WSUSAdmin/) (see figure B.6). Some options are shared with SUS and will not be covered in detail.

The WSUS server can be configured to synchronise with either Microsoft Update or another WSUS server, as discussed above. This requires information such as the server and proxy details

Figure B.7: Automatic Approval



Automatic Approval Options

Tasks

Save settings

Updates

You can specify whether and how to automatically approve installation or detection for updates. Approval takes place when an update or metadata about it is downloaded to your Windows Server Update Services server.
Note: If the installation and detection rules conflict, the installation rule will be used.

Approve for Detection

☑ Automatically approve updates for detection by using the following rule:
Classifications: Critical Updates, Security Updates
Computer groups: All Computers

Approve for Installation

☐ Automatically approve updates for installation by using the following rule:
Classifications: Critical Updates, Security Updates
Computer groups: All Computers

Revisions to Updates

Updated versions of updates that you have approved are occasionally released. You can choose whether to automatically approve revisions. If you do not choose to automatically approve the revised version, the older version will continue to be approved.
[✓ Automatically approve the latest revision of the update]
Continue using the older revision and manually approve the new update revision

Windows Server Update Services Updates

Windows Server Update Services updates are needed to ensure computers can be updated correctly. If WSUS updates are not approved, some updates may not be correctly detected by computers.
☑ Automatically approve WSUS updates

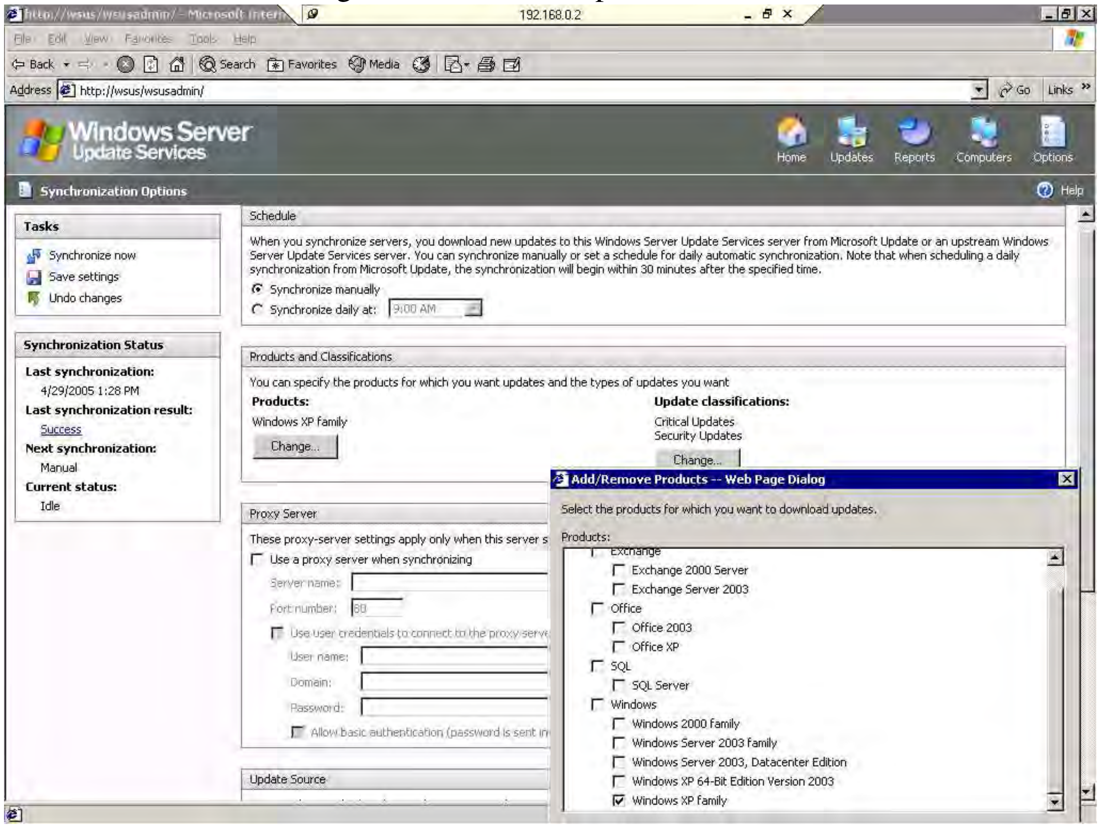
http://wsus/wsusadmin/ - Microsoft Intern
192.168.0.2
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://wsus/wsusadmin/
Go Links
Windows Server Update Services
Home Updates Reports Computers Options
Help
Add/Remove Classifications...
Add/Remove Computer Groups...
Done
Trusted sites

and a schedule for how often the WSUS server should synchronise. The syntax for entering an upstream server is http://servername[:port], with [:port] only used if the WSUS server is not using port 80.

WSUS now supports updates for Office, Exchange and SQL Server, as compared to SUS which had far fewer updates. Microsoft hopes to expand this to all of their products, and are looking into methods for securely distributing third-party updates while maintaining the distribution security of signed updates[234]. This requires that the products, for which WSUS should distribute updates, be selected by adjusting the settings for which products, languages, and class of update e.g. critical updates, security updates, service packs should be managed (see figure B.8). Given the much increased number of updates, there is an option to automatically approve certain classes of updates(see figure B.7).

WSUS provides two methods for grouping computers. The first is server side targeting. This

Figure B.8: Product Update Selection


http://wsus/wsusadmin/ - Microsoft Intern
192.168.0.2
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://wsus/wsusadmin/
Go Links
Windows Server Update Services
Home Updates Reports Computers Options
Synchronization Options
Help

Tasks
- Synchronize now
- Save settings
- Undo changes

Synchronization Status

Last synchronization: 4/29/2005 1:28 PM
Last synchronization result: Success
Next synchronization: Manual
Current status: Idle

Schedule

When you synchronize servers, you download new updates to this Windows Server Update Services server from Microsoft Update or an upstream Windows Server Update Services server. You can synchronize manually or set a schedule for daily automatic synchronization. Note that when scheduling a daily synchronization from Microsoft Update, the synchronization will begin within 30 minutes after the specified time.

[✓ Synchronize manually]
Synchronize daily at: 9:00 AM

Products and Classifications

You can specify the products for which you want updates and the types of updates you want

Products: Windows XP family
Change...

Update classifications: Critical Updates, Security Updates
Change...

Proxy Server

These proxy-server settings apply only when this server s

Use a proxy server when synchronizing
Server name:
Port number: 80
Use user credentials to connect to the proxy serve
User name:
Domain:
Password:
Allow basic authentication (password is sent in

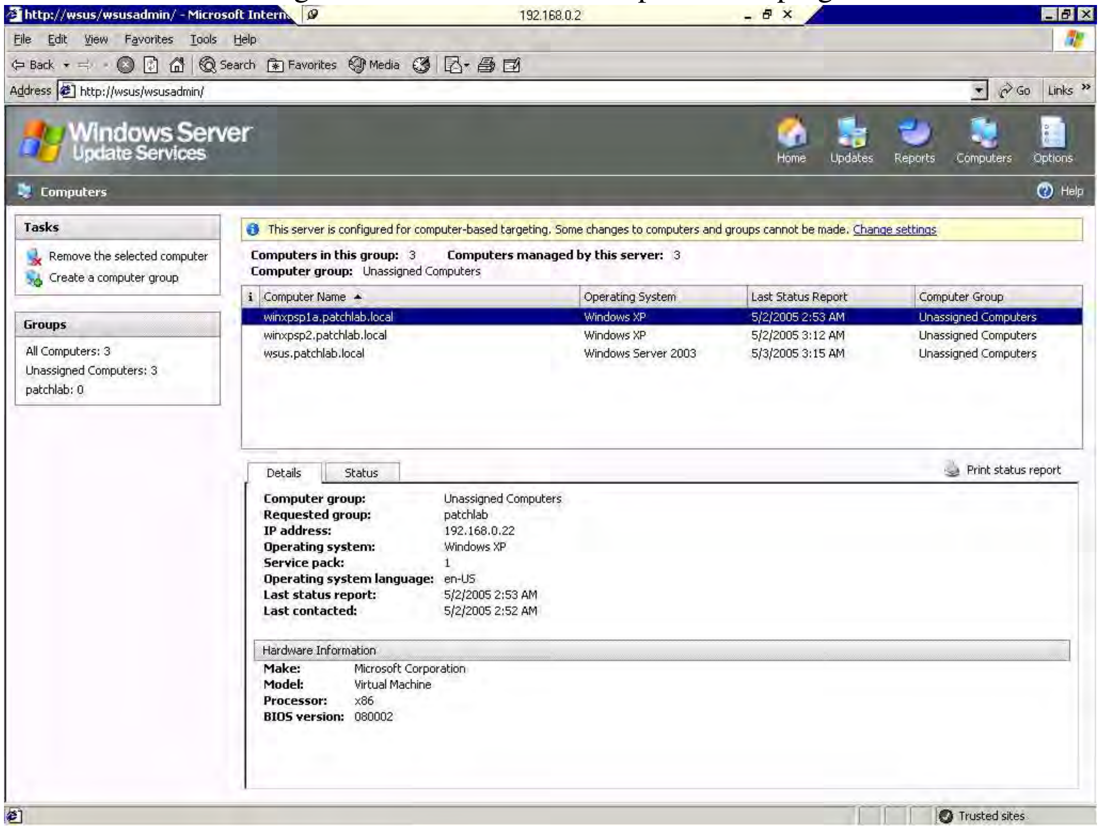
Update Source

Add/Remove Products -- Web Page Dialog

Select the products for which you want to download updates.

Products:
- Exchange
  - Exchange 2000 Server
  - Exchange Server 2003
- Office
  - Office 2003
  - Office XP
- SQL
  - SQL Server
- Windows
  - Windows 2000 family
  - Windows Server 2003 family
  - Windows Server 2003, Datacenter Edition
  - Windows XP 64-Bit Edition Version 2003
  - [✓ Windows XP family]

Figure B.9: Client-Side Computer Grouping


http://wsus/wsusadmin/ - Microsoft Intern
192.168.0.2
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://wsus/wsusadmin/
Go Links
Windows Server Update Services
Home Updates Reports Computers Options
Computers
Help
Tasks
Remove the selected computer
Create a computer group
Groups
Print status report
Details Status
Trusted sites

All Computers: 3
Unassigned Computers: 3
patchlab: 0

This server is configured for computer-based targeting. Some changes to computers and groups cannot be made. Change settings

Computers in this group: 3 **Computers managed by this server:** 3
Computer group: Unassigned Computers

Computer Name	Operating System	Last Status Report	Computer Group
winxpsp1a.patchlab.local	Windows XP	5/2/2005 2:53 AM	Unassigned Computers
winxpsp2.patchlab.local	Windows XP	5/2/2005 3:12 AM	Unassigned Computers
wsus.patchlab.local	Windows Server 2003	5/3/2005 3:15 AM	Unassigned Computers

Computer group:	Unassigned Computers
Requested group:	patchlab
IP address:	192.168.0.22
Operating system:	Windows XP
Service pack:	1
Operating system language:	en-US
Last status report:	5/2/2005 2:53 AM
Last contacted:	5/2/2005 2:52 AM

Hardware Information

Make:	Microsoft Corporation
Model:	Virtual Machine
Processor:	x86
BIOS version:	080002

allows an administrator to manually place machines that have contacted the WSUS server into chosen groups. The second, more powerful, option allows the clients themselves to advertise to be put in a certain group (see figure B.9). This setting is then controlled on the client either through group policy or registry settings. In both cases an administrator needs to create the group on the server.

The new distribution options afforded by WSUS allow for bandwidth consideration to be better accounted for. Deferred updates allow meta-data to be downloaded separately from the update files. This allows approvals to be disseminated, and the update is only downloaded if required by an AU client connected to the WSUS sever (or a downstream WSUS server). Express installation is Microsoft catching up to FreeBSD with binary patching. It allows for deltas to be sent to the AU clients. These deltas only contain information that should be changed within selected files rather than a replacement for the entire file. Express installation does incur a cost, in the form of a large initial download from to the WSUS server, as a delta for each possible version of the files needs to be distributed.

B.4.2 Client Side

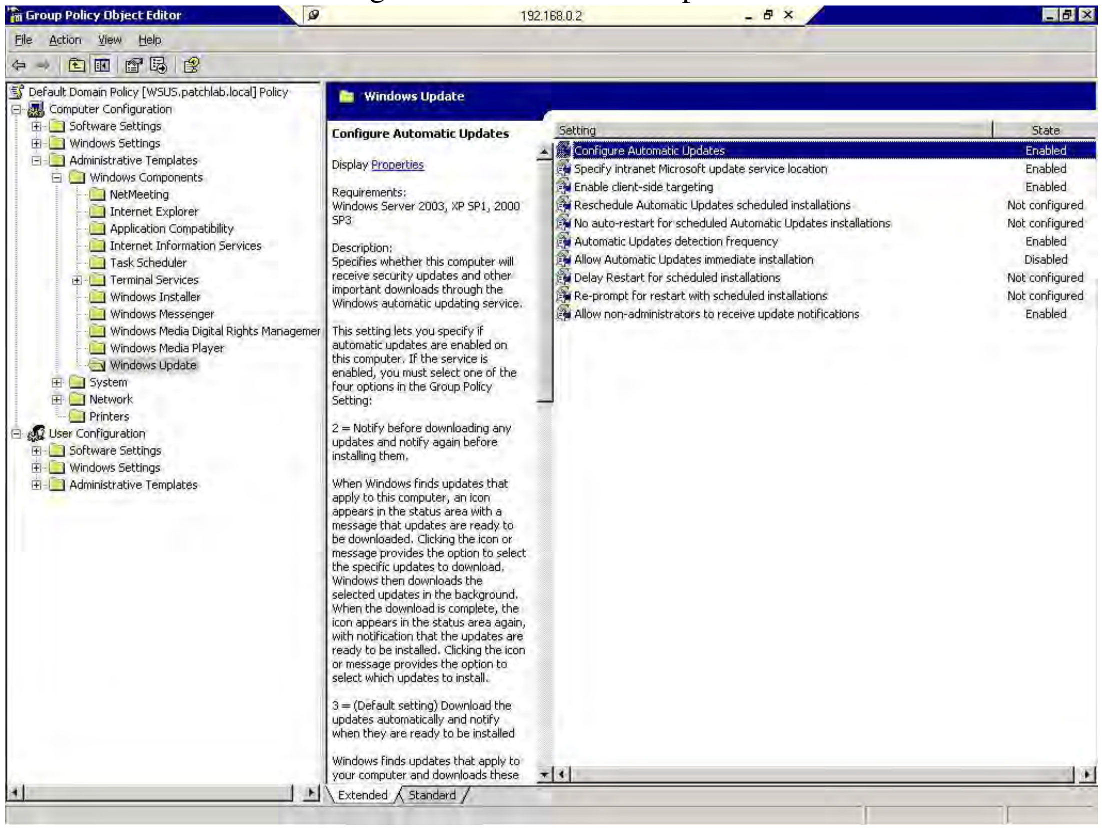
The new background intelligent transfer service (BITS v2.0) and automatic update client allow for several new configuration options on the client side (see figure B.10). The addition of these options appears to be Microsoft's response to criticism of the less flexible options provided in previous versions. In particular, the fewer restarts and greater configurability should make the process more pleasant for the desktop user. These options can be modified in several ways; active directory group policy, local group policy, or registry settings. These configuration methods are referred to as administrative policies, which are distinct from the user's configuration. A few of the options are common to SUS, therefore the focus will be on the changes and new options. Modifying these via group policy can be done by opening the group policy editor and navigating to *Computer Configuration/Administrative Templates/Windows Components/Windows Update*, after loading the windows update administrative template, wuau.adm (this will automatically be upgraded if done previously with SUS). Modifying the settings via the registry requires that the key *HKEY_LOCAL_MACHINE\Software\Policies\Microsoft\Windows\WindowsUpdate[\AU]* be edited.

There are new options related to how update notifications are displayed. These notifications can occur either before downloading and installation, just before installation, or not at all. The first option prevents the automatic update client's user-interface from being locked when administrative policies are used to configure the client. This allows a local administrator to choose their own notification settings. The next allows for non-administrators to be included in the group of users allowed to receive update notifications.

With the introduction of grouping, client-side targeting is a method whereby an AU client will advertise which group it should be a member of, allowing clients to self-populate groups. There is thus an option to specify which group the AU client should request membership of.

WSUS now takes better advantage of the agent on the AU clients, and utilises a periodic check where an AU client will connect and allow the WSUS server to interrogate its patch status. This option is specified in hours. The AU client will connect between the specified time and a 20% offset. Thus if the option is 10 hours the AU client will connect every 8 to 10 hours.

Figure B.10: New BITS Options



Default Domain Policy [WSUS.patchlab.local] Policy
- Computer Configuration
  - Software Settings
  - Windows Settings
  - Administrative Templates
    - Windows Components
      - NetMeeting
      - Internet Explorer
      - Application Compatibility
      - Internet Information Services
      - Task Scheduler
      - Terminal Services
      - Windows Installer
      - Windows Messenger
      - Windows Media Digital Rights Managemer
      - Windows Media Player
      - Windows Update
    - System
    - Network
    - Printers
- User Configuration
  - Software Settings
  - Windows Settings
  - Administrative Templates

Windows Update

Configure Automatic Updates

Display Properties

Requirements:
Windows Server 2003, XP SP1, 2000 SP3

Description:
Specifies whether this computer will receive security updates and other important downloads through the Windows automatic updating service.

This setting lets you specify if automatic updates are enabled on this computer. If the service is enabled, you must select one of the four options in the Group Policy Setting:

2 = Notify before downloading any updates and notify again before installing them.

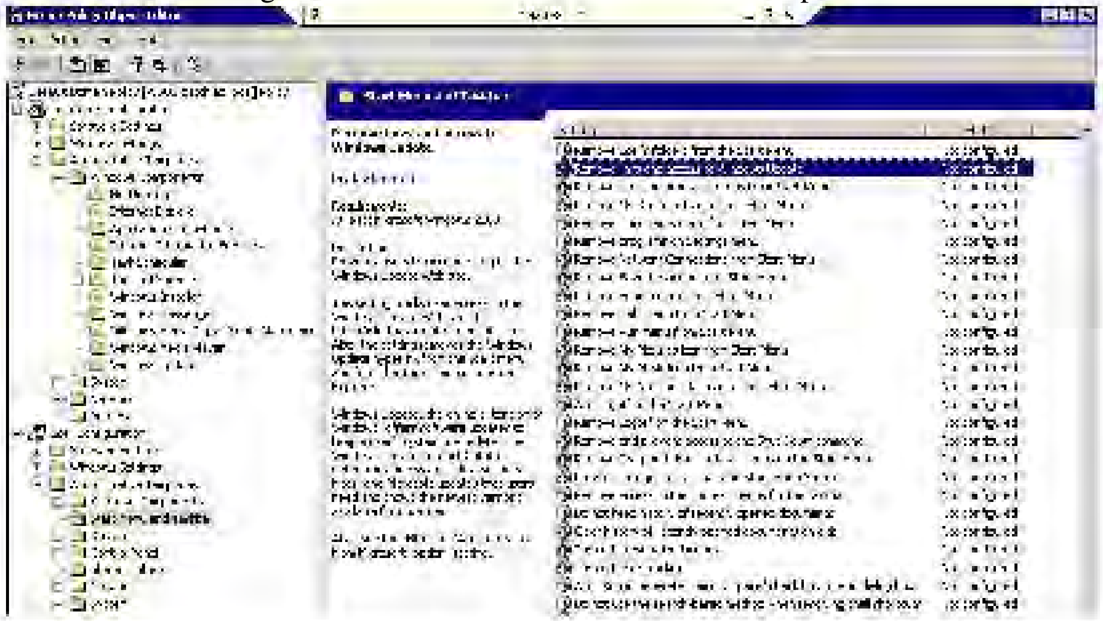
When Windows finds updates that apply to this computer, an icon appears in the status area with a message that updates are ready to be downloaded. Clicking the icon or message provides the option to select the specific updates to download. Windows then downloads the selected updates in the background. When the download is complete, the icon appears in the status area again, with notification that the updates are ready to be installed. Clicking the icon or message provides the option to select which updates to install.

3 = (Default setting) Download the updates automatically and notify when they are ready to be installed

Windows finds updates that apply to your computer and downloads these

Setting	State
Configure Automatic Updates	Enabled
Specify intranet Microsoft update service location	Enabled
Enable client-side targeting	Enabled
Reschedule Automatic Updates scheduled installations	Not configured
No auto-restart for scheduled Automatic Updates installations	Not configured
Automatic Updates detection frequency	Enabled
Allow Automatic Updates immediate installation	Disabled
Delay Restart for scheduled installations	Not configured
Re-prompt for restart with scheduled installations	Not configured
Allow non-administrators to receive update notifications	Enabled

Figure B.11: Remove Access to Windows Update



A separation has been made between updates that require a restart and those that don't. Non-restarting updates can be installed immediately without notifying the relevant user, if the user is configured to receive installation notifications. This allows the administrator to automatically install updates that don't require a restart without disturbing the desktop user, unless an update requires a restart. This should reduce disruptions to the end user.

In the case of scheduled installations, two new options have been provided. One allows a delay to be inserted before continuing with a scheduled restart, and the other allows the amount of time before the user is re-prompted for a scheduled restart to be specified. Though minor, these changes are occasionally useful.

The option to remove links and access to Windows Update was available in SUS (see figure B.11), but is often overlooked and is therefore mentioned here. This will remove the link to Windows Update in the start menu and will prevent non-approved updates being installed from Windows Update. This setting can be found in the group policy editor at *Computer Configuration/Administrative Templates/Start Menu and Taskbar*.

In addition to administrative policies, the update client can be manipulated via the command line. This is done by running *wuauclt.exe* with command line switches. The two switches are:

/resetauthorization which will delete the client side cookie, which normally expires after an hour, and contains information such as the target group (more information on this can be found in section B.7); and */detectnow* which will force the AU client to connect to the WSUS server and check for new approvals. When these switches are used together they must be used in the order they were mentioned i.e. *wuauclt.exe /resetauthorization /detectnow*. This is particularly useful for debugging machines and forcing an update.

B.5 Patching

The process of patching machines is done in six steps: synchronisation, approval, detection, distribution, installation, verification. This section will look at each step, and how WSUS supports it.

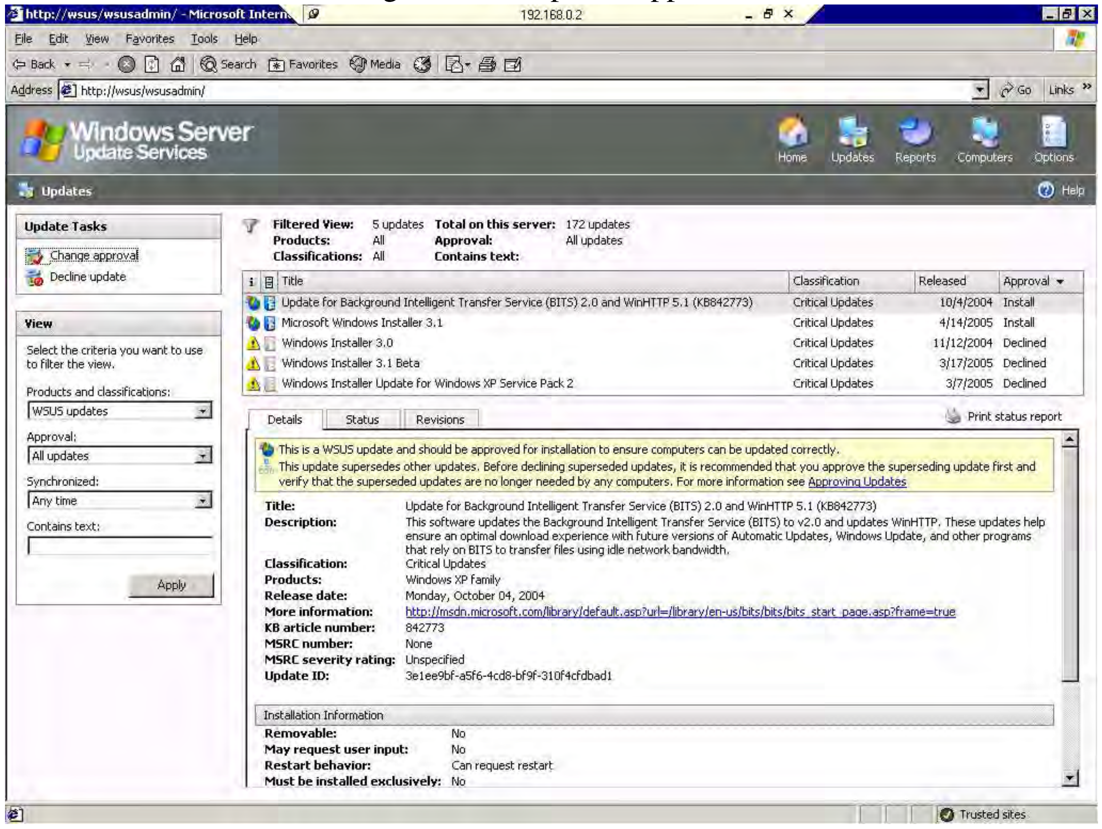
B.5.1 Synchronisation

During synchronisation meta-data is downloaded from a central distribution point, in this case Microsoft Update, and disseminated to other WSUS servers and AU clients. This process can also download the updates to the server, allowing AU clients to fetch them locally, if WSUS has been configured to do so. WSUS uses BITS to transfer the meta-data and updates in the background, and supports resuming the process if it is interrupted. The progress of a synchronisation is displayed on the front page of the WSUS administrative interface (see figure B.5).

B.5.2 Approval

WSUS allows three types of approval to be applied to each update: detect only, install, and remove (see figure B.12). No updates currently support the remove option, but will in the future, as it is a function of the new Windows Installer. The update's approvals can apply to all machines or one group. A group can also inherit its approvals from the global configuration. The interface is far easier to use, allowing updates to be filtered by product, classification, approval, date received, and by a text-based search. The filtered updates can then be sorted by column.

Figure B.12: Update Approval


http://wsus/wsusadmin/ - Microsoft Intern
192.168.0.2
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://wsus/wsusadmin/
Go Links
Windows Server Update Services
Home Updates Reports Computers Options
Updates
Help
Update Tasks
Change approval
Decline update
View
Apply
Trusted sites

Select the criteria you want to use to filter the view.

Products and classifications: WSUS updates

Approval: All updates

Synchronized: Any time

Contains text:

Filtered View: 5 updates **Total on this server:** 172 updates
Products: All **Approval:** All updates
Classifications: All **Contains text:**

Title	Classification	Released	Approval
Update for Background Intelligent Transfer Service (BITS) 2.0 and WinHTTP 5.1 (KB842773)	Critical Updates	10/4/2004	Install
Microsoft Windows Installer 3.1	Critical Updates	4/14/2005	Install
Windows Installer 3.0	Critical Updates	11/12/2004	Declined
Windows Installer 3.1 Beta	Critical Updates	3/17/2005	Declined
Windows Installer Update for Windows XP Service Pack 2	Critical Updates	3/7/2005	Declined

Details Status Revisions Print status report

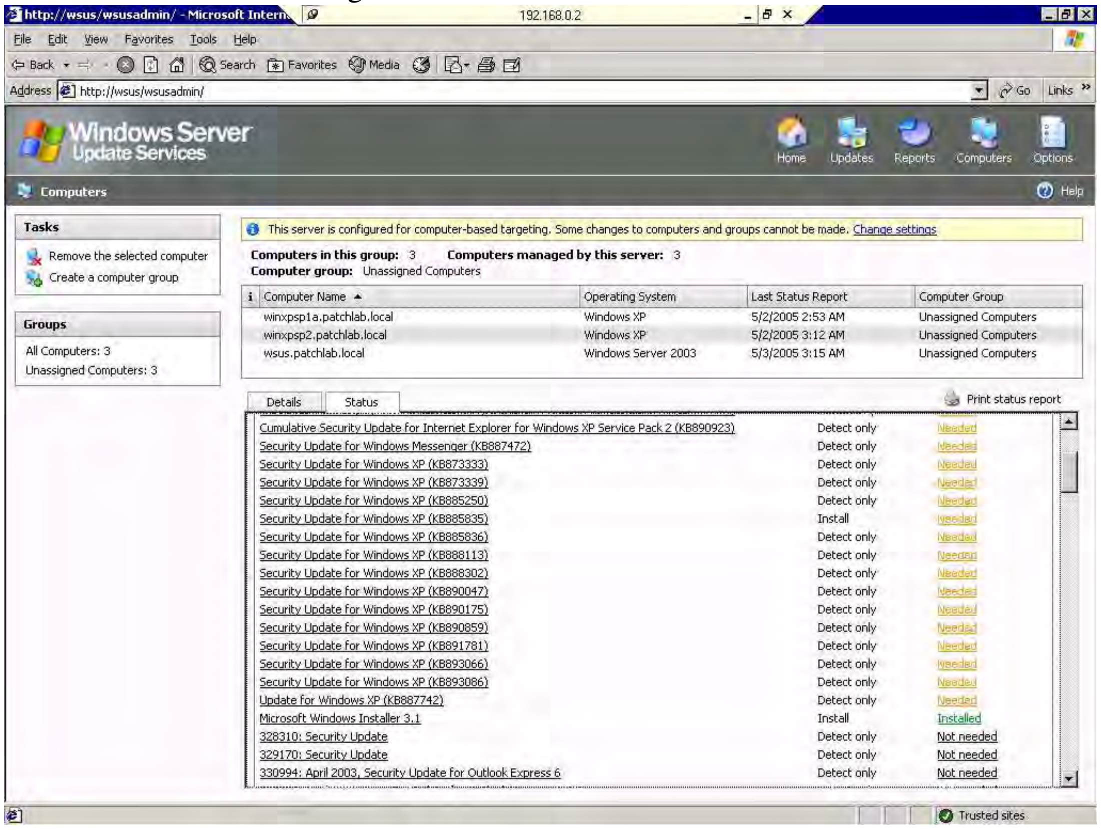
This is a WSUS update and should be approved for installation to ensure computers can be updated correctly.
This update supersedes other updates. Before declining superseded updates, it is recommended that you approve the superseding update first and verify that the superseded updates are no longer needed by any computers. For more information see Approving Updates

Title:	Update for Background Intelligent Transfer Service (BITS) 2.0 and WinHTTP 5.1 (KB842773)
Description:	This software updates the Background Intelligent Transfer Service (BITS) to v2.0 and updates WinHTTP. These updates help ensure an optimal download experience with future versions of Automatic Updates, Windows Update, and other programs that rely on BITS to transfer files using idle network bandwidth.
Classification:	Critical Updates
Products:	Windows XP family
Release date:	Monday, October 04, 2004
More information:	http://msdn.microsoft.com/library/default.asp?url=/library/en-us/bits/bits/bits_start_page.asp?frame=true
KB article number:	842773
MSRC number:	None
MSRC severity rating:	Unspecified
Update ID:	3e1ee9bf-a5f6-4cd8-bf9f-310f4cfdbad1

Installation Information

Removable:	No
May request user input:	No
Restart behavior:	Can request restart
Must be installed exclusively:	No

Figure B.13: Patch Status Detection


http://wsus/wsusadmin/ - Microsoft Internet
192.168.0.2
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://wsus/wsusadmin/
Go Links
Windows Server Update Services
Home Updates Reports Computers Options
Computers
Help
Tasks
Remove the selected computer
Create a computer group
Groups
Details Status
Print status report
Trusted sites

All Computers: 3
Unassigned Computers: 3

This server is configured for computer-based targeting. Some changes to computers and groups cannot be made. Change settings

Computers in this group: 3 **Computers managed by this server:** 3
Computer group: Unassigned Computers

Computer Name	Operating System	Last Status Report	Computer Group
winxpsp1a.patchlab.local	Windows XP	5/2/2005 2:53 AM	Unassigned Computers
winxpsp2.patchlab.local	Windows XP	5/2/2005 3:12 AM	Unassigned Computers
wsus.patchlab.local	Windows Server 2003	5/3/2005 3:15 AM	Unassigned Computers

Cumulative Security Update for Internet Explorer for Windows XP Service Pack 2 (KB890923)	Detect only	Needed
Security Update for Windows Messenger (KB887472)	Detect only	Needed
Security Update for Windows XP (KB873333)	Detect only	Needed
Security Update for Windows XP (KB873339)	Detect only	Needed
Security Update for Windows XP (KB885250)	Detect only	Needed
Security Update for Windows XP (KB885835)	Install	Needed
Security Update for Windows XP (KB885836)	Detect only	Needed
Security Update for Windows XP (KB888113)	Detect only	Needed
Security Update for Windows XP (KB888302)	Detect only	Needed
Security Update for Windows XP (KB890047)	Detect only	Needed
Security Update for Windows XP (KB890175)	Detect only	Needed
Security Update for Windows XP (KB890859)	Detect only	Needed
Security Update for Windows XP (KB891781)	Detect only	Needed
Security Update for Windows XP (KB893066)	Detect only	Needed
Security Update for Windows XP (KB893086)	Detect only	Needed
Update for Windows XP (KB887742)	Detect only	Needed
Microsoft Windows Installer 3.1	Install	Installed
328310: Security Update	Detect only	Not needed
329170: Security Update	Detect only	Not needed
330994: April 2003, Security Update for Outlook Express 6	Detect only	Not needed

B.5.3 Detection

Periodically an AU client will connect to the server and provide a list of platform details, installed updates, hardware, and drivers. This is then used by the WSUS server to display which updates are needed by the AU client and which have been successfully installed (see figure B.13). This is particularly useful for determining the patch status of an organisation. The frequency with which an AU client connects to the WSUS server is configured on the client (see B.4.2). This is also where the AU client synchronises with its WSUS server.

B.5.4 Distribution

Updates are distributed over HTTP using the background intelligent transfer service (BITS), which supports resuming of interrupted downloads and dynamic throttling of downloads to use spare bandwidth. Updates can either be downloaded from a local WSUS server or Microsoft Update depending on the topology (see section B.3.1). Distribution has been made more flexible with the introduction of download on demand, wherein updates are only downloaded to the server when needed, and express updates, which make use of binary patching (see section B.4.1).

B.5.5 Installation

Many of the patching improvements in WSUS are due to the new Windows Installer ver 3. Microsoft has converged their many patching methods into two which are supported by the new installer[235]. The new MSI packages will also support uninstallation of updates[235], hence the new 'remove approval' setting. In addition, these packages will require fewer restarts and will support binary patching[235], hence the introduction of express updates. Other powerful switches have been added, and more detail can be found from Microsoft[236].

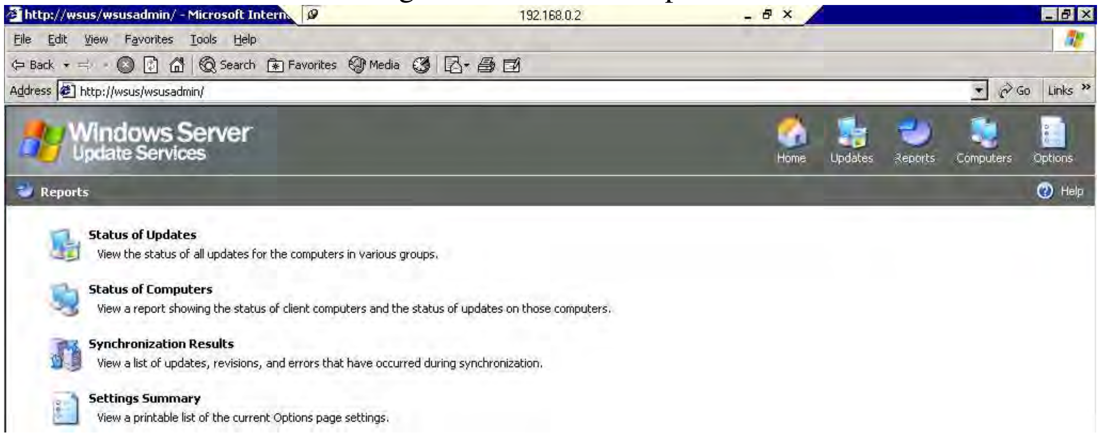
B.5.6 Verification

An important part of any patch management solution is the ability to verify that the patch was actually installed. In WSUS this is achieved through the same interface used for detection (see figure B.13). The AU clients check in after installing updates and after a machine restart in which updates are installed.

B.6 Reporting

The single largest problem with SUS was its complete lack of reporting. WSUS offers four reports officially labelled as such (see figure B.14). The two most useful are a breakdown of updates or computers, which allows an administrator to drill down to see statistics for groups and individual AU clients or updates(see figures B.15 and B.16). These reports can be filtered by approval and groups, and can then be sorted by each column. This is not the only reporting in, WSUS as many other screens provide reporting features, such as the computer and update screens (see figures B.13 and B.12). On the back-end, all of the information is stored in a SQL database, allowing ad-hoc queries to be address through third-party tools (such as Microsoft

Figure B.14: WSUS reports


http://wsus/wsusadmin/ - Microsoft Internet
192.168.0.2
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://wsus/wsusadmin/
Go Links
Windows Server Update Services
Home Updates Reports Computers Options
Reports
Help

Status of Updates
View the status of all updates for the computers in various groups.

Status of Computers
View a report showing the status of client computers and the status of updates on those computers.

Synchronization Results
View a list of updates, revisions, and errors that have occurred during synchronization.

Settings Summary
View a printable list of the current Options page settings.

Systems Management Console). This is a great improvement over SUS, but many administrators will probably require more.

B.7 Packet Capture

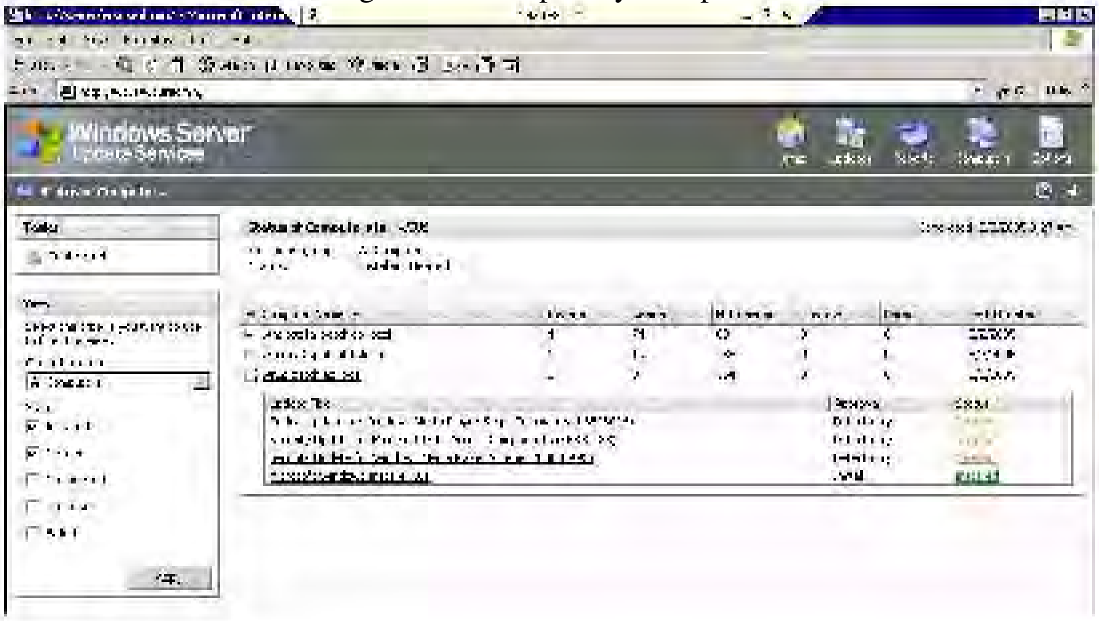
To get a better look at how WSUS does its work, Ethereal was used to perform a packet capture of the communications between an AU client and the WSUS server. This revealed several improvements over SUS. Further, it demonstrated the working of WSUS which have not been published in much detail as yet. The testing here was performed on a variety of WUS and WSUS pre-releases, and thus some of the bugs may have been resolved.

B.7.1 Steps Performed

The relevant tasks performed during the packet capture were:

1. A new Windows XP SP1a AU client is joined to the active directory domain.
2. AU client self-updated.
3. The new AU client installs Windows Installer and BITS updates, required a restart.

Figure B.15: Report by Computer



4. Logged in with some automatic update activity. The logged in administrator was not informed, although the icon appeared briefly. A restart was required.

5. Logged in and 24 new updates were downloaded.

6. Updates were installed, restart was required.

7. WSUS server synchronised with Microsoft Update.

8. *wuauclt.exe /detectnow* was run from command line on the AU client.

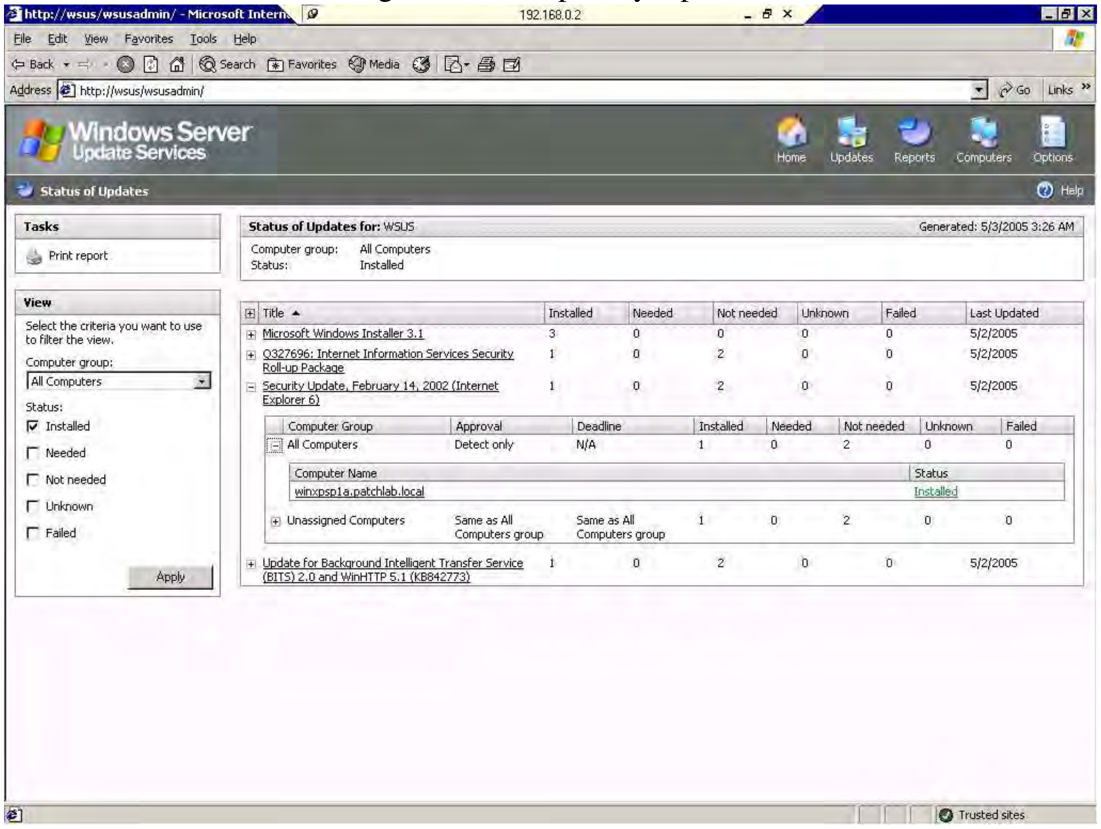
9. One critical update detected, downloaded and installed.

10. The same critical update was detected, downloaded, and installed multiple times until approval was revoked on WSUS server.

B.7.2 Resulting Network Traffic

By comparing the resulting packet capture to the steps performed above, the interactions between the WSUS server and the AU client were discovered. Below is a chronological list of recorded HTTP request traffic between the WSUS server and AU client, and its analysis.

Figure B.16: Report by Update


http://wsus/wsusadmin/ - Microsoft Intern
192.168.0.2
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://wsus/wsusadmin/
Go Links
Windows Server Update Services
Home Updates Reports Computers Options
Status of Updates
Help
Trusted sites

Tasks

Print report

View

Select the criteria you want to use to filter the view.

Computer group: All Computers

Status:

- [✓] Installed
- [ ] Needed
- [ ] Not needed
- [ ] Unknown
- [ ] Failed

Apply

Status of Updates for: WSUS — Generated: 5/3/2005 3:26 AM

Computer group: All Computers
Status: Installed

Title	Installed	Needed	Not needed	Unknown	Failed	Last Updated
Microsoft Windows Installer 3.1	3	0	0	0	0	5/2/2005
Q327696: Internet Information Services Security Roll-up Package	1	0	2	0	0	5/2/2005
Security Update, February 14, 2002 (Internet Explorer 6)	1	0	2	0	0	5/2/2005
Update for Background Intelligent Transfer Service (BITS) 2.0 and WinHTTP 5.1 (KB842773)	1	0	2	0	0	5/2/2005

Computer Group	Approval	Deadline	Installed	Needed	Not needed	Unknown	Failed
All Computers	Detect only	N/A	1	0	2	0	0
Unassigned Computers	Same as All Computers group	Same as All Computers group	1	0	2	0	0

Computer Name	Status
winxpsp1a.patchlab.local	Installed

- /iuident.cab - This stands for 'Industry Update Identification' and is how the client's version is identified. This .cab file along with the rest below was timestamped by Verisign and signed by Microsoft. If this were the first communication of a machine with a WSUS AU client rather than a SUS AU client (e.g. Windows XP with SP2) then these first three steps would not be seen and the traffic would start with a call to wuident.cab.

- Once it is determined that this is a SUS client, the self-update from point 2 of the above section (B.7.1) is performed. The client is instructed to download the relevant .cab files (starting with wacomp.cab, which contain version information for the individual client files) of the new automatic update client. In this client configuration the files were stored in /selfupdate/au/x86/XP/en/ on the IIS server.

- /wutrack.bin - After the self-update, the client requests wutrack with a parameterised query string. With SUS, the request of wutrack.bin was used for reporting and statistics on the patch process. The parameters provide information on aspects of the clients behaviour, including platform, activity, and the KB of the patch being installed (more information can be found on page 83 of the SUS deploy guide[233]). This method is how third-party SUS reporting tools were developed (e.g. K. Hoover's[237] or my D. White's [238]). With WSUS the item and activity parameters are not used, but platform information is provided. This was the only request to wutrack.bin seen in the whole capture, and appears to be left for backwards compatibility.

- /wuident.cab - This stands for Windows Update Identification, and contains AU client version information. This request includes a date stamp as a parameter.

- /wusetup.cab - This contains an .inf and .cat file which contain setup information, such as dll version and registration information, for the new automatic update client. This request also includes a date stamp as a parameter.

- From here the new automatic update client communicated with the WSUS sever using a SOAP based web service. The format used to describe the method calls is: *[returned information] MethodName (passed information)*

  - [config] GetConfig
  - [auth cookie] GetAuthCookie
  - [cookie] GetCookie (encrypted(auth cookie))
    After this the returned cookie is encrypted and sent as the preamble to all future

  transactions. This cookie will contain information such as the target group of the AU client, and expires after an hour.

  – RegisterComputer (a SOAP XML file is passed with the full platform information)

  – [required update ID's] SyncUpdates (system information, such as platform information, installed updates and installed drivers) ...

    This is how the WSUS server knows what updates are needed on the client. This method is called several times. The first time it is called the client sends empty update ID parameters. The last time it is called it contains strings of hardware drivers installed on the client.

  – [metadata] GetExtendedUpdateInfo (update meta-data)

    This includes information such as the EULA and description of each update.

  – [confirmation] ReportEventBatch (meta-data and sync updates status)

    Information about the status of the client registration is returned. The client passes a large XML file to the server here, detailing the status of the updates and once again providing platform information.

  – The first batch of updates is then downloaded as per point 3 above. In this case it is the Windows Installer 3.1 and BITS 2.0 updates. Once installed these will allow the full WSUS functionality to be used. Files are downloaded from sub-directories of the /Content/ virtual directory in chunks, presumably to allow resuming of downloads if the process is interrupted.

  – [confirmation] ReportEventBatch (update download status)

    Information about the status of the download of the patches. This is sent before the update is installed, but after it is downloaded. According to the WSUS deploy guide, the AU client should request meta-data from the WSUS server again after downloading but before installation[224]. This is to ensure that approvals revoked during the download are not ignored. While there were no separate requests representing this, it is presumed that it would occur here.

  – [confirmation] ReportEventBatch (update installation status)

    Before the client restarts and after the updates have been installed, another report is made. A separate report is made for each installed update.

- After a restart, the behaviour seen in point 4 (of the previous section B.7.1) is seen. No notification was received by the logged-on administrator, which conflicted with how group policy had been configured. It was assumed that updates for immediate install were being installed, as that option *had* been activated (see section B.4.2). However a restart was required, which should not happen if this activity was as a result of immediate updates, as they do not require a restart. This resulted in the following use of the web service:

  - [confirmation] ReportEventBatch (update installation status)

    This is a report on the, now complete, installation of the updates installed before the reboot.
  - [location on the web server of updates] GetFileLocations (update ID's and file digests)
  - The updates are then downloaded. Once again the AU client should check that none of the approvals for the downloaded updates have been revoked during the download. It is presumed that this check would be part of the GetFileLocations method.

- The machine is then restarted and 24 updates are available as per point 5, after which these calls are made:

  - [confirmation] ReportEventBatch (update installation progress) ...

    This is presumed to be reporting on the status of the installation of updates from the previous point.
  - [update location] GetFileLocations (update ID's and file digests)
  - The updates are then downloaded and installed.
  - [confirmation] ReportEventBatch (update installation progress)

- The machine is then restarted and another call to ReportEventBatch is made before point 8 is run. Running *wuauclt /detectnow* resulted in:

  - /wuident.cab
    /wusetup.cab

    These request are made with a date stamp as a parameter.
  - [required update ID's] SyncUpdates (system information, such as platform information, installed updates and installed drivers) ...
  - [metadata] GetExtendedUpdateInfo (update meta-data)
  - One update is then downloaded and installed.

– [confirmation] ReportEventBatch (update installation progress)

These are the items of interest. The full packet capture is available from the following URL:http://singe.za.net/mast packetcapture.tar.gz `http://singe.za.net/masters/files/WSUS-packetcapture.tar.gz` (warning: this is a 60MB file) for further analysis.

B.7.3 Analysis

From the information above, a pattern of behaviour can be mapped.

When the AU client first contacts the WSUS server it makes two requests, each with a date stamp as a parameter. The files are returned timestamped and signed.

1. wuident.cab
2. wusetup.cab

After this, all future interactions (apart from BITS downloading the updates) are done via a web service.

After which, if the AU client does not have a cookie, or its cookie has expired, the following handshake is made with the WSUS server:

1. GetConfig
2. GetAuthCookie
3. GetCookie
4. RegisterComputer

If the AU client still has a valid cookie, the above does not occur. The cookie is then pre-pended to all future transactions.

If the WSUS server has synchronised with an upstream server since the AU client's last synchronisation, a new synchronisation is performed. This looks like:

1. SyncUpdates
2. GetExtendedUpdateInfo
3. ReportEventBatch
4. Updates are downloaded.

If the AU client does not need to synchronise but has pending updates, a call is made to:

1. GetFileLocations
2. Updates are downloaded.

A reporting call is made after every action, and would be made after an update sync, update download and update installation. After the updates are downloaded the call is made:

1. ReportEventBatch

After the installation of the updates another report is made:

1. ReportEventBatch

If a restart is required to install any of the updates, another call is made after the machine has rebooted and, presumably, installed the updates.

1. ReportEventBatch

B.7.4 Packet Capture Summary

B.7.4.1 Interface

On the whole, WSUS seems to be better designed. It utilises an open SOAP based web service, keeps track of each interaction, and provides far more information on the patching process. SUS on the other hand, required third-party log analysers to interpret an obscure query string. The use of a standard web service should make it easier for third-party extensions to be created. The large amount of information generated should allow for many different reporting options beyond what WSUS currently offers.

B.7.4.2 Security

There are two security worries here - the first is disclosure of sensitive information, and the second is interference with the patch process. The downside of the extra information mentioned above is that a lot of information about client machines is being sent as clear-text. This information includes a list of hardware, installed drivers, and some software being used. There is enough information to allow an attacker to build a replica system to test attacks on. This is a worry, but it can be mitigated by good network design. The second worry is less troublesome, as a man in the middle attack (which the cookie exchange may be vulnerable to) would not be able to circumvent the security of the signed patches.

B.8 Resources

There are several fairly useful resources for WSUS available. Several were quite valuable while writing this document.

1. Microsoft's WSUS page `http://www.microsoft.com/wsus/`
2. The WSUS Wiki `http://wsus.editme.com/`
3. SUS Server `http://www.susserver.com/`
4. Patch Management Mailing List `http://www.patchhmanagement.org/`

B.9 Conclusion

WSUS is definitely a large step in the right direction. It has many great improvements over SUS, which seem to indicate that Microsoft is listening to the consumer and responding to their communities security needs. The interface is easy to use and provides some great functionality. The extra features provided on the client-side are equally welcome. Microsoft has developed a good architecture from which their patching strategy can be better managed and built upon. The most notable problem is that WSUS still only supports a limited range of Microsoft's products and is sorely lacking support for third-party updates. Some of these problems are resolved in Microsoft's Systems Management Server (SMS).