A UNIFIED ARCHITECTURE FOR AUTOMATIC SOFTWARE UPDATES

Dominic White

Computer Science Department, Hamilton Building
Rhodes University, Grahamstown, 6140
e-mail : project@singe.rucus.net
12 June 2004

ABSTRACT
This paper attempts to address the issue of hardening the internal security of an organisation's network by easing its patch management. Traditionally security has been modeled on a "hard outer shell" approach, with a firewall protecting the otherwise vulnerable internal network. With the advent of worms using such techniques as social engineering to bypass the organisational firewall and installing trojans, this approach is no longer sufficient. As a result of these new attacks, emphasis should be placed on improving the security of the internal network. Most research agrees that prompt patching of security vulnerabilities would significantly reduce the vulnerability of these machines. However, this requires system administrators not only to keep abreast of the flood of patches, but to ensure they are deployed to every machine, in what could be a very large network. These difficulties are worsened by problems the patches themselves often create. This is a difficult task and the failure of system administrators to perform it is echoed in the recent spate of worm attacks, with some taking advantage of vulnerabilities for which patches had been released up to six months earlier. To cope in this environment an organisation needs a comprehensive patch management policy. A technology agnostic view of this policy is described in order to better understand what is required of an automated solution. A few vendors have released tools to find vulnerabilities, find patches and distribute them, the best known example of which is Microsoft's Software Update Service. However, these tools are generally inflexible, expensive and only deal with a limited part of the patch management process. This paper seeks to discuss the implementation of an open source, cross platform tool to solve this problem. This will involve discussions in four areas; the need for such a system, the patch management process, existing systems, and the proposed solution. The proposed system architecture is then broken down into four areas; sourcing patches, patch packaging, testing patches, patch distribution and the development of the system. The first area will discuss how to pull patches from multiple vendors such as Windows Update, Symantec AV, FreeBSD ports and ISC. The second will involve discussion on creating packages; whether to replace entire binaries, to employ binary patching techniques or to use source distribution. The third section will discuss how this tool can improve testing and provide stop-gap measures for protecting the organisation during the testing process through the use of IDS signatures. The fourth section will discuss methods of distributing these packages, such as direct download or peer-to-peer implementations like Bittorrent. The last section will discuss a modular, platform-independent implementation of the system.

1 INTRODUCTION

Systems and network administrators currently face two large problems in trying to manage the security of an organisation's many machines.

- How to stay ahead of the rapid tide of security patches being release by multiple vendors for multiple products?
- How to apply those patches to many machines in a timely manner in order to prevent a security breach, without risking service failure unnecessarily?

However, before any solution is developed, the need for such a solution and the methods in which existing solutions attempt to solve the problem must be examined. Thus the first section of this paper will establish this need by examining current literature from the field and analysing recent security threats, from this a patch management process will be derived. The second section will overview some of the more popular methods for solving the problem and how they either fail or could be improved upon. The information from the previous section will be used to develop criteria by which current solutions can be evaluated. The last will discuss a preliminary architecture for a solution and focus on how it automates or improves the patch management process.

2 THE NEED

Until recently the approach to computer security was very much organised on a "hard outer-shell" model. At its most basic level this was demonstrated by a firewall protecting an otherwise highly vulnerable internal network of computers. This was made more complex and more effective by creating tiers, with one firewall protecting external servers and another protecting internal computers, all with the intention of keeping malicious code out. However these are not proving effective in preventing malicious programs from harming networks. This appears to be due to a few reasons, the increase in mobile computing, the use of social engineering techniques to spread, and an increasing number of high-level protocols. The first reason is due to a firewall's inability to block non-network traffic and effectively allows for internal devices such as laptops or removable media to travel outside the firewall, become infected and allow these infection to piggy back their way through the firewall, this is often solved with special firewall rules for laptops and wireless subnets, however cannot be totally effective. The second reason is due to the recent use of social engineering attack vectors. Instead of attempting to poke holes in firewalls, the malicious code exploits human trust over less secure media such as e-mail or instant messengers, where the user and not organisational policy enforcing software decides on the trust. This brings us to the third problem of an increasing number of high level protocols such as peer-2-peer networks and instant messaging which require stateful firewalls to understand the new protocols and examine traffic over longer windows. This is an increasingly complex task. Due to the fallibility of firewalls in providing complete security, the security of an organisation's internal network cannot be ignored and must be hardened.

The recent spate of worm attacks have demonstrated the effect of weak internal security and debilitated many computer networks. There are several example of this; the SQL Slammer worm that took advantage of a vulnerability for which a patch had been available for six months[1], the Blaster worm for which a patch had been available for three months and Microsoft's Software Update Service had been introduced[2] and the Raman worm which attacked Red Hat Linux systems using three published exploits[3]. All of these vulnerabilities were able to circumvent an organisational firewall, indicating the need to harden the organisations internal network security[4]. Current research suggests that it takes over a month to halve the number of vulnerable computers connected to the Internet and that failing to patch machines leads to upwards of 90% of security breaches[5]. This research suggests that the difficulty in patching machines stems from the fact that half of the most prevalent vulnerabilities are replaced every year by new vulnerabilities and that 80% of announced vulnerabilities have exploits available within 60 days[5]. Last year CERT reported 3784 vulnerabilities with reported security incidents rising from 82 to 137 thousand[6]. These statistics are showing how more vulnerabilities are being announced, with less time in which to fix them. This require the average system administrator to be aware of which of the many vulnerabilities affect his system, what potential effect these vulnerabilities could have, which patches correct the problem, and how to rapidly deploy them to many machines without compromising critical services. With hundreds of vulnerabilities being reported each week this is an almost impossible task.

However that is not the end of the problem, even if a system administrator manages the near impossible task described above, many of the patches being deployed have confusing versioning, a lack of documentation, don't work or break critical services. The slammer debacle provides an excellent example of just such a failing. MS02-039 was announced in July 2002, soon after three more patches for related SQL vulnerabilities were released in 2002, MS02-043 in August, MS02-056 at the start of October and MS02-061 in mid October. MS02-061 was originally kludgy and unpleasant to install (seven revisions later it still contains several pages of documentation describing how to install and remove the patch), however if an administrator applied all of these they were theoretically protected. On October 30th a non-security hot-fix for SQL server, Q317748 was released by Microsoft. This included a library that was still vulnerable and undid the work of the original patches, leaving systems that had been successfully patched vulnerable once more. These patches could have been avoided by deploying the newly released Service Pack 3. Given the large number of changes in a service pack, most organisations perform tests before deploying service packs and use the patches in the mean time. Those organisations that went for the Service Pack did not escape unscathed either. It was discovered that Best Software's MAS 500 accounting pack wouldn't run with Service Pack 3 and required a reformat of the machines to remove.[7]

This indicates that current patching and vulnerability management faces two problems, a volume problem and a process problem[10]. In an article by Bruce Schneier entitled "The Security Patch Treadmill" published in his newsletter *Cryptogram*[8], he discusses the problem and believes that in a perfect world "systems would rarely need security patches. The few patches they did need would automatically download, be easy to install, and always work." Thus the proposed solution intends to automate a large part of this process, creating proper change control and providing the necessary risk management information to allow an administrator to focus on their decision making.

3 PATCH MANAGEMENT PROCESS

In order to create a solution to automate a process, the process must be fully understood. This section will describe what a patch management process should look like. Specific technologies will not be discussed but rather a process oriented, technology agnostic look at what needs to be done, whether automated or manual, will be discussed. The main aim of this process is "to create a consistently configured environment that is secure against known vulnerabilities in operating system and application software"[9].

To this end seven steps have been identified. These steps will be described and are largely based on an amalgamation of the work done it two separate papers by Jason Chan[9] of @stake and Daniel Voldal[10] of the SANS institute.

3.1 Information Gathering

This is the information gathering phase. This information is needed to make decisions and provide proper risk analysis. This is an ongoing phase that will have inputs from many of the other steps.

3.1.1 Asset & Host Management

For a patch management process to be effective you need to know what machines are utilising your organisation's network and what they are running. This is necessary at the very least to figure out which machines are affected by which vulnerabilities and hence require patching. This should not be a once off process and should be continuous to ensure detection of new machines entering the network. As such, this should be a passive method that does not require software to be installed onto machines, thus allowing the discovery of all new machines. Typically the kind of information collected about each machine should include the host name, physical location, IP and MAC address and the software revision level. For servers additional information such as the server function and running services should be recorded[10]. Each system (or groups of systems) should then be classified into one of three criticality levels according to Voldal[10].

1. Mission critical. Machines providing services critical to the business operation. e.g. Amazon's Web Servers
2. Business critical. Machines providing important services that can tolerate short breaks in service. e.g. E-mail servers.
3. Operational critical. Machines providing non-critical services. e.g print servers

This classification will be useful in determining the seriousness of a vulnerability and whether a patch should be installed immediately or not.

3.1.2 Vulnerability Notification

The other side of information gathering relates to the vulnerabilities and patches being released. A system administrator must know as soon as a vulnerability is disclosed and what patches are needed to repair it. This information should include what versions of the software are affected, a criticality rating of the vulnerabilities seriousness and what steps can be taken as a stop-gap measure. This can be achieved, at a minimum, by subscribing to the relevant vendor security announcement lists and preferably to public disclosure lists such as SecurityFocus BugTraq [9]. This information coupled with the information obtained in the previous step will allow an administrator to quickly discover the seriousness of any vulnerability to their network.

3.2 Scheduling

To ensure that patching is done regularly in a controlled and predictable manner a patch schedule should exist. This should be made up of two different cycles. The first should be a normal cycle who's purpose is to ensure the application of non-critical, standard patches and updates. This can be either a time or event bases cycle e.g. quarterly or when service packs are released. 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. Within each of these cycles a hierarchy of patch priority should be developed to determine what order patches and machines are worked on[9]:

- Vendor Criticality - how serious the vendor has marked the vulnerability as.
- Exploit Availability - whether the vulnerability can be actively exploited either directly or through the availability of proof-of-concept code.
- System Importance - what criticality level do the affected machines fall under.
- System Exposure - how exposed is the system to the external Internet e.g. organisational firewall vs internal print server.

3.3 Testing

This is a critical part of any patch management process. Ideally the testing should be done in an environment that exactly mirrors the production environment, however this is very often not possible. At a minimum the test environment should represent all mission critical servers. If an adequate test environment cannot be established other methods may include rolling patches out to the lowest criticality, easily recoverable machines first, then continue up the criticality hierarchy in a waterfall style roll-out. Alternatively software such an VMWare or Microsoft's Virtual PC can be used to create a "lab-in-a-box" where patches can be tested. This step is especially important if an automated patch deployment solution will be used. The most common complaint about automatic patching is that faulty patches will be deployed automatically[7]. This is due to this testing step being ignored. The deployment of a patch should not be considered testing, whether manual or automatic. The actual mechanics of the testing process should be determined in the organisation, these vary from a simple check that the patch installs and the system reboots to a series of automated scripts checking the critical functionality of the officially supported software.

3.4 Change Management

During this step, the various plans for patch implementation are drawn up. These consist of three important areas. The first are contingency and back-out plans. These should be prepared for a worst case scenario. Documentation describing what is being installed and how to remove it should be drawn up, backups should be made, personnel should be on standby if something does go wrong, and the help-desk should be notified of the upgrade and have the relevant information to respond to calls. The second is risk mitigation. This requires performing the roll-out in a way that will limit possible complications and work within the existing infrastructure. For example, ensuring that the file server distributing the patch has enough bandwidth, and if not, staggering the times in which machines update. The last area describes plans relating to the monitoring and acceptance of patches. These plans detail what criteria must be met for the patch to be considered successful and how these criteria will be monitored. This will provide a specific milestone for the completion of the upgrade[9].

3.5 Installation

This is the step that is best understood by the average system administrator. This is also the step many automated solutions enable without considering patch management as a whole. Thus it is important in the context of this paper to view this step as part of a larger patch management process. This step should be well controlled to prevent drift from the consistency afforded by correct change management. To this end the patches should be deployed in a predictable manner so as to limit disruption to the business's processes. To prevent inconsistent deployment access controls should be used to disallow users or other programs from installing their own patches. To enforce this consistency policy, guidelines should be drawn up, coupled with regular checks. Patching of mission critical systems should be done manually, during off-hours[10].

3.6 Audit & Assessment

This step is used to ensure that the patches were deployed successfully. This can be done through the use of agent software on the machines, a vulnerability scanner, or both. Chan[9] argues that this step should contain the asset and host management steps described in point one, however Voldal[10] places this activity under step one as this paper has. This paper follows Voldal's assessment as system discovery is critical for more security aspects than patching and should be viewed as a dependency of patch management. This phase should also generate reports, documenting any problems that occurred to prevent repeat mistakes. These reports should be regularly forwarded to upper-management to ensure they know the patch management process is functioning correctly[10].

3.7 Consistency & Compliance

In this step the changes and lessons learnt are converted into feedback to improve the process. This can be manifested physically as an update of ghost images, build scripts and documentation. This ensures that new machines are compliant and deployed under the patch management umbrella in a smooth and timely manner. This step should also include documentation being submitted to the organisations change management process to ensure an organisation wide acceptance of the patches. This will also provide an audit trail and locatable documentation in case of failure.

4 EXISTING SOLUTIONS

In response to the need established above, many vendors are currently marketing solutions, these range from ad-hoc shell scripts which leverage existing tools, such as security scanners, to complete configuration managers and usually take one of three forms:

1. Client/Server architecture
2. Security scanners combined with a method for pushing patches
3. Complete configuration managers

These systems are generally either very expensive proprietary implementations or scripts with limited scope. Few of the solutions strike the correct balance between price and functionality. Some examples of each type will be examined below[1]. However, before they are examined, a framework from which to evaluate them must be established by defining a set of criteria which a patch management solution should adhere to. The Gartner Group has described nine capabilities that an automated patch management solution should contain[11]:

1. 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.
2. Information on the software and patch revision level of each software component on each system.
3. Automatic evaluation of patch dependencies and tracking of which patches are out-of-date or superseded.
4. A dynamically refreshed patch inventory and ability to classify the patch according to severity.
5. Reports on what patches are needed on which systems. This should take into account the system's role.
6. The solution should provide for system groupings to allow for abstracting many machines into one group. The system should also allow for role based-administration to allow different parts of the work-flow to be executed by different roles (e.g. Quality Assurance).
7. A scalable patch distribution and installation method, providing for patch roll-back if necessary.
8. The system should be cross-platform to support the usually multi-platform server environment in most organisations.
9. The solution should leverage existing software for patch management and only introduce a software agent where necessary.

These criteria are incomplete. They discuss such obvious capabilities as 'patch installation' but leave out other obvious criteria. As such the list should be modified to include additional criteria. The solutions should be:

- Secure to prevent its centralisation from being misused.
- Inexpensive to ensure its widespread use.
- Contain multiple vendor support for heterogeneous vendors.
- Detailed and powerful feedback mechanisms for decision making.
- Flexible to ensure integration with other organisational software such as firewalls and Intrusion Detection Systems.
- Robust to failure, easy to use and buzzword compliant.

It should be noted that while aspects such as robustness and security should be part of any software, they are particularly important in this case.

[1]It should be noted that during the course of this research there are plans to install and test these solutions, but at this point in time the product literature and reviews only, were evaluated.

4.1 Client-Server

Client-Server solutions involve a specific client which must be installed on the destination machines and report to some form of centralised server. These are the most popular type of solution. I will examine three of the most ambitious and popular solutions in this category and show how they do not fulfill our criteria.

Software Update Service[12] is Microsoft's patch management solution for the Windows platform. It is intended to automate the installation and ease the administration of critical patches on networks of windows machines. However it does not meet our criteria at any level. SUS does not handle patches other than critical updates for the Microsoft Windows family of operating systems. This excludes Microsoft's own recommended updates or updates for their other software (IIS or Office for example), with updates from other vendors not even being considered. However, this is set to change when the Windows Update Service is released, but will still not support other vendor's patches. Further the 'free' server can only run on the non-free Windows platform and provides inadequate and inflexible reporting[13]. This renders neither the server, client nor patches useful for non-Microsoft platforms, thus failing our criteria by not being cross-platform, not supporting multiple vendors, not being inexpensive and providing inadequate reporting. SUS is not useless however and would be far more effective as part of a patch management solution, as it has created a convenient interface from which to pull Microsoft updates.

Patchlink[14] is a cross-platform solution which claims to support all Microsoft, UNIX/Linux, Novell NetWare, and MacOS X operating systems. Update automates more than 200 vendors' upgrades, including those of IBM, Adobe, Corel, Symantec, McAfee, WinZip, Citrix, UNIX and Novell. The product costs $1,295 for a server version, plus $11 per machine per year (at a volume of 1,000 seats). This seems to fulfill most of our criteria, with others requiring testing to properly examine. Bigfix[15] is a Microsoft only solution, which differentiates itself by only sending small delta files, called Fixlets, across a network. It's $2,500 for a server plus $10 per machine annually for its Patch Manager component. These solutions do not fulfill the criteria of being inexpensive. This would have the effect of significantly reducing its widespread adoption, which may result in defended organisations becoming islands while the rest of the Internet crashes around them.

4.2 Server only

Server only solutions do not require the installation of client side agents. These usually consist of a security scanner combined with a method for pushing patches to the machines. This makes them easy to develop as the functionality of existing security scanners can be leveraged.

HFNetChkPro[16] is a Microsoft only solution but supports patches from other vendors such as Sun Microsystems. It uses the HFNetChk security scanner used by Microsoft in its Microsoft Baseline Security Analyzer (MBSA). It works by scanning for vulnerabilities and applying patches where appropriate. It is relatively inexpensive at $16 per seat. UpdateExpert[17] is very similar to the above but requires very little additional software and uses IIS and SQL and only supports patches for Microsoft. It costs $9 per seat per year for a subscription to their vulnerability database. There are several other products very similar to these such as, Security Bastion[2], Lan Net Scan[3] and Security Update Manager[4]. On the limited scope shell script front there are tools such as Quick Patch[5] for FreeBSD. This attempts to automatically install updates recommended by the FreeBSD security list.

These products, unlike the solutions mentioned above, fulfill the 'inexpensive' criteria. However they fail in a few others, the first being that they are not cross platform nor do they support many vendors. This is not intrinsic to the class however. The server only approach fails because it does not provide the level of reporting required. A security scanner or shell script can only discover a small subset of information and cannot be sure whether this information is correct, while agent based software would have far more access to the information required for robust reporting. For example a simple denial-of-service attack could

[2]`http://www.securitybastion.com/`

[3]`http://www.gfi.com/lannetscan/`

[4]`http://www.configuresoft.com/product_sum_overview.htm`

[5]`http://www.roq.com/projects/quickpatch/`

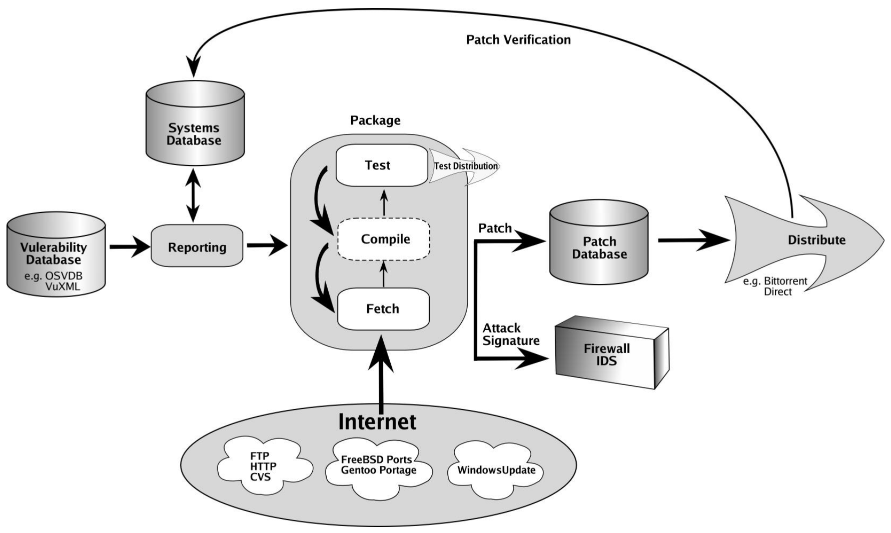
be conducted on such a system by a malicious program faking a particular vulnerable service on a large number of machines, while agent-based software would be in a better position to detect the intrusion.

4.3 Configuration Managers

This class includes complete system configuration managers such as Microsoft's Systems Manager Server (SMS) or IBM's Tivoli. These are agent based systems that allow the administrator almost complete control of every machine. With these systems, patch management would be a subset of their functionality, and is rather like swatting a fly with a Buick. These systems are far from inexpensive and don't include a homogeneous interface for installing multiple patches from multiple vendors.

5 PROPOSED SOLUTION

The proposed solution is to develop an open source system to fulfil the requirements listed in the previous section. The planned architecture is to allow for a highly modular approach. This will allow the initial system to be written for a few operating systems and methods of patching, with the first intended to be FreeBSD and its Ports system then Gentoo Linux and its Portage system. The reason for this choice is that FreeBSD's Ports system already performs much of what is needed in this project and due to its maturity will provide a good starting ground. Gentoo Linux was chosen due to the high level of development occurring within its community and by deploying the project there, it is hoped that more interest will be created in the project, thus expanding the pool of developers. Below is a graphic description of the server architecture. The client is not depicted but will consist of a small agent capable of performing the required tasks for its operating system. The packaging and agent components will be modularised. To support a new operating system on both the client and server the core agent code(client) and core packaging code(server) will require an extension to handle the OS specific tasks such as enumeration of software and creation of packages. It should be noted that this section is undergoing constant revision and input on the architecture is greatly desired.



5.1 Vulnerability Notification

This part of the project will require a method of discovering new vulnerabilities and notifying an administrator. There are several databases, such as the Open Source Vulnerability Database (OSVDB)[18] or the Vulnerability and eXposure Markup Language (VuXML) database. These databases will be examined and either a method for supporting multiple databases, which brings with it duplicate patch resolution issues, or choosing one database will be developed. Currently the OSVDB seems the best choice as it provides an XML-RPC interface and high-quality notifications.

5.2 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 in 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 then a major advantage of such a project due to the decision making benefits. The reporting will consist of a method for an administrator to check the patch level of machines and their various services across the organisation. This will be implemented in a systems database, containing the information mentioned under the 'Asset and Host Management' section described above. This database will collate information after a patch installation to enable accurate compliance reporting.

5.3 Pulling Patches

Patches are distributed over various media and the problem of how to pull patches from multiple vendors such as Windows Update, Symantec AV, FreeBSD ports and Bind's ISC is presented. To avoid this a modular approach will be taken to allow the piecemeal creation of fetching methods. Initially these plug-ins will consist of a FreeBSD ports and Gentoo Linux portage rsync plug-in, but will later be expanded to include WindowsUpdate, CVS and others.

5.4 Creating Packages

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 advantages of binary patching are 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 each other'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."[7] It is hoped that instead of a consortium, a community could be created to share their experiences.

5.5 Testing Packages and Stop-Gap protection

The same mechanisms used for distributing patches to the main systems can be used to distribute the patches to the test lab. It is very difficult to automate this stage and have it still remain effective, thus the checking will most likely be a manual task, requiring change control authorisation to proceed to final roll-out. While the testing occurs the organisation is left vulnerable and is often in a situation where it cannot turn off a critical service. Here Intrusion Detection Signatures can be used in conjunction with an Intrusion Detection Mechanism and Firewall to automatically drop the route of any malicious looking traffic. As this is a stop-gap measure the log files can be fairly easily scanned for false positives. The end goal is to have a generic IDS XML to describe an IDS signature for the exploitation of a vulnerability distributed with patches or on databases such as the OSVDB. This XML can then be translated to the relevant format for the organisational IDS e.g. Snort.

5.6 Patch Distribution

All of the current solutions discussed in the previous section distributed their patches via either a single server or several servers depending on the size of the organisation. This method is very inefficient and subjects to dangerous denial-of-service (DoS) attacks. The advances in peer-to-peer distribution should not be ignored, and an investigation into the use of Bittorrent[19] as a distribution tool will be conducted. This will have the advantage of a reduced bandwidth load on the distributing server[20], and provide greater security as many more machines will need to be compromised to distribute a malicious binary (assuming the initial upload is secure). The security can be additionally enhanced through the use of a public key infrastructure. 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.

6 CONCLUSION

From the above it can be seen that there is a definite need for a method to ease the discovery and installation stages of patching, and that current patching process is untenable. This method requires a robust testing mechanism to prevent the automatic deployment of faulty patches. These stem from the need to secure the internal network's machines due to the increasing difficulty in preventing malicious data from circumventing the firewall. This need is reinforced by the fact that there are many vendors, including Microsoft, trying to provide a solution for this problem. However these solutions are not satisfactory as they do not cater for the patch management process as a whole. Thus the framework for an open-source solution will be developed to incorporate all aspects of the process. By leveraging the support of the open source community a solution can be developed which is not only effective, but effective across the board. This solution's main differences in approach will be to integrate ideas from other parts of the computer science field to benefit the current patching process, to cater for the whole patch management process and to integrate with existing security tools to provide a holistic defense of the organisation rather than simply patching the holes.

References

[1] Microsoft. (2002, oct) Microsoft security bulletin MS02-061. Microsoft Corp. [Online]. Available: http://www.microsoft.com/technet/security/bulletin/MS02-061.mspx

[2] Microsoft. (2003, jul) Microsoft security bulletin MS02-025. Microsoft Corp. [Online]. Available: http://www.microsoft.com/technet/security/bulletin/MS02-026.mspx

[3] K. Poulson. (2001, oct) Linux worm uses its noodle. Column. [Online]. Available: http://www.securityfocus.com/news/139

[4] M. Strebe and C. Perkins, *Firewalls 247*. 1151 Marina Village Parkway, Alameda, CA 94501: SYBEX Inc., 2000, p. 226.

[5] G. Eschelbeck, "The laws of vulnerabilities," in *Black Hat Briefings*, J. Moss, Ed. 2606 Second Avenue, 406, Seattle, WA 98121 USA: Black Hat, Inc, jul 2003.

[6] (2004, jan) CERT/CC statistics 1988-2003. Website. CERT. [Online]. Available: http://www.cert.org/stats/cert_stats.html

[7] S. Berinato, "Patch and pray," *CSO Online*, aug 2003.

[8] B. Schneier, "The security patch treadmill," *Cryptogram*, vol. 35, mar 2001.

[9] J. Chan. (2004, jan) Essentials of patch management policy and practice. pmessentials.asp. @stake. [Online]. Available: http://www.patchmanagement.org/

[10] D. Voldal, "A practical methodology for implementing a patch management process," White Paper, SANS, sept 2003. [Online]. Available: http://www.sans.org/rr/papers/index.php?id=1206

[11] M. Nicolett and R. Colville, "Robust patch management requires specific capabilities," Research Note T-19-4570, Gartner, mar 2003.

[12] "Software update services deployment white paper," White Paper, Tech. Rep., jan 2004. [Online]. Available: http://www.microsoft.com/windowsserversystem/sus/susdeployment.mspx

[13] J. Hassell. (2004, feb) Automating windows patch management: Part I. Column. [Online]. Available: http://www.securityfocus.com/infocus/1760

[14] Patchlink. (2004) Vendor Website. [Online]. Available: http://www.patchlink.com/company/about.html

[15] BigFix. (2004) Vendor Website. [Online]. Available: http://www.bigfix.com/

[16] Shavlik. (2004) Hfnetchkpro. Vendor Website. [Online]. Available: http://www.shavlik.com/pHFNetChkPro.aspx

[17] St. Bernard Software. (2004) Update expert. Vendor Website. [Online]. Available: http://www.stbernard.com/products/updateexpert/products_updateexpert.as%p

[18] Open Source Vulnerability Database. (2004) Vendor Website. [Online]. Available: http://osvdb.org/

[19] B. Cohen. (2004) Bittorrent. Vendor Website. [Online]. Available: http://bitconjurer.org/BitTorrent/

[20] B. Cohen, "Incentives build robustness in bittorrent," White Paper, Tech. Rep., may 2003. [Online]. Available: http://bitconjurer.org/BitTorrent/bittorrentecon.pdf