The following paper was originally published in the
Proceedings of the Large Installation System Administration of Windows NT Conference
Seattle, Washington, August 5–8, 1998

Patch32 : A System for Automated Client OS Updates

Gerald Carter
Auburn University

For more information about USENIX Association contact:

1. Phone: 510 528-8649
2. FAX: 510 548-5738
3. Email: office@usenix.org
4. WWW URL: http://www.usenix.org/

Patch32 : A System for Automated Client OS Updates

Gerald Carter
Engineering Network Services
Auburn University
jerry@eng.auburn.edu

Abstract

The adage "a chain is only as strong as its weakest link" is true for network security, the link being the host on the network. To secure a network, hosts must be thoughtfully installed and kept updated with the appropriate patches. For hosts running Microsoft Windows 95® or Microsoft Windows NT Workstation® keeping patches current is problematic.

Unlike most Unix variants, neither Windows 95 nor NT Workstation ship with a network extensible update mechanism. Though third party solutions are available, they can be costly to implement for large networks. This paper presents a free update mechanism for hosts running Windows 9x or NT Workstation served by Samba (see Appendix A).

Developed to patch Microsoft's 32 bit operating systems, the name Patch32 was adopted. Patch32 was developed for an existing network dominated by Sun Microsystems' SPARC servers running Solaris®, however, Patch32 can be used in any environment that provides SMB file services.

1. Introduction

The initial design criteria for Patch32 was simple: Provide for completely automated, remotely administered updates to Microsoft's 32 bit operating systems. Essential to any automated update mechanism is the ability to determine what operating system the machine is running and what updates have already been installed so that the necessary updates can be determined. To be effective, any update mechanism must be able to access the necessary updates from a central location without human intervention and with a mechanism to guarantee that the program will be run on the client systems with reasonable regularity.

This paper addresses issues that arose during the implementations of *Patch32*. The remaining sections are organized as follows. Section 2 discusses solutions to the various requirements of such an update system. Section 3 describes security concerns related to *Patch32* and ways to address these concerns. For those wishing to implement the system, suggestions for customization are described in Section 4. Finally, lessons learned from the implementation process as well as plans for future enhancement are included in Section 5. The appendix contains the perl source code for the Patch32 script, example patch listings, URL's and other references for the reader interested in finding out more about the software and ideas presented here

2. Implementation

2.1. File Access

The file services needed to support Patch32 are provided by a dedicated 196MB Sun Ultra 170 running Solaris 2.5.1. Samba, which is available under the GNU General Public License, provides file and print services to Server Message Block (SMB) clients including Microsoft's Windows for Workgroups®, Windows 95 and Windows NT. Samba also includes the capability to act as a Primary Domain Controller (PDC) for Windows NT Domains, although this support is currently in the testing stages and is not included in the main source distribution as of version 1.9.18p7.

SMB services were chosen rather than other file sharing protocols such as NFS due to the native support for SMB within the Microsoft operating systems. This support, combined with the Microsoft Network Client, allowed built-in tools such as domain login scripts and System Policies to be used to configure the execution of patch scripts on remote machines.

Two separate copies of Samba were configured on the physical server, each have its own network interface. This was accomplished by utilizing two settings in the samba configuration file (smb.conf). The "interfaces" parameter specifies the correct network interface to which the samba processes should attach and the value

for the "socket address" parameter determines the IP address to which the samba processes should bind. The reader should be aware that using this second parameter may cause problems with network browsing. The problem has been reproduced when the domain controller is bound to the second network interface which has been created as a virtual interface. Therefore it may be necessary to use a central WINS server in order for a client to resolve names correctly even if it is on the same subnet as the samba server.

At the College of Engineering, the first samba server, \\USERSERVER, provided access to department shares, user shares, and various network printers. The second copy of samba, \\GUESTSERVER, provided access to various system and development tools such as the Java SDK, Perl5 for Win32, administrative scripts and command line utilities.

Both servers are configured with user level security however, \\GUESTSERVER is configured to only allow guest connections. This is accomplished by defining GUEST_SESSETUP to be the integer 2 in the file local.h, which is part of the samba source code distribution, and by isolating \\GUESTSERVER from the normal list of accounts. Therefore any user attempting to connect is validated as the guest account specified in the smb.conf file. This mechanism allows machines to access specific shares without validation. Since all files accessible on \\GUESTSERVER have been deemed as public access, the setup does not create a security concern.

At this point is may be obvious why it was decided to run two separate samba servers. Guest connections are not often desired to shares containing data or to printers where page accounting has been enabled. It is possible to recreate this same type of behavior without maintaining two servers, but separation was deemed necessary for management purposes.

2.2. Patch Preparation

The updates discussed in this paper were released by Microsoft and downloaded from the their web site (see Appendix A). This section will discuss the preparation of the archived files necessary to integrate the patches into the model presented here.

A system update contains three basic components. The first is the collection of updated files. This may include dynamic link libraries, executables, information files, device drivers or any other imaginable portion of the local system.

The second component is the patch installation program. Windows 95 updates normally rely on the rundll.exe and setupx.dll files, which are by default located in the \windows and \windows\system directories respectively. Windows NT Service Packs and Hotfixes normally include an installation executable.

The final necessary component for an installation program is a listing of directory locations and registry keys where the updated files and information is to be placed. This listing may be internal to the installation executable or, as is the case with both Windows 95 and Windows NT fixes, external in the form of an information file (INF).

Both the Windows 95 and the Windows NT updates released by Microsoft are packed in a self-installing executable file. The method for extracting the internal files is different for each operating system.

To install a Windows 95 update such as the Service Pack 1, a user would normally simply launch the update executable. At this point the setup program will extract the archived files into the %TEMP% directory. When prompted to continue with the update, the user may then make a copy of the files that have been extracted and cancel the setup program.

In order to extract the archived files from either a Windows NT Service Pack or Hotfix, one must simply run the archived executable from a console window and pass the appropriate parameter to the program. For example, to extract the files contained in the update named "hotfix1_i.exe", the standard method would be run "hotfix1_i.exe /x" from a command prompt. Please consult the service pack or hotfix documentation for the correct syntax.

2.3. Script Implementation

Once machines are able to the access the updated files from a central location, the next step is to automate the process of applying the patches. For this process, Perl (see Appendix A) was chosen as the implementation language. Reasons for selecting Perl include fast development time due to familiarity with the language on a Unix platform as well as the availability of modules to interface with specific Windows 95 / NT entities such as the event log and the system registry.

```
determine what OS we are running;
set the patch directory;
set the install method;
open $patchdir\$patchFile;
foreach ( entry in patch listing ) {
   split the entry into parts;
   if ( registry key does not exist ) {
      install the patch;
   }
}

if ( any patches have been installed ) {
   execute rebootMessage;
}
```

Figure 1: Patch32 algorithm

As stated previously, the first thing that is necessary to install updates to an operating system is to determine what version of the OS is currently running. The following segment of code will return all the information needed.

```
($osString,$majorVer,$minorVer,$osBuild,
$osId) = Win32::GetOSVersion ();

$osString   OS revision string
$majorVer   Major version number
$minorVer   Minor version number
$osBuild    OS build number
$osType     integer (win32s = 0; Win95
                    = 1; WinNT = 2)
```

Figure 2: Win32::GetOSVersion() and return values

The OS specific patch location is determined by testing $osType and then creating a directory path by catenating the Patch32 root directory specified at the beginning of the script with the OS name (either win95 or winnt) and $osBuild. For example, Windows 95 OSR1 has a build number of 67109814. The full path to the correct patches would be

```
$basedir\win95\67109814
```

From this patch directory, the program will open a list of updates to be applied to the client. The format of a patch listing entry is given below.

```
<registry key>:<install patch directory>
```

Figure 3: Patch entry format

Each entry is delimited by a carriage return. Lines beginning with a semicolon (;) or a pound sign (#) are ignored as comments as are blank lines. The “registry key” field is the absolute path to a key in the local system registry whose existence implies a previous application of the patch. The hive key HKEY_LOCAL_MACHINE is abbreviated as HKLM.

```
; This fix corrects GETADMIN problem
HKLM\SOFTWARE\...\Hotfix\Q146965:admnfix
```

Figure 4: Example patch listing entry. A portion of the registry key field has been deleted to prevent line wrapping.

Once the patch entry is read, it is split into its component parts. Using the given registry key, the program checks for its existence in the local registry. The nonexistence of the registry key indicates that the patch has not been installed. If it does exist, then the patch has previously been applied and the program does not attempt to install it. Because of the frequency with which the patch program will be run, it is necessary that the script be efficient when evaluating the current state of the client. Therefore no attempt is made to verify the integrity of previously installed updates or to examine the version of installed files. The “install patch directory” field is the name of the directory containing the update. This is a path relative to the previously determined OS specific patch directory and will be used during the patch installation process described in the next section.

2.4. Patch Installation

During the development of Patch32, one goal was to provide a method of deploying new patches with minimal effort. By default, all Windows NT hotfixes released by Microsoft store a patch identification number in the following registry key

```
HKEY_LOCAL_MACHINE
   Software
      Microsoft
         Windows NT
            CurrentVersion
               Hotfixes
                  <patch number>
```

Therefore it is possible to determine the presence of a hotfix by the existence of its ID in the registry. The Service Packs create a string value in

```
HKEY_LOCAL_MACHINE
   Software
```

```
      Microsoft
         Windows NT
            CurrentVersion
"CSD Version" = "<Service Pack Name>"
```

Checking for the existence of this value will only determine if any service pack has been installed rather than a specific one. Therefore the setup INF file for the Service Pack is modified to create a key in

```
HKEY_LOCAL_MACHINE
   Software
      Microsoft
         Windows NT
            CurrentVersion
               Hotfixes
                  <SvcPackNo>
```

The existence of an installed service pack may then be determined by examining only one registry key. The same method may be used when an update creates many keys but only one need be examined.

By default, Windows 95 update information is stored in

```
HKEY_LOCAL_MACHINE
   Software
      Microsoft
         Windows
            CurrentVersion
               Setup
                  Updates
                     UPD<patch number>
```

These default keys are used only to reduce the amount of effort needed to integrate a new patch into the existing setup. Site specific keys may be used as long as they are created by the update once it has been applied to the local system.

Once the registry key for the patch and the location from which to install the patch is known, the setup program for the update can be executed using the system() function. Patch32 allows for a single install method to be specified for Windows 95 and one for Windows NT.

In the case of Windows 95, the INF file provided with the update is used for the installation. This file is parsed using a modified version of the Windows 95 default method that is stored in the system registry.

```
rundll.exe setupx.dll
           InstallHinfSection 132
           DefaultInstall <filename>
```

Figure 5: Default Windows 95 install method for INF files.

Setupx.dll is instructed to process the "DefaultInstall" section of the file given by the last parameter. The only difference in the install method used by the Patch32 script is that the fourth parameter is 133 rather than 132. This has the effect of not prompting the user to reboot after the INF file is processed.

For Windows NT service packs and hotfixes, the update.exe utility provided with Service Pack 3 for Windows NT 4.0 can be used to install both

```
update.exe -z -u –q <filename>
```

Figure 6: Windows NT install procedure for a non-interactive update without a reboot after patch completion.

These methods are used to reduce the work needed for new patches. They are not the only possible installation methods. Custom setup programs may be may be used as long as the method to install each update per OS is the same. Modifications to the patch installation method are discussed in Section 4.

The Patch32 script continues until all entries have been processed. At the end of the program, if any patches have been installed, the contents of a string variable which specify a reboot message are executed by another call to the system() function. Windows 95 clients are informed that the system has been patched and changes will take effect after the next system reboot. Windows NT displays a message that the system will reboot due to patches having been applied and is restarted using the shutdown utility included in the *Windows NT 4.0 Server Resource Kit* **[1]**.

(a)
```
$basePatchDir\win95\w95upd.exe
```

(b)
```
start $basePatchDir\winnt\shutdown
      /l /y /r /t:30 "Patches completed.
      System rebooting."
```

Figure 7: (**a**) Windows 95 and (**b**) Windows NT reboot messages executed if any updates were applied.

2.5. Script Execution

The last issue to be resolved is to guarantee that the client machines will execute the patch script on a regular basis. This program must be run during the operating system boot process but after network support has been enabled.

Windows 95 clients are able to run the program during a domain login script or from the

```
HKEY_LOCAL_MACHINE
   Software
      Microsoft
         Windows
            RunOnce
```

registry key which can be set using a remote update of the system policies.

In order to install updates on Windows NT clients, the update program must have administrative privileges on the local host. Therefore the patch program may not be run during a normal user's domain login script. The *Windows NT 4.0 Server Resource Kit* contains a service utility named AutoexNT that, upon start up, executes a batch file named %SYSTEMROOT%\System32\AutoexNT.bat. By configuring the service to start up automatically, AutoexNT.bat will be executed during the boot process. It is from this batch file that the Patch32 script is launched. By running the AutoexNT service under a local administrative account, the program is able to update system files and settings.

```
@echo off
set LOG=%SYSTEMROOT%\log\autoexnt.log
set PERL=\\GUESTSERVER\perl\bin\perl.exe
set PATCH32=\\GUESTSERVER\bin\patch32.pl


echo :::::::::::::::::::: >> %LOG%
date /t >> %LOGFILE%
time /t >> %LOGFILE%
echo :::::::::::::::::::: >> %LOG%
net start LanManWorkstation >> %LOG%
%PERL%  %PATCH32% >> %LOG%
```

Figure 8: Example AutoexNT.bat script

3. Security

3.1. Permissions

When run under Windows NT, the patch program must run as a local administrative account in order to update system files. This results in three security issues that must be addressed. These are the integrity of the patch files themselves and their associated setup programs, the security of the registry keys which are defined by the system updates and the security of the batch file run by the AutoexNT service. Obviously the last two issues are irrelevant in Windows 95 since such access control does not exist and because the AutoexNT service is not used.

In order to insure the integrity of the update packages, write access to the files should be restricted to administrative accounts only. Without this protection, a virus or Trojan Horse program could easily be distributed to clients by modifying the patch files on the server.

It is also important to maintain the security of the system registry keys created by the updates because these are the only means by which the Patch32 program can determine whether or not an update has been installed. If the security of these keys were comprised, it would be possible to avoid the installation of an update simply by creating the correct registry key. It is important to remember that the patch script makes no attempt to verify the status of currently installed patches. It is not technically difficult to perform this type of system examination, but would result in an increased execution time and degraded system performance.

The last security issue to address is the AutoexNT service. As stated previously, this service runs as a local administrative account and executes the contents of the AutoexNT.bat file whatever they may be. It is therefore extremely important that only administrative accounts be allowed to modify this file for reasons similar to those given for ensuring the integrity of updates packages.

3.2. Logging

In a distributed environment, it is important not only to be able deploy updates effectively, but also to be able to determine which machines were successfully updated and those which failed. Patch32 supports log-

ging to the Windows NT EventLog via Perl's Win32::WriteEventLog() function.

```
3000  The patch32 script completed
      successfully
3001  The patch32 script failed
3002  The %1 patch was installed
      successfully
3003  The %1 patch installation failed
3004  The system was rebooted due to
      patches having been applied
```

Figure 9: Patch32 EventID's and descriptions.

4. Customization

During the initial installation, Patch32 requires that certain variables, which determine paths to OS updates, be defined. These variables, located at the beginning of the source file, allow the administrator to specify the base location of the Patch32 directory hierarchy and the install methods and parameters for Windows 95 and Windows NT updates. The server administrator may also choose to modify the reboot message that is executed upon the successful application of an update.

One advantage to Patch32, besides the fact the all software components are freely available, is that the patch script itself is very short, about two hundred lines at the time of this writing. Because the code is easy to understand, site specific changes can be made easily. This makes Patch32 extremely flexible. For example, one of the many, freely available SMTP console mail programs such as Blat (see Appendix A) could be used to augment the system's current logging abilities. In this way, network administration would be notified via e-mail upon update completions or failures.

The patch installation method is also very flexible. The current configuration at the College of Engineering uses methods available with the updates released by Microsoft. A simple alternative to this would be to use a batch script named "update.bat" that would execute specific patch installation programs. This would provide the ability to have an individual installation mechanism for each OS patch. The only restrictions would be that the update program runs non-interactively under Windows NT and that it creates the registry key necessary to allow future executions of Patch32 to determine its existence.

Another possibility of expanding the current installation method would be to determine it based on the OS build number. Then each version for Windows 95 and Windows NT would have a separate method for installing updates. This would be most helpful when supporting Windows NT 3.51 and 4.0 clients from the same server.

5. Conclusion and Future Development

In conclusion, the Patch32 system provides a flexible means of deploying system updates to Windows 95 and Windows NT clients on a regular basis. Thus far Patch32 has been implemented to manage Windows NT 4.0 and Windows 95 clients, but has also been tested with Windows NT 3.51, Windows NT 5.0 Beta 1 and Windows 98 clients. The current system supports over two hundred Windows 95 machines and approximately forty Windows NT 4.0 Workstations.

One issue that arose from the experience of implementing this system is the question of 'What is a patch?' A system update is really nothing more that a reconfiguration of the client machine. Files and registry keys are created, updated and deleted. In essence, this is software installation. It is perceivable that the Patch32 system could be used to configure software packages on client machines. This, however, is left as an exercise for the reader.

Appendix A: URL's

This section lists relevant URLs for freely available software discussed in this paper.

Samba is available under the GNU General Public License and comes with full source code. Information is available at http://samba.anu.edu.au/samba.

The *Microsoft OS updates* for Windows 95 and Windows NT are available from Microsoft's web server and FTP server. See http://support.microsoft.com and ftp://ftp.microsoft.com/bussys/winnt/winnt-public/fixes.

ActiveState distributes *Perl5 for Win32* which available at http://www.ActiveState.com.

All source code for the *Patch32* system are available on-line as will be future versions. See

http://www.eng.auburn.edu/users/cartegw/Patch32 for information about obtaining the files.

Blat is a public domain console SMTP mailer for the win32 platform. It is designed to run under Windows NT and comes with full source code. Visit http://gepasi.dbs.aber.ac.uk/softw/blat.html for more information.

References

[1] Windows NT 4.0 Server Resource Kit, Microsoft Press, Redmond, Washington, 1996. ISBN 1-57231-344-7

Appendix B: Patch32 source listing

```
##############################################################################
#      Filename          : patch32.pl
#      Author            : Gerald ( Jerry ) Carter
#                          jerry@eng.auburn.edu
#      Date Created      : November 17, 1997
#      Last Update       : June 6, 1998
#
#      ATTN: Some lines have been wrapped for readability.  The real
#      source may be downloaded from
#           http://www.eng.auburn.edu/users/cartegw/Patch32
#
#      Perl5 for Win32 script to patch Windows 95 / NT clients on the
#      College of Engineering network.  Client supported are
#           Windows 95 OSR1, OSR2, OSR2.1
#           Windows 98
#           Windows NT Workstation 3.51, 4.0, 5.0 Beta 1
#
#      The following variables must be set for each OS
#           $basePatchDir            Base directory of all patches
#           $patchDirectory          Location of the patch files
#           $installPatch            Patch installation method
#           $installParameter        Patch install parameters
#           $rebootMessage           win32 command that will run if any
#                                       patches are installed
#
#      $installPatch . $patchDirectory . '\' . $pSourceDir .
#         '\' . $installPatchParameter
#
#      ============================================================
#      LICENSE
#      ============================================================
#      This program is free software; you can redistribute it
#      and/or modify it under the terms of the GNU General Public
#      License as published by the Free Software Foundation; either
#      version 2 of the License, or (at your option) any later version.
#
#      This program is distributed in the hope that it will be useful,
#      but WITHOUT ANY WARRANTY; without even the implied warranty of
#      MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
#      GNU General Public License for more details.
#
#      You should have received a copy of the GNU General Public
#      License along with this program; if not, write to the Free
#      Software Foundation, Inc., 675 Mass Ave, Cambridge,
#      MA 02139, USA.
#
##############################################################################


# Needed module for registry functions
use Win32::Registry;
```

```
# Setup some script variables
$DEBUG = 1;
$reboot = 0;
$eventLog = 0;
$eventSource = "Patch32";
$computer = $ENV{'COMPUTERNAME'};
$eventLogFile = "Application";

##############################################################################
# Script variables - Fill these in with values appropriate for
#                    your site

# Base patch directory
$basePatchDir = "\\\\ivy\\patch32";

# Win95 patch install method
$win95InstallPatch = 'rundll.exe setupx.dll,InstallHinfSection
                      DefaultInstall 133 ';

# Win95 patch install method parameter
$win95InstallPatchParm = 'install.inf';

# Win95 reboot message to be executed if patches are applied
$win95RebootMsg = "$basePatchDir\\win95\\w95upd.exe";

# WinNT patch install method
$winntInstallPatch = '';

# WinNT patch install method parameter
$winntInstallPatchParm = 'update -z -u -q';

# WinNT reboot message to be executed if patches are applied
$winntRebootMsg = "start $basePatchDir\\winnt\\shutdown /l /y /r /t:30
                   \"Patches completed.  System rebooting.\"";

##############################################################################

# First we must determine what OS we are running
($osString, $majorVer, $minorVer, $osBuild, $osId) = Win32::GetOSVersion ();
if ( $DEBUG >= 2 ) {
   print "ID = $osId\n";
}
# win32s
if ( $osId == 0 ) {
   print "Can't patch win32s!.";
   exit ( -1 );
}
# Windows 95
elsif ( $osId == 1 ) {
   # Print the OS information
   print "OS version is Windows 95$osString.\n";

   # Set the $patchDirectory
   $patchDirectory = "$basePatchDir\\win95\\$osBuild";
   $installPatch = $win95InstallPatch;
   $installPatchParameter = $win95InstallPatchParm;
   $rebootMessage = $win95RebootMsg;
}
# Windows NT
elsif ( $osId == 2 ) {
   # Needed by Windows NT EventLog
   require "NT.ph";


   # Print the OS information
   print "OS version is Windows NT $majorVer.$minorVer ";
   print "(build $osBuild: $osString)\n";
```

```
   $patchDirectory = "$basePatchDir\\winnt\\$osBuild";
   $installPatch = $winntInstallPatch;
   $installPatchParameter = $winntInstallPatchParm;
   $rebootMessage = $winntRebootMsg;

   $eventLog = 1;
   Win32::OpenEventLog ( $ntlog, $computer, $eventLogFile) || warn $!;
}
# Unknown
else {
   print "Unable to determine what OS we're running!\n";
   print "Script exiting...\n";
   exit ( -1 );
}

if ( $DEBUG >= 1 ) {
   print "Patch Directory = $patchDirectory\n";
   print "Install Patch method = $installPatch\n";
   print "Install Patch parameters = $installPatchParameter\n";
   print "Reboot Message = $rebootMessage\n";
}

# Now we are ready to gread the patch.list file
open ( PATCHLIST, "$patchDirectory/patch.list" ) || die $!;

# Now we loop through the listing of patches
while ( $patch = <PATCHLIST> ) {

   # Check for comment
   $tmpChar = substr ( $patch, 0, 1);
   if ( ("$tmpChar" ne ";") && ($tmpChar ne "#") ) {

      # Get the individual patch information
      chop ( $patch );
      ( $pRegKey, $pSourceDir ) = split ( /:/, $patch );

      # ...This should be able to handle other hives as well as HKLM...
      # ...a kludge for now...
      $pRegKey =~ s/HKLM\\//;
      $regObject = $HKEY_LOCAL_MACHINE;

      # If we can open the given registry key, then we assume that
      # the patch is in place and go on
      if ( $pRegKey ne "" ) {
         if ( $DEBUG >= 2 ) {
            print "Patch Registry Key = \n     $pRegKey\n";
            print "Patch Source Directory = \n     $pSourceDir\n";
         }
         if ( $regObject->Open ( "$pRegKey", $patchVer ) ) {
            print "Patch $pSourceDir already installed!\n";
         }
         else {
            $reboot = 1;
            print "Installing $pSourceDir...\n";
            $installTmp = $installPatch . $patchDirectory . "\\" .
                          $pSourceDir . "\\" . $installPatchParameter;
            print "$installTmp\n";
            system $installTmp;
            print "\n";
            if ( $eventLog == 1 ) {
               if ( $regObject->Open ( "$pRegKey", $patchVer ) ) {
                  Win32::WriteEventLog ( $computer, $eventSource,
                         &EVENTLOG_SUCCESS, &NULL, 3002, &NULL, 0,
                         "$pSourceDir" );
               }
               else {
```

```
                Win32::WriteEventLog ( $computer, $eventSource,
                       &EVENTLOG_ERROR_TYPE, &NULL, 3003, &NULL, 0,
                       "$pSourceDir" );
             }
          }
       }
    }
 }


# Check to see if we installed patches
if ( $reboot ) {
   system $rebootMessage;
   if ( $eventLog == 1 ) {
      # Event : rebooting the system
      Win32::WriteEventLog ( $computer, $eventSource,
             &EVENTLOG_INFORMATION_TYPE, &NULL, 3004, &NULL, 0, "" );
   }
}

if ( $eventLog == 1 ) {
   # Event : Successful script completion
   Win32::WriteEventLog ( $computer, $eventSource, &EVENTLOG_INFORMATION_TYPE,
                          &NULL, 3000, &NULL, 0, "" );
   Win32::CloseEventLog ($ ntlog );
}

exit (0);

########## end of patch32.pl ##############################################
##############################################################################
```

Appendix C: Example patch.list

```
;
;     Patch listing for Windows NT 4.0 ( build 1381 )
;
;     Author          : Gerald ( Jerry ) Carter
;                       jerry@eng.auburn.edu
;                       College of Engineering Network Services
;                       Auburn University
;     Date Created    : February 20, 1998
;     Last Update     : June 1, 1998
;

; Service Pack 3
HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Hotfix\ServicePack3:sp3

; This fix corrects GETADMIN problem
HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Hotfix\Q146965:admnfix

; This fix corrects the chargen/telnet port issue (135)
HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Hotfix\Q154460:chargen

; Workaround for Pentium problem with invalid opcode
HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Hotfix\Q163852:pentfix

; Fix for newtear and bonk attacks.  Also ICMP attacks
HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Hotfix\Q179129:tearfix

; This fix is for lsahack.exe problem..  Also increased encryption on
; the LSA secrets in the registry.
HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Hotfix\Q184017:lsa2fix
```