KAPITOLA 7

PowerShell a sítě

V této kapitole:

- Síťová konfigurace Windows
- Klienty síťových služeb
- Přenos souborů a BITS
- Inventarizace a diagnostika sítě

Správa síťových nastavení a sledování síťového provozu neodmyslitelně patří do popisu práce správce. V tomto světle se PowerShell jeví jako – opatrně řečeno – mírně zanedbaný administrátorský prostředek. Jeho tvůrci téměř ignorovali „síťové potřeby" uživatelů a správců, tedy alespoň na první pohled to tak vypadá. V této kapitole si ukážeme, že situace není tak špatná. Ve skutečnosti asi nechtěli tvůrci PowerShellu jen podruhé vytvářet to, co je hotové a stačí to uchopit. Nastavení sítě a správa provozu je zabalena do řady zajímavých tříd, k nimž můžeme přistoupit přes dobře ovladatelná rozhraní. Síťové adaptéry jsou výborně reprezentovány rozhraním WMI, firewall je přístupný přes rozhraní COM a .NET Framework pak nabízí škálu zajímavých tříd roztodivných možností – od webového klientu po statistiky síťového provozu. Tato kapitola nám ukáže, že není třeba hledat nové, neexistující příkazy PowerShellu, neboť řadu síťových úkolů zvládneme dobře s existujícími možnostmi.

Tip: Autor musí na tomto místě připustit, že ve Windows existuje jeden nástroj pro konfiguraci sítí a síťových služeb, bez nějž se v PowerShellu neobejdeme. Program Netsh je tak všestranný a potřebný, že jej přinejmenším čas od času budeme muset použít pro některá nastavení. Jde to opravdu výborný konfigurační prostředek.
http://technet.microsoft.com/en-us/library/cc725935%28WS.10%29.aspx

Správa síťové konfigurace Windows

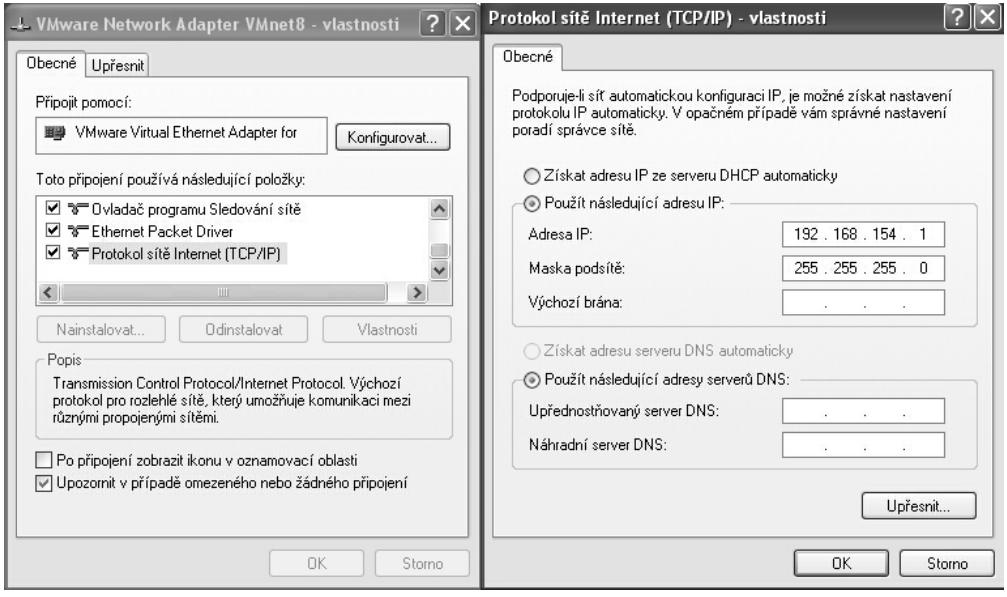
Úvod

Tvůrci PowerShellu se nijak zvlášť nevěnovali problematice nastavení síťových rozhraní a jejich správy, takže PowerShell je v tomto směru odkázán na zprostředkované možnosti již dříve používaných rozhraní.

Síťová nastavení lze jak číst, tak ovládat pomocí všestranného rozhraní WMI a jeho tříd. Mezi nejdůležitější patří třída *Win32_NetworkAdapterConfiguration*, která nabízí řadu nastavení síťových adaptérů. Poslouží nám velmi dobře především při konfiguraci protokolu IP, tedy hlavně adres a serverů DNS pro překlady jmen.

Pokud provádíme prvotní inventuru, můžeme nejdříve využít filtru pro hledání tak, abychom obdrželi do roury pouze objekty síťových karet s opravdu aktivním protokolem IP.

```
=>Get-WmiObject win32_networkadapterconfiguration -Filter "ipenabled = 'true'"

...
DHCPEnabled      : True
IPAddress        : {0.0.0.0}
DefaultIPGateway :
DNSDomain        :
ServiceName      : EL2000
Description      : 3Com Gigabit LOM (3C940) - Virtual Machine Network Services Driver
Index            : 4
...
```



Obrázek 7.1: Rozhraní WMI poskytuje přístup k tradičním konfiguračním parametrům síťových rozhraní

Výběr konkrétního adaptéru pro další práci provedeme nejlépe pomocí jeho vlastnosti *Index*, jejíž hodnota je jedinečná (v řeči rozhraní WMI jde o vlastnost „klíčovou" – *key propery*).

```
=>Get-WmiObject win32_networkadapterconfiguration -Filter "index = 4"

DHCPEnabled      : True
IPAddress        : {0.0.0.0}
DefaultIPGateway :
DNSDomain        :
ServiceName      : EL2000
Description      : 3Com Gigabit LOM (3C940) - Virtual Machine Network Services Driver
Index            : 4
```

Konfigurace síťového adaptéru je zpřístupněna pomocí řady metod, jež nám nabízí zde použitá třída WMI. PowerShell 2 přináší velmi užitečný cmdlet, pomocí něhož bude volání potřebných metod poměrně snadné. Nejdříve si připravíme parametry pro metodu, která nám nastaví novou statickou konfiguraci IP, a posléze ji zavoláme na síťovém adaptéru, který načteme do roury jako objekt.

```
$addr = @()
$addr += "192.168.1.10"
$mask = @()
$mask += "255.255.255.0"

=>Get-WmiObject win32_networkadapterconfiguration -Filter "index = 4"  | `
ForEach-object {Invoke-WmiMethod -Name enablestatic -ArgumentList $addr,$mask `
-InputObject $_ }
```

```
__GENUS          : 2
__CLASS          : __PARAMETERS
__SUPERCLASS     :
__DYNASTY        : __PARAMETERS
__RELPATH        :
__PROPERTY_COUNT : 1
__DERIVATION     : {}
__SERVER         :
__NAMESPACE      :
__PATH           :
ReturnValue      : 0
```

Povšimněme si několika důležitých věcí. Cmdlet pro přímé volání metody WMI se jmenuje *Invoke-WMIMethod* a jméno metody uvádíme jako parametr za přepínačem *-Name*. Další důležitý přepímač *-ArgumentList* nese potřebné parametry metody samotné – my jsme si je připravili předem do dvou proměnných. V úvodu ukázky je vidět, že tyto proměnné jsou definovány jako pole, neboť metoda na vstupu pole vyžaduje (a díky tomu dovoluje nastavit více adres IP na jeden adaptér). Na závěr nesmíme přehlédnout důležitou hodnotu ve výpisu po zavolání metody – *ReturnValue* nám ukazuje, jak operace dopadla, a ve spolupráci s dokumentací k dané metodě nám poskytne cenné informace o výsledku volání metody. Uděláme dobře, když si tuto hodnotu uložíme do proměnné k dalšímu prozkoumání a testování.

Nový cmdlet pro volání metod WMI není jedinou cestou, jak tyto metody spustit. Stejný úkon dobře provedeme i tak, že si síťovou konfiguraci uložíme do objektové proměnné a přímo na ní pak voláme příslušnou metodu s přímým předáním parametrů.

```
=>$sit = Get-WmiObject win32_networkadapterconfiguration -Filter "index = 4"
=>$vystup = $sit.EnableStatic("172.16.10.5","255.255.0.0")
=>$vystup | fl returnvalue

returnvalue : 81
```

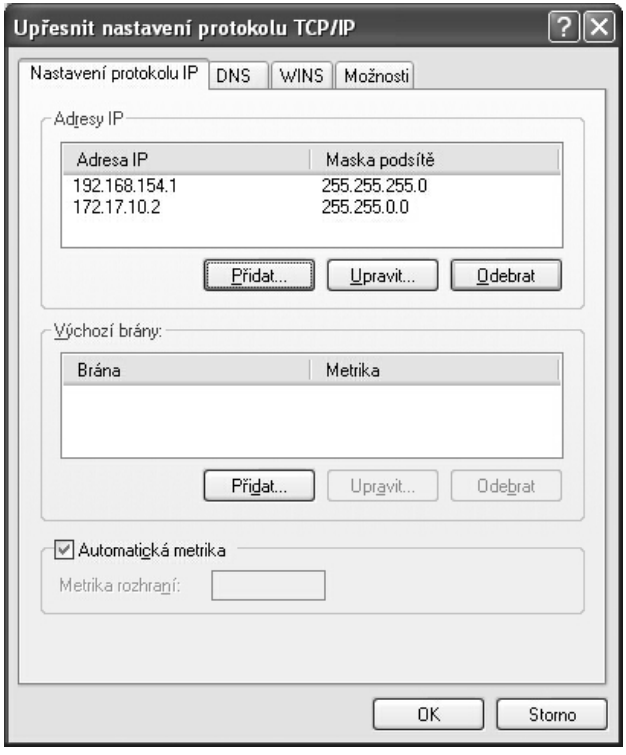
Návratová hodnota je tentokrát jiná než nula, což naznačuje, že operace skončila nezdarem – přímý průzkum nové konfigurace a následné nahlédnutí do dokumentace však prozradí, že přiřazení se podařilo, přestože je indikována chyba. Je to proto, že adaptér již měl statickou konfiguraci, a nemohlo tedy dojít k jejímu zapnutí (z konfigurace dynamické, DHCP). Adresa se však přiřadila správně.

Přiřazení více adres IP síťovému adaptéru

Síťové rozhraní spravované operačním systémem Windows může používat více adres IP najednou, a tím tedy patřit do více sítí, jejichž příslušnost určuje síťová maska. Takovéto přiřazení můžeme také provést pomocí rozhraní WMI, jak si ukážeme zde.

1. Založíme si proměnné typu pole, do nichž načteme budoucí adresy IP a síťové masky. Proměnné jako pole volíme proto, že právě takovýto vstupní formát vyžaduje příslušná metoda WMI.

```
$adresy = @()
$masky = @()
```



Obrázek 7.2: Síťové rozhraní můžeme vybavit i více adresami IP pro obsluhu klientů ve více sítích

2. Připravíme si zdrojový soubor CSV, v němž budou adresy IP a příslušné síťové masky. Tento soubor poté načteme do připravených polí.

```
=>cat .\adresy.csv

adresa,maska
172.16.10.25,255.255.0.0
192.168.15.5,255.255.255.0

=>Import-Csv .\adresy.csv

adresa                                              maska
------                                              -----
172.16.10.25                                        255.255.0.0
192.168.15.5                                        255.255.255.0

Import-Csv .\adresy.csv | ForEach-Object `
{$adresy += $_.adresa; $masky += $_.maska}

=>$adresy
172.16.10.25
192.168.15.5
```

```
=>$masky
255.255.0.0
255.255.255.0
```

3. Vstupní data máme připravena v polích, a proto můžeme volat příslušnou metodu, jež přiřadí statickou konfiguraci IP.

```
$sit = Get-WmiObject win32_networkadapterconfiguration -Filter "index = 10"

$return = $sit.EnableStatic($adresy,$masky)
$return.returnvalue
0
```

4. Nové nastavení můžeme samozřejmě prověřit.

```
=>$sit = Get-WmiObject win32_networkadapterconfiguration -Filter "index = 10"
=>$sit | select IPAddress, ipsubnet | ft *

IPAddress                                             ipsubnet
---------                                             --------
{172.16.10.25, 192.168.15.5}                          {255.255.0.0, 255.255.255.0}
```

Hromadné přiřazení parametrů tedy není nijak složité, když si uvědomíme, že vstupním formátem je pole řetězců, jež reprezentují adresy a masky. Jejich příprava je v PowerShellu poměrně snadná.

Přiřazení základních parametrů síťového rozhraní

Jakékoliv síťové rozhraní potřebuje sadu výchozích parametrů, díky nimž bude schopno komunikovat se síťovými protokoly. Protokolová sada TCP/IP se dnes používá prakticky beze zbytku i ve světě Windows, a proto si ukážeme, jak nastavíme všechny důležité parametry síťového adaptéru prostřednictvím PowerShellu.

ipconf.txt, ipconf.ps1

1. Konfigurační údaje uložme do přehledného konfiguračního souboru, jehož struktura bude určovat všechny potřebné parametry TCP/IP.

```
=>cat .\ip_conf.txt

adresa=192.168.25.10
maska=255.255.255.0
brana=192.168.25.1
dns1=172.16.1.2
dns2=172.16.1.253
wins=172.16.1.253
```

2. První krok našeho skriptu načte zdrojová data do jediného textového řetězce, s nímž budeme dále pracovat.

```
=>$conf = (cat ip_conf.txt) -join "`n"

=>$conf

adresa=192.168.25.10
```

```
maska=255.255.255.0
brana=192.168.25.1
dns1=172.16.1.2
dns2=172.16.1.253
wins=172.16.1.253
```

3. Celý textový řetězec převedeme na hešovací tabulku (kolekci dvojic), s níž budeme následně snadno manipulovat při nastavování parametrů.

```
=>$conf_table = ConvertFrom-StringData $conf
=>$conf_table

Name                           Value
----                           -----
maska                          255.255.255.0
dns2                           172.16.1.253
brana                          192.168.25.1
adresa                         192.168.25.10
wins                           172.16.1.253
dns1                           172.16.1.2
```

4. Vytvoříme si proměnnou, jež bude zastupovat síťové rozhraní, s nímž budeme pracovat.

```
=>$interface = Get-WmiObject win32_networkadapterconfiguration `
-Filter 'index=10'

=>$interface

DHCPEnabled      : False
IPAddress        : {0.0.0.0}
DefaultIPGateway :
DNSDomain        :
ServiceName      : EL2000
Description      : 3Com Gigabit LOM (3C940)
Index            : 4
```

5. Nejdříve přiřadíme adresu protokolu IP a spolu s ní nepostradatelnou síťovou masku. Vzápětí prověříme, jestli operace dopadla dobře.

```
=>$return = $interface.EnableStatic($conf_table.Item("adresa"),$conf_table.
Item("maska"))
=>$return.ReturnValue
81
```

Jak jsme si již ukázali v jednom z předchozích příkladů, výstupní hodnota 81 označuje sice opakované, ale úspěšně přiřazení statické síťové adresy.

6. Budeme pokračovat dalším logickým krokem, kterým je přiřazení brány sítě.

```
=>$return = $interface.SetGateways($conf_table.Item("brana"))
=>$return.ReturnValue
0
```

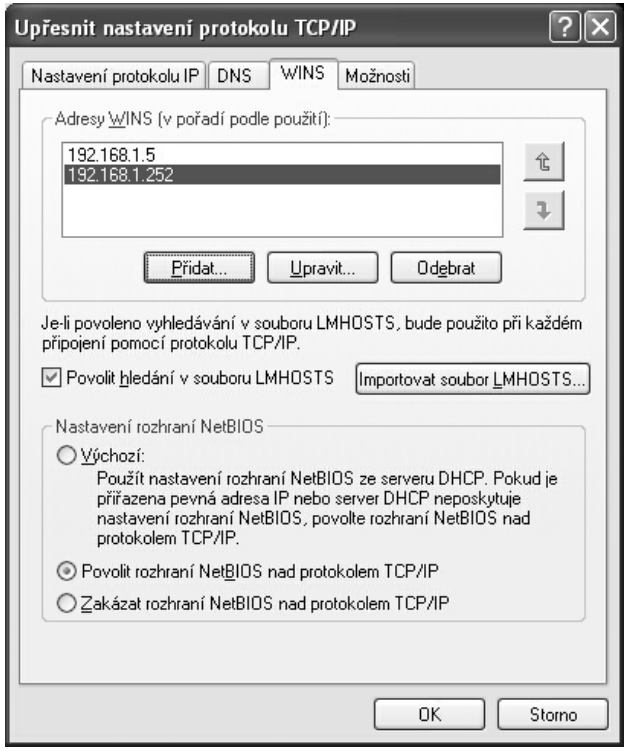
7. V dalším kroku přiřadíme adresy serverů DNS pro překlad síťových jmen.

```
=>$dns_servery = $conf_table.Item("dns1"),$conf_table.Item("dns2")
=>$return = $interface.SetDNSServerSearchOrder($dns_servery)
```

8. Obdobným způsobem můžeme též přiřadit adresu serverů služby WINS.

```
=>$return = $interface.SetWINSServer($conf_table.Item("wins")
=>$return.ReturnValue
68

=>$return = $interface.SetWINSServer($conf_table.Item("wins"),$conf_table.
Item("dns1"))
=>$return.ReturnValue
0
```

Povšimněme si důležité vlastnosti metody *SetWINSServer*: na vstupu přijímá dva parametry (primární a sekundární server), a proto nám první pokus o konfiguraci nevyšel, což ukázala návratová hodnota.



Obrázek 7.3: Konfigurace serverů služby WINS musí obsahovat dvě položky: primární a sekundární server

9. Na závěr můžeme konfiguraci ještě prověřit.

```
=>Get-WmiObject win32_networkadapterconfiguration `
-Filter 'index=10' | fl `
IPAddress,IPSubnet,DefaultIPGateway,DNSServerSearchOrder,Wins*server
```

```
IPAddress            : {192.168.25.10}
IPSubnet             : {255.255.255.0}
DefaultIPGateway     : {192.168.25.1}
DNSServerSearchOrder : {172.16.1.2, 172.16.1.253}
WINSPrimaryServer    : 172.16.1.253
WINSSecondaryServer  : 172.16.1.2
```

Záloha a obnova nastavení síťových rozhraní

Schopnosti PowerShellu a rozhraní WMI můžeme výborně využít pro zálohování a případnou obnovu nastavení rozhraní IP. V následujícím postupu nám pomohou osvědčené techniky, jako jsou export a import do souboru XML.

1. Nastavení síťových rozhraní načteme pomocí rozhraní WMI a pošleme rourou na výstup do souboru XML.

```
=>Get-WmiObject win32_networkadapterconfiguration -Filter 'ipenabled=true' `
| Export-Clixml ip_conf.xml
```

2. Uložená data můžeme kdykoliv načíst a PowerShell je transformuje do požadované objektové podoby.

```
=>$restore = Import-Clixml .\ip_conf.xml | where {$_.index -eq 10}

=>$restore.IPAddress
172.16.10.25
192.168.15.5

=>$restore.IPSubnet
255.255.0.0
255.255.255.0
```

3. Takto připravená data lze již snadno použít k opětovné konfiguraci síťového rozhraní.

```
=>$sit = Get-WmiObject win32_networkadapterconfiguration -Filter 'index=10'
=>$sit.EnableStatic($restore.IPAddress,$restore.IPSubnet)
...
ReturnValue      : 0
```

4. A na závěr kontrola nových nastavení.

```
=>$sit = Get-WmiObject win32_networkadapterconfiguration -Filter 'index=10'

=>$sit | select ipaddress,ipsubnet

ipaddress                                   ipsubnet
---------                                   --------
{172.16.10.25, 192.168.15.5}                {255.255.0.0, 255.255.255.0}
```

Ovládání firewallu Windows

Operační systém Windows disponuje již poměrně dlouhou dobu zabudovaným „personálním“ firewallem, tedy mechanizmem, který kontroluje provoz na místních síťových rozhraních. Jeho podoba se postupně vyvíjela a také byl postupně zabudován vedle klientských variant Windows i do serverových systémů. Firewall můžeme ovládat prostřednictvím rozhraní COM a je důležité si uvědomit, že se postupy mírně liší dle varianty Windows. Ukážeme si jak starší verzi XP (tam se součást nazývá Windows Firewall), tak novější Windows 7 (kde se komponenta nazývá Windows Firewall with Advanced Security).

Obrázek 7.4: Firewall systému Windows můžeme dobře ovládat prostřednictvím rozhraní COM

A. Ovládání firewallu ve Windows XP.

1. Nastavení firewallu jsou reprezentována kolekcí objektů, k nimž přistoupíme pomocí rozhraní COM. Nejdříve začneme „uchopením“ výchozí instance a poté si uložíme profil nastavení firewalu přímo do další proměnné.

```
=>$firewall = New-Object -ComObject HnetCfg.FwMgr
=>$nastaveni = $firewall.LocalPolicy.CurrentProfile
=>$nastaveni

Type                                    : 1
```

```
FirewallEnabled                                 : False
ExceptionsNotAllowed                            : False
NotificationsDisabled                           : False
UnicastResponsesToMulticastBroadcastDisabled : False
RemoteAdminSettings                             : System.__ComObject
IcmpSettings                                    : System.__ComObject
GloballyOpenPorts                               : System.__ComObject
Services                                        : System.__ComObject
AuthorizedApplications                          : System.__ComObject
```

2. Zběžný výpis v předchozím kroku nám dobře ukazuje, že některé možnosti nastavení jsou již přímo dostupné. Firewall zde můžeme snadno zapnout/vypnout nebo třeba povolit/zakázat výjimky. Pro samotné zapnutí firewallu nemusíme volat žádnou metodu, vlastnost můžeme zapsat rovnou.

```
=>$nastaveni.FirewallEnabled = $true
=>$nastaveni.FirewallEnabled
True
=>$nastaveni.FirewallEnabled = $false
=>$nastaveni.FirewallEnabled
False
```

3. Řada nastavení firewallu je dále „zakuklena" v objektech, jak je z výpisu patrné. Můžeme prověřit služby a jejich povolení a stejně tak zkontrolujeme povolené aplikace.

```
=>$nastaveni.Services | select name,enabled,remoteaddresses

Name                                                  Enabled RemoteAddresses
----                                                  ------- ---------------
Sdílení souborů a tiskáren                               True LocalSubNet
Architektura UPnP                                       False LocalSubNet
Vzdálená plocha                                          True *

=>$nastaveni.AuthorizedApplications | select name,enabled,remoteaddresses

Name                                                  Enabled RemoteAddresses
----                                                  ------- ---------------
Vzdálená pomoc                                           True *
Java(TM) 2 Platform Standard E...                        True *
Windows Live Messenger 8.1 (Ph...                        True *
Nástroj Diagnostika sítě systé...                        True *
Windows Live Messenger                                   True *
Microsoft Office Outlook                                 True *
Skype                                                    True *
```

4. Do seznamu povolených aplikací můžeme samozřejmě přidat nějakou další. Za tím účelem si vyrobíme novou proměnnou dle dané třídy a postupně ji vybavíme vlastnostmi. Na závěr ji přidáme do seznamu pomocí metody *Add*.

```
=>$app = New-Object -ComObject HNetCfg.FwAuthorizedApplication
=>$app
```

```
Name                 :
ProcessImageFileName :
IpVersion            : 2
Scope                : 0
RemoteAddresses      : *
Enabled              : True

=>$app.name = "Firefox"
=>$app.ProcessImageFileName = "C:\Program Files\Mozilla Firefox\firefox.exe"
=>$app.Enabled = $true

=>$nastaveni.AuthorizedApplications.Add($app)

=>$nastaveni.AuthorizedApplications | where {$_.name -like "firefox"}


Name                 : Firefox
ProcessImageFileName : C:\Program Files\Mozilla Firefox\firefox.exe
IpVersion            : 2
Scope                : 0
RemoteAddresses      : *
Enabled              : True
```

5. Podobně jako v předchozím případě můžeme povolit nějaký určitý port. I v tomto případě máme k dispozici třídu, která takovouto položku reprezentuje, takže začneme tvorbou nové instance.

```
=>$port = New-Object -ComObject HnetCfg.FwOpenPort

=>$port.name = "LDAP"
=>$port.Port = 389
=>$port.Enabled = $true

=>$nastaveni.GloballyOpenPorts.Add($port)

=>$nastaveni.GloballyOpenPorts | select name,port,enabled

Name                                                Port                Enabled
----                                                ----                -------
HTTP                                                 80                  True
HTTPS                                                443                 True
Windows Remote Management                            5985                True
LDAP                                                 389                 True
```

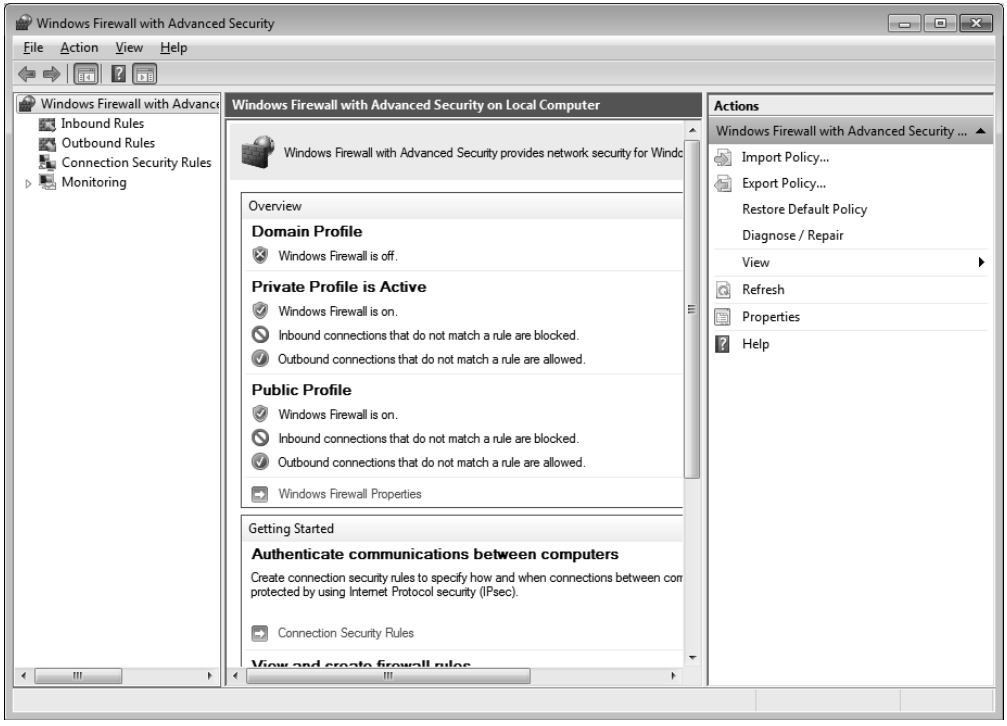
6. Pro případ, že bychom nastavení upravili více, než je žádoucí, můžeme sáhnout po výchozím nastavení.

```
=>$firewall.RestoreDefaults()
```

Tip: Při práci s firewallem doporučujeme využít následujících referenčních informací:
http://technet.microsoft.com/en-us/library/cc737845%28WS.10%29.aspx
Stránka poskytuje velmi důkladný popis potřebných objektů a jejich členů.

B. ovládání firewallu ve Windows 7

W7Firewall.ps1



Obrázek 7.5: Služba firewall byla ve Windows 7 zcela zásadně přepracována. Přesto máme i nadále možnost ji ovládat pomocí rozhraní COM.

1. Pokročilejší firewall v novějších verzích Windows se ovládá pomocí stejné objektové knihovny COM, avšak firewall a jeho součásti reprezentují nové třídy. Nejdříve si tedy založíme proměnnou a do ní uložíme výchozí nastavení firewallu. Bližší průzkum ukáže, že objekt je mírně odlišný od svého předchůdce.

```
=>$firewall = new-object -ComObject HnetCfg.FwPolicy2

=>$firewall | Get-Member
   TypeName: System.__ComObject#{98325047-c671-4174-8d81-defcd3f03186}

Name                                    MemberType            Definition
----                                    ----------            ----------
EnableRuleGroup                         Method                void Enabl
IsRuleGroupEnabled                      Method                bool IsRul
RestoreLocalFirewallDefaults            Method                void Resto
BlockAllInboundTraffic                  ParameterizedProperty bool Block
DefaultInboundAction                    ParameterizedProperty NET_FW_ACT
DefaultOutboundAction                   ParameterizedProperty NET_FW_ACT
ExcludedInterfaces                      ParameterizedProperty Variant Ex
```

```
FirewallEnabled                                ParameterizedProperty bool Firew
IsRuleGroupCurrentlyEnabled                    ParameterizedProperty bool IsRul
NotificationsDisabled                          ParameterizedProperty bool Notif
UnicastResponsesToMulticastBroadcastDisabled ParameterizedProperty bool Unica
CurrentProfileTypes                            Property              int Curren
LocalPolicyModifyState                         Property              NET_FW_MOD
Rules                                          Property              INetFwRule
ServiceRestriction                             Property              InetFwServ
```

2. Základní možnost vypnutí/zapnutí firewallu můžeme využít stejně jako u starší verze, ovšem narážíme zde hned na nutnost použití profilů. Tyto určují chování firewallu za určitých situací – firewall je tedy zapnut či vypnut vždy pro určitý profil.

```
=>$firewall.FirewallEnabled(1)
True
=>$firewall.FirewallEnabled(1) = $false
=>$firewall.FirewallEnabled(1)
False
```

Tabulka shrnuje typy profilů a jejich číselné označení, jež jsme výše použili.

Tabulka 7.1: Nastavení firewallu jsou vázána na určitý profil, který ve výchozí konfiguraci odráží potenciální nebezpečnost připojené sítě – můžeme s nimi pracovat pomocí těchto číselných označení

Profil firewallu	Číselné označení
Domain	1
Private	2
Public	4

Při práci s firewallem a některými jeho parametry se musíme důsledně držet jednoho z profilů, neboť každý z nich má nezávislou sadu nastavení.

3. Tento krok ukazuje vytvoření aplikačního pravidla. Postup je založen na tvorbě nové instance třídy reprezentující pravidla, takže je pouze potřeba při seznamování dobře prozkoumat dostupné vlastnosti.

```
=>$pravidlo = New-Object -ComObject HnetCfg.FWRule
=>$pravidlo.Name = "LDAP"
=>$pravidlo.Description = "Provoz LDAP"
=>$pravidlo.Protocol = 6 #tcp
=>$pravidlo.LocalPorts = 390
=>$pravidlo.ApplicationName = "c:\LDAP\ldap.exe"
=>$pravidlo.Action = 1 #Povolit provoz
=>$pravidlo.Profiles = 4
=>$pravidlo.Enabled = $true
=>$firewall.Rules.Add($pravidlo)

=>$firewall.Rules | where {$_.name -like "ldap"}

Name                  : LDAP
Description           : Provoz LDAP
ApplicationName       : c:\LDAP\ldap.exe
```

```
serviceName          :
Protocol             : 6
LocalPorts           : 390
RemotePorts          : *
LocalAddresses       : *
RemoteAddresses      : *
IcmpTypesAndCodes    :
Direction            : 1
Interfaces           :
InterfaceTypes       : All
Enabled              : True
Grouping             :
Profiles             : 4
EdgeTraversal        : False
Action               : 1
EdgeTraversalOptions : 0
```

Příklad dobře naznačuje, že koncepce je v tomto případě stejná jak pro starší, tak pro novější variantu firewallu. Rozdíly jsou tedy především v používání profilů s podstatně většími možnostmi přiřazení v závislosti na typu síťového rozhraní.

Tip: Také ovládání této varianty firewallu je dobře zdokumentováno, a to na následujících stránkách:

http://msdn.microsoft.com/en-us/library/aa366418%28VS.85%29.aspx

Shrnutí

Konfigurace síťových rozhraní a jejich nastavení je v PowerShellu velmi dobře možná, ovšem vyžaduje použití jiných rozhraní než jen .NET Framework. Prostředky WMI jsou již prověřeny léty užívání a dovolují kompletně nastavit síťová rozhraní. Komponenty COM reprezentují firewall ve Windows a rovněž nabízejí pokročilé možnosti kompletního nastavení.

Důležité k zapamatování

- Třída WMI jménem *WIN32_NetworkAdapterConfiguration* je klíčem ke konfiguraci síťových rozhraní.
- Některé metody výše zmíněné třídy přijímají na vstupu parametry typu pole (třeba adresy IP), což je potřeba respektovat s tím, že si takovéto vstupní proměnné nejlépe předem připravíme.
- Metody výše uvedené třídy jsou schopny při konfiguraci síťového rozhraní vracet širokou škálu návratových hodnot, jež ukazují na skutečný průběh vyžádaných změn. Určitě mějte při ruce dokumentaci a dobře otestujte, co metody vrátí a zda je tak změna úspěšně provedena.
- Firewall ve Windows se výrazně liší dle verze operačního systému a tomu odpovídá i složitost a rozsah jeho ovládání. Novější verze vyžaduje opravdu dobré pochopení celé architektury síťové ochrany, proto doporučujeme nejdříve prostudovat vstupní dokumentaci výrobce a zorientovat se nejdříve v grafické reprezentaci firewallu.

Klienty síťových služeb a protokolů

Úvod

PowerShell byl ve své první verzi nesmírně macešský v podstatě ke všem síťovým službám a protokolům. Jeho tvůrci sice ve verzi 2 připravili několik zajímavých novinek, dosud však platí, že PowerShell rozhodně není synonymem pro „síťově" orientovaný shell, tedy alespoň potud, pokud hovoříme přímo o jeho příkazech. Na druhou stranu PowerShell je výborným prostředníkem pro práci se síťovými protokoly, neboť vynikající zázemí mu poskytuje .NET Framework, který je pro práci se sítěmi vybaven více než slušně. A tato možnost se stane oporou i při naší práci se sítěmi.

Dobrým a veskrze užitečným případem je zasílání e-mailových zpráv, které je často využíváno při běžné automatizaci správcovských úloh. Ve verzi 1 je potřeba sáhnout na některý z osvědčených způsobů pomocí známých či pro nás novějších objektových modelů – „starší" variantu pomocí objektu COM si dále ukážeme, zde naznačme opětovné použití .NET Frameworku. Ten zapouzdřuje klient služby SMTP prostřednictvím několika tříd, z nichž budeme v nejjednodušší podobě potřebovat jen dvě:

mailer.ps1

```
$smtpKlient = new-object system.net.mail.SmtpClient
$zprava = New-Object system.net.mail.MailMessage

$smtpKlient.Host = "smtp.patrikmalina.eu"

$zprava.From = "it@patrikmalina.eu"
$zprava.To.Add("malina@gopas.cz")
$zprava.Subject = "Zkouska posty"
$zprava.Body = "Toto je prosta emailova zprava"

$smtpKlient.Send($zprava)
```

Na počátku jsme vytvořili dva objekty – jeden reprezentuje klient SMTP (tedy vlastně „odesílač"), druhá pak zprávu samotnou. Za zmínku stojí, že příjemci zprávy jsou uloženi jako prvky kolekce, a proto jsme při jejím sestavování volali metodu *Add* namísto přímého přiřazení vlastnosti jako třeba u předmětu zprávy (*Subject*).

Poznámka: Nutno poznamenat, že výše uvedený fragment skriptu bude pracovat spíše jen výjimečně – přesněji řečeno pouze za podmínek, kdy je na zadaném serveru SMTP volně otevřeno přeposílání (open relay). To není zcela běžné, obzvláště u serverů SMTP v Internetu. Ověřování (předávání přihlašovacích údajů) si ukážeme dále.

PowerShell ve verzi 2 se dočkal novinky, na kterou řada správců netrpělivě čekala, a to příkazu, který výše uvedenou práci zastane sám:

```
Send-MailMessage `
-From "it@patrikmalina.eu" `
-To "malina@gopas.cz" `
-SmtpServer "smtp.patrikmalina.eu" `
```

```
-Subject "E-mail z PowerShellu" `
-Credential login `
-Body "Zprava z PowerShellu" `
-UseSSl
```

Pokud by nás původ tohoto příkazu zajímal více, stačí si prohlédnout jeho anatomii a zjistíme, že jsme opět zpátky v .NET Frameworku, z nějž jsme o příklad dříve vyšli. Stačí vyvolat chybu:

```
Send-MailMessage : Na základě provedeného ověření není vzdálený certifikát platný.
At line:1 char:17
+ Send-MailMessage <<<<  -From "it@patrikmalina.eu" -To "malina@gopas.cz" -SmtpSer-
ver "smtp.patrikmalina.eu" -Subject "E-mail z PowerShellu" -Credential it@patrikma-
lina.eu -Body "Zprava z PowerShellu" -UseSsl
    + CategoryInfo          : InvalidOperation: (System.Net.Mail.SmtpClient:SmtpC-
lient) [Send-MailMessage], AuthenticationException
```

Ihned je zřejmé, že nově zabudovaný klient je přímou implementací nám již povědomé třídy. Samozřejmě jsou nasnadě další možnosti, mezi něž určitě patří připojování příloh, k nimž se dostaneme v dále popsaných konkrétních postupech.

Podobným způsobem bychom mohli dále pokračovat, neboť .NET Framework nabízí výbavu i pro další služby a aplikační protokoly, jako jsou FTP či HTTP, a pomocí univerzálnějších tříd lze dobře vytvořit třeba klient služby POP3.

Tip: .NET Framework a jeho třídy jsou opravdu dobře použitelné pro řadu zajímavých řešení. Matador komunity PowerShellu Lee Holmes třeba ukazuje v jednom ze svých článků použití univerzálního klientu při komunikaci protokolem POP3 a HTTP:

http://www.leeholmes.com/blog/ScriptingNetworkTCPConnectionsInPowerShell.aspx

Stránky tohoto autora jsou vůbec plné zajímavých řešení, nejen pro „síťování".

Zasílání e-mailu pomocí rozhraní CDO

Operační systém Windows obsahuje již poměrně dlouhou dobu knihovnu CDO, jejíž základní verze umožňuje jednoduché odesílání e-mailových zpráv pomocí rozhraní COM. Tento postup se velmi osvědčil již při skriptování pomocí WSH a není důvod na něj zanevřít v PowerShellu, pokud jej dobře známe a jsme na něj zvyklí. Přístup k objektům knihovny je i v PowerShellu velmi snadný.

Tip: Knihovna CDO se vyskytuje v různých variantách dle instalovaného softwaru firmy Microsoft. Základní podoba, která je spojena s instalací součástí Internet Explorer a Outlook Express, umí základní odesílání e-mailů. Aplikace Outlook instaluje pokročilejší verzi a nejbohatší varianta je obsažena v instalaci nástrojů pro správu MS Exchange 2003. O této posledně zmíněné verzi ještě bude řeč na jiném místě této knihy.

cdomail.ps1

1. Zavoláme knihovnu CDO a podle její třídy *message* si založíme objektovou proměnnou, jež bude reprezentovat naši poštovní zprávu.

```
=>$mail = New-Object -ComObject cdo.message
```

2. Naši zprávu budeme odesílat prostřednictvím protokolu SMTP, jenž vyžaduje konfiguraci odesílacího serveru. Pokud pracujeme v lokální síti a server SMTP nás ověří automaticky na základě přihlášení k operačního systému, není potřeba konfiguraci měnit. Pracujeme-li ovšem se vzdáleným serverem SMTP a chceme se ujistit, že naši zprávu přijme k odeslání, bude zřejmě nutné přihlášení. Tato nastavení jsou obsažena ve speciální kolekci, jejíž parametry nyní obsadíme potřebnými hodnotami.

```
$mail_config = $mail.Configuration.Fields

$configpath = [string]"http://schemas.microsoft.com/cdo/configuration"

#zaslani zpravy pres vnejsi sitove rozhrani
$mail_config.Item("$configpath/sendusing") = 2

#jmeno serveru sluzby SMTP
$mail_config.Item("$configpath/smtpserver") = "smtp.mujserver.cz"

#port sluzby SMTP
$mail_config.Item("$configpath/smtpserverport") = 25

#pouziti SSL (0 = False, 1 = True)
$mail_config.Item("$configpath/smtpusessl") = 0

#zpusob overeni (0 = anonymni, 1 = Basic, 2 = NTLM)
$mail_config.Item("$configpath/smtpauthenticate") = 1

#prihlasovaci jmeno k SMTP
$mail_config.Item("$configpath/sendusername") = "login"

#prihlasovaci heslo k SMTP
$mail_config.Item("$configpath/sendpassword") = "password"

#zapsani novych hodnot
$mail_config.Update()
```

3. Obsadíme vlastnosti hodnotami, jež určí základní parametry odesílané zprávy.

```
$mail.From = "it@patrikmalina.eu"
$mail.To = "malina@gopas.cz"
$mail.Subject = "CDO mail"
$mail.TextBody = "Mail pomoci CDO"
```

4. Nyní můžeme zprávu odeslat pomocí metody příznačného jména.

```
$mail.Send()
```

Jednorázové odeslání e-mailové zprávy

Příkaz PowerShellu *Send-MailMessage* je dobrým prostředkem pro základní odesílání e-mailových zpráv, a to především tehdy, když je server SMTP ve stejné síti, a nepředpokládáme tedy komplikace s přihlašováním a blokování *relayingu*, tj. anonymního přeposílání pošty s neověřeným původem. Příkaz potřebuje jen několik výchozích parametrů, jež předáme pomocí přepínačů.

```
Send-MailMessage `
-From "it@patrikmalina.eu" `
-To "malina@gopas.cz" `
-SmtpServer "localhost" `
-Subject "E-mail z PowerShellu" `
-Body "Zprava z PowerShellu"
```

Adresa serveru SMTP musí být určena, aby e-mailový klient vůbec mohl odeslání provést. Jestliže se nacházíme ve stejné síti jako server SMTP, není typicky třeba provádět žádné výslovné ověření – server si případně může vyžádat ověření sám pomocí protokolu NTLM nebo Kerberos a využije naše přihlášení do Windows.

Odesílání e-mailové zprávy s přihlášením k serveru

Servery služby SMTP dnes běžně vyžadují přihlášení klienta, a pokud odesíláme zprávu z cizí sítě či chceme využít jiný přihlašovací účet, než je naše aktuální přihlášená relace, je potřeba předat přihlašovací údaje.

mail01.ps1

1. Vytvoříme objektové proměnné, které budou reprezentovat zprávu, „odesílač“ a také přihlašovací údaje.

```
$smtpKlient = new-object system.net.mail.smtpClient
$zprava = New-Object system.net.mail.mailmessage
$prihlaseni = New-Object system.net.NetworkCredential
```

2. Přiřadíme hodnoty přihlašovacích údajů do proměnné a předáme tato nastavení odesílači.

```
$prihlaseni.username = "login"
$prihlaseni.Password = "heslo"
$smtpKlient.Credentials = $prihlaseni
```

3. Zbytek zprávy sestavíme jako v případě bez ověřování a následně ji můžeme odeslat.

```
$smtpKlient.Host = "smtp.patrikmalina.eu"
$zprava.From = "it@patrikmalina.eu"
$zprava.To.Add("malina@gopas.cz")
$zprava.Subject = "Zkouska posty"
$zprava.Body = "Toto je prosta emailova zprava"

$smtpKlient.Send($zprava)
```

Hromadné odesílání e-mailové zprávy

Možnosti snadného odesílání e-mailových zpráv lze dobře využít i k hromadnému rozesílání stejné či jen mírně upravené zprávy většímu množství příjemců. Tato úloha ukazuje sestavení jednoduchého „rozesílače“, který bude mírně upravovat text zprávy a odešle e-mail všem příjemcům zadaným prostřednictvím vstupního souboru.

Následující skript potřebuje dva vstupní soubory. Jeden z nich je samotným seznamem příjemců, který je sestaven ve formátu CSV, takže jeho typický řádek (pod nezbytným záhlavím) vypadá takto:

```
mail,givenname,sn
prijemce1@mojedomena.cz,Patrik,Malina
```

Druhým vstupním souborem je samotné tělo zprávy, obsahující případné „náhradní“ textové řetězce, jež budou nahrazeny za běhu daty z prvního vstupního souboru (v našem případě jménem a příjmením). Zpráva může vypadat třeba takto:

```
Příjemce: _givenname _surname

Vážený kolego,
příští pondělí bude probíhat udělování oblíbených cen Kolega roku!
...
```

Pojďme se tedy podívat, jak bude náš hromadný „rozesílač“ pracovat.

mailing.ps1

1. Nejdříve načteme seznam všech příjemců ze souboru CSV a uložíme jej do proměnné, s níž pak budeme průběžně pracovat.

   ```
   $pwd = pwd

   $path = Join-Path $pwd -ChildPath "\users.csv"
   $users = import-csv $path -erroraction stop
   ```

   Pro případ, že by se vstupní seznam nepodařilo načíst, jsme určili akci *Stop* – při chybě načítání příjemců se skript zastaví.

2. V dalším kroku nastavíme výchozí proměnnou, která bude společná všem příjemcům (adresa odesílatele), a založíme proměnnou, jež bude ukládat záznam činnosti našeho skriptu.

   ```
   $emailfrom = "odesilatel@mojedomena.cz"
   $smtpServer = "smtp.mojedomena.cz"

   $log = @()
   $log += "User_mail,GN,SN,Time_sent"
   ```

3. V tuto chvíli již můžeme nastartovat cyklus, který průběžně odešle zprávu všem příjemcům.

   ```
   ForEach ($user in $users)

   {
       $tolog = ""
   ```

Proměnná s výstupním záznamem je vynulována, abychom ji mohli naplnit skutečnými údaji o naposled osloveném příjemci.

4. Načteme text zprávy a klíčová slova nahradíme požadovanou „personifikovanou" částí. V našem případě bude doplněno jméno a příjmení.

```
$mailtext = cat zprava.txt

$mailtext = $mailtext -replace "_givenname",$user.givenname
$mailtext = $mailtext -replace "_surname",$user.sn

$emailbody = $mailtext

$subject = "Udrzba site -- zprava o planovanem vypadku"
 $emailto = $user.mail
```

Stejně jako jméno a příjmení; jsme doplnili do potřebných proměnných též další údaje o příjemci; pomocí nich následně proběhne odesílání zprávy.

5. Zpráva je připravena, takže můžeme vytvořit samotný poštovní klient, předat mu údaje a zprávu odeslat.

```
$mailer = new-object Net.Mail.SMTPclient($SMTPserver)

$msg = new-object `
Net.Mail.MailMessage($emailfrom,$emailto,$subject,$emailbody)
$msg.IsBodyHTML = $False

$mailer.send($msg)
$msg.Dispose()
```

6. Po odeslání zprávy je třeba ještě vyrobit záznam o daném příjemci do našeho souboru záznamů. Kromě identifikace přidáme též časový údaj pro případnou zpětnou kontrolu.

```
    $tolog = $emailto + "," + $user.givenname + "," + $user.sn + `
    (get-date -format g).ToString()
    echo $tolog
    $log += $tolog
}
```

7. Jakmile tímto způsobem obešleme všechny příjemce, je čas uložit nashromážděný záznam o celé akci do výstupního souboru.

```
$mailinglogdate = get-date -uformat %m_%d_%Y_%H_%M_%S
$logpath = Join-Path $pwd -ChildPath "\Mailing_$mailinglogdate.csv"
$log | set-content -Path $logpath -Force
```

Jak je patrné, výstupní soubor se záznamem činnosti bude obsahovat přímo ve svém jménu časový údaj, aby bylo na první pohled jasné, kdy akce proběhla.

Takto sestavený hromadný odesílací program je samozřejmě základní kostrou pro jakékoliv složitější konstrukce. Přinejmenším jej lze vylepšit přikládáním příloh, jak si ukážeme v jiné úloze, nebo třeba použitím těla zprávy ve formátu HTML.

Odesílání e-mailové zprávy s přílohami

Zprávy elektronické pošty bez příloh jsou těžko myslitelné, obzvláště při správě sítí – typickým produktem automatizovaných úloh jsou záznamy činnosti skriptu (logy) či třeba exporty nastavení různých součástí operačního systému, které si následně zasíláme poštou. PowerShell si může i zde vypomoci několika způsoby, my si pak ukážeme připojování příloh pomocí klientu v .NET Frameworku.

mail_with_attach.ps1

1. Založíme e-mailovou zprávu obvyklým způsobem a nastavíme výchozí parametry. Využíváme tedy třídy, jež reprezentují poštovní zprávu a samotný odesílač.

```
$smtpKlient = new-object system.net.mail.SmtpClient
$zprava = New-Object system.net.mail.MailMessage

$smtpKlient.Host = "smtp.patrikmalina.eu"

$zprava.From = "it@patrikmalina.eu"
$zprava.To.Add("malina@gopas.cz")
$zprava.Subject = "Zkouska posty"
$zprava.Body = "Toto je prosta emailova zprava"
```

2. Přílohu e-mailu reprezentuje objekt další třídy, který nyní zapojíme do naší úlohy. Přiřadíme mu cestu k uložené příloze jako vlastnost objektové proměnné, čímž si data připravíme k připojení.

```
$cesta = "c:\logfiles\log025.txt"
$priloha = New-Object system.net.mail.attachment($cesta)
```

3. Příloha je uložena v proměnné a nyní ji můžeme přidat k již založené zprávě.

```
$zprava.Attachments.Add($priloha)
```

4. Nyní můžeme zprávu konečně odeslat i s přílohou. Po odeslání je ještě potřeba zařídit, aby odesílač dále nedržel soubor přílohy ve svém výhradním užívání – jinak bychom mohli při snaze s ním opět pracovat narazit na chybu přístupu.

```
$smtpKlient.Send($zprava)
$priloha.Dispose()
```

Přílohy jsou připojovány jako prvky kolekce a můžeme jich připojit více opakováním výše uvedeného postupu.

Odeslání zprávy z Outlooku – protokolem MAPI

Řada počítačových sítí je nastavena tak, že volné odesílání e-mailových zpráv protokolem SMTP není vůbec povoleno, přesněji řečeno, je vynucováno spojení pomocí protokolu MAPI (tedy typicky klientem Microsoft Outlook). V mnoha situacích tak můžeme dojít k závěru, že snaha o ladění komunikace se serverem SMTP je zbytečná (zvláště snaha o přihlášení), neboť stejnou práci pro nás snadno udělá Outlook. Ten je zpřístupněn pomocí rozhraní COM a ve spojení s „všemocným “příkazem *Get-Member* je i jeho rozsáhlý objektový model dobře čitelný.

Outlook01.ps1

1. Nejprve prověříme, zdali aplikace Outlook běží, a pokud ano, využijeme toho a přímo se připojíme na její běžící instanci. Ušetří nám to mimo jiné přihlašování.

```
if (Get-Process outlook -ea silentlycontinue)

{
$outlook = [System.Runtime.InteropServices.Marshal]::
GetActiveObject("Outlook.Application")

$nevypinat = $true
}
```

   V tomto kroku jsme použili zajímavou třídu *Marshal*, která nám umožňuje komunikovat s objekty typu COM a kupříkladu převzít existující běžící aplikaci do naší objektové proměnné.

2. Pokud Outlook neběží, musíme si jej spustit a přihlásit se.

```
else
{
$Outlook = New-Object -com "Outlook.Application"
$Outlook.Session.Logon("Outlook","heslo")

$nevypinat = $false
}
```

3. Založíme novou objektovou proměnnou, jež bude reprezentovat e-mailovou zprávu, a přiřadíme potřebné hodnoty. Poté zprávu odešleme.

```
$zprava = $Outlook.CreateItem("olMailItem")
$zprava.To = "malina@gopas.cz"
$zprava.Subject = "PowerShell a MAPI automaticky"
$zprava.Body = "Zprava z PowerShellu a Outlooku"
$zprava.Send()
```

4. Na závěr prověříme, jestli Outlook před naší akcí běžel, a pokud ne, opět jej zavřeme. Následuje „uklizení“ nepotřebných proměnných.

```
if (-not $nevypinat)
{
$outlook.Quit()
}
Remove-Variable zprava
Remove-Variable outlook
```

Načtení obsahu poštovní schránky Outlooku

Objektový model aplikace Outlook dovoluje samozřejmě mnohem více než prosté odeslání zprávy, jež jsme si ukázali výše. V tomto příkladu si ukážeme připojení k poštovní schránce, načtení zpráv (některých jejich vlastností) a výstup těchto dat do tabulkového souboru.

readmailbox01.ps1

Obrázek 7.6: Schránka na serveru MS Exchange je přístupná prostřednictvím aplikace Outlook jak pro export dat v dávkách, tak pro přímé skriptování

1. Připojíme se k aplikaci již známým a popsaným způsobem. Je-li Outlook spuštěn, využijeme toho, není-li, spustíme jej a přihlásíme se pro případ, že by to bylo potřeba.

```
if (Get-Process outlook -ea silentlycontinue)
{
$outlook = [System.Runtime.InteropServices.Marshal]::GetActiveObject("Outlook.
Application")
$nevypinat = $true
}

else
{
$Outlook = New-Object -com "Outlook.Application"
$Outlook.Session.Logon("Outlook","heslo")
$nevypinat = $false
}
```

2. Připojíme se k relaci a uložíme do objektové proměnné výchozí poštovní složku (typicky *Inbox*).

```
$space = $Outlook.GetNamespace("MAPI")
$fld = $space.GetDefaultFolder(6)
```

3. V dalším kroku si založíme proměnnou typu pole, do níž budeme posléze zprávy ukládat. Tuto proměnnou využijeme posléze pro uspořádání seznamu dle vybrané vlastnosti. Zároveň s ukládáním do proměnné budeme zprávy rovnou posílat na výstup do souboru CSV v tom pořadí, v jakém nám je Outlook bude vracet.

```
$zpravy = @()

$fld.Items | select senderemailaddress,senderemailtype, receivedtime, subject `
Tee-Object -Variable zpravy | Export-Csv inbox.csv -NoTypeInformation
```

4. Data v proměnné nyní využijeme k setřídění a následnému opětovnému exportu do CSV.

```
$zpravy | Sort-Object receivedtime -Descending`
| Export-Csv inbox_sorted.csv -NoTypeInformation
```

5. Na závěr provedeme opět kontrolu původního stavu aplikace, případně Outlook zavřeme a uklidíme objektové proměnné.

```
if (-not $nevypinat)
{
$outlook.Quit()
}

Remove-Variable fld
Remove-Variable space
Remove-Variable outlook
```

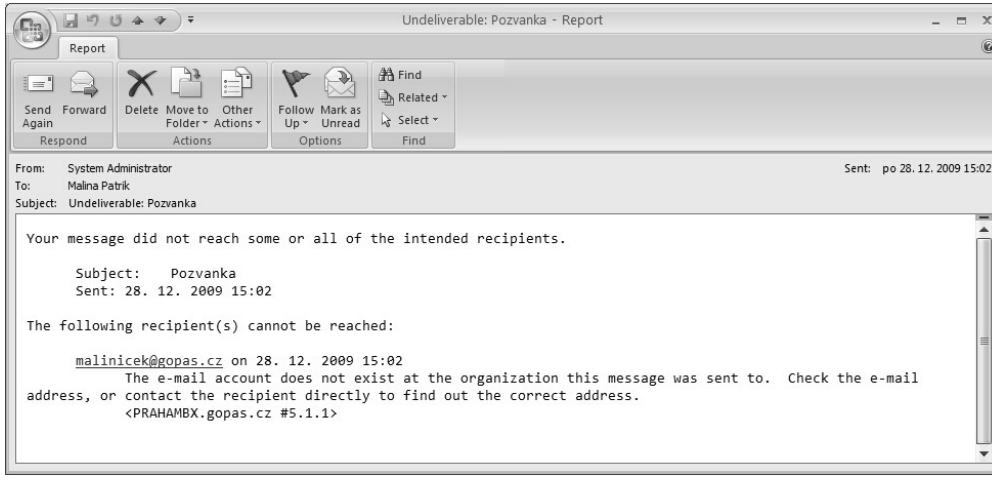
Výsledkem našeho skriptu jsou tedy dva soubory CSV, z nichž jeden obsahuje zprávy seřazené sestupně dle data a času přijetí zprávy do schránky.

Tabulka 7.2: Nejběžnější informace o poštovní zprávě a odpovídající objektové vlastnosti
Více viz: *http://msdn.microsoft.com/en-us/library/bb176688.aspx*

Property (třída MailItem)	Data
Body	Text zprávy
BodyFormat	Typ obsahu textové zprávy (Text, HTML, RTF)
EntryID	Jedinečný identifikátor položky
HTMLBody	Text zprávy, je-li v HTML
ReceivedTime	Čas přijetí zprávy
SenderEmailType	Typ odesílatele (SMTP, Exchange)
SenderEmailAddress	Jméno či adresa odesílatele
Subject	Předmět zprávy

Vyšetření obsahu zprávy v Outlooku: „nedoručenka“

Informace o nedoručení odeslané e-mailové zprávy je poměrně běžným obsahem poštovních schránek, a to nejen administrátorských. Správci poměrně často potřebují vyšetřit zprávy o nedoručení (non-delivery report, dále jen NDR), jednak proto, aby vypátrali případné havarující adresy příjemců, jednak proto, aby zjistili cílové poštovní servery, jež nemusejí pracovat úplně správně. Analýza se provádí buďto na straně serveru (třeba vyhodnocením logů), nebo přímo v poštovní schránce příjemce NDR (tedy odesílatele původních zpráv). Druhá možnost je typická hlavně při hromadném obesílání větších skupin příjemců, kdy všechny NDR putují spořádaně do jedné poštovní schránky, kde je můžeme vyhodnotit. Při takovém postupu řešíme klíčový problém – odesílatel NDR je poštovní server a samotná adresa příjemce je uvedena v těle (textu) zprávy, a to ještě v některém z možných formátů. Odtud ji musíme získat a případně převést na formát SMTP.



Obrázek 7.7: Zpráva o nedoručení má poněkud složitou strukturu, takže automatická extrakce původní adresy příjemce nám dá určitou práci

Následující skript otevře aplikaci Outlook, vstoupí do určené složky s přijatou poštou a vyhodnotí doručené zprávy NDR. Naším cílem je vyhodnotit jejich obsah a vyjmout z nich adresy původních příjemců. Budeme zohledňovat jednak obsah zprávy NDR samotné, jednak tvar adresy příjemce, který se může lišit (ne vždy jde o typickou podobu: xy@domena.abc).

Náš skript bude vracet pro každou zprávu adresu v požadovaném formátu SMTP (tedy e-mailovou adresu tak, jak jsme zvyklí). Outlook drží adresy příjemců a odesílatelů někdy v odlišných formách, a proto si případná jiná jména přeložíme na adresu SMTP. Za tímto účelem bude náš skript obsahovat funkce, jež tuto činnost zajistí.

analyzeNDR01.ps1

1. Outlook často drží adresu odesílatele a příjemce ve formátu Legacy Exchange DN (starší formát, používaný až po Outlook 2003). Tato hodnota je uložena v Active

Directory jako atribut, a proto není problém podle ní uživatele dohledat a získat jeho skutečnou adresu SMTP. Založíme tedy funkci a určíme vstupní proměnnou.

```
Function GetSMTPMailFromLEDN
([string]$ledn)
```

2. Jediný příkaz nám bude stačit na vyhledání atributu a zjištění odpovídající adresy SMTP.

```
{
$SMTP = ((get-adobject -Server GlobalCatalog.mojedomena.xyz `
-Filter legacyexchangedn=$ledn).mail)
```

Tip: Předchozí úsek naší funkce obsahuje cmdlet, který není běžnou výbavou PowerShellu. Tento cmdlet *Get-ADObject* je součást balíku PowerShell Community Extensions, o němž se dočtete v kapitole 2.3. Jeho práci mohou zastat i jiné příkazy, o nichž je řeč v kapitole 6 o Active Directory.

3. Zjištěnou hodnotu atributu *mail* převedeme na prostý řetězec, který se stane návratovou hodnotou z funkce. Tímto jsme vyřešili převod jména Legacy Exchange DN.

```
$smtpaddress = $SMTP.ToString()
return $smtpaddress
}
```

4. Další problém, který před námi stojí a zaslouží si svou funkci, je převod jména *Display Name* opět na adresu SMTP. Postup bude obdobný jako výše – další funkce nám zajistí vyhledání atributu v Active Directory a nalezení jemu příslušného atributu s adresu SMTP.

```
Function GetSMTPMailFromDisplayName (
[string]$disp)
{

$SMTP = ((get-adobject -Server GlobalCatalog.mojedomena.xyz `
-Filter "(&(displayname=$disp)(mail=*))").mail)

$smtpaddress = $SMTP.ToString()
return $smtpaddress
}
```

Při hledání jsme použili stejný cmdlet, avšak filtr jsme rozšířili – pod stejným zobrazovacím jménem se může vyskytovat více objektů Active Directory, avšak ne všechny musí mít e-mailovou adresu. Proto hledáme jen takový, který má atribut *mail* obsazen. Výše použitý atribut *Legacy Exchange DN* je proti tomu jedinečný (pokud není nastaven chybně u více účtů).

Tímto jsme vyřešili překlady nejčastějších jmen na adresy SMTP a můžeme spustit hlavní logiku skriptu.

5. Otevřeme Outlook a přejdeme do složky, v níž máme zprávy NDR odloženy.

```
$ou = New-Object -com "outlook.application"
$space = $ou.GetNamespace("MAPI")
$fld = $space.GetDefaultFolder(6)

$replies = $fld.folders.Item("NDR")
```

6. Nastavíme výchozí hodnoty pro proměnné, jež nám poslouží jako počitadla pro jednotlivé rozpoznané případy (očekáváme několik typů NDR a také možné zprávy *Out Of Office*, jež mohou mezi NDR omylem spadnout). Poté vytiskneme záhlaví celé statistiky.

```
$all = 0
$Failure = 0
$Undeliverable = 0
$OutOfOffice = 0
$Other = 0

echo "Reason;OriginalRecipient;All;Failure;Undeliverable;OutOfOffice;Other"
```

7. Spustíme cyklus, jenž bude postupně procházet všechny zprávy, a začneme počítat jejich celkový úhrn.

```
$replies.items | % {

$all ++
```

8. Provedeme první vyhodnocení předmětu zprávy. Pokud obsahuje určitá klíčová slova, budeme jej považovat za NDR navrácené poštovní branou.

```
if (($_.subject -like "*status*") -AND ($_.subject -like "*Failure*"))
{
$reason = "Mail address gateway failure"
$Failure ++
```

Obsadili jsme si proměnnou, jež bude vystupovat jako důvod ve výstupní sestavě, a také jsme zvýšili příslušné počitadlo o jednotku.

9. Načteme důležité atributy zprávy do proměnných pro výstupní sestavu.

```
$body = $_.htmlbody
if ($body -like "") {$body = $_.body}
$body = $body.ToString()

$subject = $_.subject
```

10. Tento krok obsahuje klíčovou operaci, a to vyhledání adresy odesílatele v těle (textu) NDR. Použijeme na to jednak práci s řetězcem pomocí metody *Substring*, jednak regulární výrazy. Podrobnější rozbor následuje v jednom z příštích kroků, kde se bude postup opakovat.

```
$recip = $htmlbody.substring(([System.Text.RegularExpressions.Regex] `
"To.*@.*<").Match($htmlbody).Index+4,(([System.Text.RegularExpressions.Regex] `
"To.*@.*<").Match($htmlbody).Length-5))

}
```

V tuto chvíli jen uveďme, že hledáme takový řetězec, jenž začíná slovem *To*, následuje „něco“ se zavináčem uprostřed a řetězec končí znaménkem < (jde o typickou sekvenci navrácenou poštovním serverem či branou). Takto získaná e-mailová adresa je uložena do proměnné, jež představuje původního příjemce zprávy.

11. Náš skript vyhodnotil první možný typ NDR – návratovou zprávu internetové poštovní brány či vzdáleného serveru. Nyní odlišíme zprávy typu *Out Of Office*.

```
else
    {

    if (($_.subject -like "*out*") -AND ($_.subject -like "*office*"))
    {
    $reason = "Out of office"
    $OutOfOffice ++
    $body = $_.htmlbody
     if ($body -like "") {$body = $_.body}
     $body = $body.ToString()

    $sender = $_.SenderEmailAddress
    $subject = $_.subject
```

Postupujeme stejně jako výše: rozlišujeme klíčová slova v poli *Předmět* (subject) a poté extrahujeme vlastnost zprávy (*senderemailaddress*), jež obsahuje odesílatele – v tomto případě tedy skutečného odesílatele, neboť zpráva o nepřítomnosti je odeslána v zastoupení hledané poštovní schránky. Nastal čas rozlišit typ adresy a případně ji převést na formát SMTP.

12. Otestujeme typ adresy odesílatele zprávy, a pokud bude potřeba, zavoláme funkci na překlad formátu *Legacy Exchange DN* na formát SMTP.

```
if ($_.senderemailtype -like "*EX*")
        {
        $recip = (GetSMTPMailFromLEDN($sender)).ToString()
        }
        else
        {
        $recip = $sender
        }
    }
```

Tímto jsme vyhodnotili zprávu v případě, že jde o informaci o nepřítomnosti.

13. V tomto kroku se podíváme na zprávy, jež jsou odeslány místním systémem Exchange jako nedoručitelné. Jejich formát je opět mírně odlišný, a extrakce adresy původního příjemce tedy vyžaduje drobné modifikace.

```
else
        {
        if ($_.subject -like "*undeliverable*")
        {
        $reason = "Local mailer indeliverable"
        $Undeliverable ++
```

```
$body = $_.htmlbody
     if ($body -like "") {$body = $_.body}
     $body = $body.ToString()

$subject = $_.subject
```

Opět provedeme operace jako v předešlých dvou situacích – je-li rozlišen předmět zprávy jako hledaný, zvýšíme hodnotu počítadla a načteme klíčové součásti zprávy.

14. Tento krok opět obsahuje vyhledávání řetězce s adresou původního příjemce. Vysvětlíme si podrobněji, jak operace probíhá.

```
$recip = $body.substring(([System.Text.RegularExpressions.Regex] `
".*on \d{1,2}").Match($body).Index+6,(([System.Text.RegularExpressions.Regex] `
".*on \d{1,2}").Match($body).Length-11))
$recip = $recip.Trim()
```

Regulární výraz je klíčem k řešení problému. Metoda *match* hledá shodu se vzorem v uvozovkách a tento pak zastupuje hledané jméno. Záchytným bodem je skutečnost, že NDR typicky obsahuje jméno původního příjemce následované spojkou *on* a datumových údajem (tedy číslicemi). Pro názornost uveďme příklad takové zprávy:

```
Your message did not reach some or all of the intended recipients.

      Subject:    Pozvanka
      Sent:    28. 12. 2009 15:02

The following recipient(s) cannot be reached:

      malinicek@gopas.cz on 28. 12. 2009 15:02
            The e-mail account does not exist at the organization this mes
sage was sent to.  Check the e-mail address, or contact the recipient dire
ctly to find out the correct address.
            <PRAHAMBX.gopas.cz #5.1.1>
```

Poznámka: Uvedený regulární výraz je jen jedním příkladem z mnoha možných. Jiné varianty pro pátrání po e-mailové adrese je možno najít v jiných místech této knihy.

Regulární výraz tedy zachytí hledaný řetězec a my poté pomocí metody *Substring* ještě provedeme odstranění předcházejících a následných znaků, jež jméno (adresu) obklopují (tedy zahodíme mezery před adresou a třeba ono slůvko *on* následované číslicemi za adresou). Povšimněte si, že v regulárním výrazu je kvantifikátor označující dvě varianty s jednou či dvěma číslicemi (`|d{1,2}`), protože dopředu nevíme, jestli datum bude začínat jedno či dvouciferným číslem. Právě proto ještě nakonec použijeme metodu *Trim*, jež odstraní případnou nadbytečnou mezeru.

Náš příklad uvedený výše tedy bude zpracován tak, že původní řádek:

```
malinicek@gopas.cz on 28. 12. 2009 15:02
```

bude vyhodnocen jako:

```
malinicek@gopas.cz
```

15. Získali jsme jméno či adresu původního příjemce, ale nevíme, zda má formát SMTP. Proto provedeme test a v případě potřeby zavoláme funkci, jež adresu získá.

```
if ($recip -notlike "*@*")
            {
            $recip = (GetSMTPMailFromDisplayName($recip)).ToString()
            }

        }
```

16. Zbývá nám ošetřit poslední situaci: pokud předmět zprávy nepoukazuje na typický formát NDR, je potřeba to zdokumentovat a pokusit se získat adresu odesílatele pro pozdější analýzu. Provedeme to postupy, které jsme si již ukázali.

```
else
            {

            $reason = $_.subject.ToString()
            $Other ++

            $body = $_.htmlbody
             if ($body -like "") {$body = $_.body}
             $body = $body.ToString()

             $sender = $_.SenderEmailAddress
            $subject = $_.subject

            $recip = $sender

            if ($_.senderemailtype -like "*EX*")
            {
            $recip = (GetSMTPMailFromLEDN($sender)).ToString()
            }

            if ($recip -notlike "*@*")
            {
            $recip = (GetSMTPMailFromDisplayName($recip)).ToString()
            }


            }
```

17. Vyhodnotili jsme dostupná data, takže je můžeme poslat na výstup. A když projdeme celou složku Outlooku, můžeme ji opustit a skript ukončit.

```
     }
}
echo "$reason;$recip;$all;$Failure;$Undeliverable;$OutOfOffice;$Other"
}
remove-variable replies
remove-variable fld
```

```
remove-variable space
$ou.quit()
remove-variable ou
```

Pokud naše e-mailová složka NDR obsahuje zprávu, kterou jsme výše uvedli jako příklad, vrátí náš skript následující výstup:

```
Reason;OriginalRecipient;All;Failure;Undeliverable;OutOfOffice;Other
Local mailer indeliverable;malinicek@gopas.cz;1;0;1;0;0
```

Skript můžeme spustit s přesměrováním dat namísto do textové konzoly do souboru formátu CSV.

```
=>.\analyzeNDR01.ps1 > ndr1.csv
```

Výstup můžeme také zajistit přímo ze skriptu – kroky 6 a 17 můžeme pozměnit či obohatit o následující řádek, jenž zajistí přímý výstup do CSV.

```
"Reason;OriginalRecipient;All;Failure;Undeliverable;OutOfOffice;Other" | `
Add-Content ndr.csv -Encoding Ascii

"$reason;$recip;$all;$Failure;$Undeliverable;$OutOfOffice;$Other" | `
Add-Content ndr.csv -Encoding Ascii
```

Stažení obsahu webové stránky

Běhové prostředí .NET Framework nabízí zajímavé možnosti pro práci s daty na webových serverech. PowerShell dokáže tyto možnosti dobře využít, pokud potřebujeme získat data z webových stránek, stáhnout soubor v binární podobě či zpracovat jiné internetové zdroje. Několik možností ukáží následující kroky.

1. Vytvoříme si objektovou proměnnou, jež bude obsahovat klíčový objekt – tento bude vlastně jednoduchým webovým klientem.

   ```
   $web = new-object System.Net.WebClient
   ```

2. Naše síť může pracovat s tzv. servery proxy, jež zprostředkují naše požadavky na webové stránky a obdobná data. Pokud je to náš případ, musíme odpovídající nastavení připravit, než si vyžádáme data. Poslouží nám odpovídající třída a následné přiřazení objektu do příslušné vlastnosti.

   ```
   $webproxy = new-object System.Net.Webproxy("proxy1",8080)
   $web.Proxy = $webproxy
   $web.Proxy

   Address               : http://proxy1:8080/
   ...
   ```

3. V tomto kroku stáhneme existující binární soubor pomocí jeho adresy URL a umístíme jej na souborový systém.

   ```
   $filename = "WUG_02-21-08_Procesy_PM.pdf"
   $downloadfolder = "c:\download"

   $web.DownloadFile("http://www.patrikmalina.eu/download/$filename",`
   "$downloadfolder\$filename")
   ```

4. Stejná objektová proměnná nám nabízí též prostředky ke stahování zdrojového kódu samotné webové stránky. Provedeme to jednoduše následujícím způsobem.

```
$webpage = $web.DownloadString("http://www.patrikmalina.eu")
$webpage | Set-Content -Path "$downloadfolder\stranka.html"
```

5. Možnosti tohoto webového klientu můžeme spojit i s jinými datovými formáty. PowerShell umí přímo pracovat s datovým formátem XML, čehož lze snadno využít při zpracování publikovaných příspěvků pomocí služby RSS. Jednoduše si tak můžeme připravit export aktuálních příspěvků.

```
=>$RSS = [xml]$web.DownloadString("http://scienceworld.cz/rss.xml")

=>$RSS_list = @()
=>$rss.rss.channel.item | select title,link | Tee-Object -Variable RSS_list | `
Export-Csv rss_clanky.csv

=>$RSS_list | Get-Member

   TypeName: Selected.System.Xml.XmlElement

Name        MemberType   Definition
----        ----------   ----------
Equals      Method       bool Equals(System.Object obj)
GetHashCode Method       int GetHashCode()
GetType     Method       type GetType()
ToString    Method       string ToString()
link        NoteProperty System.String link=http://scie...
title       NoteProperty System.String title=NovÃ©  moÍ...
```

Výše uvedené postupy naznačují další možnosti využití webového klientu v PowerShellu – získaná data (webové stránky atd.) můžeme dále prohledávat a kupříkladu pátrat po specifických údajích (adresy e-mailu, hypertextové odkazy atd.).

Tip: Guru PowerShellu a jeho spolutvůrce Lee Holmes uvádí na svém blogu krásný příklad využití „čtečky" stránek HTML za účelem hledání cest URL. Jde o krásnou ukázku koncepční práce a mimo jiné též využití regulárních výrazů.

http://www.leeholmes.com/blog/UnitTestingInPowerShellALinkParser.aspx

Hromadný přenos souborů pomocí služby BITS

Tvůrci PowerShellu použili koncepce modulu k tomu, aby do verze 2 přidali velmi užitečnou funkcionalitu – přenos souborů po síti prostřednictvím služby BITS. Tento modul – ačkoliv je vlastně hotovým „kusem PowerShellu" – není zaveden automaticky, a správce to tedy před jeho využitím musí udělat sám.

Poznámka: Služba BITS (Background Intelligent Transfer Service) je prostředkem, pomocí nějž přenášejí některé služby či součásti Windows souborová data po síti mezi servery a klienty. Typickým příkladem je třeba služba Windows Update, která může přenášet aktualizační balíčky ze serveru ve své síti s centrální službou WSUS (Windows Server Update Services). Technologie BITS

tedy dovoluje přenést soubory v rámci navázané relace bez starosti o jednotlivé části kolekce či případné nutné pokusy o znovudoručení.

Podrobnější informace o této technologii najdeme třeba zde:

http://msdn.microsoft.com/en-us/library/aa362708%28VS.85%29.aspx

Modul a jeho příkazy můžeme využít k jednoduchému přenosu souboru třeba následujícím způsobem.

bits01.ps1

1. Ze všeho nejdříve musíme načíst potřebný modul obsahující příkazy pro další práci. Pro zajímavost pustíme import s podrobným výpisem, abychom viděli „zabalené" příkazy.

```
=>Import-Module bitstransfer -Verbose
VERBOSE: Importing cmdlet 'Add-BitsFile'.
VERBOSE: Importing cmdlet 'Complete-BitsTransfer'.
VERBOSE: Importing cmdlet 'Get-BitsTransfer'.
VERBOSE: Importing cmdlet 'Remove-BitsTransfer'.
VERBOSE: Importing cmdlet 'Resume-BitsTransfer'.
VERBOSE: Importing cmdlet 'Set-BitsTransfer'.
VERBOSE: Importing cmdlet 'Start-BitsTransfer'.
VERBOSE: Importing cmdlet 'Suspend-BitsTransfer'.
=>
```

2. V dalším kroku si uložíme cesty k souborům, které chceme stáhnout (přenést). Za povšimnutí stojí, že služba BITS akceptuje jak cesty internetové typu HTTP, tak lokální cesty UNC.

```
=>cat .\BITS_paths.txt

http://download.services.openoffice.org/files/stable/3.1.1/OOo_3.1.1_Wi-
n32Intel_install_en-US.exe
\\10.0.0.139\Users\Public\downloads\amd64fre_GRMRSATX_MSU.msu
```

3. Nyní můžeme spustit nezávislé stahování pro každou cestu (umístění) souboru – služba BITS pracuje v asynchronním režimu, takže přenos bude pracovat jako úlohy na pozadí (joby). Náš skript také rozliší, kdy přistupujeme na počítač v lokální síti, a zaopatří správné ověření uživatele. Začněme nastavením počitadla, jež poslouží k pojmenování úloh jednotlivých stahování, a také založením proměnné, do níž si úlohy (joby) budeme ukládat.

```
$paths = cat .\BITS_paths.txt

$poc = 0
$download_jobs = @()

foreach ($path in $paths)

{
```

4. Spustili jsme cyklus, který postupně zpracuje všechny zdrojové soubory (přesněji jejich cesty) a započne jejich stahování. Původní cestu také použijeme pro získání jména cílového souboru, který bude ležet na souborovém systému.

```
$name = "Download_"
$destination = "c:\download\" + ($path.split("/\"))[-1]
```

   Povšimněte si poslední části nově vznikající cesty k souboru. Metoda *Split* zajistí rozpad původní cesty na části (prvky pole), a to podle umístění lomítka (normálního či zpětného). Následně použijeme index pole, který ukáže na poslední prvek – jinými slovy na text za posledním lomítkem v původní cestě. Takto jsme vypreparovali jen jméno souboru.

5. Nastává moment, kdy se rozhodneme podle typu cesty k souboru. Je-li cesta ke zdroji typu UNC (má dvě lomítka na počátku), budeme pravděpodobně potřebovat přihlašovací údaje k síťovému zdroji. Proto si je interaktivně vyžádáme.

```
if ($path -like "\\*")
     {

     $login = Get-Credential
     $name += $poc
```

6. Nyní můžeme spustit stahování a předat cmdletu připravené parametry – zdroj a cíl, přihlašovací údaje a také instrukci, že má úloha běžet na pozadí. Založený job si uložíme do připravené proměnné.

```
$download_jobs += start-bitstransfer -Asynchronous -DisplayName $name `
-Source $path -Destination $destination -Credential $login

     }
```

7. Druhý případ zdrojové cesty čeká na zpracování – tentokráte začneme stahování bez předávání uživatelského přihlášení, neboť předpokládáme anonymní internetový zdroj.

```
else
     {

     $name += $poc
     $download_jobs += start-bitstransfer -Asynchronous -DisplayName $name `
-Source $path -Destination $destination

     }
```

8. Nastartovali jsme stahování jako nezávislý job, a proto můžeme zvýšit počitadlo o jedničku a zpracovat další zdroj.

```
$poc++
}
```

9. Náš cyklus byl vyčerpán, neboť všechny úlohy jsou spuštěny, takže se můžeme podívat na obsah naší proměnné, jež celou dobu joby „sbírala“.

```
$download_jobs
```

```
JobId                   : a0528764-a804-4a4f-b5d1-094e0995249d
DisplayName             : Download_0
Description             : This is a file transfer that uses the Background
Intelligent Transfer Service (BITS).
TransferType            : Download
JobState                : Transferring
OwnerAccount            : STROJ1\Patrik
Priority                : Foreground
RetryInterval           : 600
RetryTimeout            : 1209600
TransientErrorCount     : 0
ProxyUsage              : SystemDefault
ErrorContext            : None
ErrorCondition          : NoError
InternalErrorCode       : 0
ErrorDescription        :
ErrorContextDescription :
BytesTotal              : 142938840
BytesTransferred        : 31210228
FilesTotal              : 1
FilesTransferred        : 0
CreationTime            : 5. 1. 2010 22:40:45
ModificationTime        : 5. 1. 2010 22:41:39
TransferCompletionTime  : 1. 1. 0001 0:00:00
FileList                : {http://download.services.openoffice.org/files/
stable/3.1.1/OOo_3.1.1_Win32Intel_install_en-US.exe}
ProxyList               :
ProxyBypassList         :

JobId                   : 44d5553a-185e-4475-b28d-7f0593dc4a5e
DisplayName             : Download_1
Description             : This is a file transfer that uses the Background
Intelligent Transfer Service (BITS).
...
```

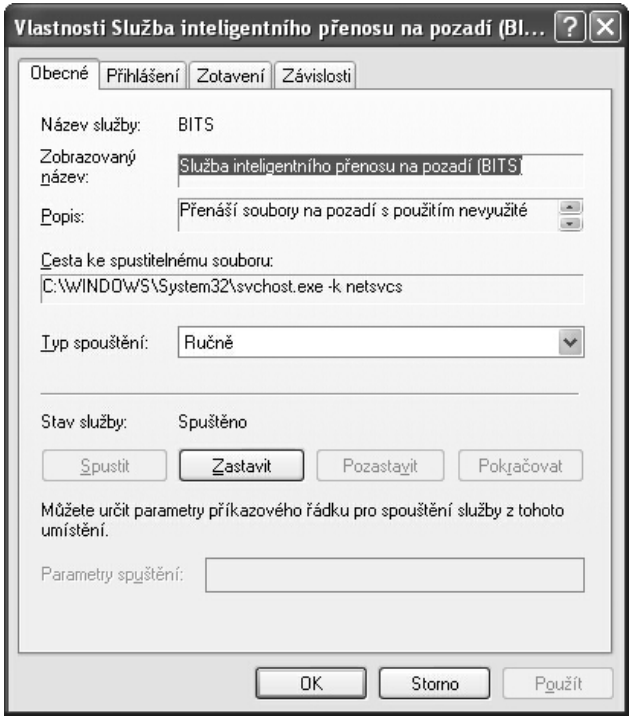
Objekt typu úloha služby BITS je poměrně bohatý, jak je z výpisu patrné, a poskytuje řadu cenných informací o průběhu přenosu souborů.

Průběžné ovládání služby a úloh BITS

Služba BITS může spravovat řadu úloh v rozličném stavu zpracování a PowerShell nabízí všestrannou sadu příkazů, pomocí nichž lze dobře sledovat a ovládat veškeré dění při přenosu souborů. Ukážeme si několik typických kroků při interaktivní práci.

1. Nejlépe se zorientujeme, pokud získáme základní přehled o běžících úlohách.

```
=>Get-BitsTransfer

JobId               DisplayName   TransferType  JobState        OwnerAccount
-----               -----------   ------------  --------        ------------
fb38244f-a7d0-49... Download_0    Download      Transferring    STROJ1\Patrik
75a75de1-e314-41... Download_1    Download      Transferring    STROJ1\Patrik
```



Obrázek 7.8: Služba BITS se ve Windows objevila již dávno, avšak její bezprostřední využití bylo dlouho velmi obtížné. PowerShell nabízí výborné prostředky pro její snadné využití.

2. Jednotlivé úlohy lze snadno uchopit dle jejich zobrazovacího jména a v případě potřeby pozastavit.

```
=>Get-BitsTransfer download_0

=>Get-BitsTransfer download_0 | Suspend-BitsTransfer

JobId               DisplayName TransferType   JobState        OwnerAccount
-----               ----------- ------------   --------        ------------
fb38244f-a7d0-49... Download_0  Download       Suspended       STROJ1\Patrik

=>(Get-BitsTransfer download_0).jobstate
Suspended
```

Z ukázky je dobře patrná výhoda jobů, jež jsou zakládány jako objekty – práce s nimi je velmi snadná, pokud použijeme rouru či přistoupíme k četným vlastnostem objektu.

3. Přenos dat lze stejně snadno znovu nastartovat, případně úlohu úplně odstranit.

```
=>Get-BitsTransfer download_0 | Resume-BitsTransfer -Asynchronous

=>Get-BitsTransfer download_0 | Remove-BitsTransfer
```

4. Rozhodně neplatí pravidlo, že jedna úloha služby BITS (jeden job) může stahovat jen jediný soubor. Každé úloze lze přiřadit řadu souborů, a to i dodatečně, během stahování.

```
=>$download_jobs

JobId               DisplayName         TransferType        JobState
-----               -----------         ------------        --------
e0730b76-5f9a-46... Download_0          Download            Transferring
393ca5e7-a27c-4e... Download_1          Download            Transferring

=>Add-BitsFile -BitsJob $download_jobs[0] `
-Source http://download.services.openoffice.org/files/extended/3.2.0rc1/en-US.exe `
-Destination c:\download\en-US.exe

=>Get-BitsTransfer download_0 | fl displayname,jobstate,filelist

DisplayName : Download_0
JobState    : Transferring
FileList    : {http://download.services.openoffice.org/files/stable/3.1.1/OOo_
3.1.1_Win32Intel_install_en-US.exe, http://download.services.openoffice.org/files/
extended/3.2.0rc1/OOo_3.2.0rc1_20091217_Win32Intel_install_en-US.exe}
```

Po našem zákroku má úloha dva soubory ke stahování. Pokud nám nezáleží na odlišení jednotlivých úloh, můžeme takto snadno předat celou řadu souborů jedinému jobu a celou práci přenechat službě BITS.

Jednorázové spuštění úlohy služby BITS

Výše jsme si ukázali, že jediná úloha služby BITS může zpracovat kolekci souborů a zajistit jejich přenos. Tato možnost se projevuje i ve vlastnostech cmdletů, jimiž PowerShell službu a její úlohy ovládá. Novou úlohu lze pak spustit velmi snadno.

1. Připravíme si zdrojový soubor s daty ke stahování ve formátu CSV. Po jeho načtení vypadají data třeba takto:

```
=>import-csv download.csv | fl

name        : LastVersion
source      : http://download.services.openoffice.org/files/stable/3.1.1/OOo_
3.1.1_Win32Intel_install_en-US.exe
destination : c:\download\OOo_3.1.1_Win32Intel_install_en-US.exe

name        : DevelopVersion
source      : http://download.services.openoffice.org/files/extended/3.2.0rc1/
OOo_3.2.0rc1_20091217_Win32Intel_install_en-US.exe
destination : c:\download\OOo_3.2.0rc1_20091217_Win32Intel_install_en-US.exe
```

2. Samotný přenos spustíme velmi snadno a opět zde s úspěchem využijeme objektů a roury.

```
=>$prenos = import-csv download.csv | % {Start-BitsTransfer -source $_.source `
-destination $_.destination -Asynchronous -DisplayName $_.name}
```

```
=>$prenos

JobId               DisplayName   TransferType JobState       OwnerAccount
-----               -----------    ------------ --------      ------------
5fb478db-f617-4b... LastVersion   Download     Transferring   STROJ1\Patrik
ac0415fd-5fd2-40... DevelopVersionDownload     Transferring   STROJ1\Patrik
```

3. Pokud nám nezáleží na přiřazování některých parametrů a chceme prostě rychle a snadno zavolat hromadné stahování, můžeme přímo využít předání parametrů pomocí roury.

```
=>$prenos = import-csv download.csv | Start-BitsTransfer -Asynchronous
=>$prenos

JobId               DisplayName    TransferType  JobState      OwnerAccount
-----               -----------    ------------  --------      ------------
ce444b80-2c86-47... BITS Transfer  Download      Transferring  STROJ1\Patrik
555005df-fa4e-4d... BITS Transfer  Download      Transferring  STROJ1\Patrik
```

Shrnutí

PowerShell sám o sobě nenabízí příliš mnoho cmdletů, které by přímo pracovaly se síťovými klienty nebo plnily jejich funkci (pěknou výjimkou je třeba služba BITS). Na druhou stranu .NET Framework nabízí řadu zajímavých možností a použití příslušných tříd není jinak komplikované. Práce se síťovými protokoly je sama o sobě dosti silnou motivací k prozkoumání možností .NET Frameworku – obzvláště při práci s elektronickou poštou a obsahem webu zde najdeme velké možnosti.

Důležité k zapamatování

- PowerShell nabízí hned několik možností, jak odesílat e-mailové zprávy. Třídy .NET Frameworku jsou poměrně přehledné a výsledný kód dobře čitelný, na druhou stranu znalci knihovny CDO ji mohou snadno využívat i zde a využít tak veškeré zkušenosti.
- Ověření uživatelů služby SMTP je často klíčovým problémem nefunkčních skriptů na „mailování“. Důkladně si prověřte nastavení zabezpečení serveru SMTP, jejž budete využívat, a důkladně nastavte způsob ověření uživatele. Můžete-li, využijte ve vnitřní síti vnitřní server SMTP, který nebude vyžadovat komplikované ověření. Toto může ušetřit dlouhé minuty testování a pátrání.
- Aplikace Outlook zahrnuje bohaté objektové rozhraní, které můžeme výborně využít. Komplikace mohou nastat při ověřování uživatele/přihlašování, pokud jej třeba využíváme v konfiguraci „RPCoverHTTP“, tedy mimo vnitřní síť s dodatečným zapouzdřením protokolu MAPI. V takových situacích je lepší, když Outlook před spuštěním skriptu běží a k ověření již došlo předem.
- Regulární výrazy určitě stojí za námahu, protože jsou nepostradatelným prostředkem pro analýzu textu zpráv, hledání e-mailových adres či dalších specifických textů.
- .NET Framework v sobě ukrývá řadu zajímavých tříd, jež reprezentují základní síťové klienty.

- Služba BITS nám výborně pomůže při hromadném přenosu souborů. Pracuje jak s internetovými, tak s lokálními zdroji, a především můžeme její úlohy spustit jako job „na pozadí“ a neustále je kontrolovat či ovládat.

Průzkumy sítě a inventarizace

Úvod

Již jsme zmínili, že verze 1 síťovou problematiku víceméně ignorovala – jednoznačně se nejednalo o prioritu vývojového týmu, a tak je potřeba si pomoci jinými prostředky. Kupříkladu rozhraní WMI poskytuje dobrou možnost pro základní diagnostiku, kterou známe jinak jako běžný Ping:

```
=>Get-WmiObject win32_pingstatus -Filter "address='localhost'" | `
fl address,protocoladdress,statuscode

address         : localhost
protocoladdress : 127.0.0.1
statuscode      : 0
```

Ačkoliv bychom mohli použít běžný Ping.exe, výhodou výše popsaného postupu je samozřejmě zpracování výstupu v objektové podobě. Ve verzi 2 najdeme analogii použití objektu WMI zapouzdřenou do samostatného příkazu, jak si hned ukážeme. Bližší ohledání prozradí, že jsme se od původního objektu příliš nevzdálili:

```
=>Test-Connection -ComputerName localhost

Source        Destination     IPV4Address      IPV6Address      Bytes            Ti
                                                                                 me
                                                                                 (m
                                                                                 s)
------        -----------     -----------      -----------      -----            --
STROJ1        localhost       127.0.0.1        {}               32               0
STROJ1        localhost       127.0.0.1        {}               32               0
STROJ1        localhost       127.0.0.1        {}               32               0
STROJ1        localhost       127.0.0.1        {}               32               0


=>Test-Connection -ComputerName localhost | Get-Member


   TypeName: System.Management.ManagementObject#root\cimv2\Win32_PingStatus

Name                            MemberType     Definition
----                            ----------     ----------
Address                         Property       System.String Address {get;set;}
BufferSize                      Property       System.UInt32 BufferSize {get;set;}
NoFragmentation                 Property       System.Boolean NoFragmentation
{get;set;}
PrimaryAddressResolutionStatus Property       System.UInt32 PrimaryAddressResoluti-
onStatus {get;...
```

```
ProtocolAddress                 Property       System.String ProtocolAddress
{get;set;}
ProtocolAddressResolved         Property       System.String ProtocolAddressResolved
{get;set;}

...
ConvertFromDateTime             ScriptMethod   System.Object ConvertFromDateTime();
ConvertToDateTime               ScriptMethod   System.Object ConvertToDateTime();
IPV4Address                     ScriptProperty System.Object IPV4Address
{get=$iphost = [System.N...
IPV6Address                     ScriptProperty System.Object IPV6Address
{get=$iphost = [System.N...
```

Ano, příkaz *Test-Connection* není ničím jiným než zakukleným voláním výše vyzkoušené třídy WMI.

Zůstaneme-li u diagnostiky sítě, dalším obvyklým prostředkem je kontrola překladu síťových jmen. I v tomto případě nachází PowerShell oporu v .NET Frameworku a jeho vhodné třídě, jak je patrné z následujícího příkladu:

```
=>[System.Net.Dns] | Get-Member -Static

   TypeName: System.Net.Dns

Name                  MemberType Definition
----                  ---------- ----------
BeginGetHostAddresses Method     static Sys
BeginGetHostByName    Method     static Sys
BeginGetHostEntry     Method     static Sys
BeginResolve          Method     static Sys
EndGetHostAddresses   Method     static ipa
EndGetHostByName      Method     static Sys
EndGetHostEntry       Method     static Sys
EndResolve            Method     static Sys
Equals                Method     static boo
GetHostAddresses      Method     static ipa
GetHostByAddress      Method     static Sys
GetHostByName         Method     static Sys
GetHostEntry          Method     static Sys
GetHostName           Method     static str
ReferenceEquals       Method     static boo
Resolve               Method     static Sys

=>[System.Net.Dns]::GetHostAddresses("localhost") | select ipaddress*

IPAddressToString
-----------------
127.0.0.1
```

Zůstaňme u diagnostiky sítě ještě chvilku a zapátrejme po další nesmírně užitečné pomůcce, kterou je nástroj příkazového řádku Netstat. I data, která jsou shromážděna touto pomůckou, by se nám určitě hodila v objektové podobě tak, abychom je mohli

dále zpracovat v PowerShellu. .NET Framework ukrývá řadu zajímavostí i pro tyto situace, obzvláště pak třídu, kterou si nyní ukážeme:

```
=>[System.Net.NetworkInformation.IPGlobalProperties]::GetIPGlobalProperties()`
| Get-Member | ft name, membertype -AutoSize

Name                    MemberType
----                    ----------
Equals                      Method
GetActiveTcpConnections     Method
GetActiveTcpListeners       Method
GetActiveUdpListeners       Method
GetHashCode                 Method
GetIcmpV4Statistics         Method
GetIcmpV6Statistics         Method
GetIPv4GlobalStatistics     Method
GetIPv6GlobalStatistics     Method
GetTcpIPv4Statistics        Method
GetTcpIPv6Statistics        Method
GetType                     Method
GetUdpIPv4Statistics        Method
GetUdpIPv6Statistics        Method
ToString                    Method
DhcpScopeName             Property
DomainName                Property
HostName                  Property
IsWinsProxy               Property
NodeType                  Property

=>$netstat = `
[System.Net.NetworkInformation.IPGlobalProperties]::GetIPGlobalProperties()

=>$netstat.GetActiveTcpConnections() | ft local*,remote*,state -AutoSize

LocalEndPoint   RemoteEndPoint          State
-------------   --------------          -----
10.0.0.139:2650 66.102.13.113:80    CloseWait
10.0.0.139:2651 66.102.13.113:80    CloseWait
10.0.0.139:2742 74.125.87.138:80    CloseWait
10.0.0.139:2743 74.125.87.138:80    CloseWait
10.0.0.139:2804 212.24.152.83:443 Established
10.0.0.139:2805 212.24.152.83:443 Established
10.0.0.139:2806 212.24.152.83:443 Established
10.0.0.139:2807 212.24.152.83:443 Established
10.0.0.139:2808 212.24.152.83:443 Established
10.0.0.139:2809 212.24.152.83:443 Established
10.0.0.139:2810 212.24.152.83:443 Established
10.0.0.139:2811 212.24.152.83:443 Established
10.0.0.139:3076 212.24.152.83:443 Established
...
```

V našem průzkumu můžeme pokročit ještě o kousek dále, neboť i získané informace o spojení lze ještě „rozložit na prvočinitele“.

```
=>$netstat.GetActiveTcpConnections() | Get-Member

   TypeName: System.Net.NetworkInformation.SystemTcpConnectionInformation

Name           MemberType Definition
----           ---------- ----------
Equals         Method     bool Equals(System.Object obj)
GetHashCode    Method     int GetHashCode()
GetType        Method     type GetType()
ToString       Method     string ToString()
LocalEndPoint  Property   System.Net.IPEndPoint LocalEndPoint {get;}
RemoteEndPoint Property   System.Net.IPEndPoint RemoteEndPoint {get;}
State          Property   System.Net.NetworkInformation.TcpState State {get;}

=>$netstat.GetActiveTcpConnections() | `
% {"$($_.localendpoint.address);$($_.localendpoint.port)"}

10.0.0.139;2650
10.0.0.139;2651
10.0.0.139;2742
10.0.0.139;2743
10.0.0.139;2804
10.0.0.139;2805
10.0.0.139;2806
10.0.0.139;2807
10.0.0.139;2808
10.0.0.139;2809
10.0.0.139;2810
...
```

A chceme-li mít třeba představu, na kterých portech naslouchají aplikace lokálního systému, nezbývá než zavolat následující metodu:

```
=>$netstat.GetActiveTcpListeners() | ft * -AutoSize

AddressFamily Address         Port
------------- -------         ----
...
InterNetwork 0.0.0.0         5985
InterNetwork 0.0.0.0        44334
InterNetwork 0.0.0.0        44501
InterNetwork 0.0.0.0        47001
InterNetwork 10.0.0.139       139
InterNetwork 127.0.0.1       1032
InterNetwork 127.0.0.1       5152
InterNetwork 192.168.154.1    139
InterNetwork 192.168.199.1    139
...
```

Hromadné prověření dostupnosti síťových klientů

Předvedené možnosti základního testování sítě nám dovolují sestavit sice jednoduché, ale často potřebné řešení, pomocí něhož prověříme dostupnost části síťových adres a případně je přeložíme na jména. Skript bude pracovat s jednoduchým výchozím konfiguračním souborem, v němž si určíme rozsah testování a adresu sítě.

ipdisc.txt, batch_ping.ps1

1. Sestavíme konfigurační soubor a určíme v něm adresu testované sítě, rozsah adres klientů a také požadavek, zda bude proveden překlad jmen.

```
=>cat .\ipdisc.txt
net=10.0.0.
start=130
end=140
names=true
```

2. Načteme obsah konfiguračního souboru a založíme výstupní soubor pro záznam nalezených informací.

```
$conf = (cat .\ipdisc.txt) -join "`n"
$conf_table = ConvertFrom-StringData $conf

"adresa,ping,jmeno" | Add-Content net_discovery.csv
```

3. Nyní začneme pracovat s jednotlivými adresami postupně v cyklu, jehož tělo zajistí testování. Rozsah cyklu bude určen spodní a horní hranicí tak, jak jsme je načetli z konfiguračního souboru.

```
for ($klient = [int]$conf_table.start;$klient -le [int]$conf_table.end;`
$klient++)

{
```

4. V tomto kroku si sestavíme celou adresu IP ze síťové a klientské části, pro jistotu ji vypíšeme na konzolu a následně zavoláme testování síťové dostupnosti.

```
$adresa = $conf_table.net + [string]$klient
Write-Host $adresa

$con = Test-Connection $adresa -Quiet
```

   Povšimněme si přepínače *-Quiet*, který zajistí výstup hodnoty typu *Boolean* podle toho, zda „Ping“ zafungoval, namísto chybových hlášení.

5. Nyní ověříme, zda konfigurační soubor vyžaduje překlad jmen (parametr *names* nastaven na *true*), a případně jej provedeme. Využijeme nám již dobře známé třídy .NET Frameworku.

```
if ($conf_table.net)

     {
             $name = [System.Net.Dns]::resolve($adresa)
     }
```

Nezapomeňme, že metoda *Resolve* vrací v případě úspěchu jméno a v případě opačném adresu, kterou jsme jí předložili.

6. Získaná data tedy konečně můžeme přidat do našeho výstupního souboru a poté cyklus uzavřeme. Po ukončení cyklu naše práce končí.

```
$adresa + "," + $con + "," + $name.hostname | Add-Content net_discovery.csv

}

Write-Host "Hotovo"
```

7. Výstupní soubor pak bude vypadat třeba takto:

```
adresa,ping,jmeno
10.0.0.130,False,10.0.0.130
10.0.0.131,False,10.0.0.131
10.0.0.132,False,10.0.0.132
10.0.0.133,False,10.0.0.133
10.0.0.134,False,10.0.0.134
10.0.0.135,False,10.0.0.135
10.0.0.136,False,10.0.0.136
10.0.0.137,False,10.0.0.137
10.0.0.138,True,mygateway1.ar7
10.0.0.139,True,NTB1
10.0.0.140,True,Stroj1
```

Síťové adresy a ověření příslušnosti k podsíti

Při správě síťového prostředí často narážíme na okruh typických úloh, jejichž součástí je prověření, zdali určitá adresa IP náleží k dané síti (či podsíti). Vstupní údaje bývají zadány různými způsoby, ovšem adresu IP protokolu verze 4 (tedy dodnes běžnější adresu IP) máme typicky k dispozici jako čtyři desítkové cifry oddělené tečkami. Adresa sítě bývá zadána obdobně, případně ji určujeme délkou tzv. síťové masky, což bývá počet bitů. Při snaze určit, zda síťová adresa počítače náleží do dané sítě, která je určena maskou, narazíme na několik problémů, jejichž řešení naznačí tato úloha.

Náš skript hromadně zpracuje data, která uložíme ve zdrojových souborech. Jeden z nich obsahuje prostý seznam adres IP, jejichž síťovou příslušnost zkoumáme, a vypadá takto:

```
192.168.1.10
172.16.10.5
...
```

Druhý, referenční soubor zahrnuje seznam nám známých síťových adres s jejich označením. Jeho řádky vypadají takto:

```
Reg,Country,Mask
Europe,CZ,192.168.129.5/17
Asia,JP,172.16.210.0/17
```

Je patrné, že soubor má strukturu CSV (hodnot oddělených čárkou), a jednotlivé sloupce tedy budou reprezentovat příslušné parametry pro každou síť. Skript tedy bude postup-

ně načítat adresy z prvního souboru, prohledá referenční tabulku a zjistí, jestli se daná adresa nachází v některé z vyjmenovaných sítí. Pokud ano, obdržíme její identifikaci ve výstupní sestavě.

sip.txt, nets.csv, ipnetcheck.ps1

1. Prvním krokem bude načtení obsahu souborů do proměnných. Tento postup není nezbytně nutný, ale především u referenční tabulky sítí je výhodnější, neboť urychlí opakovaný přístup k těmto datům (jak dále uvidíme, budeme je potřebovat stále dokola).

```
#adresy k pruzkumu
$saddrs = cat sip.txt

#referencni tabulka siti
$nets = import-csv nets.csv
```

   Povšimněte si, referenční tabulka je importována jako data CSV, díky čemuž PowerShell okamžitě konvertuje tabulku na objektovou reprezentaci.

2. V dalším kroku spustíme hlavní cyklus, který bude procházet postupně všechny adresy ze zdrojového souboru a porovnávat je se sítěmi, které známe.

```
ForEach ($saddr in $saddrs) {

    $out_line = "$saddr,,Not found"
```

   Ihned po započetí cyklu jsme nastavili obsah výstupní proměnné na výchozí hodnotu, která zůstane nezměněna, pokud nenajdeme žádnou příslušnou síť.

3. V tomto kroku spouštíme hlavní hledací cyklus, v jehož průběhu budou načítány sítě z referenčního seznamu a porovnávány s adresou IP.

```
ForEach ($net in $nets)  {
```

   Proměnná *$net* bude tedy reprezentovat aktuální síť z referenční tabulky.

4. Během této důležité fáze provádíme klíčovou operaci, jež umožňuje porovnání sítě se zkoumanou adresou. Povšimněme si znovu, jak vypadá vstupní struktura referenční tabulky:

```
Europe,CZ,192.168.129.5/17
```

   Potřebujeme tedy vypreparovat jednak adresu sítě, jednak samotnou délku masky vyjádřenou celým číslem:

```
192.168.129.5/17
```

   Provedeme to rozdělením řetězce dle znaku „/“ tak, že obdržíme levou a pravou část v samostatných proměnných:

```
$Cmask = ([string]($net.mask.Split("/"))[0])
 $CIDRbits = ([int]($net.mask.Split("/"))[1])
```

   Levá část je nyní uložena v první proměnné a obsahuje samotnou adresu sítě IP (tedy vše vlevo od lomítka), druhá proměnná pak jen délku masky (hodnotu za lomítkem). Rozdělení jsme provedli voláním metody *Split* na textovém řetězci

a výsledné hodnoty v poli dvou prvků jsme pak volali jejich indexem (ve hranatých závorkách).

5. V tuto chvíli máme připraveny všechny vstupní údaje pro porovnání adresy a sítě, a můžeme tedy zavolat kód, který provede porovnání. V našem skriptu to zajistíme voláním funkce, která jednoduše odpoví, zda síťová adresa do uvedené sítě patří, nebo ne.

```
if (NetCheck $saddr $Cmask $CIDRbits) `
{$out_line = "$saddr,$($net.Country),$Cmask";break}
```

   Celé ověření pro nás zajistí funkce *NetCheck*, která je středobodem logiky naší úlohy, a proto se na ni nyní zaměříme.

6. Z předchozího kroku je zřejmé, že funkce obdrží tři vstupní hodnoty. Její definice vstupních proměnných vypadá takto:

```
Function NetCheck (
  [string]$SourceAddress,[string]$Network,[int]$maskBits){
```

   Obě adresy (hledanou i referenční) začneme zpracovávat jako text, masku pak jako celočíselnou hodnotu.

7. Dalším krokem bude převod síťových adres na objekty, neboť .NET Framework zahrnuje třídu reprezentující adresu IP. To se nám vzápětí bude hodit k jejich dalšímu zpracování, neboť daná třída nabízí převod jednotlivých částí adresy IP (samostatných oktetů oddělených tečkami) na jedno číslo, s nímž dále budeme pracovat.

```
$saBytes = [System.Net.IPaddress]::Parse($SourceAddress)
$maskBytes = [System.Net.IPaddress]::Parse($Network)
```

8. Nyní zavoláme adresy v jejich objektové reprezentaci a uložíme jednotlivé části adresy (oktety) jako jednotlivé prvky pole, s nímž budeme dále pracovat.

```
$saOctets = $saBytes.GetAddressBytes()
$maskOctets = $maskBytes.GetAddressBytes()
```

9. Původní adresu máme tedy „rozsekánu“ na jednotlivé oktety, z nichž složíme celkovou reprezentaci ve dvojkové soustavě, potřebnou pro porovnání se síťovou adresou a maskou. Nejdříve si založíme cílovou proměnnou a posléze vstoupíme do cyklu, jenž postupně zpracuje všechny čtyři oktety do binární podoby a seřadí je za sebe.

```
$saBin = ""
ForEach ($saOctet in $saOctets){


   $saOctetFull = FillByte ([convert]::tostring($saOctet,2))
   $saBin += $saOctetFull

}
```

   Vysvětleme zde několik důležitých kroků v předchozím kousku kódu. Převod do binární podoby zajišťuje metoda *ToString*, jejímiž parametry jsou samotný oktet

adresy IP (tedy třeba hodnota „172“), a dále určení dvojkové soustavy (ono číslo 2). Tato metoda bohužel při převodu provádí nepříjemnou věc: zahazuje nepotřebné nuly, takže oktet se nemusí přeložit na plnou délku požadované binární číslice (tedy osm nul či jedniček). Abychom toto vyřešili, zavoláme další funkci jménem *FillByte*, již vzápětí vysvětlíme.

10. Funkce *FillByte* provádí vlastně jednoduchou operaci – dokud není původní oktet v plné délce osmi číslic, doplňuje zleva chybějící nuly.

```
Function FillByte (
[string]$Byte){

    while ($byte.length -lt 8) {

    $Byte = "0" + $Byte

    }

Return $Byte
```

Po vyplnění je hodnota navrácena v kompletní reprezentaci s osmi binárními číslicemi.

11. Po převodu zdrojové (hledané) adresy na její plnou binární podobu provedeme totéž ještě se síťovou adresou – i tentokráte si vypomůžeme funkcí pro doplňování chybějících nul v jednotlivých oktetech.

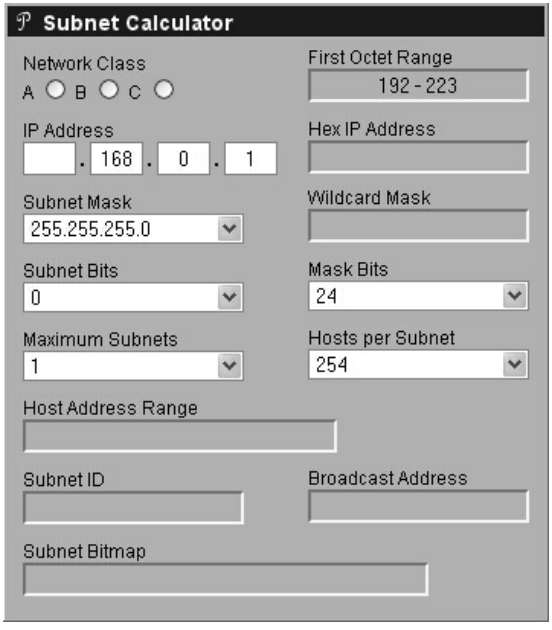
```
    $MaskBin = ""
    ForEach ($maskOctet in $maskOctets){


        $maskOctetFull = FillByte ([convert]::tostring($maskOctet,2))
        $maskBin += $maskOctetFull

    }
```

12. Provedli jsme veškeré přípravné kroky a můžeme konečně porovnat obě adresy a zjistit, jestli sdílejí stejnou síťovou část. Provedeme to porovnáním odpovídajících částí řetězců, jež odřízneme na potřebnou délku pomocí *SubString*. Samotnou délku určuje délka síťové masky, kterou jsme po celou dobu drželi jako celočíselnou hodnotu.

```
$saBin.SubString(0,$maskBits) -match $maskBin.SubString(0,$maskBits)
```

Dobře si povšimněte, že celé porovnání je zajištěno pomocí operátoru *match*, který poskytne výstupní logickou hodnotu. Nalezneme-li shodu, pak celá funkce vrátí hodnotu *Pravda* (*True*). Tento jediný řádek tedy odřízne z obou adres jejich síťovou část (určený počet bitů zleva) a porovná je mezi sebou. Zdrojová adresa tedy může vypadat třeba takto: 11000000101010000000000100001010. Je-li maska určena jako 16, pak se bude porovnávat jen tučně označená část: **1100000010101000**0000000100001010. Pokud síťová adresa vypadá takto: **1100000010101000**1000000100000101,

je zřejmé, že síťová část se shoduje a **testovaná adresa počítače bude ležet v testované síti.** Funkce tedy vrátí logickou hodnotu *Pravda*.



Obrázek 7.9: Počítání složitějších adres a masek často překračuje naši fantazii. Pro větší názornost můžeme sáhnout po grafickém kalkulátoru.

13. Již dříve jsme si ukázali, že výstup funkce je součástí rozhodovacího bloku podmínky *If*. Pokud se tedy vrátí logická hodnota *Pravda*, je proveden kód v těle podmínky – získaná hodnota je považována za nalezené řešení, cyklus prohledávání sítí je ukončen (*break*) a na výstup je zapsán výsledek.

```
if (NetCheck $saddr $Cmask $CIDRbits) `
{$out_line = "$saddr,$($net.Country),$Cmask";break}

    }

  $out_line
```

Výstupní hodnota pak bude zahrnovat původní adresu, nalezenou síť a její identifikaci. Porovnejme tedy ještě jednou vstupní údaje, průběh změn ve skriptu a výstupní hodnoty, jak jsou uvedeny v tabulce 7.3.

Tabulka 7.3: Logika našeho řešení je již poměrně pokročilá, proto si pomůžeme názorným přehledem vstupních a výstupních hodnot

	Testovaná adresa	Testovaná síť
Vstupní řádek	192.168.129.5	Europe,CZ,192.168.129.5/16
Původní podoba adresy	192.168.129.5	192.168.1.10/16

	Testovaná adresa	Testovaná síť
Binární podoba	11000000101010000000000100001010	11000000101010001000000100000101
Síťová část	1100000010101000	1100000010101000
Výstupní řádek	192.168.1.10,CZ,192.168.129.5	

V případě, že síť nebyla nalezena (a funkce *NetCheck* vrátí hodnotu *Nepravda*), cyklus pokračuje testováním dalších dostupných sítí v referenčním seznamu a kontrolní hodnota na výstupu bude třeba takováto: „172.16.10.5„Not found““

Poznámka: Postup uvedený v tomto příkladu není zdaleka jediným řešením dané úlohy. Mohou se lišit jak formy vstupních dat (ne vždy je maska zadána jako počet bitů, někdy jde o oktety jako adresy), tak postupy s využitím různých tříd .NET Frameworku na převod adres a jejich porovnání. Dané řešení je jen jednou z cest k cíli.

Shrnutí

Tvůrci PowerShellu nevytvořili téměř žádné speciální cmdlety či postupy pro diagnostiku sítě. Zcela spoléhají na prostředky dostupné v rozhraní WMI a v třídách .NET Frameworku. Pokud plánujeme rozsáhlejší využití, určitě je dobré zběžně prohlédnout dokumentaci ke třídám, o nichž hovoříme v našich postupech. Na druhou stranu základní kontrola (ping, překlad jmen) je jednoduchá a nevyžaduje složité skriptování.

Důležité k zapamatování

- PowerShell si pomáhá při diagnostice sítě a síťových rozhraní především třídami .NET Frameworku, jež ukrývají velmi dobrou výbavu. Není nezbytně nutné spouštět klasické nástroje jako Netstat a posléze složitě zpracovávat jejich textový (neobjektový) výstup.
- Cmdlet *Test-Connection* přináší jednu zajímavou výhodu oproti své mateřské třídě WMI, na jejímž základě je zbudován. Přepínač *Quiet* potlačuje případná chybová hlášení a na výstupu dostáváme stručnou a výstižnou informaci, jestli klient na *Ping* reaguje nebo ne.
- Sama adresa IP může být uložena jako objekt dle odpovídající třídy v .NET Frameworku. Přináší to výhody při jejím dalším použití.

Otázky a odpovědi

Otázka:

Jaké příkazy PowerShellu slouží ke konfiguraci síťových rozhraní v systému Windows?

Odpověď:

PowerShell žádné cmdlety pro přímou konfiguraci síťových rozhraní neobsahuje. Síťová rozhraní můžeme ovládat prostřednictvím bohatě „vybavených“ tříd rozhraní WMI.

Otázka:

Můžeme v PowerShellu ovládat firewall operačního systému Windows? Pokud ano, ve kterých verzích?

Odpověď:

Firewall systému Windows můžeme ovládat pomocí skriptů a konzoly ve kterékoliv verzi, v níž se vyskytuje. Nicméně objekty, které firewall reprezentují, ze liší dle verze Windows (XP a novější).

Otázka:

Lze z PowerShellu odeslat e-mailovou zprávu?

Odpověď:

Ano, PowerShell lze použít jako poštovní klient protokolu SMTP pro odeslání zprávy. K dispozici je jednak starší knihovna CDO, jednak novější třídy .NET Frameworku. Ve verzi 2 je k dispozici nový příkaz *Send-MailMessage*.

Otázka:

Které příkazy můžeme využít ke stahování obsahu webových stránek?

Odpověď:

PowerShell neobsahuje žádné speciální příkazy pro práci s běžnými internetovými aplikačními síťovými protokoly. Můžeme však využít řadu zajímavých tříd .NET Frameworku.

Otázka:

K čemu slouží služba BITS a sada cmdletů k jejímu ovládání?

Odpověď:

Služba Background Inteligent Transfer Service slouží k pokročilému přenosu souborů mezi počítači se systémem Windows. Využívá ji třeba distribuovaná služba WSUS pro šíření záplat a instalačních balíčků a můžeme ji přímo využít i pro svá data.

Otázka:

Jak můžeme pomocí roury zpracovat výstup příkazu Ping při hromadné kontrole dostupnosti síťových adres?

Odpověď:

Přímé zpracování textového výstupu příkazu Ping je zbytečně složité a navíc zbytečné. Stejnou službu nabízí jednak jedna z tříd WMI, jednak nový příkaz *Test-Connection*, jejichž výstup je objektový a můžeme jej přímo ovládat v rouře.