System Security Administration for NT

Harlan Carvey
Hcarvey@netsol.com

Network Solutions
http://www.networksolutions.com

Abstract

System security administration has, for the most part, been largely ignored as network administration has flourished. The result of this is that there are large installations of NT that need to be retrofitted with some form of security administration and management system. There are various third party tools available to assist in this endeavor, but they are somewhat general and not tailored to meet the needs of a particular user or organization. System administrators must therefore learn to mold their infrastructure to the tool, rather than the other way around. Often times, the tools may also be quite expensive, and difficult to learn and maintain.

1. Introduction

Security administration for a single NT system can be cumbersome, at best. There is no single interface for configuring and monitoring the security posture of an NT system. For example, the audit policy for a standalone NT system is set via the User Manager, while log-specific settings and all monitored activity is recorded in the EventLog. Further, each object (file, directory, share, Registry key) has it's own interface for enabling access control lists (ACLs) and audit settings. Rolling out a common audit standard across an NT enterprise, and then monitoring the EventLogs, can be a daunting task. However, there are many third-party commercial tools available to assist the system administrator. Two drawbacks of these tools are their vagueness and cost. Commercially available solutions are very general in nature, not providing the ability to focus in on the specific needs of the administrator. Some check for compliance with "best practices" but the definition of "best practices" is purely arbitrary. Other tools provide a wealth of information. In fact, they provide too much information, inundating the administrator with facts, requiring him to sift through this information and attempt to extract the real issues at hand. None of the tools take other aspects of the administrator's infrastructure into account, such as corporate security policies, the existence of a firewall, ACLs on routers, VLAN implementations on switches, etc. The cost of purchasing and installing the tools, coupled with the "cost" of deciphering the collected data, makes for a very low return on investment (ROI). Updates to the tools to look for new vulnerabilities may even be an added expense.

System administrators need a more viable solution. The appropriate solution for a security administration system must:

- Be simple, and easy to construct and maintain.
- Be centralized.
- Be scalable.
- Be configurable, to meet the specific needs of a user.
- Provide for verification of security policy compliance (compliance checking and verification).
- Notify the administrator of systems that are not in compliance.
- Where appropriate, modify system settings to bring systems into compliance with policies.
- Reduce overall cost for administering the system (i.e., no expensive per-seat license fees, etc).

1.1 Purpose

The purpose of this paper is to provide a framework, using Perl, for administering system security across an NT enterprise. Due to space limitations, all code examples cannot be presented in this paper. Additional scripts will be available via the author's web site at http://www.patriot.net/users/carvdawg/perl.html.

1.2. Background

The solution will be implemented in Perl, a scripting language that has been used on Unix platforms for administration purposes of years, and has been made available for the Win32 platform. ActiveState provides a fairly complete distribution of Perl for Win32 systems, as well as several avenues of support. There are also several additional modules that provide a quantum leap in functionality and usability, in the NT environment, to the ActiveState distribution. These modules provide a convenient wrapper around the Win32 API functions, making them more accessible, and easier to use and understand. This access to the Win32 API facilitates collection, and if necessary, modification, of security-relevant data for NT machines across the enterprise.

2. Solution

The solution for the centralized security administration of NT systems can be separated into three phases: collection, filtering/analysis, and modification. The three phases are kept distinctly separate in order to maintain simplicity, scalability and functionality. Separating the phases also makes it easier to construct a working set of tools, by allowing for testing and verification of one phase before moving on to the next. Additional functionality can be added to one phase without requiring any changes to the other phases.

The collection phase is used to extract raw configuration data from NT machines across the enterprise:

- Generate MD5 checksums of specific files to ensure file integrity.
- Registry settings (checking for security-specific settings, trojans, viruses, etc)
- File, directory, share and Registry key ACLs
- Services (running, stopped, and paused), the account each service runs under, and the executable called by the service
- Audit policies and EventLog entries
- User rights and user account information
- File (boot.ini) and directory ("Start Menu" for each user) contents

This data is saved in a central location, most likely on an administration workstation, in a suitable format. The data may be written to an Access database or an Excel spreadsheet, but the simplest method is to choose a standard delimiter, such as a colon or semi-colon, and write the data to flat text files. The data is simple to parse and review, and easily compressed and encrypted for archival. The examples in this paper will send all data to STDOUT, allowing it to also be redirected to a flat file.

The data can be easily analyzed using filters written in Perl, making use of its inherent ability to parse data and files. These filters access and process the data based on predefined conditions, making full use of the pattern matching abilities of regular expressions. Filters may be designed to check for:

- Known security issues, such as ACLs (on files, directories, and Registry keys) or specific Registry values.
- More complex issues, such as the ACLs on a file as it's related to the function of the particular system (PDC, BDC, or workstation) and a Registry key entry.
- General trends across the enterprise, such as the level of Service Packs and HotFixes installed.
- Compliance with corporate security policies, to include the audit policy for each system, or the existence of modems on workstations vice RAS servers.

The processed information can then be presented in whichever format meets the needs of the user, to include:

- Email
- Pager
- HTML
- XML
- Word document
- Excel spreadsheet
- Access database
- EventLog entries

The system administrator can also use Perl to act upon the information derived from filtering and analysis, making modifications as necessary. Services can be stopped or started, Registry values added or deleted, auditing can enabled, and EventLogs can be collected and filtered, all from a single workstation.

Due to it's simple, modular nature, this system can be easily updated as new information (new vulnerabilities, updated security policies) is incorporated.

This system can also be used to establish a baseline, which managers and administrators can use to look for system changes, or develop and implement security policies and standards.

Once the data is filtered and analyzed, system administrators can begin the modification phase. Using the same Perl modules that were used to collect data from systems, the system administrator can bring the NT systems into compliance with security standards, by updating settings (i.e., Registry entries, file/directory/share/Registry permissions, audit policy, etc). As this is done from a centralized location, only one account, with a strong password, is required to perform this entire range of security administration.

Once all settings have been implemented and tested, the collection and filtering phases can again be used to verify compliance with security standards, by executing those phases as part of a regular security scan. Using the baseline information, the system administrator need only to look for anomalies and unusual settings, indicating that a lack of compliance with security policies and standards.

As an example, assume the system administrator wishes to not only check for the existence of trojan programs on her NT servers and workstations, but she also wants to ensure that these programs (or any other programs, for that matter) can not be installed on the systems. She uses Perl scripts collect relevant data (Registry key entries and permissions, file and directory ACLs, etc) from across the enterprise, to include offices connected to the master domain by WAN and VPN links. Once she's collected data from all systems, she filters the Registry entries based on published signatures of known trojan programs, and then reviews the entries for suspicious or unauthorized entries. Not finding anything significant, she then sets about modifying Registry and directory ACLs on all systems to prevent the installation of trojan programs in the future. She follows this up with periodic scans of the Registry keys and Security EventLog to ensure compliance, and incorporates new data, as it becomes available. The entire process is simple to maintain, and automated to reduce errors and increase efficiency.

3. Implementation

The examples presented in this paper are not meant to be all-inclusive. Rather, they are presented as a framework or set of tools from which system administrators can build solutions that meet their specific needs. Also, space requirements limit the number and functionality of the scripts that can be presented. Additional scripts will be available via the web, at http://www.patriot.net/users/carvdawg/perl.html.

The following examples lay the groundwork from which specific solutions and systems can be built.

3.1 Checksums

The integrity of critical system files is a serious security concern, as there are publicly available web sites that demonstrate the construction of a rootkit for NT (http://www.rootkit.com). Rootkits have been a thorn in the sides of Unix administrators, as they are used to patch the kernel and trojanize commands in order to hide malicious activities. DIGEST.PL (Script 1) demonstrates a method for generating MD5 hashes of critical files. These hashes can be saved to a protected file or database and be used to verify the integrity of critical system files on a regular basis.

3.2 Registry Settings

The NT Registry can be as much a treasure trove as it is a technological minefield. There are several Registry values that have a direct or indirect impact on the security posture of the system.

3.2.1 Individual Registry Values

REGKEYS.PL (Script 2) shows how Registry values can be collected from remote NT systems. The values can then be verified, and a message printed to STDOUT for each incorrectly set Registry value. This information can be logged to a database or file, or the value simply modified as needed. For example, the system administrator can check NT systems for compliance by checking the Service Pack level on each system.

3.2.2 Trojan Keys

TROJANKEYS.PL (Script 3) demonstrates collecting the contents of Registry keys. The values can be scanned for known trojan signatures using the grep() function. Trojan signatures are available from anti-virus sites such as F-Secure (http://www.f-secure.com). For example, using grep(), the system administrator can quickly scan incoming data using the following signatures for network backdoor, remote administration trojan programs (signature in parens):

- ATAKA (SNDVOL.EXE)
- NetBus (Patch)
- DeepThroat (SystemDLL32)
- NetSphere (NSSX)
- GateCrasher (Command)
- Portal of Doom (ljsgz.exe)
- Girlfriend (Windll.exe)
- HackATack (Expl32.exe)
- Phase Zero (MsgServ|msgsvr32.exe)
- MyPics Variant (agent5|zip01.exe)
- NewApt (tpawen|panth)
- SubSeven (mtmtask.dl)

By simply changing the Registry key examined, system administrators can verify which HotFixes are installed on each NT system.

3.3 ACLs

Access control lists (ACLs) define which users and groups have what level of access to specific resources. PERMS.PL (Script 4) demonstrates a method for collecting the ACLs from files, directories, Registry keys, and shares, and presenting the ACLs in an easy to understand format. Credit goes to Dave Roth for both his Win32::Perms module, as well as the code for the getperms() method.

3.4 Services

SERVICES.PL (Script 5) demonstrates a method for collecting information from services available on NT systems, such as the service's status, the service executable, and the account that the service runs under. Other methods in the Win32::Lanman module will allow the system administrator to start or stop services. Dave Roth's Win32::Daemon module provides a nice interface for creating services from Perl scripts.

3.5 Auditing and Logging

NT systems have the capability of performing auditing and logging functions. Administrators can use the information available in the EventLog to see if there are problems with applications, look for clean or dirty system shutdowns, or even as a rudimentary intrusion detection system.

3.5.1 Audit Policies

The AUDITPOL.PL (Script 6) script shows how the audit policy can be determined. Auditing can be enabled if the current state shows it to be disabled, and the particular events to be audited can be set.

3.5.2 EventLog Entries

The DUMPEVENTS.PL (Script 7) script demonstrates a means of collecting EventLog entries from multiple machines within the domain. The script collects the entries and sends them to STDOUT. However, the entries can be deposited in an Excel spreadsheet or Access database, making the entries more manageable. Specific events can be filtered during the collection process, such as Dr. Watson messages, failed logon attempts, etc. If ACLs are set to prevent users from writing to specific directories and Registry keys, attempts to do so (as when the user attempts to install software, or when a trojan attempts to install itself) appear as EventID 650 in the Security EventLog.

3.6 User Privileges and User Account Information

The administrator may need to determine the privileges that users have available in order to diagnosis access issues. The Win32::Lanman module provides methods for displaying privileges assigned to particular users or groups, or assigning privileges as necessary.

USERS.PL (Script 8) demonstrates how an administrator might collect information about user accounts from the local NT system. Other Win32::Lanman methods allow the administrator to enumerate global and local groups, or add users and groups.

3.7 Directory and File Contents

STARTUPCHK.PL (Script 9) shows how an administrator can examine the contents of the Start Up directories for all profiles on an NT system. This script will also show the executable files pointed to by shortcuts.

Administrators can also use Perl to get the contents of particular files, such as boot.ini or lmhosts, in order to aid in troubleshooting problems.

4. Conclusions

The system and example scripts presented in this paper are quite simple, yet scalable and easy to maintain. All requirements for a security administration system have been met, to include reducing overall cost. At the same time, system administrators using this model have an understanding of the necessity for maintaining the security of their NT systems. They also have a complete security administration toolkit consisting of resources for NT-specific security information, a powerful scripting language (Perl), and a wide variety of tools and techniques to make their jobs easier and more efficient.

Using other modules, system administrators can perform a wide variety of other centralized security administration tasks. For example, there are modules available that allow the system administrator to interface with the Performance Monitor on remote NT systems to check running processes. System administrators can also use Win32::OLE to interface locally with IIS 4.0's metabase and make configuration changes, or use LWP::UserAgent to test their IIS 4.0 web servers for susceptibility to the latest exploit (note: Rain Forest Puppy's whisker.pl is uses Socket.pm to provide the functionality of LWP::UserAgent).

5. Appendix A: Resources

5.1 Perl on NT

All scripts associated with this paper were written and tested using ActiveState's ActivePerl build 522 on Windows NT 4.0. At the time of this writing, ActivePerl build 522 was available from ftp://ftp.activestate.com/ActivePerl/Windows/5.005/Intel/.

5.2 Extensions

The following extensions are available as part of the default distribution of ActiveState's ActivePerl:

- Win32::TieRegistry
- Win32::Shortcut

The Digest::MD5 module is not installed as part of the standard distribution, but is available via PPM ("ppm install Digest-MD5").

The following extensions are available from Dave Roth's web site, and installable via PPM:

- Win32::AdminMisc
- Win32::Perms

The Win32::Lanman module is available in a zipped archive from Dave Roth's FTP site (ftp://ftp.roth.net/ntperl/Others/Lanman/lanman_1_05.zip) or it can be installed via PPM from http://jenda.mccann.cz/perl/.

5.3 Books and Papers

Schwartz, R. L., Olson, E., and Christiansen, T., "Learning Perl on Win32 Systems", O'Reilly & Associates, 1997

Roth, D., "Win32 Programming: The Standard Extensions", MacMillian Technical Publishing, 1998

Roth, D., "A Network Machine Management System", USENIX's 2nd Large Installation System Administration of Windows NT (LISA-NT) *Conference*, July, 1999

Jumes, J. G., Cooper, N. F., Chamoun, P., and Feinman, T. M., "Windows NT 4.0 Security, Audit, and Control", Microsoft Press, 1999

5.4 Web Sites

The author's web site lists several scripts which could not be listed in this paper due to space requirements (http://www.patriot.net/users/carvdawg/perl.html).

Dave Roth's site is the home of several extremely useful Perl modules, as well as his paper from the Usenix LISA-NT '99 Conference (http://www.roth.net)

Joe Casadonte's web site lists many modules available for NT (http://www.netaxs.com/~joc/perlwin32.html)

6. Appendix B: Scripts

6.1 Digest.pl (Script 1)

```
#! c:\perl\bin\perl.exe
# digest.pl
# Generate MD5 checksums on files to verify file
# integrity
# Use with system files, web pages, etc.  Can also use
# files on mapped drives, but must use complete path.
use strict;
use Digest::MD5;

my $server = shift || Win32::NodeName;
my @files = ("io.sys","ntldr","boot.ini","ntdetect.com”,
            "winnt\\system32\\fpnwclnt.dll");
my $path;
$server =~ y/a-z/A-Z/;
($server eq Win32::NodeName) ? ($path = "c:\\") :
  ($path = "\\\\$server\\c\$\\");

foreach (@files) {
  my $file = $path.$_;
  if (-e $file) {
    my $hash = hash($file);
    print "$server\t$file => $hash\n";
  }
  else {
    print "$file could not be found.\n";
  }
}

# sub() to generate the actual hash
sub hash {
  my ($file) = @_;
  open (FILE, $file) or die "Can't open $file: $!\n";
  binmode(FILE);
  my $digest = Digest::MD5->new->
     addfile(*FILE)->b64digest;
  return $digest;
```

```
}
```

6.2 Regkeys.pl (Script 2)

```
#! c:\perl\bin\perl.exe
# RegKeys.pl
# Collect Registry key values for analysis
use strict;
use Win32::TieRegistry(Delimiter=>"/");

my $server = shift || Win32::NodeName;
my $remote;

if ($remote = $Registry->{"//$server/LMachine"}) {
 \&getRegValues($server);
}
else {
  print "Could not connect to $server Registry.\n";
}

sub getRegValues {
 my($value,$data);
 my %regkeys = ();
# Keys are kept in a datafile, in format: Value;Path
# Ex: EnableDCOM;SOFTWARE/Microsoft/Ole
 my $datafile = "RegKeyValues";
 if (-e $datafile) {
   open(FL,$datafile) || die "Could not open “.
      “ $datafile: $!\n";
   while(<FL>) {
     chomp;
# Skip comments and blank lines
     next if ($_ =~ m/^#/);
     next if ($_ =~ m/^\s+$/);
      my($key,$path) = split(/;/,$_);
     $regkeys{$key} = $path;
    }
     close(FL);
    foreach my $key (keys %regkeys) {
      $value = $remote->{$regkeys{$key}};
      $data = $value->{$key};
      if (defined $data) {
        $data = hex($data) if ($data =~ m/^0x/);
         print "$key = $data\n";
        }
        else {
          print "$key = NotFound\n";
        }
      }
   }
}
```

6.3 TrojanKeys.pl (Script 3)

```
#! c:\perl\bin\perl.exe
# trojankeys.pl
# Check Registry keys where trojans
# like to hide for suspicious entries
use strict;
use Win32::TieRegistry(Delimiter=>"/");

my ($remote);
my $server = shift || Win32::NodeName;

my %hives = ("LMachine" => "HKLM",
             "CUser" => "HKCU");

foreach my $hive (keys %hives) {
 print "Checking $hives{$hive} hive...\n";
 ($remote = $Registry->{"//$server/$hive"}) ?
   (\&getTrojanKeys($server)) :
   (print "Could not connect to “.
     “ $hives{$hive} hive.\n");
  print "\n";
}

sub getTrojanKeys {
 my($path) = 'SOFTWARE/Microsoft/’.
   ‘Windows/CurrentVersion';
 my(@keys) = ('Run','RunOnce','RunOnceEx',
                  'RunServices');
 foreach my $k (@keys) {
   my $key = $remote->{"$path/$k"};
   if (defined $key) {
     my @vals = $key->ValueNames;
     if($#vals != -1) {
       foreach my $val (@vals) {
         my $data = $key->{$val};
        $data = "NotFound" unless($data);
         print "$k:$val:$data\n";
       }
     }
   }
 }
}
```

6.4 Perms.pl (Script 4)

```
#! c:\perl\bin\perl.exe
# Perms.pl
# usage: perl perms.pl [obj]
# ex: perl perms.pl c:\winnt
use strict;
use Win32::Perms;

my $obj = shift ||
 die "You must enter an object: File, Dir,”.
     “ or Reg key.\n";
```

```
\&getperms($obj);

# Credit goes to Dave Roth for providing the
# following code to translate the DACL mask
# into something readable; ie, Explorer-like
# listing
sub getperms {
    my($obj) = @_;
    my($Acct,@List,$Path,$iTotal,@String);
    my($Perm) = new Win32::Perms($obj);

    my (%PERM) = (R   =>  0,
                  W   =>  1,
                  X   =>  2,
                  D   =>  3,
                  P   =>  4,
                  O   =>  5,
                  A   =>  6);

    my(%MAP) = ('FILE_READ_DATA'    =>  'R',
                'GENERIC_READ'      =>  'R',
                'KEY_READ'          =>  'R',
                'KEY_QUERY_VALUE'   =>  'R',
                'FILE_WRITE_DATA'   =>  'W',
                'KEY_WRITE'         =>  'W',
                'GENERIC_WRITE'     =>  'W',
                'KEY_SET_VALUE'     =>  'W',
                'DELETE'            =>  'D',
                'FILE_DELETE_CHILD' =>  'D',
                'FILE_EXECUTE'      =>  'X',
                'FILE_TRAVERSE'     =>  'X',
                'GENERIC_EXECUTE'   =>  'X',
                'CHANGE_PERMISSION' =>  'P',
                'TAKE_OWNERSHIP'    =>  'O',
                'FILE_ALL_ACCESS'   =>  'A',
                'GENERIC_ALL'       =>  'A',
                'STANDARD_RIGHTS_ALL'
                    =>'A');

    die "Can not obtain permissions for '$Path'\n"
        if(!$Perm);

    $Perm->Dump(\@List);

    foreach $Acct (@List) {
        my($Perm);
        my(@String) = split(//, "-"
            x scalar(keys(%PERM)));
        my($Mask,@M,@F);
        my($DaclType);

        next if($Acct->{Entry} ne "DACL");
        $iTotal++;
        DecodeMask($Acct,\@M,\@F);
        foreach $Mask (@M) {
            $Perm |= 2**$PERM{$MAP{$Mask}};
        }

        foreach $Mask (keys(%PERM)) {
            $String[$PERM{$Mask}] = $Mask
              if ($Perm & 2**$PERM{$Mask}) ;
        }

        $DaclType = $Acct->{ObjectName};
        if( 2 == $Acct->{ObjectType} ) {
# We have either a file or directory. Therefore we
#  need to figure out if this
# DACL represents an object (file) or a
# container (dir)...
            ($Acct->{Flag} & DIR) ?
                ($DaclType = "Directory") :
                ($DaclType = "File");
        }
        my $permstr = join("",@String);
        print "$Acct->{Account}:$DaclType:”.
            “$permstr\n”;
    }
    print "Everyone has full permissions.\n" if(!$iTotal);
}
```

6.5 Services.pl (Script 5)

```
#! c:\perl\bin\perl.exe
# Services.pl
# Retrieve info on services on $server
use strict;
use Win32::Lanman;

my $server = shift || Win32::NodeName;

\&Services($server);

sub Services {
  my($server) = @_;
# Array to translate the current state of the service into
# something readable
  my(@state) = ("","Stopped","Start_Pending",
          "Stop_Pending",  "Running",
           "Continue_Pending",
          "Pause_Pending","Paused");

# Array to translate the startup options for the service
# into something readable
  my(@startup) = ("","","Automatic","Manual",
    "Disabled");
  my($err,@services,$service,%info);
  if (Win32::Lanman::EnumServicesStatus(
```

```
    "\\\\$server","",&SERVICE_WIN32,
   &SERVICE_STATE_ALL,\@services)) {
   foreach $service (@services) {
     if (Win32::Lanman::QueryServiceConfig(
       "\\\\$server","",${$service}{name},
        \%info)) {

# Print out in an easy to read format
       print "${$service}{display}\n";
       print "\tName\t${$service}{name}\n";
       print "\tState\t".$state[${$service}{state}]."\n";
       print "\tAccount\t$info{account}\n";
       print "\tFile\t$info{filename}\n";
       print "\tStart\t".$startup[$info{start}]."\n";
       print "\n";
     }
     else {
       $err = Win32::FormatMessage
         Win32::Lanman::GetLastError();
       $err = Win32::Lanman::GetLastError()
          if ($err eq "");
       print "$server: QueryServiceConfig Error: $err";
     }
   }
 }
 else {
  $err = Win32::FormatMessage
    Win32::Lanman::GetLastError();
  $err = Win32::Lanman::GetLastError()
    if ($err eq "");
  print "$server: Services Error: $err";
 }
}
```

6.6 AuditPol.pl (Script 6)

```
#! c:\perl\bin\perl.exe
# auditpol.pl
# Determine the audit policy of an NT system
use strict(vars);
use Win32::Lanman;

my $server = shift || Win32::NodeName;

\&AuditPol($server);
sub AuditPol {
 my($server) = @_;
 my(%info);
 my(%hash) = (
 "AuditCategoryLogon" => "Logon and Logoff",
 "AuditCategoryObjectAccess" => "File and
     Object Access",
 "AuditCategoryPrivilegeUse" => "Use of
     User Rights",
 "AuditCategoryAccountManagement" =>
     "User/Group Mgmt",
 "AuditCategoryPolicyChange" => "Security Policy
       Changes",
 "AuditCategorySystem" => "Restart and
       Shutdown System",
 "AuditCategoryDetailedTracking" =>
       "Process Tracking");

 my(%settings) = (0 => "None",
                  1 => "Success",
                  2 => "Failure",
                  3 => "Success and Failure");

if (Win32::Lanman::LsaQuery-
 AuditEventsPolicy("\\\\$server",\%info)) {
  my $audit = $info{auditingmode};
  if (!$audit) {
   print "$server: Auditing is NOT enabled!\n";
  }
  else {
    print "\n";
    printf "%-35s %-20s\n","Audit Events","Settings";
    printf "%-35s %-20s\n","------------","--------";
    my $options = $info{eventauditingoptions};
    foreach my $key (keys %hash) {
       printf "%-35s %-20s\n",
         $hash{$key},$settings{$$options[&$key]};
    }
  }
 }
 else {
  my $err = Win32::FormatMessage
     Win32::Lanman::GetLastError();
   print "$server: Audit Error:  $err\n";
 }
}
```

6.7 DumpEvents.pl (Script 7)

```
#! c:\perl\bin\perl.exe
# Dumpevents.pl
use strict;
use Win32::Lanman;
use Win32::Perms;
my $server = shift || Win32::NodeName;
Win32::Perms::LookupDC(0);

\&GetEvents($server,"Security");

sub GetEvents {
 my($server,$evtlog) = @_;
 my(@events,$event,$desc);
```

```
  my %types = (1 => "(Error)",
               4 => "(Information)",
               8 => "(Success Audit)",
               16 => "(Failure Audit)");

  my %category = (0 => "(None)",
                  1 => "(System Event)",
                  2 => "(Logon/Logoff)",
                  3 => "(Object Access)",
                  4 => "(Privilege Use)");

if(Win32::Lanman::ReadEventLog("\\\\$server",
  $evtlog, 0xffffffff, 0, \@events)) {
  foreach $event (@events) {
    print"Computer: ".${$event}{computername}."\n";
    print "Category:".${$event}{eventcategory}."
         ".$category{${$event}{eventcategory}}."\n";
    my $id = (${$event}{eventid} & 0xffff);
# Use the EventID to filter on specific types of events
    print "Event ID:    ".$id."\n";
    print "EventType: ".${$event}{eventtype}."
         ".$types{${$event}{eventtype}}."\n";
    print "Source: ".${$event}{source}."\n";
    print "SourceName:  ".${$event}{sourcename}."\n";
    print"Generated:   ".localtime(${$event}
      {timegenerated})."\n";
    print "Written: ".localtime(${$event}
      {timewritten})."\n";
    print "Flags: ".${$event}{reservedflags}."\n";
    print "User: ".Win32::Perms::
      ResolveAccount(${$event}{usersid})."\n";
    print "Description: ";
    if (Win32::Lanman::GetEvent
      Description("\\\\$server", $event)) {
       $desc = ${$event}{eventdescription};
        print $desc."\n";
    }
    else {
      my $strings = ${$event}{strings};
       print "\n";
      foreach (@$strings) {
         print "\t+".$_."\n";
       }
    }
# print "Data:".unpack("H".2*length(${$event}
# {data}), ${$event}{data})."\n"
#    if (${$event}{data} ne "");
     print "\n\n";
   }
 }
  else {
    my $err = Win32::FormatMessage
        Win32::Lanman::GetLastError();
    $err = Win32::Lanman::GetLastError()
        if ($err eq "");
    print "$server:  ReadEventLog error: $err.\n";
  }
}
```

6.8 Users.pl (Script 8)

```
#! c:\perl\bin\perl.exe
# Users.pl
# Get user info from local system
use strict;
use Win32::Lanman;

my $server = shift || Win32::NodeName;
print "$server users...\n";

\&GetUserInfo($server);

sub GetUserInfo {
  my($server) = @_;
 my(@users,$user,$pwage);

  if (Win32::Lanman::NetUserEnum
    ("\\\\$server",0,\@users)) {
    foreach my $user (@users) {
     $pwage = (split(/\./,${$user}{'password_age'}))
        /(3600*24);
     print "${$user}{'name'}\n";
      print "\tComment     => ${$user}{'comment'}\n";
      print "\tUID         => ${$user}{'user_id'}\n";
      print "\tPasswd Age  => $pwage\n";
#     print "\tLogon Svr   =>
#       ${$user}{'logon_server'}\n";
      print "\tLast Logon  =>
        ".mlocaltime(${$user}{'last_logon'})."\n";
      print "\tLast Logoff =>
        ".mlocaltime(${$user}{'last_logoff'})."\n";
      print "\tAccount does not expire.\n"
        if (${$user}{'acct_expires'} == -1);
      print "\tACCOUNT DISABLED.\n"
        if (${$user}{'flags'} &
           UF_ACCOUNTDISABLE);
     print "\tUser cannot change password.\n"
        if (${$user}{'flags'} &
         UF_PASSWD_CANT_CHANGE);
      print "\tAccount is locked out.\n"
        if (${$user}{'flags'} & UF_LOCKOUT);
      print "\tPassword does not expire.\n"
       if (${$user}{'flags'} &
       UF_DONT_EXPIRE_PASSWD);
     print "\tPassword not required.\n"
       if (${$user}{'flags'} &
        UF_PASSWD_NOTREQD);
      print "\n\n";
```

```
    }
  }
  else {
    my $err = Win32::FormatMessage
       Win32::Lanman::GetLastError();
    $err = Win32::Lanman::GetLastError()
         if ($err eq "");
      print "NetUserEnum Error:  $err\n";
  }
}

sub mlocaltime {
  ($_[0] == 0) ? (return "Never") :
     (return localtime($_[0]));
}
```

6.9 StartUpChk.pl (Script 9)

```
#! c:\perl\bin\perl.exe
# StartUpChk.pl
# Checks contents of StartUp folder
# for each profile
use strict;
use Win32::Shortcut;

my $server = shift || Win32::NodeName;
my $me = Win32::NodeName;

my $startdir;
($server eq $me) ? ($startdir = "c:\\") :
  ($startdir = "\\\\$server\\c\$\\");

my $start = $startdir."winnt\\profiles\\";
my $startup = "\\start menu\\programs\\startup";
my ($dir,$err);

if (-e $start && -d $start) {
  opendir(ST,"$start");
  foreach $dir (sort readdir(ST)) {
    next if ($dir eq "." || $dir eq "..");
    my $dir2 = $start.$dir;
    if (-e $dir2 && -d $dir2) {
      my $newdir = "$start".$dir."$startup";
      if (-e $newdir && -d $newdir) {
        opendir(SUP,$newdir);
             my @files = readdir(SUP);
             closedir(SUP);
             if (@files) {
               foreach my $file (@files) {
                 next if ($file eq "." || $file eq "..");
                 if ($file =~ m/lnk$/) {
                      my $shortcut = Win32::Shortcut
                         ->new($newdir."\\".$file);
                  ($shortcut) ? (print
                    "$dir:$file:$shortcut->{Path}\n") :
                 (print "Error with
                    Shortcut: ".Win32::FormatMessage
                    Win32::GetLastError."\n");
               }
                 else {
                  print "$dir:$file\n";
                  }
               }
             }
           }
           else {
             print "$newdir does not exist or is not”.
                  “ a directory.\n";
           }
        }
         else {
           print "$dir2 does not exist or is not a”.
             “ directory.\n";
         }
       }
    closedir(ST);
  }
  else {
     print "$start does not exist or is not a direc-
tory.\n";
  }
```