Skrypty powershell

- Audyt grup z zagnieżdzoną grupą z innej domeny
- Skrypt wykonujący audyt wszystkich wpisów w DNS
- Skrypt do wysłania wiadomości e-mail
- Sprawdzenie otwartych / nasłuchujących portów wraz z aktywnym procesem
- Skrypt do odrzucenia starych paczek i czyszczenia bazy WSUS
- Skrypt dodający aliasy SMTP użytkownikom w parametrach AD
- Wyszukiwania zadania w Task Scheduler o odpowiedniej nazwie
- Wyszukiwanie frazy we wszystkich GPO.
- Pobieranie klucza BitLocker z AD

Audyt grup z zagnieżdzoną grupą z innej domeny

> Komendę uruchamiamy jako Administrator w powershell na systemie z zainstalowanymi przystawkami RSAT.

```
# Pytaj użytkownika o nazwę grupy, kontroler domeny (DC) i poświadczenia
$Group = Read-Host -Prompt "Podaj nazwę grupy"
$DC = Read-Host -Prompt "Podaj kontroler domeny badź jej nazwę (dc.example.local / example.local)"
$Credential = Get-Credential -Message "Podaj poświadczenia użytkownika do dostępu do domeny $DC"

# Pobierz członków grupy z określonego kontrolera domeny
$membership = Get-ADGroup $Group -Properties Member -Server $DC -Credential $Credential
$membership.Member | Group-Object { ($_ -split '(?=DC=)',2)[1] } |
ForEach-Object {
   [adsi] $ldap = 'LDAP://{0}' -f $_.Name
   [string] $domain = $ldap.Name

   foreach($member in $_.Group) {
      $obj = Get-ADObject $member -Server $domain -Credential $Credential
      [pscustomobject]@{
         MemberOf       = $membership.Name
         Domain         = $domain
         Name           = $obj.Name
         DSN            = $obj.DistinguishedName
         SamAccountName = $obj.SamAccountName
         ObjectClass    = $obj.ObjectClass
      }
   }
}
```

> Wersja z zapisem pliku do CSV

```
# Pytaj użytkownika o nazwę grupy, kontroler domeny (DC) i poświadczenia
$Group = Read-Host -Prompt "Podaj nazwę grupy"
$DC = Read-Host -Prompt "Podaj kontroler domeny badź jej nazwę (dc.example.local / example.local)"
$Credential = Get-Credential -Message "Podaj poświadczenia użytkownika do dostępu do domeny $DC"

# Pobierz członków grupy z określonego kontrolera domeny
$membership = Get-ADGroup $Group -Properties Member -Server $DC -Credential $Credential

# Zbieramy obiekty członków grupy
$results = foreach ($domainGroup in $membership.Member | Group-Object { ($_ -split '(?=DC=)',2)[1] }) {
    [adsi] $ldap = "LDAP://$($domainGroup.Name)"
    [string] $domain = $ldap.Name

    foreach ($member in $domainGroup.Group) {
        $obj = Get-ADObject $member -Server $domain -Credential $Credential
        [pscustomobject]@{
            MemberOf       = $membership.Name
            Domain         = $domain
            Name           = $obj.Name
            DSN            = $obj.DistinguishedName
            SamAccountName = $obj.SamAccountName
            ObjectClass    = $obj.ObjectClass
        }
    }
}

# Pytanie do użytkownika czy zapisać wynik
$saveCsv = Read-Host -Prompt "Czy chcesz zapisać wynik do pliku CSV? (t/n)"

if ($saveCsv -eq 't') {
    $path = Read-Host -Prompt "Podaj pełną ścieżkę do pliku CSV"

    try {
        $results | Export-Csv -Path $path -NoTypeInformation -Encoding UTF8
        Write-Host "Wynik zapisano do pliku: $path" -ForegroundColor Green
    }
    catch {
        Write-Host "Wystąpił błąd podczas zapisywania pliku:" -ForegroundColor Red
        Write-Host $_.Exception.Message -ForegroundColor Red
    }
```

```
}
else {
    Write-Host "Wynik nie został zapisany do pliku." -ForegroundColor Yellow
}

# Możesz też wyświetlić wyniki na ekranie jeśli chcesz
$results
```

Skrypt wykonujący audyt wszystkich wpisów w DNS

Skrypt uruchamiamy jako Administrator w powershell, z zainstalowanymi przystawkami RSAT - DNS.

Skrypt zapisujący wyniki do pliku:

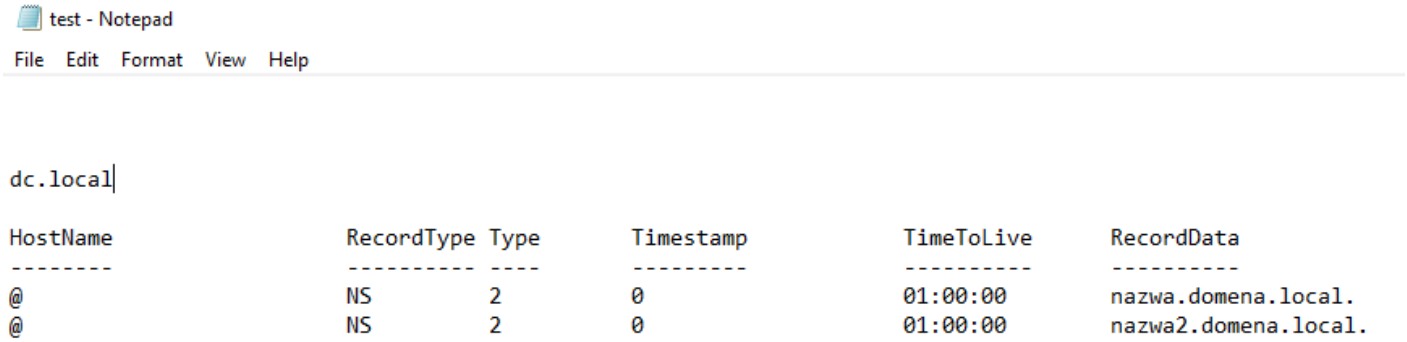
```
### Pobieranie argumentu adresu serwera DNS ###
if ($args.Count -eq 2) {
    $DnsServer = $args[0]
    $SavePath = $args[1]
} else {
    # Jeśli uruchomiono w ISE lub bez argumentów, prosi o wpisanie ręcznie
    Write-Host "Uruchom skrypt poprzez ./NazwaSkryptu.ps1 AdresSerweraDNS ŚcieżkaDoPliku.txt bądź podaj je teraz: "
    $DnsServer = Read-Host "Podaj adres serwera DNS"
    $SavePath = Read-Host "Podaj ścieżkę do zapisu"
}

# Sprawdzenie czy podano ścieżkę do zapisu
if (-not $SavePath) {
    Write-Host "Nie podano ścieżki do zapisu. Skrypt zostanie przerwany." -ForegroundColor Red
    exit
}

# Pobranie stref DNS z serwera
$Zones = @(Get-DnsServerZone -ComputerName $DnsServer)

ForEach ($Zone in $Zones) {
    # Wypisywanie strefy do konsoli i pliku
    Write-Host "`n$($Zone.ZoneName)" -ForegroundColor "Green"
    "`n$($Zone.ZoneName)" | Out-File -Append -FilePath $SavePath

    # Pobranie rekordów DNS w strefie i zapis do pliku
```

```
    Get-DnsServerResourceRecord -ComputerName $DnsServer -ZoneName $Zone.ZoneName | Out-File -Append -FilePath $SavePath
}

Write-Host "Zapis zakończony. Dane znajdują się w: $SavePath" -ForegroundColor Cyan
```

Przykładowe uruchomienie :

```
Uruchom skrypt poprzez ./NazwaSkryptu.ps1 AdresSerweraDNS ŚcieżkaDoPliku.txt bądź podaj je teraz:
Podaj adres serwera DNS: xelo.gbcc.local
Podaj ścieżkę do zapisu: test.txt
```

```
Zapis zakończony. Dane znajdują się w: test.txt

PS C:\Users\adm_koniecznyt>
```



Skrypt który każdą zone pokazuje w osobnym, interaktywnym oknie

> Uwaga! Przy większej ilości zone w DNS może zająć dużo czasu przez przeklikanie się przez wszystkie okna

> Okna otwierają się dopiero po zamknięciu poprzedniego

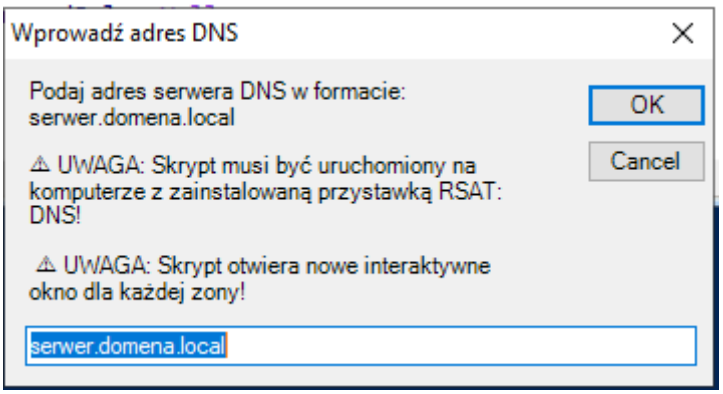
```
Add-Type -AssemblyName Microsoft.VisualBasic
Add-Type -TypeDefinition @"
using System;
using System.Runtime.InteropServices;

public class WinAPI {
    [DllImport("user32.dll")]
    public static extern bool SetForegroundWindow(IntPtr hWnd);
}
"@ -Language CSharp
```

```
# Tworzenie formularza i ustawienie go na aktywne okno
[System.Windows.Forms.Application]::EnableVisualStyles()
$null = [System.Windows.Forms.Form]::new()

# Wyświetlenie okna dialogowego do wpisania adresu serwera DNS z instrukcją
$DnsServer = [Microsoft.VisualBasic.Interaction]::InputBox(
    "Podaj adres serwera DNS w formacie: serwer.domena.local`n`n⚠ UWAGA: Skrypt musi być uruchomiony na komputerze z zainstalowaną przystawką RSAT: DNS!`n`n ⚠ UWAGA: Skrypt otwiera nowe interaktywne okno dla każdej zony!",
    "Wprowadź adres DNS",
    "serwer.domena.local"
)

# Ustawienie okna na pierwszym planie po zamknięciu InputBox
$process = Get-Process | Where-Object { $_.MainWindowTitle -match "Wprowadź adres DNS" }
if ($process) {
    [WinAPI]::SetForegroundWindow($process.MainWindowHandle)
}

# Jeśli użytkownik nic nie wpisał i kliknął "Anuluj", skrypt się zatrzyma
if (-not $DnsServer) {
    Write-Host "Nie podano adresu serwera DNS. Skrypt zostanie zakończony." -ForegroundColor Red
    exit
}

# Pobranie stref DNS z serwera
$Zones = @(Get-DnsServerZone -ComputerName $DnsServer)

ForEach ($Zone in $Zones) {
    Write-Host "`n$($Zone.ZoneName)" -ForegroundColor "Green"

    # Pobranie rekordów DNS w strefie
    $records = Get-DnsServerResourceRecord -ComputerName $DnsServer -ZoneName $Zone.ZoneName

    if ($records) {
        # Wymuszenie oczekiwania na zamknięcie Out-GridView przed otwarciem kolejnego
        $records | Out-GridView -Title "Rekordy DNS - $($Zone.ZoneName)" -PassThru | Out-Null
    } else {
        Write-Host "Brak rekordów w strefie $($Zone.ZoneName)" -ForegroundColor Yellow
    }
```

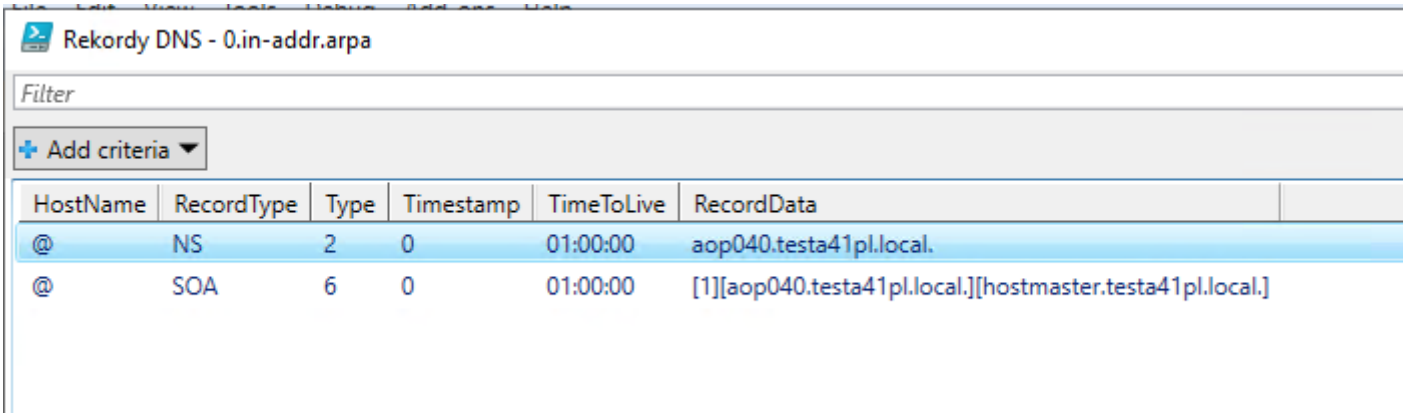
```
}


Write-Host "Koniec działania skryptu." -ForegroundColor Cyan
```

Przykładowe użycie:

> Pytanie o serwer DNS



> Interaktywne okno z wynikami



Skrypt do wysłania wiadomości e-mail

Uruchamiamy skrypt z poziomu Powershell ISE

```
$EmailFrom = "AdresNadawcy@domena.com"
$EmailTo = "AdresOdbiorcy@domena.com"
$SMTPServer = "smtp.domena.com" ## Adres serwera SMTP
$SMTPPort = 587 ## Port serwera
$SMTPUsername = "no-reply@domena.com"
$SMTPPassword = "SilneHasłoDoSkrzynki"
$Subject = "Temat Wiadomości"
$Body = "Treść wiadomości"

$SMTPPassword = ConvertTo-SecureString $SMTPPassword -AsPlainText -Force
$Credential = New-Object System.Management.Automation.PSCredential ($SMTPUsername, $SMTPPassword)

Send-MailMessage -From $EmailFrom -To $EmailTo -Subject $Subject -Body $Body -SmtpServer $SMTPServer -Port $SMTPPort -Credential $Credential -UseSsl
```

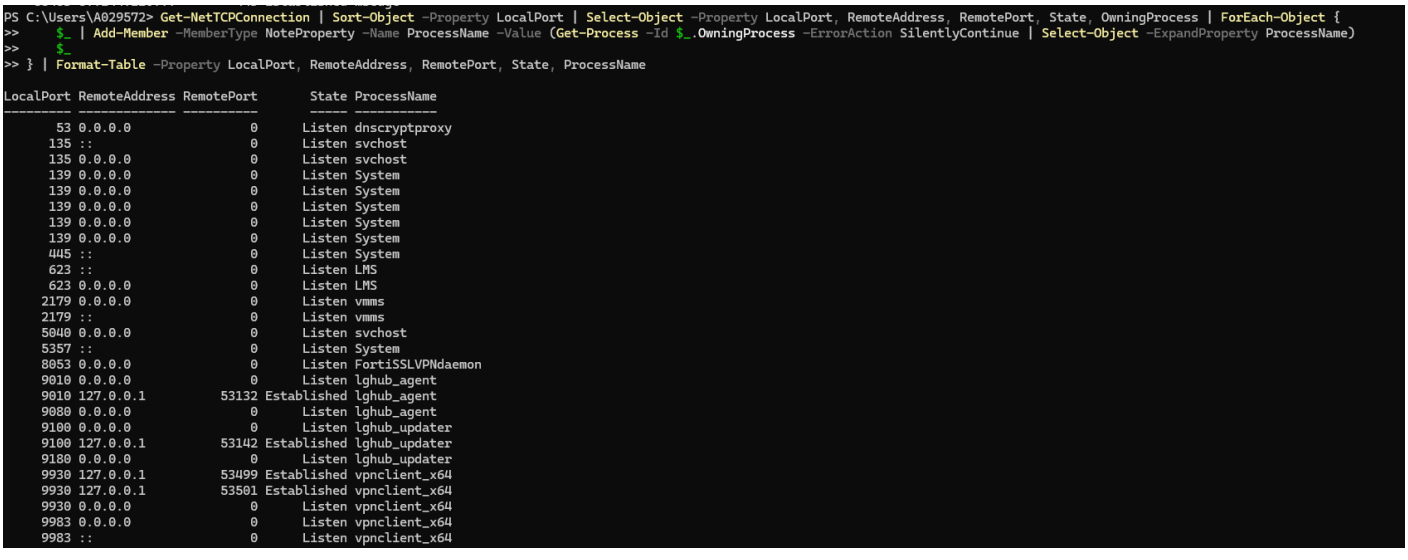
Sprawdzenie otwartych / nasłuchujących portów wraz z aktywnym procesem

Skrypty uruchamiamy w powershell.

Samo pobranie danych bez parsowania wyników:

```
Get-NetTCPConnection | Sort-Object -Property LocalPort | Select-Object -Property LocalPort, RemoteAddress, RemotePort, State, OwningProcess | ForEach-Object {
    $_ | Add-Member -MemberType NoteProperty -Name ProcessName -Value (Get-Process -Id $_.OwningProcess -ErrorAction SilentlyContinue | Select-Object -ExpandProperty ProcessName)
    $_
} | Format-Table -Property LocalPort, RemoteAddress, RemotePort, State, ProcessName
```

Przykładowy wynik:



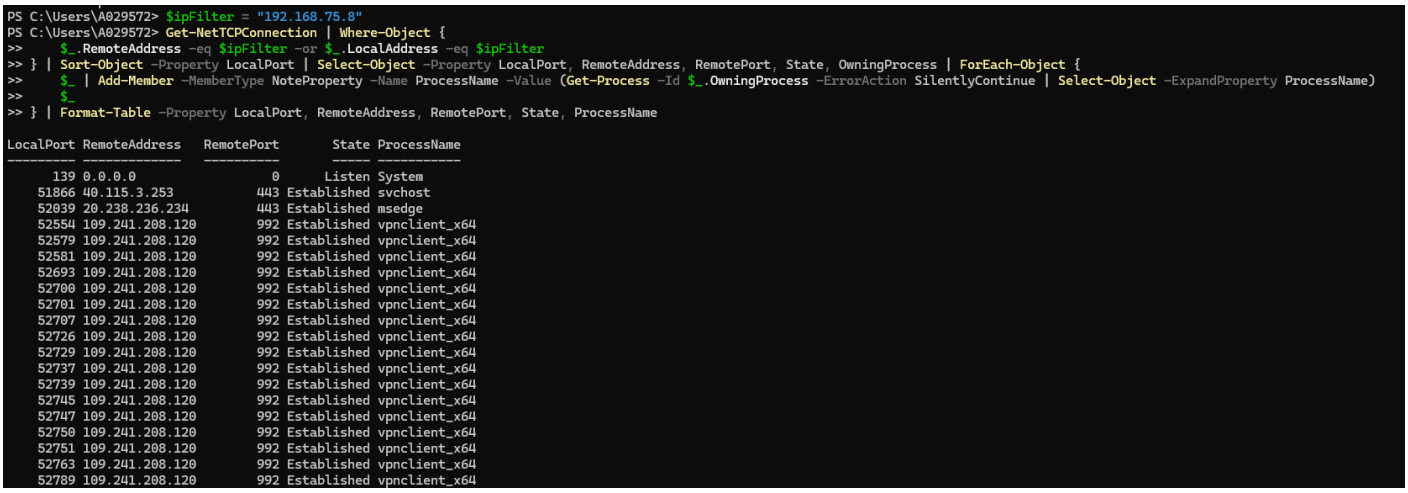
```
PS C:\Users\A029572> Get-NetTCPConnection | Sort-Object -Property LocalPort | Select-Object -Property LocalPort, RemoteAddress, RemotePort, State, OwningProcess | ForEach-Object {
>>     $_ | Add-Member -MemberType NoteProperty -Name ProcessName -Value (Get-Process -Id $_.OwningProcess -ErrorAction SilentlyContinue | Select-Object -ExpandProperty ProcessName)
>>     $_
>> } | Format-Table -Property LocalPort, RemoteAddress, RemotePort, State, ProcessName

LocalPort RemoteAddress RemotePort       State ProcessName
--------- ------------- ----------       ----- -----------
       53 0.0.0.0               0      Listen dnscryptproxy
      135 ::                    0      Listen svchost
      135 0.0.0.0               0      Listen svchost
      139 0.0.0.0               0      Listen System
      139 0.0.0.0               0      Listen System
      139 0.0.0.0               0      Listen System
      139 0.0.0.0               0      Listen System
      139 0.0.0.0               0      Listen System
      445 ::                    0      Listen System
      623 ::                    0      Listen LMS
      623 0.0.0.0               0      Listen LMS
     2179 0.0.0.0               0      Listen vmms
     2179 ::                    0      Listen vmms
     5040 0.0.0.0               0      Listen svchost
     5357 ::                    0      Listen System
     8053 0.0.0.0               0      Listen FortiSSLVPNdaemon
     9010 0.0.0.0               0      Listen lghub_agent
     9010 127.0.0.1         53132 Established lghub_agent
     9080 0.0.0.0               0      Listen lghub_agent
     9100 0.0.0.0               0      Listen lghub_updater
     9100 127.0.0.1         53142 Established lghub_updater
     9180 0.0.0.0               0      Listen lghub_updater
     9930 127.0.0.1         53499 Established vpnclient_x64
     9930 127.0.0.1         53501 Established vpnclient_x64
     9930 0.0.0.0               0      Listen vpnclient_x64
     9983 0.0.0.0               0      Listen vpnclient_x64
     9983 ::                    0      Listen vpnclient_x64
```

Pobieranie danych z filtrowaniem po IP:

```
$ipFilter = "Szukanie danego IP"


Get-NetTCPConnection | Where-Object {
```

```
    $_.RemoteAddress -eq $ipFilter -or $_.LocalAddress -eq $ipFilter
} | Sort-Object -Property LocalPort | Select-Object -Property LocalPort, RemoteAddress, RemotePort, State, OwningProcess | ForEach-Object {
    $_ | Add-Member -MemberType NoteProperty -Name ProcessName -Value (Get-Process -Id $_.OwningProcess -ErrorAction SilentlyContinue | Select-Object -ExpandProperty ProcessName)
    $_
} | Format-Table -Property LocalPort, RemoteAddress, RemotePort, State, ProcessName
```

Przykładowy wynik:



Pobieranie danych i zapisanie wyników do pliku

```
# Definiowanie ścieżki do pliku CSV
$csvPath = "C:\LokalizacjaDoPliku\NazwaPliku.csv"

# Pobranie wszystkich połączeń TCP i dodanie informacji o procesie
Get-NetTCPConnection | Sort-Object -Property LocalPort | Select-Object -Property LocalPort, RemoteAddress, RemotePort, State, OwningProcess | ForEach-Object {
    $_ | Add-Member -MemberType NoteProperty -Name ProcessName -Value (Get-Process -Id $_.OwningProcess -ErrorAction SilentlyContinue | Select-Object -ExpandProperty ProcessName)
    $_
} | Export-Csv -Path $csvPath -NoTypeInformation

Write-Host "Zapisano połączenia do pliku CSV: $csvPath"
```

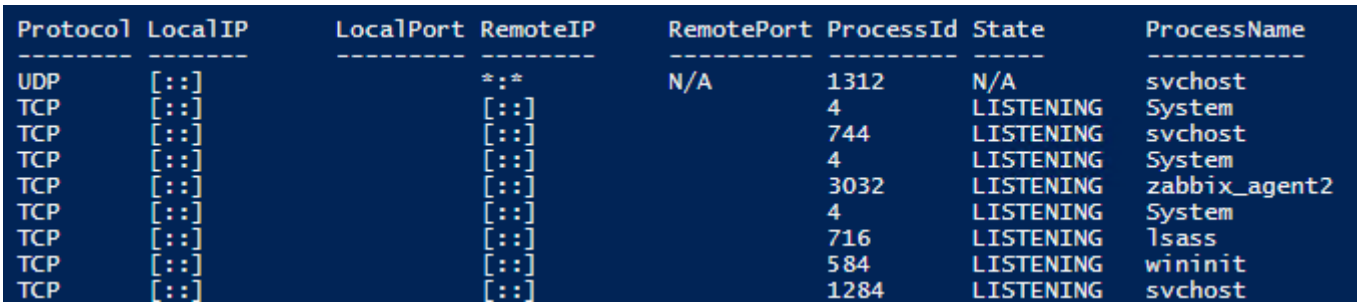
Pobieranie danych za pomocą netstat -ano z parsowaniem wyników w CLI:

```
$netstatOutput = netstat -ano
```

```
$connections = @()
foreach ($line in $netstatOutput) {
    if ($line -match '^\s*(TCP|UDP)') {
        $parts = $line -split '\s+'

        # Przypisanie wartości do odpowiednich zmiennych
        $protocol = $parts[1]
        $localAddress = $parts[2]
        $remoteAddress = $parts[3]
        $processID = ""
        $state = ""

        if ($protocol -eq 'TCP') {
            $state = $parts[4]
            $processID = $parts[5]
        } elseif ($protocol -eq 'UDP') {
            $state = "N/A"
            $processID = $parts[4]
        }

        # Funkcja do dzielenia adresu i portu, zachowująca nawiasy IPv6
        function Split-AddressPort ($address) {
            if ($address -match '^\[(.+)\]:(\d+)$') {
                # IPv6 z nawiasami np. [::1]:443
                return @("[{0}]" -f $matches[1], $matches[2])
            } elseif ($address -match '^(.+):(\d+)$') {
                # IPv4 i inne adresy z portem np. 192.168.1.1:80
                return @($matches[1], $matches[2])
            } else {
                # Adresy bez portów np. *:*
                return @($address, "N/A")
            }
        }

        $localIP, $localPort = Split-AddressPort $localAddress
        $remoteIP, $remotePort = Split-AddressPort $remoteAddress

        # Pobranie nazwy procesu lub ustawienie komunikatu
        $processName = "Unknown"
        if ($processID -eq "0") {
```

```
            $processName = "Process terminated | PID = 0"
        } elseif ($processID -match '^\d+$') {
            try {
                $processName = (Get-Process -Id $processID -ErrorAction SilentlyContinue).Name
            } catch {
                # Proces może już nie istnieć lub być niedostępny
            }
        }

        $connections += [PSCustomObject]@{
            Protocol    = $protocol
            LocalIP     = $localIP
            LocalPort   = $localPort
            RemoteIP    = $remoteIP
            RemotePort  = $remotePort
            ProcessId   = $processID
            State       = $state
            ProcessName = $processName
        }
    }
}

$connections | Sort-Object -Property LocalIP | Format-Table -AutoSize -Property Protocol, LocalIP, LocalPort,
RemoteIP, RemotePort, ProcessId, State, ProcessName
```

Przykładowy wynik:



Protocol	LocalIP	LocalPort	RemoteIP	RemotePort	ProcessId	State	ProcessName
UDP	[::]		*:*	N/A	1312	N/A	svchost
TCP	[::]		[::]		4	LISTENING	System
TCP	[::]		[::]		744	LISTENING	svchost
TCP	[::]		[::]		4	LISTENING	System
TCP	[::]		[::]		3032	LISTENING	zabbix_agent2
TCP	[::]		[::]		4	LISTENING	System
TCP	[::]		[::]		716	LISTENING	lsass
TCP	[::]		[::]		584	LISTENING	wininit
TCP	[::]		[::]		1284	LISTENING	svchost

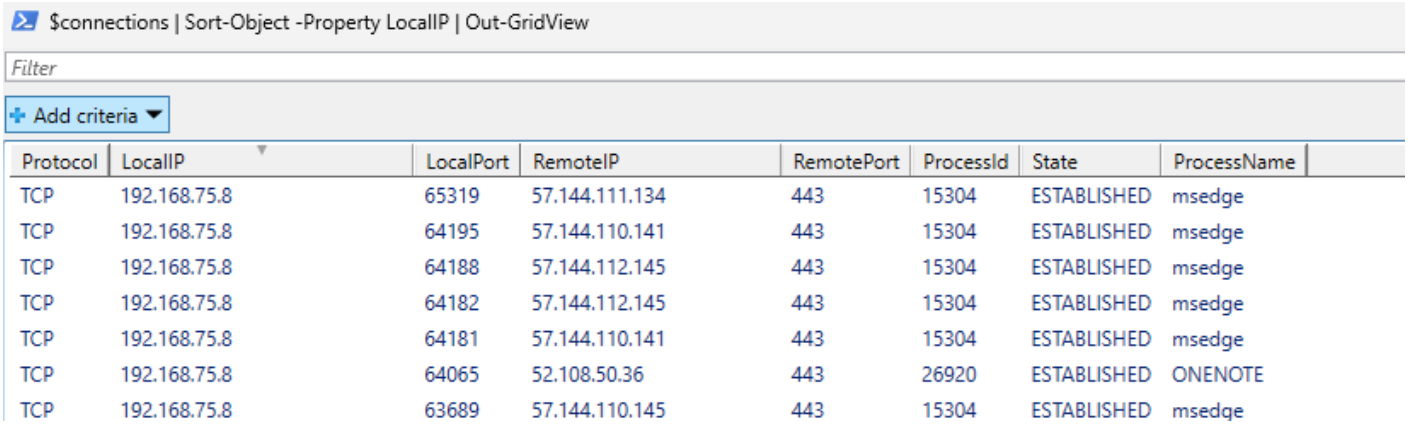
Pobieranie danych za pomocą netstat -ano z interaktywną tabelą

```
$netstatOutput = netstat -ano

$connections = @()
foreach ($line in $netstatOutput) {
    if ($line -match '^\s*(TCP|UDP)') {
```

```
$parts = $line -split '\s+'

# Przypisanie wartości do odpowiednich zmiennych
$protocol = $parts[1]
$localAddress = $parts[2]
$remoteAddress = $parts[3]
$processID = ""
$state = ""

if ($protocol -eq 'TCP') {
    $state = $parts[4]
    $processID = $parts[5]
} elseif ($protocol -eq 'UDP') {
    $state = "N/A"
    $processID = $parts[4]
}

# Funkcja do dzielenia adresu i portu, zachowująca nawiasy IPv6
function Split-AddressPort ($address) {
    if ($address -match '^\[(.+)\]:(\d+)$') {
        # IPv6 z nawiasami np. [::1]:443
        return @("[{0}]" -f $matches[1], $matches[2])
    } elseif ($address -match '^(.+):(\d+)$') {
        # IPv4 i inne adresy z portem np. 192.168.1.1:80
        return @($matches[1], $matches[2])
    } else {
        # Adresy bez portów np. *:*
        return @($address, "N/A")
    }
}

$localIP, $localPort = Split-AddressPort $localAddress
$remoteIP, $remotePort = Split-AddressPort $remoteAddress

# Pobranie nazwy procesu lub ustawienie komunikatu
$processName = "Unknown"
if ($processID -eq "0") {
    $processName = "Process terminated | PID = 0"
} elseif ($processID -match '^\d+$') {
    try {
```

```
            $processName = (Get-Process -Id $processID -ErrorAction SilentlyContinue).Name
        } catch {
            # Proces może już nie istnieć lub być niedostępny
        }
    }

    $connections += [PSCustomObject]@{
        Protocol    = $protocol
        LocalIP     = $localIP
        LocalPort   = $localPort
        RemoteIP    = $remoteIP
        RemotePort  = $remotePort
        ProcessId   = $processID
        State       = $state
        ProcessName = $processName
    }
  }
}

$connections | Sort-Object -Property LocalIP | Out-GridView
```

Przykładowy wynik:



Protocol	LocalIP	LocalPort	RemoteIP	RemotePort	ProcessId	State	ProcessName
TCP	192.168.75.8	65319	57.144.111.134	443	15304	ESTABLISHED	msedge
TCP	192.168.75.8	64195	57.144.110.141	443	15304	ESTABLISHED	msedge
TCP	192.168.75.8	64188	57.144.112.145	443	15304	ESTABLISHED	msedge
TCP	192.168.75.8	64182	57.144.112.145	443	15304	ESTABLISHED	msedge
TCP	192.168.75.8	64181	57.144.110.141	443	15304	ESTABLISHED	msedge
TCP	192.168.75.8	64065	52.108.50.36	443	26920	ESTABLISHED	ONENOTE
TCP	192.168.75.8	63689	57.144.110.145	443	15304	ESTABLISHED	msedge

Skrypt do odrzucenia starych paczek i czyszczenia bazy WSUS

Skrypt służy do automatycznego odrzucania przestarzałych aktualizacji w WSUS oraz uruchamiania procesu czyszczenia WSUS, jeśli dokonano jakichkolwiek zmian.

Wymagania:

- Uprawnienia administracyjne do serwera WSUS.
- Zainstalowany moduł PowerShell dla WSUS.
- Połączenie z serwerem WSUS.

Zakres działania:

Skrypt odrzuca aktualizacje, jeśli spełniają one którykolwiek z poniższych warunków:

- Są zastąpione przez nowsze aktualizacje (superseded).
- Są oznaczone jako wygasłe (expired) według definicji Microsoft.
- Dotyczą systemów operacyjnych x86 lub Itanium.
- Są przeznaczone dla systemu Windows XP.
- Są pakietami językowymi.
- Dotyczą starszych wersji przeglądarki Internet Explorer (wersje 7, 8, 9).
- Są aktualizacjami specyficznymi dla regionów (zawierają nazwy krajów).
- Są aktualizacjami beta.
- Są przeznaczone dla systemów wbudowanych (embedded).

> Jeśli aktualizacja obejmuje wiele systemów operacyjnych i spełnia jeden lub więcej powyższych warunków, wersje aktualizacji, które nie spełniają tych kryteriów, nie zostaną odrzucone przez ten skrypt.

Przykład użycia:

Aby uruchomić skrypt, użyj poniższej komendy w PowerShell:

```
.\Decline-Updates -WSUSServer WSUSServer.Company.com -WSUSPort 8530
```

Gdzie:

- WSUSServer.Company.com - to adres Twojego serwera WSUS
- 8530 - to numer portu WSUS (domyślnie 8530 dla HTTP, 8531 dla HTTPS).

Efekty działania:
- Skrypt analizuje listę dostępnych aktualizacji.
- Odrzuca aktualizacje zgodnie z podanymi kryteriami.
- Jeśli odrzucono jakiekolwiek aktualizacje, skrypt uruchamia czyszczenie WSUS w celu zwolnienia miejsca na serwerze.

Treść skryptu

```
Param(
    [Parameter(Mandatory=$false,
    ValueFromPipeline=$true,
    ValueFromPipelineByPropertyName=$true,
    ValueFromRemainingArguments=$false,
    Position=0)]
    [string]$WSUSServer = "Localhost", #default to localhost
    [int]$WSUSPort=8530,
    [switch]$reportonly
    )


Function Decline-Updates{
    Param(
        [string]$WsusServer,
        [int]$WSUSPort,
        [switch]$ReportOnly
    )
    write-host "Connecting to WSUS Server $WSUSServer and getting list of updates"
    $Wsus = Get-WSUSserver -Name $WSUSServer -PortNumber $WSUSPort
    if($WSUS -eq $Null){
        write-error "unable to contact WSUSServer $WSUSServer"
    }else{
        $Updates = $wsus.GetUpdates()
        write-host "$(($Updates | where {$_.IsDeclined -eq $false} | measure).Count) Updates before cleanup"
        $updatesToDecline = $updates | where {$_.IsDeclined -eq $false -and (
        $_.IsSuperseded -eq $true -or   #remove superseded updates
        $_.PublicationState -eq "Expired" -or #remove updates that have been pulled by Microsoft
        $_.LegacyName -imatch "ia64" -or #remove updates for itanium computers (1/2)
        $_.LegacyName -imatch "x86" -or  #remove updates for 32-bit computers
```

```
        $_.LegacyName -match "XP" -or   #remove Windows XP updates (1/2)
        $_.producttitles -match "XP" -or #remove Windows XP updates (1/2)
        $_.Title -match "Itanium" -or   #remove updates for itanium computers (2/2)
□$_.Title -match "ARM64" -or   #remove updates for itanium computers (2/2)
        ($_.Title -imatch "language" -and $_.Title -notmatch "(PL|ENG)") -or  #remove langauge packs
□$_.Title -match "Sprachpaket" -or
        $_.title -match "Internet Explorer 7" -or #remove updates for old versions of IE
        $_.title -match "Internet Explorer 8" -or
        $_.title -match "Internet Explorer 9" -or
        $_.title -match "Japanese" -or #some non-english updates are not filtered by WSUS language filtering
        $_.title -match "Korean" -or
        $_.title -match "Taiwan" -or
        $_.Title -match "Beta" -or     #Beta products and beta updates
        $_.title -match "Embedded"     #Embedded version of Windows
        )}

        write-host "$(($updatesToDecline | measure).Count) Updates to decline"
        $changemade = $false
        if($reportonly){
           write-host "ReportOnly was set to true, so not making any changes"
        }else{
           $changemade = $true
           $updatesToDecline | %{$_.Decline()}
        }

        #Decline updates released more then 3 months prior to the release of an included service pack
        # - service packs updates don't appear to contain the supersedance information.
        Foreach($SP in $($updates | where title -match "^Windows Server \d{4} .* Service Pack \d")){
           if(($SP.ProductTitles |measure ).count -eq 1){
              $updatesToDecline = $updates | where {$_.IsDeclined -eq $false -and $_.ProductTitles -contains
$SP.ProductTitles -and $_.CreationDate -lt $SP.CreationDate.Addmonths(-3)}
              if($updatesToDecline -ne $null){
                 write-host "$(($updatesToDecline | measure).Count) Updates to decline (superseded by
$($SP.Title))"
                 if(-not $reportonly){
                    $changemade = $true
                    $updatesToDecline | %{$_.Decline()}
                 }
              }
           }
```

```
        }

        #if changes were made, run a WSUS cleanup to recover disk space
        if($changemade -eq $true -and $reportonly -eq $false){
            $Updates = $wsus.GetUpdates()
            write-host "$(($Updates | where {$_.IsDeclined -eq $false} | measure).Count) Updates remaining,
running WSUS cleanup"
            Invoke-WsusServerCleanup -updateServer $WSUS -CleanupObsoleteComputers -
CleanupUnneededContentFiles -CleanupObsoleteUpdates -CompressUpdates -DeclineExpiredUpdates -
DeclineSupersededUpdates
        }

    }
}

Decline-Updates -WSUSServer $WSUSServer -WSUSPort $WSUSPort -reportonly:$reportonly
```

Skrypt dodający aliasy SMTP użytkownikom w parametrach AD

Skrypt wykonujemy z poziomu PowerShell ISE po zmianie odpowiednich parametrów. Skrypt musi być uruchomiony na serwerze z zainstalowanym funkcją RSAT, z dostępem do domeny w której chcemy go uruchomić

```
# Importowanie modułu Active Directory
Import-Module ActiveDirectory

# Definicja zmiennych dla domen
$oldDomain = "staradomena.pl"
$newDomain = "nowadomena.pl"

# Określenie wyróżnionej nazwy (DN) docelowej jednostki organizacyjnej (OU)
$ou = "OU=Użytkownicy_Firma,OU=Kontakty,OU=Organizacja,DC=domena,DC=local"

# Pobranie wszystkich użytkowników (aktywnych i nieaktywnych) w określonym OU wraz z ich UserPrincipalName i ProxyAddresses
$users = Get-ADUser -Filter * -SearchBase $ou -Properties UserPrincipalName, ProxyAddresses, SamAccountName

# Iteracja przez każdego użytkownika (zarówno aktywnego, jak i nieaktywnego) i dodanie aliasu
foreach ($user in $users) {

    # Upewnienie się, że użytkownik ma ustawiony UserPrincipalName
    if ($user.UserPrincipalName -match "@$oldDomain$") {

        # Wyodrębnienie nazwy użytkownika poprzez usunięcie starej domeny z UserPrincipalName
        $username = $user.UserPrincipalName -replace "@$oldDomain", ""

        # Utworzenie aliasu w formacie username@newDomain
```

```
        $alias = "smtp:$username@$newDomain"

        # Pobranie aktualnych ProxyAddresses (aliasów e-mail)
        $proxyAddresses = $user.ProxyAddresses

        # Upewnienie się, że ProxyAddresses jest zainicjalizowane
        if (-not $proxyAddresses) {
            $proxyAddresses = @()
        }

        # Sprawdzenie, czy alias już istnieje
        if ($proxyAddresses -notcontains $alias) {
            try {
                # Dodanie nowego aliasu
                Set-ADUser -Identity $user -Add @{ProxyAddresses=$alias}
                Write-Host "Dodano alias $alias dla użytkownika $username"
            } catch {
                Write-Host "Błąd podczas dodawania aliasu $alias dla użytkownika $username: $_" -ForegroundColor
Red
            }
        } else {
            Write-Host "Alias $alias już istnieje dla użytkownika $username"
        }
    } else {
        Write-Host "Pomijanie użytkownika $($user.SamAccountName), ponieważ nie ma poprawnego
UserPrincipalName kończącego się na @$oldDomain" -ForegroundColor Yellow
    }
}
```

Wyszukiwania zadania w Task Scheduler o odpowiedniej nazwie

> Skrypt zapisuje pliki w aktualnie używanym folderze. Należy przejść w PWSH do katalogu docelowego

> Skrypt uruchamiamy w PowerShell z uprawnieniami Administratora

```
# Definiujemy nazwę zadania do wyszukania
$taskName = 'NazwaTasku'

# Pobierz nazwę hosta i skonstruuj nazwę pliku
$hostname = $env:COMPUTERNAME
$filepath = "tasks_$hostname.txt"

# Wypisz wszystkie zadania wraz z nazwami i ścieżkami, zapisz do pliku
Get-ScheduledTask | Select-Object TaskName, TaskPath | Out-File -FilePath $filepath -Encoding UTF8

# Oddziel sekcje w pliku
Add-Content -Path $filepath "`nWeryfikacja czy istnieje task o określonej nazwie:`n"

# Sprawdź, czy zadanie o podanej nazwie istnieje, dopisz wynik na końcu pliku lub stosowny komunikat
$task = Get-ScheduledTask | Where-Object { $_.TaskName -eq $taskName }
if ($task) {
    $task | Out-File -FilePath $filepath -Append -Encoding UTF8
} else {
    Add-Content -Path $filepath "Brak zadania o nazwie $taskName"
}
```

Wyszukiwanie frazy we wszystkich GPO.

Skrypt tworzy katalog gdzie eksportuje pliki xml GPO po czym przeszukuje w nich ustawioną frazę. Katalog oraz pliki utworzą się w aktywnym katalogu PWSH

Skrypt uruchamiamy w PWSH z uprawnieniami administratora, na kontrolerze domeny

```
# Definiujemy wzorzec wyszukiwania na samej górze
$searchPattern = "SvcUpddate"

Write-Host "`nTworzę folder na raporty GPO w bieżącym katalogu..."

# Próba pobrania ścieżki skryptu, jeśli brak to używamy bieżącego katalogu
if ($MyInvocation.MyCommand.Path) {
    $scriptFolder = Split-Path -Parent $MyInvocation.MyCommand.Path
} else {
    Write-Host "Nie wykryto ścieżki skryptu, używam bieżącego katalogu."
    $scriptFolder = (Get-Location).Path
}

$folder = Join-Path $scriptFolder "GPOReports"
New-Item -Path $folder -ItemType Directory -Force | Out-Null

Write-Host "Pobieram wszystkie obiekty GPO z domeny..."
$allgpos = Get-GPO -All

# Plik do zapisania raportu z listą i wynikiem w folderze skryptu
$reportFile = Join-Path $scriptFolder "GPO_export_and_scan_report.txt"

# Czyszczenie pliku jeśli istnieje
if (Test-Path $reportFile) {
    Remove-Item $reportFile
}
```

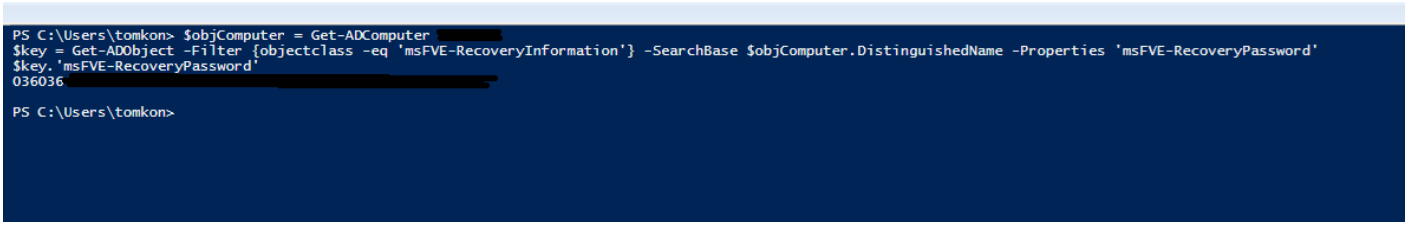
```
Write-Host "Eksportuję wszystkie GPO do osobnych plików XML oraz zapisuję listę do pliku raportu..."
Add-Content -Path $reportFile "Lista wyeksportowanych GPO:`n"
$allgpos | ForEach-Object {
    $path = Join-Path $folder ("{0}_{1}.xml" -f $_.DisplayName, $_.Id)
    Write-Host ("Eksportuję GPO: {0}" -f $_.DisplayName)
    Get-GPOReport -Guid $_.Id -ReportType Xml -Path $path
    Add-Content -Path $reportFile $_.DisplayName
}

# Dodaj pustą linię do rozdzielenia sekcji
Add-Content -Path $reportFile "`nWyniki wyszukiwania wzorca '$searchPattern':`n"

Write-Host "Szukam wzorca '$searchPattern' we wszystkich wyeksportowanych raportach..."
$patternMatches = Select-String -Path (Join-Path $folder "*.xml") -Pattern $searchPattern
if ($patternMatches -and $patternMatches.Count -gt 0) {
    $patternMatches | ForEach-Object { Add-Content -Path $reportFile $_.ToString() }
} else {
    Add-Content -Path $reportFile "Brak wzorca '$searchPattern' w żadnym z raportów GPO."
}

Write-Host "`nGotowe! Pełny raport w pliku: $reportFile"
```

Pobieranie klucza BitLocker z AD

> Należy uruchomić powershell jako user z uprawnieniami do odczytu atrybutów

```
$objComputer = Get-ADComputer NAZWAKOMPUTERA
$Bitlocker_Objects = Get-ADObject -Filter {objectclass -eq 'msFVE-RecoveryInformation'} -SearchBase
$objComputer.DistinguishedName -Properties 'msFVE-RecoveryPassword'
$Bitlocker_Objects
```

Wynik:



> Bardziej skomplikowana wersja - z pytaniem o nazwę komputera oraz poświadczenia domenowe:

```
# Pobieranie nazwy komputera
$ComputerName = Read-Host "Nazwa komputera [$env:COMPUTERNAME]"
if ([string]::IsNullOrWhiteSpace($ComputerName)) {
    $ComputerName = $env:COMPUTERNAME
}

# Pobieranie poświadczeń domenowych
$Credentials = Get-Credential -Message "Poświadczenia do domeny AD"

# Wyszukanie odpowiedniego komputera
$objComputer = Get-ADComputer $ComputerName -Credential $Credentials

# Pobranie kluczy BitLocker z dodatkowymi polami
$Bitlocker_Object = Get-ADObject `
```

```
    -Filter {objectclass -eq 'msFVE-RecoveryInformation'} `
    -SearchBase $objComputer.DistinguishedName `
    -Properties 'msFVE-RecoveryPassword','msFVE-RecoveryGuid','msFVE-VolumeGuid','whenCreated' `
    -Credential $Credentials |
    Sort-Object whenCreated -Descending |
    Select-Object `
        @{Name='CN'; Expression={ $ComputerName }},
        @{Name='Bitlocker Password'; Expression={ $_.'msFVE-RecoveryPassword' }},
        @{Name='Volume GUID'; Expression={ $_.'msFVE-VolumeGuid' }},
        @{Name='Recovery GUID'; Expression={ $_.'msFVE-RecoveryGuid' }},
        @{Name='Created'; Expression={ $_.whenCreated }}

# Wypisanie danych
$Bitlocker_Object | Format-Table -AutoSize
```

Wynik: