Centralised patch management

Jelmer Barhorst & Martin Pels
University of Amsterdam

Revision 1.1 – 4th July 2005

Abstract

In large, heterogeneous networks it is hard to keep track of patch rollouts on all the different computers. This document tries to kick-start the salvation of this problem by designing a centralised, platform independent framework for downloading, approving and rolling out software updates and patches on all computers in a network.

The document first describes APT, YUM, and Windows Update, to investigate how clients on the most common operating systems normally receive their updates. After this, three existing patch management systems are discussed. These are Microsoft Windows Server Update Services, Radia Patch Manager, and LANDesk Patch Manager. The three systems are investigated on how they deal with issues from seven areas. These are:

- Patches
- End users
- Distribution
- Administration
- User interface/Framework
- Infrastructure
- Reporting

Based on the information gathered during this investigation a list with functional requirements for an ideal patch management system is composed. This list is also subdivided in the seven areas listed above. Finally, a Proof of Concept implementation for a centralised, platform independent patch management system is discussed. This implementation incorporates the most basic patch management functions, and contains a module for distributing patches to hosts that use APT for receiving updates.

Preface

This report was made on behalf of the master course system and network administration of the University of Amsterdam (UvA). It is the end result of a research project conducted over a period of four weeks in June 2005.

Acknowledgments

We would like to thank the University of Amsterdam for making this project possible. Further more we would like to thank everyone at UvA's IC[26] that assisted us during the project. Special thanks go to Frank Pinxt for the guidance and advice. We would also like to thank Marc Smet from HP. Finally, thanks to everyone who's name should be in here, but is not.

Contents

Chapter 1

Introduction

In large, heterogeneous computer networks it is hard to keep track of patch rollouts on all the different hosts (servers and workstations). Often there are numerous different operating systems and even more applications, all potentially with their own updating tools. Centralized patch management systems (PMS) are available which make the job much easier, but they generally focus only on one, or a few types of operating systems.

It would be nice to have a centralized, platform independent framework for downloading, approving and rolling out software updates and patches on all computers (hosts) in a network. The goal of this framework is to facilitate patch management for both operating systems (e.g. Windows, UNIX, MacOS) and applications (e.g. virus-scanners, anti-spyware tools). The framework uses existing update mechanisms like APT[1] and Windows Update[18] to download updates, and allows administrators to approve patches for distribution, and define system groups so that updates can be distributed using a *one, some, many*[24] approach[1].

1.1 Contents

This document starts with an introduction to three client update tools (APT, YUM and, Windows Update). After this, three existing patch management systems (Microsoft Windows Server Update Services, Radia Patch Manager, and LANDesk Patch Manager) are discussed, in order to obtain a good view on how patch management is usually handled. In chapter 4 the most important functional requirements for an ideal patch management system are described. Chapter 5 discusses a small Proof of Concept for a centralized PMS. Finally, the document ends with a conclusion and some notes on future work that may be done on this topic.

[1]First deploy the patch on one host, then a few more and if it the deployment is still successful keep on deploying.

Chapter 2

Client update mechanisms

2.1 Introduction

Before getting into the workings of patch management systems, which is done in chapter 3, it is good to first have a look at how clients normally receive their updates. This chapter discusses three of the commonly used update tools for some popular operating systems. These are APT[1], YUM[29], and Windows Update[18]. Each section describes where an update tool is used, and how it works internally.

2.2 APT

The first update tool described in this chapter is the Advanced Packaging Tool (APT). APT was originally created for the Debian[7] GNU/Linux distribution to facilitate fast and easy installation of packages, while automatically handling dependencies, and taking care of customized configuration files. APT is currently used in a large number of Linux distributions, and has even been ported[2] to support the RedHat Package Manager (RPM) system as well.

APT works by retrieving lists of available packages from one or more repositories on the Internet. Each Package file contains information on all available packages for a particular operating system version and architecture. The file also holds information on dependencies between different packages. This allows APT to discover which packages need to be installed together, and which packages may not. For every operating system version there is also a Release file available. This file lists the Package files for all available repositories and architectures for the particular operating system version, together with MD5 and SHA1 checksums for these files. The Release file is digitally signed using PGP[13] or GnuPG[8].

Packages that are selected for installation are downloaded from the repository on which they reside. Installation can be done automatically, or with user intervention.

2.3 YUM

Yellow dog Updater, Modified (YUM) is a tool similar to APT, designed for managing the installation of RPM packages on Linux. It is available for all RPM-based Linux distributions, such as the RedHat Enterprise series[22], Yellow dog[28] and Mandriva Linux[16].

While YUM and APT seem very much alike on the surface, they are quite different internally. Instead of using separate files for retrieving package information YUM uses information stored in the headers of RPMs. In order to find out which packages are available YUM downloads the headers of all RPMs that are not already installed on the system. It then uses the information stored in these headers to discover dependencies and conflicts. The integrity of the headers is verified using PGP/GnuPG.

Like with APT, installation of new or upgraded packages can be done automatically, or with user intervention.

2.4 Windows Update

The following may not come as a surprise: Windows Update only supports Windows based operating systems and applications. Windows Update is only able to distribute updates for the core of the operating system, and the applications that are distributed with the operating system, such as Internet Explorer and Windows Media Player. The newest version is called Microsoft Update and is able to distribute updates for Office, SQL Server, etc. as well.

How the update mechanism Microsoft uses works exactly is not documented very well. In 2003 Mike Hartmann made an in-depth analysis on how Windows Update works internally[12]. In short a piece of software is executed by the browser which makes an inventory of the whole system (i.e. it lists all hardware, installed software and patches) and a list with all the necessary updates is presented to the user. The user then chooses which updates should be installed.

Since a few years Microsoft developed a client-side update agent which automatically checks for new updates. Optionally, it can automatically download and install new updates without user intervention. When an update requires the host to be rebooted, the user will be notified and can choose to immediately reboot or delay it.

Chapter 3

Existing patch management systems

3.1 Introduction

The previous chapter described three well-known update clients for rolling out patches on different operating systems. This chapter takes an in-depth look at three PMS's that centralise the rollout of patches to clients. These tools are Microsoft Windows Server Update Services (WSUS), LANDesk Patch Manager, and Radia Patch Manager. These systems where investigated to obtain a good view on how patch management is usually handled. This information is used in the next chapter to create a list of criteria for an ideal PMS.

This chapter starts with a list of points on which the three systems where researched. This list may not be comprehensive, but is believed to contain the most important properties of a PMS. In the remainder of the chapter the list is applied to the three different systems. The chapter ends with a conclusion on how the three systems compare on the researched areas.

The information in this chapter was gathered from interviews, product brochures and manuals, and experience of the authors with the discussed products.

3.2 Research topics

This section describes the topics on which the three PMSs where investigated. The different subtopics are listed in order of importance, starting with the most important aspects.

The criteria discussed are:

- Patches
- End users
- Distribution
- Administration
- User interface/Framework
- Infrastructure
- Reporting

Patches

- How long does it take after a patch is released before it appears in the PMS?
- Is it technically possible to roll back patches?
- Are the source and integrity of the patches verified? (e.g. using digital signatures and checksums)
- Does the PMS support multiple platforms? (combination of hardware architecture and operating system)
- Is it possible to distribute patches for applications as well?

End users

- Is it possible for a user, and to which extent (in terms of severity and risk) to delay or reject a patch?
- How are reboots handled?
- Is a user able to initiate a rollback of a patch in cases such as incompatibly?

Distribution

- Are patches distributed using a patching agent supplied with the PMS, or using standard updating tools?
- Is it possible to prioritize updates instead of first-come-first-served?
- Does the PMS monitor mobile devices (such as laptops) so that when they reappear in the network new patches are distributed immediately?
- Does the PMS use the *one, some, many* concept for rolling out patches? (patch one host and if it is successful patch more hosts; in case of failure start over again)

Administration

- Is it possible to delay or reject patches?
- Is it possible to automatically approve new patches? (based on previous approvals and/or severity and risk)
- Is it possible to add pre- and postscripts to patches? (i.e. to update configuration files)
- Is it possible to add custom patches? (i.e. patches for in-house developed software)

User interface/Framework

- Is the interface fool-proof?
- Is the framework protected by some form of access control?
- Is it possible to backup and restore all settings, historical data and cached patches?
- Does the framework have an Application Programming Interface (API) for customization and/or new (custom) modules?
- Are platforms and patches linked to CVE[6], BugTraq[23] or other security sources?
- Is there a web-interface available?

Infrastructure

- How is the load on the network infrastructure handled? (unicast, anycast, multicast, peer-to-peer, etc.)
- Are multiple update servers possible? (chaining, load balancing, fall-back, redundancy, etc.)
- Is it possible to cache updates locally?
- How does the PMS deal with low bandwidth users?
- How is an inventory of the hosts in a network made?
- How does the framework integrate with an existing infrastructure? (LDAP, NIS, Active Directory)

Reporting

- Does the client give feedback? (i.e. the distribution of a patch is stopped after a certain number of reported failures)
- Does the PMS generate alerts when a new (critical) patch is available?
- Is the PMS able to produce different types of reports?

3.3 Windows Server Update Services

Microsoft's Windows Server Update Services (WSUS)[17] is a PMS that can be obtained and used free of charge and was released recently. The release date is almost three years after the release of its predecessor Software Update Service (SUS) which was a relatively basic system. In this chapter most of the features WSUS offers will be described.

3.3.1 Patches

Since WSUS is a Microsoft product it only supports Windows and other Microsoft applications. Therefore updates do not have to be approved by a third-party vendor.

Release time

Because all patches come directly from Microsoft the time between release and appearance in WSUS is directly linked to the configured synchronization frequency. The minimum frequency is once a day, so the maximum delay after release is 24 hours.

However, if necessary, the administrator of WSUS can initiate a manual synchronization.

Rollback

Microsoft uses a software installation and update mechanism which is called 'Windows Installer'. This unified mechanism also supports the concept of rolling back patches. However, when the update does not support a rollback, WSUS can only stop further deployment..

Verification

All recent software released by Microsoft, thus including updates, are digitally signed. When a patch is downloaded from the Microsoft server or another WSUS server, the digital signature is verified. When someone has been tampering with the patch, it shall not be distributed through the network.

Platform support

WSUS is a product developed by Microsoft and therefore it only supports their platforms. The supported platforms are Windows 2000 Service Pack 3 (SP3) and later, Windows XP and later and Windows Server 2003. All earlier platforms such as Windows NT4, Windows 9x, etc. are no longer supported by Microsoft and therefore it is logical that WSUS does not support them as well.

Application support

The architecture of WSUS allows it to update applications as well as the core of the operating system. These applications are however limited to Microsoft's own applications and it is not known if the company is going to allow other software vendors to distribute their updates and patches through WSUS.

3.3.2 End users

An agent that is installed on every host controls all user interaction. Most users will only be confronted with a warning that the host has to be rebooted, because unprivileged users can not influence which updates are installed.

Delay/rejection of patches

When a user is logged in he can not delay or reject patches. They are just installed, if he likes it or not. When the user has enough privileges (e.g. is a local administrator) he can control which updates are and which are not installed. In most cases users will not have such privileges.

Reboots

The agent has several options when it comes down to rebooting. The first option is to delay the installation of updates which need a reboot until the user shuts down the system (i.e. at the end of the day).

When installation of updates is scheduled and there are updates which need a reboot and no users are logged in, the host will be rebooted automatically. It is possible to disable this setting, because servers for example have remote users who do not expect a server to be rebooted in the middle of a day.

If there is a user logged in he is confronted with the message that the host needs to be rebooted and a question whether the reboot should be done immediately. When the user does not want to reboot at that moment and go on with his work (or just to save files he is working on) the same question will be asked an amount of time later. This period can be configured by the administrator.

Initiating rollbacks

For non-administrators it is impossible to remove an update. When the situation arises that it is necessary the WSUS administrator should be contacted to exclude the host from that specific update. However, this is not always possible since not all updates Microsoft has released can be rolled back.

3.3.3 Distribution

Distribution of patches with WSUS is not very complex, but for a lot of companies that want centralized patch management it should be more than sufficient.

Distribution tool

In the previous section is already made clear that an agent is used. This agent is not specifically for WSUS, but it is also used to download updates directly from Microsoft.

The configuration of this agent can be done locally on every host, but when an Active Directory is used the same configuration options are available in a so called Group Policy (GPO). Using GPO's, it is possible to define global settings that apply to all hosts and specific settings for (groups of) special hosts such as servers. The settings that can be defined cover scheduling of the installation, location of the WSUS server, the frequency for detection of new updates and how reboots should be handled,

Prioritization

The mechanism used by Microsoft handles updates by the first-come-first-served concept. It is however possible to specify a deadline before when an update must be installed.

Mobile devices

WSUS is a passive system (i.e. it does not monitor hosts actively) in which the agent must initiate communications with the server.

Distribution technique

The concept of *one, some, many* is not used by WSUS. Instead updates can be approved for detection only. By doing this one can easily see what the impact of an update is. Because WSUS allows hosts to be categorized in groups, one can decide to rollout an update on a pilot group. If there were no problems encountered in the pilot group the update can be approved for all hosts in the network.

3.3.4 Administration

Administration of WSUS is relatively easy. This is partial because of the interface which will be discussed in the next section, but mostly because WSUS only distributes patches and nothing more than that.

Delay/rejection of patches

The fundamental feature of a PMS to reject patches was already present in SUS. Microsoft has enhanced this so that patches can be approved for detection only or installation on a per group of hosts basis.

Another nice feature is that older versions of patches are automatically rejected when a new version is approved.

Automatic approval

It is possible to let WSUS automatically approve patches for detection or even for installation. This can be specified for each group of hosts.

Custom configuration files & patches

Only the patches that originate from Microsoft can be distributed. Therefore it is not possible to add custom patches or scripts.

3.3.5 User interface/Framework

As to be expected, the framework has to be installed on a Windows server and can only be accessed using the Internet Explorer. It uses the Internet Information Services and a somewhat limited version of SQL Server.

Usability

As expected Microsoft has done a decent job building a pretty easy to understand, unambiguous and straightforward user interface.

Access control

Authentication for the WSUS user interface uses the mechanisms of Windows. This means that when the WSUS server is a part of a domain, the central authentication database is used to verify the credentials the user supplied. It should be noted that WSUS sends the passwords in clear text to the server. It is highly recommended to enable the use of a encrypted connection as soon as possible.

Backup

WSUS uses Microsoft SQL Server to store all information and a directory on the file-system to store all locally cached updates. Therefore backups can be made easily using default available tools.

API

There are API's available for both WSUS and the agent. Those API's can be used to build almost everything not included in WSUS.

Links to vulnerability lists

The management interface provides the administrator with information about updates. With most updates comes a link to the Microsoft website where more detailed information can be found. For most security updates a link to CVE is included on that same website.

Web-interface

The whole user interface is implemented as a web-application. Unfortunately it only works with the Internet Explorer on a Windows host.

3.3.6 Infrastructure

If deployed right WSUS can be very scalable while not supporting advanced distribution techniques like multicast and peer-to-peer.

Network load

The communications between a WSUS server and an agent is always unicast. No advanced techniques are used to reduce the load on the network.

Multiple servers

To reduce the load on a single server or network segment, WSUS supports chaining of multiple servers. The disadvantage of this is that reports covering all servers and the hosts served by them are not available.

Local caching

All patches that are to be distributed to the clients are cached locally on a WSUS server.

Low bandwidth users

For downloading the patches from the WSUS server the agent uses a framework called Background Intelligent Transfer Services (BITS). This framework guaranties that downloading the patch won't consume all available bandwidth. Furthermore, when the connection with the WSUS server is lost, BITS can restart the download at the point where the connection was lost. It is also possible to define time constraints. For example, one could configure BITS to use a minimal amount of bandwidth during office hours and outside these hours to use all available bandwidth.

Inventory building

Finding out which hosts should get their updates from the WSUS server is not part of the server itself. Instead, an administrator has to configure each host by hand or by creating a GPO in Active Directory.

3.3.7 Reporting

Functionality that SUS lacked and what most administrators wanted to have were reporting capabilities. Microsoft has included this in WSUS, but alerting is still not available.

Client feedback

When an update is installed, the agent connects to WSUS and reports if the installation was successful or not. Internally the WSUS server updates the status of the host with the provided information which can be used to generate reports.

Alerting

By default there is no option in WSUS available to send an alert to the administrators when new updates are available or when the installation of updates failed. However, if this functionality is wanted, one could write a program using the WSUS API that implements alerting.

The only way to find out if there are new patches is to browse to the web-interface.

Report diversity

The most important enhancement since SUS is the ability to generate reports. Besides a summary of all settings and the report on how many new updates became available in a certain period there are two important types of reports. The first is a report which uses an update as basis and see which hosts or groups of hosts have installed it or not, which need it or not, and if there were failures. Another report is available which uses the host or group as basis and lists the updates by the same criteria.

It is however not possible to let the system e-mail a report on a regular basis to administrators and/or managers.

3.4 Radia Patch Manager

Radia Patch Manager[9] was originally developed by a company called Novadigm which was bought by HP a while ago. Radia Patch Manager is part of a much larger system and is available as an extension. During the investigation of this PMS a consultant of HP was available to answer most questions about the system.

3.4.1 Patches

Radia uses the concept of feeds. These feeds are XML files obtained from several sources, namely a local one, one from HP and the vendor's own feeds. The order of the feeds as listed here is also the order in which Radia uses them. The HP feed contains fixes for vendor feeds miss tags or have invalid characters, in which case the XML file is invalid.

Radia supports any operating system, as long as a feed and a client are available for the operating system and architecture.

Release time

The feeds are by default monitored for updates on a weekly basis. In most cases this is sufficient, because Microsoft for instance releases patches only once a month at the moment. This means that when the vendor releases a patch, it will be available in Radia after one week at most.

The fixes made by HP which are published through their feed are available within one and a half week. This adds a maximum extra half week between a patch release and its incorporation in the PMS.

Rollback

Radia supports the concept of rolling back patches. According to the manual[10] this is possible for all Windows versions. For RedHat Enterprise Linux, which is also supported by Radia, this is not possible due to the fact that the RedHat Package Manager (RPM) does not support it.

Verification

If the patch contains a checksum, it is validated. For most patches, this is the case.

Platform support

As previously said, Radia supports Microsoft Windows and RedHat Enterprise Linux. In the future Novell SUSE Linux will be supported. At this moment only x86 based platforms are supported.

Application support

Since Microsoft recently started distributing patches for its Office suite via the same mechanism that is used for the Windows Updates, support for it is currently being built into Radia. Because of the nature of RedHat's package management updates for applications are already distributed to the clients.

3.4.2 End users

End users have no influence on Radia's patch management. Any changes a user makes will be undone by the system. As for reboots, different reboot policies can be instated for different patches.

Delay/rejection of patches

If a user has administrator privileges on his host patches can be uninstalled (if the patch supports it). However, when the Radia agent synchronizes with the server this anomaly will be detected, reported and fixed (i.e. the patch will be installed again). This behavior is a result of the fact that the whole Radia suite has a concept that is called 'Desired State', which means that every client exactly reflects the configuration as specified in Radia.

A user can of course prevent the whole process of rolling out patches by not connecting to the network or removing the Radia agent.

Reboots

Within Radia there are several ways to handle reboots. It is possible to specify on a per patch basis whether the host has to be rebooted and if that is the case it is possible to specify whether the user can save his work or that the host should immediately reboot without giving users this ability. Besides these types of reboots it is possible to give the user a warning that the host should be rebooted. The user can optionally be presented with the choice to cancel the reboot so he can just go on with his work. The last option the administrator can set is a timeout which will automatically approve the warning message that was presented to the user. This can be useful when the user is out of office and is not able to respond to the warning message when a reboot is necessary.

Linux users are not warned in the way described above. If a reboot is scheduled, it will be performed without giving the user time to save their work.

Initiating rollbacks

As described previously, a user is practically not able to reject or roll back a patch. Instead, the so called 'Desired State' of the host has to be altered by an administrator.

3.4.3 Distribution

Radia uses a straightforward mechanism for distributing clients. Prioritization is not possible, and updates are only spread if a client connects to the server.

Distribution tool

As may be clear by now, Radia uses a agent that is installed at every host. This agent makes an inventory of the whole system (e.g. software, installed patches, etc.) and reports this to the Radia server. Software and patches that are not installed when the inventory is compared to the Desired State for that host will be downloaded and installed.

Prioritization

Updates and patches are rolled out in the same order as they are acquired. Prioritization of critical updates is not possible.

Mobile devices

Radia does not actively probe hosts. It waits for them to connect to the Radia server themselves.

Distribution technique

Radia does not use the *one, some, many* approach. Instead, a pilot group is defined for impact analysis and pilot testing. Radia has a subsystem called the Configuration Analyzer. With this program an administrator can easily see if programs and patches potentially conflict.

Once a patch is scheduled for distribution it will not be automatically stopped by Radia when there are problems.

3.4.4 Administration

Radia is very flexible when it comes to administration. Features like automatic scheduling and creating custom patches are available.

Delay/rejection of patches

Since every patch or update has to be approved before they are being rolled out, delaying or rejecting of them is possible.

Automatic approval

Radia is completely built as a framework that is based on policies. It is possible to create a policy that automatically approves patches.

Custom configuration files & patches

Adding in-house developed patches or custom configuration files is possible by using the local feed.

3.4.5 User interface/Framework

Radia's user interface is probably its weakest feature. While powerful, it is hard to use. The framework offers enough flexibility.

Usability

Setting up the whole Radia framework is hard and difficult. Therefore it is advisable to hire someone with a lot of expertise.

The user interface in which everything can be controlled is only accessible by a tool which at first looks a bit like the Windows registry editor (Regedit). An easier to use web-interface is available, but while having the most basic functions available it is by far not as powerful.

Access control

Access control can be completely integrated with an existing infrastructure, for example with LDAP or Active Directory.

Backup

Backing up the data stored in the system is possible. The two databases used by Radia are easy to backup using standard tools. The feeds are also stored locally, so the latest version is always available.

If the status of a host is lost it is not possible to view reports which cover that information. This will be resolved as soon as the agent that is installed on the host reconnects.

API

There is a Java based API available to write programs that are able to communicate directly with the configuration server. This server is the pivot of the whole Radia framework.

Links to vulnerability lists

Both Microsoft and RedHat are compatible with CVE and provide this information in their respective feeds. The information is accessible via the web-interface.

Web-interface

As said, there is a web-interface in which it is possible to acquire new patches, approve them and see status reports. For daily administration this should be sufficient, but for more complex maintenance the web-interface is not usable.

3.4.6 Infrastructure

The infrastructure of Radia is quite advanced. It allows for things like multicasting, caching, and deployment of multiple servers.

Network load

Radia has functionality to distribute patches and software via multicast to multiple clients at once.

Multiple servers

The whole Radia framework consists of several components (databases, services, etc.). These can all be distributed to different servers. It is also possible to add multiple servers which deal with the clients after a master server has redirected the client.

Local caching

The patches are downloaded from vendors, and stored as objects in the central database.

Low bandwidth users

To conserve bandwidth Radia uses several techniques. The first one is bandwidth throttling. This means that only a limited amount of the available bandwidth will be used for downloading patches, and when an application needs the bandwidth used by Radia it will be made available.

Furthermore, it is able to restart the download of a patch when, for example, during the download of it the connection failed. This prevents that a patch must be downloaded from the beginning after the download of the first piece could not be completed.

Inventory building

An inventory of all hosts a company has is not made automatically. A host is added to the system after installing the agent. Installing it can be done by pushing it to a host using a Remote Procedure Call (RPC) based mechanism.

Integration

Radia is able to use LDAP based sources to gather information for authentication and targeting patches to specific (groups of) hosts.

3.4.7 Reporting

Reporting and alerting features are well implemented in Radia.

Client feedback

As said previously, the agent compares the current state of a host with its 'Desired State'. Anomalies that are found will be shown.

A report to the Radia server will also be sent if the installation of a patch fails.

Alerting

Radia is able to send alerts by means of SNMP to a central server which can do further processing and alerting.

Report diversity

Reports on all subsystems of Radia are accessible via a local reporting server if this server is available. By default there are reports available on acquisition, background information on patches and, most importantly, a view of all devices.

3.5 LANDesk Patch Manager

The third PMS described in this chapter is LANDesk Patch Manager[14]. This PMS is part of the LANDesk Management Suite, which allows for central administration of not just patches, but also things like licenses and configuration information. The Patch Manager can also be installed separately.

3.5.1 Patches

LANDesk is different from the other two PMSs in that it does not distribute patches in their original form. Instead, LANDesk builds standardized packages for all operating system and application patches. These packages are then downloaded by the Patch Manager. Management of patches on a host is done using a supplied client agent.

Release time

Because patches are not downloaded directly from the vendor it takes longer for patches to reach the hosts in a network. LANDesk promises a maximum of 24 hours between the release of a new patch, and the availability of a LANDesk package for this patch.

It is also possible to download the patches directly from a vendor, and then distribute them. However, this approach requires quite a lot of maintenance, and disables some features in the client agent, like rolling back of patches.

Rollback

If LANDesk packages are used to distribute patches to clients it is always possible to roll the host back to its previous state. The agent stores all original versions of changed files to achieve this. It is even possible to rollback patches that are normally impossible to roll back (e.g. some Windows updates). It is however not recommended to perform a rollback of these patches, since there is no vendor-support available in case the rollback is unsuccessful.

If normal updates are used instead of LANDesk packages, it is not possible to perform a rollback using the client agent.

Verification

All packages that the client agent receives are verified on file-level. This means that for every file in the package the checksum is verified.

Platform support

LANDesk supports a variety of operating systems. Windows, Macintosh OS X and the major RPM-based Linux distributions (RedHat Enterprise, SUSE, Mandriva) are supported. Unfortunately, since LANDesk uses its own agent software, only the most commonly used architectures are supported.

Application support

LANDesk is able to roll out patches for applications. For which applications patches can be rolled out depends on the packages LANDesk provides. Currently these are only Microsoft applications for Windows, and only applications from default supplied RPMs for the Linux distributions.

3.5.2 End users

All hosts in the network are equipped with the client agent. This agent requires no user-intervention to operate, and gives users very little influence.

Delay/rejection of patches

One of the things the user does not have influence on is the installation of patches. It is not possible to delay or reject installation, except by turning off the agent (if the user's rights permit this), or disconnecting the host from the network.

Reboots

LANDesk has a wide number of options on the field of handling reboots. The following options are available:

Ask Users are given a dialog screen which asks if they wish to reboot. This allows users to save their work first.

Never reboot An automatic reboot after installation of patches is never performed. This introduces the risk that patches are activated days or weeks after their initial installation.

Reboot for all patches An automatic reboot is scheduled after installation of all patches.

Only if needed An automatic reboot is only performed if this is required by one of the patches that was installed.

Which reboot setting is used can be set for a group of systems globally, or in the client agent on a host. The setting on the client overrides the global setting. This local setting can, unfortunately, not be changed centrally.

Initiating rollbacks

Users are not allowed to initiate the rolling back of patches installed on their systems. They may be able to manually roll back a patch by editing or removing files. This is however corrected by the client agent the next time it connects to the central server to look up the list of needed patches.

3.5.3 Distribution

LANDesk uses a sophisticated mechanism to distribute patches among clients (more on this in section 3.5.6), and communications between clients and the server can be set up in multiple ways.

Distribution tool

As described earlier, LANDesk does not use the standard update tools of operating systems and applications. To distribute updates it uses its own client agent.

Prioritization

It is not possible to prioritize updates so that more important updates are installed first. It is however possible to set dependencies, so that specific updates are installed before others.

Mobile devices

LANDesk has two different methods to recognize a host that enters the network. The first method is simply waiting for a client agent to connect to the server. If the host enters the network the agent looks for the central server to check if any new patches need to be installed.

The second method LANDesk uses is periodic polling. The system can do a periodic sweep of the network, looking for active client agents.

Distribution technique

LANDesk allows administrators to define different groups of hosts. This makes it possible to identify pilot groups to test patches on before they are rolled out onto groups of production hosts.

Once installation of a patch on a number of systems is scheduled, the *one, some, many* approach is not applied. If ten hosts are scheduled to receive a certain patch, and installation fails on one of them, the rollout will still continue on the other nine.

3.5.4 Administration

Administrators have several options to influence the rolling out of patches in LANDesk. These options are described here.

Delay/rejection of patches

As to be expected, LANDesk supports one of the most fundamental features of patch management, namely the delaying and rejecting of patches. Delaying or rejecting a patch can be done for all systems, or for specific groups of systems.

Automatic approval

LANDesk is not able to automatically approve newly available patches based on criteria as previous approvals or severity of the bug that a patch fixes. All patches need to be approved by hand by an administrator.

Custom configurationfiles & patches

It is possible for an administrator to create custom LANDesk packages, to roll out versions of patches with different configuration files, or to roll out patches for applications that are unsupported by LANDesk. Building custom packages can however be a high-maintenance job.

3.5.5 User interface/Framework

While clients using different operating systems are supported, the server part of LANDesk only runs on Microsoft Windows servers. It uses a database to store information on patches, and is generally managed through a desktop application on the server.

Usability

While not as easy as the interface of WSUS, LANDesk's user interface is quite straightforward. Performing the most common tasks does not require much practice, and there is enough documentation available to help an administrator with the rest.

Access control

LANDesk can be access controlled in two ways. The first is a simple password. The second is using Windows NT's built-in authentication features. This allows administrators to control which users in the domain are allowed to log on to the LANDesk server.

Backup

LANDesk does not provide a backup facility itself. It is however quite easy to do a manual backup of the system. This can be done by making a dump of the database the tool uses, and backing up two directories on the file-system that LANDesk uses.

API

An API for LANDesk to add custom features is not available. The system does offer a large number of options to communicate with it using scripts. This allows an administrator to, for example, implement functionality for alerting.

Links to vulnerability lists

LANDesk Patch Manager only downloads and distributes patches that are delivered to it from LANDesk. It does not directly monitor vulnerability lists for problems in applications it manages. LANDesk packages are however always supplied with a link to related information on vulnerability lists. It is therefore possible to click on a downloaded patch and view the related security bulletin.

Web-interface

The Patch Manager is shipped with a web-interface included. The preferred method of use is however the desktop program installed on the server. This program offers more flexibility than the web-interface.

3.5.6 Infrastructure

To be able to manage a large number of clients in a network LANDesk uses a number of features to reduce network and server loads. These are discussed here.

Network load

In order to distribute patches to a large number of clients, LANDesk uses a technique called Targeted Multicasting[15]. When using this technique the LANDesk server selects a client in an IP-subnet as subnet representative. The subnet representative downloads packages, and distributes them among all hosts in his subnet, using multicast.

Multiple servers

LANDesk uses so-called Core Servers to manage patch distribution to clients. These Core Servers are able to handle a maximum of 2000 clients. If more clients need to be managed it is possible to set up a Roll-up Server. This server holds a copy of the data on the Core Servers. This is useful for reporting features.

It is unfortunately not possible for the Roll-up Server to control the Core Servers. This means that patch distribution needs to be scheduled on each Core Server separately, or on the Roll-up server only, in which case the Core Servers are not used.

Local caching

LANDesk caches all patches in a local directory. This directory is accessible through HTTP and SMB.

Low bandwidth users

To facilitate the rolling out of patches to low-bandwidth hosts (e.g. laptops connecting through GPRS) LANDesk uses a throttling mechanism to distribute the patches. This way only a small subset of the host's connection is devoted to the patch rollout.

Inventory building

LANDesk's patching system uses three techniques to make an inventory of clients on the network. The first is browsing an Active Directory on the network to look up hostnames. The second method is simply scanning a network subnet for the existence of clients. If a host is found without a client installed, the client is pushed onto this host using a RPC-based mechanism. The third method of discovering a client is waiting for it to connect to the server after an administrator installed the client agent.

Integration

The LANDesk system integrates only partly with an existing infrastructure. For example, it uses Active Directory to look up the names of hosts. But no other available information is used. Also, the system does not store any information in the Active Directory.

3.5.7 Reporting

On the field of reporting LANDesk provides the most common types of reports. Unfortunately, alerting features are not available.

Client feedback

LANDesk's client agent is able to notify the server of a patch that failed to install. Unfortunately, the distribution process is not very intelligent. The server does not abort on a failed installation, nor does it send out an alert. What it does is infinitely retrying the installation, even when it fails.

Alerting

As said, no alerts are sent out when a patch installation fails. Also, no alerts are sent when new patches are available and need approval for distribution. LANDesk does however offer extensive support for custom scripts, so it may be possible to create a custom-built alerting feature, for example by directly querying the SQL database.

Report diversity

LANDesk is able to generate reports on which hosts are in the network, and what their patching status is. It is also able to report on the distribution status of specific patches. Reports can be scheduled and generated automatically, and can be published in various formats (e.g. HTML, PDF, DOC).

3.6 Conclusion

Table 3.1 shows a summary of how well the three PMSs perform on the areas listed in section 3.2. Since giving an in-depth product comparison was not the main goal of this document, the conclusions listed here provide only an indication on which of the system performs best. A decision on which of these systems one should use in an organisation should therefore not be based solely on these conclusions. Grading of the different topics is done using `++`, `+`, `0`, `-`, and `--`. Logically, `++` is best, and `--` is worst.

As the table shows, all three systems perform quite well on all covered areas. WSUS is the most user-friendly system of the three. That, and its extensive API make it the best system on the area "User Interface/Framework". However, on the topics reporting and infrastructure it lacks behind on the other two systems.

Area	**WSUS**	**Radia**	**LANDesk**
Patches	+	+	+
End users	+	++	++
Distribution	0	+	+
Administration	+	+	+
User interface/Framework	++	0	+
Infrastructure	0	++	+
Reporting	0	+	0

Table 3.1: PMS comparison

Radia is the most advanced system of the three. The large number of reboot options for clients, and the 'Desired State' concept make it score high on the areas "End users" and "Infrastructure". The large feature-set does however make the system complex, which reduces its user-friendliness.

LANDesk, like Radia, is very feature-rich. It is however not as advanced on some points. A good example of this is the handling of large numbers of clients. LANDesk's reporting capabilities are also not as good as Radia's. These deficiencies are compensated by its user-friendliness.

Both Radia and LANDesk have an advantage on WSUS in that they support multiple platforms, instead of just Windows-based operating systems and applications.

Which of the three systems should be used in a production environment is largely dependent on the specific situation within an organisation. For example, if an organisation only deploys Windows hosts WSUS is the better choice, and if the organisation deploys several thousands of hosts with varying operating systems Radia may be the system of choice.

Chapter 4

Ideal patch management

The previous chapter discussed three different PMSs, in order to find out how patch management is usually done. In this chapter the information gathered during this research is used to set up a list of requirements for an ideal patch management. The requirements are grouped according to the same areas used in chapter 3.

4.1 Patches

This section is about the patches itself. The area's covered are the acquisition of patches, how patches are rolled back, how patches are validated and why multiple platforms should be supported.

4.1.1 Acquiring

A PMS can acquire newly released patches in three different ways. The first method is downloading the patches directly from the vendor of the application for which the patch applies. This is usually the case for products like virus-scanners. New virus definitions generally do not affect a system's operating system or any other installed applications. Therefore it is usually safe to download these updates directly from application vendors and distribute them immediately, using their own update mechanisms. The biggest advantage of this method is that patches can be acquired shortly after their release, instead of having to wait for a third party to approve the patch. The disadvantage is that a PMS needs to support the update mechanisms for all different applications that are deployed in a network and require an updating mechanism. This may become hard to maintain.

The second method for acquiring patches is to retrieve them through the update mechanism of the operating system the affected application runs on. This acquisition method is used widely for patching operating systems and applications provided with it. It is also often used to acquire updates for separate applications from the same vendor as the operating system (e.g. Microsoft SQL Server and Microsoft Office). Patches that are downloaded through this method are sure to have been tested for compatibility with the operating system, and other applications that are provided with the operating system by default (e.g Windows Media Player). Another advantage is that the PMS only needs to support a single update acquisition mechanism per operating system. A disadvantage of this method is that newly released updates and patches take longer to make it to the PMS, since they need to be approved by the operating system vendor first.

The third method of acquiring patches is to download them from a third party that tests and repackages the update. This is usually the vendor of the PMS. LANDesk uses this method.

This method has two main advantages. The first is that more compatibility tests can be performed, including tests with applications that are not distributed with an operating system by default. The second advantage is that because the third party can repackage all patches and updates only a single acquisition mechanism needs to be built into the patch management system.

Unfortunately, because patches are tested more thoroughly, and need to be repackaged, getting a newly released patch to the PMS using this method may take even more time than if the second method is used. This may be unacceptable for critical patches. Another disadvantage is that relying on the vendor of the PMS for updates introduces the risk of vendor lock-in.

Looking at the advantages of the three methods described above one can conclude that the second acquisition mechanism is most suitable for use in a centralised PMS, and should be used where possible (some applications may not support this method). Despite the fact that the third method described allows for better testing and simpler acquisition, it is not the best choice because of the added time between release of a patch and its actual availability, and the risk of vendor lock-in.

4.1.2 Rollback

There are situations in which a patch that is rolled out on a host needs to be pulled back. This may be because the patch conflicts with other installed applications, or because it introduces unwanted changes in the patched application. In such situations the PMS should provide the possibility to initiate the rollback of the patch. It may however not always be possible for a PMS to provide this possibility, since some patches introduce such dramatic changes that rolling back these patches is practically impossible. This is currently the case for many patches for Microsoft Windows.

4.1.3 Verification

When patches are downloaded and installed onto hosts it is important to verify that the patches have not been altered during downloading (accidentally or on purpose), and that the patches do in fact originate from the vendor that provides them. A good PMS should therefore check the integrity of downloaded patches, and verify that the patches have been downloaded from the actual vendor. To do this the PMS may provide mechanisms such as checksums or digital signatures.

4.1.4 Platform support

Very few companies today deploy completely homogeneous networks. Often, many different operating systems and architectures are deployed. An ideal PMS supports update mechanisms for all these platforms. By integrating the patch management of all these platforms into one system, administration can be much easier.

Since it is impossible to incorporate update mechanisms for every operating system and architecture on the planet it is advisable to include the most commonly used platforms, and provide the possibility of adding modules for other platforms. The latter may not be necessary if a working system is already available for an obscure platform. Building a custom module to support just a few hosts may be more work than simply using a second PMS to manage these hosts.

4.2 End users

The users of hosts present in a network just want to do their job. Therefore they do not want to be disturbed by silly pop-ups telling them they should save their work and reboot their workstation. Sometimes users also want to delay or even rollback patches. This section covers these subjects.

4.2.1 Delay/rejection of patches

A PMS could give users the option of delaying or rejecting the rollout of a patch on their computer. Users may want this option if they know that a certain patch will break self-installed applications that they use. Having users install their own applications is however not a good idea. If they need a specific application they should take to their administrator to have it installed. In that case the application will also be incorporated in the testing of a patch, making the option of user-controlled delays and rejects unnecessary (and unwanted).

Users should of course be able to contact an administrator if they find out that an installed patch introduced problems.

4.2.2 Reboots

Some patches require that a host is rebooted after installation. This of course affects the ability of a user to work. Reboots of hosts can be handled in different ways. The first possibility is to force the host to reboot after installation of a new patch. This might be a good solution if a critical patch needs to be rolled out as fast as possible (e.g. in case of a virus outbreak), but does not give the user the ability to save their work, which could lead to major grievance.

A second option to handle reboots is to give users a warning message that their host needs to be rebooted. Users are then able to save their work and click OK to reboot their host. The downside of this is that users can postpone a reboot indefinitely, which may be a problem when the installed patch is critical. A solution to this would be setting a deadline, so that the host will be rebooted ultimately at a certain time.

A third option is not to bug users with reboots at all, and postpone a reboot till after office hours. More like the previous option this may cause unnecessary security-hazards in case of a critical patch.

An ideal PMS should have the possibility to use all three of the above options. An administrator should be able to select the option that applies best for each patch, so that patches are rolled out as fast as possible if this is necessary, but the ability for users to work is not affected if there is no absolute need for it.

4.2.3 User initiated rollback

As described above users should not be able to delay or reject a patch themselves. This is also advisable for the rolling back of patches. Like with the delaying or rejecting of a patch the PMS should not allow the user to initiate a rollback. They should be able to notify an administrator that a certain patch caused problems. An administrator can then decide if a rollback is necessary.

4.3 Distribution

In this section is discussed whether or not a client-side agent should be installed, in what order and for which (group of) hosts patches should be scheduled for distribution, and how mobile devices such as laptops should be handled.

4.3.1 Distribution tool

Two forms of distribution can be used to download patches to desktop hosts. The most straightforward approach is to use standard update tools, provided by the installed operating system or application (e.g. Windows Update, APT, or virus-scanner update tools). The advantage of this is that no additional client software needs to be installed on the system. A disadvantage is that the standard applications are very diverse, and do not always have reporting and rollback options.

The second possibility is to provide a client agent with a PMS. This client agent can communicate with the patch management server, report on installed software and patches, and facilitate rollbacks. Disadvantage of this is that extra software needs to be written to create agents that run on the different platforms. Especially for uncommon architectures it may not be so easy to add the platform to the PMS.

Ideally a PMS uses a combination of the two approaches. It can use the standard available tools to perform normal updates, so that rolling out patches is always possible, even on an architecture for which no client application is available. For the most common platforms a client application should be available, so that reporting and rolling back patches are possible on these platforms.

4.3.2 Prioritization

One can think of situations where a new, important patch needs to be rolled out fast, but where this is impossible because a number of older patches is still pending for installation, or waiting for a reboot. A good PMS should support differentiation between ordinary updates that are handled on a first-come, first-served basis, and high priority updates that demand immediate installation. If multiple updates of different priorities are installed, the method of rebooting for the patch with the highest priority should be used. See section 4.2.2 for a description of the different reboot types.

4.3.3 Mobile devices

Ideally, new hosts that enter the network are placed into a quarantine network. This is a separate network, separated from the normal network. Hosts on the quarantine network have limited, or no access to the normal network and the rest of the world. A quarantine network is used to check and fix the patching status of a host before it enters the normal network and is able to infect other hosts.

The easiest method of monitoring the network for new hosts and getting them patched is to wait for a connection from a client agent. This approach will however not work if the new device does not have this client installed. These devices may be discovered as soon as the user logs on to the network's domain. The PMS can then install the client agent using a push method. Currently the only platform capable of this approach is Microsoft Windows, which offers Remote Procedure Calls that can be used for this purpose.

4.3.4 Groups

Any reasonable PMS should be able to differentiate between different groups of hosts. If there are different groups it is possible to define different rules on how patches are distributed. It would be good practice to at least create two groups: one group containing all servers, and one holding all workstations. This enables an administrator to roll out certain patches only to a select number of hosts, or even let some patches be automatically distributed to workstations, but require a manual approval for servers.

Groups should be organized in a hierarchical way so that subgroups can exist.

4.3.5 One, some, many

Patches should be rolled out using the *one, some, many* approach as explained by T. Limoncelli and C. Hogan in their book *The Practice of System and Network Administration*[24]. Following this approach, patches are first installed on one host for testing. If the test is successful the patch is installed on a small number of hosts. This continues in groups with increasing size, until all hosts have been successfully patched. This way, if a patch fails it is not necessary to roll back the patch on all hosts. The most important advantage is that the remaining hosts still work and the business can continue without (much) interruption.

Ideally, the PMS also uses this approach for rolling out patches within a group, as explained previously. In this case, it is not advisable to halt the rollout of patches immediately after one failed installation. A better approach is to halt the rollout if a certain amount of installations fails (e.g. more than 5 percent).

4.4 Administration

This section is about approving and rejecting, customizing and adding custom patches.

4.4.1 Approving patches

A PMS should allow the possibility to delay or reject the rolling out of patches to hosts. This functionality is necessary to give administrators the time to solve things like compatibility problems with a patch and other software used by the organization, while still being able to roll out patches released after the conflicting patch. This is of course only possible if subsequent patches do not depend on the conflicting patch.

To ease the burden of patch management on administrators it may be desirable to automatically install some types of patches. This may be patches that only provide small changes, or patches that fix serious security issues. While automatic installation eases the burden on administrators, it also comes with some serious risks. If automatically installed patches turn out to cause problems there may not be an administrator around to halt the rollout, issue a rollback and fix the problem. Halting the rollout should not be an issue when the previously described *one, some, many* approach is used.

Conclusively, a PMS may provide the possibility of automatic installation, but this option should preferably only be enabled for hosts on which patches are tested, and never in a production environment.

4.4.2 Pre- and postscripts

It is possible that patches supplied by vendors cause problems on some hosts. Also, administrators may want to replace default configuration files in patches with files containing customized settings. For this reason it is desirable that a PMS supports the option of distributing scripts along with the patches to fix these issues. Modifying the patch itself is not advisable, since this will change the checksum or digital signature of the patch, which may cause the client to reject the patch.

4.4.3 Custom patches

Some companies use software that is developed in-house, or specially built for the organisation. It is desirable that patches and updates for this custom software can also be distributed using the same method as ordinary patches. A PMS should therefore incorporate the possibility to insert home-made patches for custom software.

4.5 User interface/Framework

In this section some aspects of a good user interface are discussed. It covers access control, backups and customization and proposes several security archives with information about patches.

4.5.1 Design

Good software does not only have a lot of useful functionality but also has a good user interface. How the interface should be designed is out the scope of this project. There are a lot of guides on how an interface should be designed, but the basic criteria are that it should be unambiguous, straightforward and buttons should be in logical places.

The management interface of a PMS should be implemented as a web-application. The advantage is that it works on all platforms for which a webbrowser is available. Another advantage of this approach is that an administrator does not need to install a client application before being able to use it. While the interface will in most cases be used solely by a small number of system administrators, a web-interface is easier to maintain than a locally installed interface, because all maintenance activities such as upgrades are done server-side.

4.5.2 Customization

The framework should be as modular and extensible as possible so that it can always become compatible with new technologies and platforms it has to support. Therefore an Application Programming Interface (API) that covers the whole framework should be available for everyone to build anything.

The following list gives an idea of the possibilities of an API. The list is not long and therefore by for not exhaustive.

- Network inventorying
- Authentication
- Reporting / Statistics
- Alerting
- Platform / application support
- Network transport mechanisms
- ...

4.5.3 Access control

In the old days everyone trusted each other. It is now generally acknowledged that people can not be trusted. Therefore it should be possible to authenticate users who are authorized to use the management interface of the PMS.

Most large organisations that have a decent IT infrastructure also have a centralized authentication mechanism. Most commonly used in todays networks is LDAP[1], therefore this should be supported by the PMS. However, there should be enough flexibility (i.e. an API) to implement other authentication mechanisms.

Besides the LDAP based mechanism which is not available in all organisations there should be a possibility to have a local mechanism. This could be as simple as a list of usernames and passwords. Nonetheless, this mechanism should be based on the API that is also used for the LDAP based mechanism.

4.5.4 Backups

Disasters like hard-disk crashes, which prevents the PMS to do its job are inevitable. Therefore, the PMS should have functionality to backup and restore all settings, historical data and locally cached patches and updates. If locally cached data needs to be backed up should be configurable for each module. Some vendors remove old versions of patches and updates when a new version is released. Therefore these should also always be backed up.

4.5.5 Links to vulnerability lists

It should be possible to see, at least in case of a security related patch, what bug the patch fixes. A well-known website is for this kind of information is CVE, where all security updates and their descriptions for the Debian project releases are available. Other well-known websites are SecurityFocus BugTraq database, CERT Advisories[5] and US-CERT Vulnerability Notes Database[27].

[1]Microsoft's Active Directory is based on LDAP and it can therefore be accessed with LDAP as well.

4.6 Infrastructure

This section is probably the most technical in this specification. It covers the load on the network and servers involved, whether or not updates should be cached locally and how a PMS should deal with mobile users that have a low or expensive bandwidth connection. Also, the inventorying of a network is discussed.

4.6.1 Network load

When a new patch is to be rolled out it could have a serious impact on the load of the network infrastructure. A solution for this may be to use multicast. With multicast the server sends the patch simultaneously to all clients so that the server does not have to send it over and over again to every distinct client as is done with unicast. The best-case scenario is that the server has to send it only once to all clients. However, it is not very likely that all clients will listen to the multicast at once (e.g. the host was not powered on at the time the multicast took place). Multicast is widely used for distribution of audio and video content. The problem with multicast is that it is possible to loose some data when a packet is lost in transit. For audio and video this is not a big problem but for patches it is. To overcome these problems the Internet Engineering Task Force has set up a working group that investigates reliable multicast transport[11].

Another way to reduce load is by using some kind of peer-to-peer mechanism. Many projects that have to distribute large quantities of data (e.g. CD/DVD images of a Linux or BSD distribution) use it to offload their own servers. When all clients in a network participate in a peer-to-peer network, redundancy on the availability of the patch is automatically realized.

4.6.2 Multiple servers

When neither multicast or peer-to-peer are an option it should also be possible to set up multiple servers, which could be arranged in several ways. One could think of a hierarchical setup where there is a master on which everything is managed, such as approval of patches and targeting to specific clients, etcetera. However, the clients do not use the master to receive the updates. Instead, several slave servers are set up in this model to communicate with the clients. These slave servers will distribute the updates exactly as dictated by the master server.

4.6.3 Local caching

The previous section discussed solutions for the load on the server(s) with which the clients connect and download the patches from. These servers need to download the patches themselves as well, either directly from the vendor, or from a neighboring or parent update server. It would be very inefficient when the server would have to download the patch over and over again for each client that requests it. Therefore a cache, which stores the patch itself on a local hard-disk must be implemented.

4.6.4 Low bandwidth users

Almost all companies have people working out of the office. Most of these people have a laptop. These laptops are increasingly more often being connected to the company's network using technologies like GPRS and UMTS. Unfortunately, companies pay for every byte is transferred over the connection (at least, that is the case in The Netherlands), and compared to for example a ADSL connection it is very expensive.

While having enough bandwidth with GPRS and even more with UMTS, there are still users who use dial-up connections via ISDN or POTS (analogous telephony) for working outside the office (e.g. at home). The costs per megabyte are lower but the time before a patch of several megabytes is completely downloaded while not disturbing (i.e. delays due to too little available bandwidth) other programs who need the connection as well.

That is why care must me taken when deploying patches to users which are connected via low or expensive bandwidth connections. For example, when a patch is several kilobyte in size it should not be a big problem, especially when it is a patch which fixes a high risk security hole. For patches which have a low or even no priority and/or are too large to download the user should be notified by for example a pop-up message that there are patches standing-by which should be deployed on the device. In this message a date is shown before when the patch must me deployed, and thus before when the user should return to the office. In some cases, such as non-trivial services, one should be able to temporarily fix the problem by for example disabling the service until it is patched. This way, the need for a deadline for such patches is removed.

4.6.5 Inventory building

The inventorying of the network should be implemented as a modular framework (i.e. have a API). In this framework several modules should be able to work at the same time. One could think of a module that gets its information from LDAP or Active Directory and a module that scans an IP-network for hosts and identifies them somehow. Because it is not unthinkable that more than one plug-in discover the same host, the modules should have a priority.

A big problem is that once a network inventory has been made, and a client connects, one can not be sure, based solely on the IP-address of it, that it is the same client as the last time a client connected from that IP-address. These addresses can be changed either manually, or because the network uses DHCP to automatically assign hosts their address. The MAC-address of the network interface card can not be used as well, because that address is not visible to hots that are not on the same network (e.g. behind a router).

This problem can only be circumvented with cooperation of the client agent. The agent should register itself with the PMS using a unique identifier so that when the IP-address changes the client can still be identified. In this case the MAC-address of the network interface card can be use to generate such a unique identifier. It would be wise to make the installation of the client on a host part of an automatic OS load procedure.

4.7 Reporting

Managing a system without being able to see what is going on is not possible. Therefore the system should provide this information to an administrator by sending alerts in case of an emergency or report. This section discusses that matter.

4.7.1 Alerting

The framework should be able to generate an alert for every event that can occur in the system. One could think of new released (critical) patches that are ready to be approved, hosts that have crossed an install deadline, patches that are no longer rolled out due to failure, etc. These events should be classified in terms of severity and risk. The fact that a new patch is available should be an event, but the fact that the patch is critical could be another event with higher classification.

When a event occurs and meets certain criteria an administrator should be warned. This could be done by sending an e-mail or an SMS to an administrator's mobile phone. Because SMSs are expensive it should be possible to assign each type of alerting mechanism its own criteria. Besides that there should also be a mechanism that dampens the amount of alerts that are sent out. When an event occurs and administrators are notified, it should not be necessary to notify him again within a short period.

It should also be possible to integrate the PMS with an existing monitoring system. When a company has a central monitoring system that is based on SNMP it would be nice if the PMS is able to send SNMP Traps.

All events should be logged in a file which can be viewed inside the application. The log-file should be archived automatically at a configurable time interval. It should also be possible for an administrator to manually initiate the archiving process and delete old archived log-files. When browsing the log-file the whole list of events could be too large to find a single entry. Therefore it should be possible to apply filters based on several criteria which could be (a group of) hosts, platform, classification, date and time, etc.

4.7.2 Reports

To keep the system comprehensible it should be able to generate different types of reports. Reports can show for example which hosts are completely up-to-date and which not, how long it takes before a patch is 25%, 50%, 75% and completly rolled out, etc.

A report builder in which one can define custom reports is very useful when the types of reports available by default are not sufficient for managing the update process. Therefore an interface should be available in which all types of available data can be combined in a report or graph.

The reports available by default should be at least:

- List of patches per platform/application
- List of new hosts in the network (not categorized in a specific group)
- List of hosts in the network:
  - Installed operating systems and applications
  - Patched hosts: which patches are installed
  - Not fully patched hosts: which patches are missing
- Groups of hosts:
  - Hosts in each group
  - Approved patches per group

It should also be possible to optionally send reports on a regular interval by e-mail.

Chapter 5

Proof of Concept

5.1 Introduction

The previous chapter gave a description on the functional requirements for an ideal patch management system. This chapter kick-starts the development of this system. Due to time constraints it was not feasible to completely build this ideal system. Instead, a Proof of Concept (PoC) application was created. Goal of this PoC was to show that it is possible to create a framework for rolling out patches to clients on multiple platforms, using the update tools that are natively installed on these clients.

The application was released under the BSD License, and is available online[4]. It has been successfully tested using a Ubuntu Hoary (5.04)[25] host as client. Clients using other Debian-based operating systems are supported with little modification to the patching module.

The first part of this chapter describes the different components that the PoC consists of. The second part discusses the requirements on clients and the server to deploy the PoC. Section 5.4 discusses future work that should be done in order to make this Proof of Concept suitable for use in a production environment.

5.2 Components

The PoC consists of three components. The first is a central web-interface from which an administrator controls the PMS. The back-end of the system is a database, which holds information on operating systems and patches. The third part of the system consists of operating system dependent modules for retrieving and distributing patches. The three components are further described in this section. The PoC is written in Perl[20], and uses a MySQL[19] database back-end.

5.2.1 Web-interface

The first component of the PMS is the web-interface. The web-interface allows an administrator to perform four different tasks:

- Synchronize the database of patches
- Show a list of available patches for a particular operating system, version and architecture
- Select and deselect the patches that need to be distributed to hosts
- Save the changes and build the client repository for the particular operating system

5.2.2 Database

The heart of the PoC is the database. It holds all information on operating systems, available patches, and patches that are to be distributed to hosts. The database holds the following three tables:

OS This table holds all operating systems for which packages are available. Stored in it are the operating system's name, its version, and its architecture. This allows the system to differentiate between patches for different operating system versions and architectures.

PACKAGE The package table holds information on available patches for all operating systems. Stored properties are the patch name, its version, and the meta-data needed by the operating system dependent module to allow distribution of the patch. Every patch is linked to the operating system it belongs to.

ACCEPTED The accepted table lists all packages that have been selected for distribution by an administrator, together with the date on which they where approved for distribution.

5.2.3 Patching modules

The third part of the PoC consists of operating system dependent modules for retrieving available patches, and controlling the distribution of patches to clients. A module is needed for each type of update tool used on the clients (e.g. APT, YUM, Windows Update). The modules are automatically loaded by the web-interface, and can be used by calling one of two functions that need to be available in all modules. These functions are described below.

The PoC currently holds only one patching module. This is due to the time available for this project. The update tool chosen for this module was APT. The reason for this is twofold. Firstly, the PMSs discussed in this report do not offer support for APT-based clients, and the authors do not know of any other PMS that does support this update tool. The second reason APT was chosen is because it is very flexible, and easy to implement.

Synchronization

As discussed above, patching modules can be called using one of two functions that are available in all modules. The first of these is the synchronize function. This function downloads the list of available patches from a vendor, and stores it in the central database. The version of this function in the APT patching module downloads the list of available security packages from a repository on the Internet, and stores the information on each package in the database. A detailed list of the the tasks it performs is as follows:

- Download the Release file for the relevant operating system from a repository on the Internet
- Verify the digital signature of the Release file with the public key from the operating system vendor, using GnuPG
- Download the Package file for the relevant operating system version and architecture
- Verify the integrity of the Package file using the MD5 and SHA1 hashes stored in the Release file
- For every package in the Package file store the package name, version, and the complete package information (the meta-data) in the database

Repository building

The second standard function for a patching module is the build function. This function makes sure that clients will only receive patches that are approved for distribution. For APT this comes down to generating new Release and Package files, which only contain packages that are approved by an administrator. The exact tasks it performs are:

- Create a new Package file using the meta-data from the packages that are listed in the accepted table of the database
- Create a new Release file, using the header from the old Release file
- Create MD5 and SHA1 hashes from the new Package file, and store them in the new Release file
- Create a new digital signature for the Release file, using a private key that matches a public key known to the clients in the network

5.3 Configuration

In order to set up the PoC a number of things need to be configured. This section describes how to deploy the PoC on a server, and how to configure a client to start using the system.

5.3.1 Server

On the server that will hold the PMS three items need to be configured. First a webserver needs to be deployed, which allows administrators to use the system, and clients to download files from the system. Secondly, a database that will hold the patch information needs to be installed. Finally, GnuPG needs to be configured to be able to verify and create digital signatures.

Webserver

The webserver on which the PoC is to be installed should support Perl CGI-scripts. The PoC mimics the structure of a normal APT repository. However, it only publishes the Package and Release files, and not the packages itself. To allow a client to download packages as well, the webserver should be configured to forward requests for packages to a repository. This can be a repository on the Internet, or a local repository set up using apt-proxy[3] or a similar tool.

Database

The PoC requires a MySQL database. To be able to use it the database should contain the three tables mentioned in section 5.2.2, and the correct username, password and database name should be set in the database module of the PoC. The database module uses Perl DBI[21] to access the MySQL database.

GnuPG keys

To allow the PMS to verify and create digital signatures for the Release file two things need to be set up. First the public key of the repository from which the original Package and Release files are downloaded needs to be installed in the keyring on the system. This keyring should reside in the directory `modules/APT/.gnupg/`. Installing the public key allows the system to verify the files that it downloads from the repository.

To be able to create digital signatures for the new Release file, a new key-pair should be created and added tot the keyring.

5.3.2 Client

In order to configure a client to use the PoC two things need to be done. The first is changing the URL of the APT repository to point to the PMS. The second is adding the public key of the PMS to the database of trusted keys.

APT repository

The URL of the APT repository that a client uses to download new security packages usually resides in the file `/etc/apt/sources.list`. This URL should be changed to point to the PMS. For Ubuntu Hoary, the new URL is as follows:

```
deb http:///<pms.yoursite.tld>/pms/modules/APT/ubuntu hoary-security main
```

Trusted key

To allow a client to verify files created by the PMS the public key of the PMS should be added to the database of trusted keys on the client host. For Ubuntu Hoary this database resides in the file `/etc/apt/trusted.db`. Keys can be imported into this database using GnuPG.

5.4 Requirements for production use

The PoC described in this chapter is a working prototype. However, it is not fully ready to be deployed in a production environment. The PoC currently uses a single, hard-coded source URL for downloading lists of available packages from a repository. In order to make the tool suitable for a production environment this should be changed so that URLs for multiple operating systems, and multiple URLs for a single operating system can be used.

Another important feature that this PoC misses is access control. This can easily be solved by securing the webserver directory in which the tool resides, but is preferably built into the tool itself.

Chapter 6

Conclusion

An good patch management system needs to incorporate a large number of features. The most important of these where gathered in this document, subdivided into seven areas:

- Patches
- End users
- Distribution
- Administration
- User interface/Framework
- Infrastructure
- Reporting

Microsoft Windows Server Update Services, Radia Patch Manager, and LANDesk Patch Manager all incorporate a large number of these features, but also all miss some properties. Which of the three systems should be deployed in a production environment is largely dependent on the specific situation within an organisation.

Building the Proof of Concept showed that it is possible to create a centralised patch management system that is capable of managing patch distribution on hosts with varying operating systems and architectures.

6.1 Future work

Future work that may be done following up on the research described in this document should mainly focus on further development of the Proof of Concept. Any of the features of an ideal patch management system that were described in this document may be incorporated into the system, starting with the points mentioned in section 5.4 (use of multiple Debian sources and access control).

Bibliography

[1] *Advanced Packaging Tool (APT) Howto*,
http://www.debian.org/doc/manuals/apt-howto/

[2] *AptRpm*,
https://moin.conectiva.com.br/AptRpm/

[3] *Apt-proxy*,
http://apt-proxy.sourceforge.net/

[4] *Centralised patch management system: Proof of Concept*, J. Barhorst and M. Pels,
http://www.os3.nl/~martin/files/pms_0.1.tar.gz

[5] *CERT/CC Advisories*,
http://www.cert.org/advisories/

[6] *Common Vulnerabilities and Exposures*,
http://www.cve.mitre.org/

[7] *Debian GNU/Linux*,
http://www.debian.org/

[8] *Gnu Privacy Guard (GnuPG)*,
http://www.gnupg.org/

[9] *HP OpenView Patch Manager Using Radia*,
http://www.managementsoftware.hp.com/products/radia_patm/

[10] *HP OpenView Patch Manager Using Radia: Installation and configuration manual*, HP, 2005

[11] *IETF Working Group: Reliable Multicast Transport*,
http://www.ietf.org/html.charters/rmt-charter.html

[12] *Inside Windows-Update*, Mike Hartmann, 2003,
http://www.tecchannel.de/ueberblick/archiv/402064/index.html

[13] *International PGP Home Page*,
http://www.pgpi.org/

[14] *LANDesk Patch Manager*,
http://www.landesk.com/Products/Patch/Index.aspx

[15] *LANDesk Targeted Multicasting*,
http://www.axle-it.nl/Assortiment/Producten/LanDesk/LDMS/Multicast.html

[16] *Mandriva Linux*,
http://www.mandriva.com

[17] *Microsoft Windows Server Update Services*,
http://www.microsoft.com/windowsserversystem/updateservices/

[18] *Microsoft Windows Update*,
http://windowsupdate.microsoft.com/

[19] *MySQL*,
http://www.mysql.com/

[20] *Perl*,
http://www.perl.org/

[21] *Perl DBI*,
http://dbi.perl.org/

[22] *RedHat*
http://www.redhat.com/

[23] *SecurityFocus BugTraq mailinglist*,
http://www.securityfocus.com/archive/1

[24] *The Practice of System and Network Administration*, T. Limoncelli and C. Hogan, 2001, ISBN 0201702711

[25] *Ubuntu Linux*,
http://www.ubuntulinux.org/

[26] *University of Amsterdam, Informatiseringscentrum*,
http://www.ic.uva.nl/

[27] *US-CERT Vulnerability Notes Database*,
http://www.kb.cert.org/vuls/

[28] *Yellow Dog Linux*,
http://www.yellowdoglinux.com/

[29] *Yellow dog Updater, Modified (YUM)*,
http://linux.duke.edu/projects/yum/