Stosowanie zasady bezpiecznych haseł

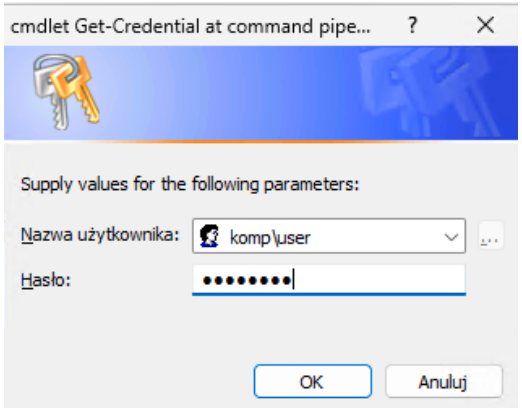
Przygotowanie

Enable-PSRemoting -Force -SkipNetworkProfileCheck

A. Używanie haseł w PowerShell
1. Poświadczenia PS

PowerShell sprawił, że określanie i używanie poświadczeń za pomocą polecenia Get-Credential jest niezwykle łatwe i bezpieczne.
Jeśli musisz określić alternatywne poświadczenia, jest to rozwiązanie:

```
$credential=Get-Credential
```



Polecenie cmdlet Get-Credential zwraca obiekt PSCredential.

Istnieje metoda dla tego typu obiektu, która pozwoli wyodrębnić zapisane hasło w postaci zwykłego tekstu.

```
$credential=Get-Credential
$credential.GetNetworkCredential()
$credential.GetNetworkCredential() | FL
```

```
PS C:\Windows\system32> $credential=Get-Credential
cmdlet Get-Credential at command pipeline position 1
Supply values for the following parameters:

PS C:\Windows\system32> $credential.GetNetworkCredential()

UserName                                               Domain
--------                                               ------
user                                                   komp


PS C:\Windows\system32> $credential.GetNetworkCredential() | FL


UserName       : user
Password       : zaq1@WSX
SecurePassword : System.Security.SecureString
Domain         : komp
```

Jeśli potrzebujesz szybko zweryfikować swoje dane uwierzytelniające, byłoby to przydatne.

Możesz zbudować obiekt PSCredential bez użycia polecenia cmdlet Get-Credential w następujący sposób:

```
$pass=ConvertTo-SecureString 'P@ssw0rd1' -AsPlainText -Force
$cred=New-Object -TypeName PSCredential -ArgumentList @('komp\Administrator',$pass)
```

```
PS C:\Windows\system32> $pass=ConvertTo-SecureString 'P@ssw0rd1' -AsPlainText -Force
$cred=New-Object -TypeName PSCredential -ArgumentList @('komp\Administrator',$pass)
```

Nie jest to świetny sposób na zrobienie tego pod względem bezpieczeństwa, ale w środowisku deweloperskim może to być przydatne podczas automatyzacji zadań.

2. Praca z hasłami (zapisz w zeszycie notatkę z tego punktu)

Czasami nie ma potrzeby używania pełnego obiektu PSCredential i wystarczy hasło.

Ogólnie rzecz biorąc, podczas określania hasła w PowerShell parametry akceptują tylko bezpieczne ciągi (to dobrze).
Konwersja hasła na ten typ danych może być żmudna, ale jest to niezbędny krok ze względów bezpieczeństwa.
Oto 2 najczęstsze sposoby tworzenia bezpiecznych ciągów w PowerShell.

Obie metody mają swoje plusy i minusy.

Podejście do odczytu hosta

Plusy:

a. Łatwe do zapamiętania
b. Hasło nigdy nie pojawia się w historii poleceń w postaci zwykłego tekstu

Minusy:

a. Wymaga danych wejściowych użytkownika po uruchomieniu
b. Nie można sprawdzić, czy wystąpiła literówka

Podejście ConvertTo-SecureString

Plusy:

a. Może być użyty w skrypcie bez udziału użytkownika
b. Łatwo zweryfikuj, czy wpisano poprawnie

Minus:

a. Hasło jest w postaci zwykłego tekstu w historii poleceń

3. Czas na funkcje

Metoda Read-Host przechwytywania haseł, jest bezpieczniejsza niż konwersja ze zwykłego tekstu, ale powoduje, że jest możliwość podania błędnych haseł i przypadkowego blokowania kont.

Jest sposób na przeprowadzenie zarówno weryfikacji hasła, jak i niezapisywania hasła w postaci zwykłego tekstu.

```
$secure=Read-Host -AsSecureString -Prompt Password
$bstr=[Runtime.InteropServices.Marshal]::SecureStringToBSTR($secure)
$plain=[Runtime.InteropServices.Marshal]::PtrToStringAuto($bstr)
Write-Output $plain
```

```
PS C:\Windows\system32> $secure=Read-Host -AsSecureString -Prompt Password

PS C:\Windows\system32> $bstr=[Runtime.InteropServices.Marshal]::SecureStringToBSTR($secure)

PS C:\Windows\system32> $plain=[Runtime.InteropServices.Marshal]::PtrToStringAuto($bstr)

PS C:\Windows\system32> Write-Output $plain
zaq1
```

Dzięki temu podręcznemu zestawowi poleceń możemy mieć ciastko i je zjeść.

Możemy to wykorzystać, aby dwukrotnie poprosić o hasło, a następnie sprawdzić, czy hasła pasują, zanim przejdziemy dalej.

```
$pass=Read-Host -Prompt 'Enter a Password' -AsSecureString
$pass2=Read-Host -Prompt 'Re-type Password' -AsSecureString
$bstr=[Runtime.InteropServices.Marshal]::SecureStringToBSTR($pass)
$plain=[Runtime.InteropServices.Marshal]::PtrToStringAuto($bstr)
[Runtime.InteropServices.Marshal]::ZeroFreeBSTR($bstr)
$bstr2=[Runtime.InteropServices.Marshal]::SecureStringToBSTR($pass2)
$plain2=[Runtime.InteropServices.Marshal]::PtrToStringAuto($bstr2)
[Runtime.InteropServices.Marshal]::ZeroFreeBSTR($bstr2)
if($plain -cne $plain2){
  Write-Error -Message "Passwords don't match...Try again."
}else{
  Write-Output $pass
}
```

```
PS C:\Windows\system32> $pass=Read-Host -Prompt 'Enter a Password' -AsSecureString

PS C:\Windows\system32> $pass2=Read-Host -Prompt 'Re-type Password' -AsSecureString

PS C:\Windows\system32> $bstr=[Runtime.InteropServices.Marshal]::SecureStringToBSTR($pass)

PS C:\Windows\system32> $plain=[Runtime.InteropServices.Marshal]::PtrToStringAuto($bstr)

PS C:\Windows\system32> [Runtime.InteropServices.Marshal]::ZeroFreeBSTR($bstr)

PS C:\Windows\system32> $bstr2=[Runtime.InteropServices.Marshal]::SecureStringToBSTR($pass2)

PS C:\Windows\system32> $plain2=[Runtime.InteropServices.Marshal]::PtrToStringAuto($bstr2)

PS C:\Windows\system32> [Runtime.InteropServices.Marshal]::ZeroFreeBSTR($bstr2)

PS C:\Windows\system32> if($plain -cne $plain2){
  Write-Error -Message "Passwords don't match...Try again."
}else{
  Write-Output $pass
}
System.Security.SecureString

PS C:\Windows\system32> |
```

Teraz, aby przekształcić to w funkcję, możemy zrobić coś takiego:

```
function Read-Password {
  $pass=Read-Host -Prompt 'Enter a Password' -AsSecureString
  $pass2=Read-Host -Prompt 'Re-type Password' -AsSecureString
  $bstr=[Runtime.InteropServices.Marshal]::SecureStringToBSTR($pass)
  $plain=[Runtime.InteropServices.Marshal]::PtrToStringAuto($bstr)
  [Runtime.InteropServices.Marshal]::ZeroFreeBSTR($bstr)
  $bstr2=[Runtime.InteropServices.Marshal]::SecureStringToBSTR($pass2)
  $plain2=[Runtime.InteropServices.Marshal]::PtrToStringAuto($bstr2)
  [Runtime.InteropServices.Marshal]::ZeroFreeBSTR($bstr2)
  [bool]$pValid=$true
```

```
  $builder=New-Object -TypeName Text.StringBuilder
  if ($plain -cne $plain2){
    $pValid=$false
    [void]$builder.Append('Passwords do not match. ')
  }
  if ($plain.Length -lt 8){
    $pValid=$false
    [void]$builder.Append('Password too short. ')
  }
  if($plain.Length -gt 127){
    $pValid=$false
    [void]$builder.Append('Password too long. ')
  }
  if($plain.ToLower() -ceq  $plain){
    $pValid=$false
    [void]$builder.Append('Must have an upper case letter. ')
  }
  if($plain.ToUpper() -ceq $plain){
    $pValid=$false
    [void]$builder.Append('Must have a lower case letter. ')
  }
```

```
  if($plain -notmatch '[\W\d]'){
    $pValid=$false
    [void]$builder.Append('Must contain a special or numeric character. ')
  }
  if($pValid -eq $false){
    Write-Error -Message $builder.ToString()
  }else{
    Write-Output $pass
  }
}
```

Uruchamiam funkcje

```
Read-Password
```

Podaje hasło 12345678

```
PS C:\Windows\system32> Read-Password
Read-Password : Passwords do not match. Must have an upper case letter. Must have a lower case letter.
At line:1 char:1
+ Read-Password
+ ~~~~~~~~~~~~~
    + CategoryInfo          : NotSpecified: (:) [Write-Error], WriteErrorException
    + FullyQualifiedErrorId : Microsoft.PowerShell.Commands.WriteErrorException,Read-Password
```

Uruchamiam funkcje

```
Read-Password
```

Podaje hasło zaq1@WSX

```
PS C:\Windows\system32> Read-Password
System.Security.SecureString
```

Funkcja ma na stałe zakodowane podstawowe wymagania dotyczące haseł. Te zakodowane na sztywno ograniczenia powinny działać z większością domyślnych zasad haseł. Niektóre środowiska mogą być bardziej rygorystyczne.

Wymagania dotyczące hasła są następujące:

8 znaków długości lub więcej (mniej niż 127 znaków)

Musi zawierać wielką literę

Musi zawierać małą literę

Musi zawierać cyfrę lub znak specjalny

Jeśli którykolwiek z tych warunków nie zostanie spełniony, na ekranie zostanie wyświetlony błąd, a komunikat wskaże wszystkie warunki, które nie zostały spełnione (w tym, jeśli nie są zgodne).

B. Bezpieczne zapisywanie poświadczeń za pomocą PowerShell

Czasami chcesz bezpiecznie zapisywać i pobierać informacje w PowerShell. Zapisywanie takich rzeczy jak hasła i inne dane uwierzytelniające w postaci zwykłego tekstu nie jest wcale dobrym pomysłem. Aby tego uniknąć, możesz użyć funkcji bezpiecznego ciągu programu PowerShell. Najczęstszym sposobem na to jest polecenie:

$Secure = Read-Host –AsSecureString

Tworzy to bezpieczny ciąg. Po wprowadzeniu polecenia wszystkie wpisywane znaki są konwertowane na bezpieczny ciąg, a następnie zapisywane w zmiennej $secure . Dzięki temu poleceniu wprowadzone znaki nie są wyświetlane na ekranie.

```
PS C:\Windows\system32> $Secure = Read-Host -AsSecureString

PS C:\Windows\system32> $Secure
System.Security.SecureString
```

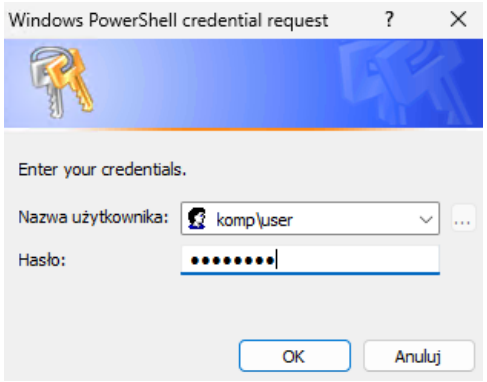
Ponieważ zmienna $secure zawiera bezpieczny ciąg, program PowerShell wyświetla tylko tekst System.Security.SecureString podczas próby jego wyświetlenia. Informacje, które mają być zabezpieczone, są teraz zapisywane jako chroniona zmienna o nazwie $secure w PowerShell.
Jak można to bezpiecznie zapisać w pliku, aby można go było użyć ponownie później i nadal zachować ochronę, nawet na dysku?

Można użyć polecenia Export-Clixml, zastosowaniem tego na komputerach z systemem Windows jest bezpieczne eksportowanie poświadczeń i zabezpieczanie ciągów jako XML.

Lepszym sposobem na bezpieczne przechowanie wartości, którą chce zabezpieczyć, jest:

$Secure = get-credential -credential komp\user

```
PS C:\Windows\system32> $Secure = get-credential -credential komp\user
```



Który poprosi Cię o informacje, jak pokazano powyżej. Można zauważyć, że nazwa użytkownika został złożona dzięki parametrowi -Credential.

W ten sposób otrzymasz zmienną z nazwą użytkownika (tutaj user) i bezpiecznym ciągiem, który jest poświadczeniem PowerShell.

Następnie zapisz informacje poprzez:

$Secure | Export-CliXml -Path .\kompuser.xml

```
PS C:\Windows\system32> $Secure | Export-CliXml -Path .\kompuser.xml
```

lub/i zapisując plik na komputerze w sieci

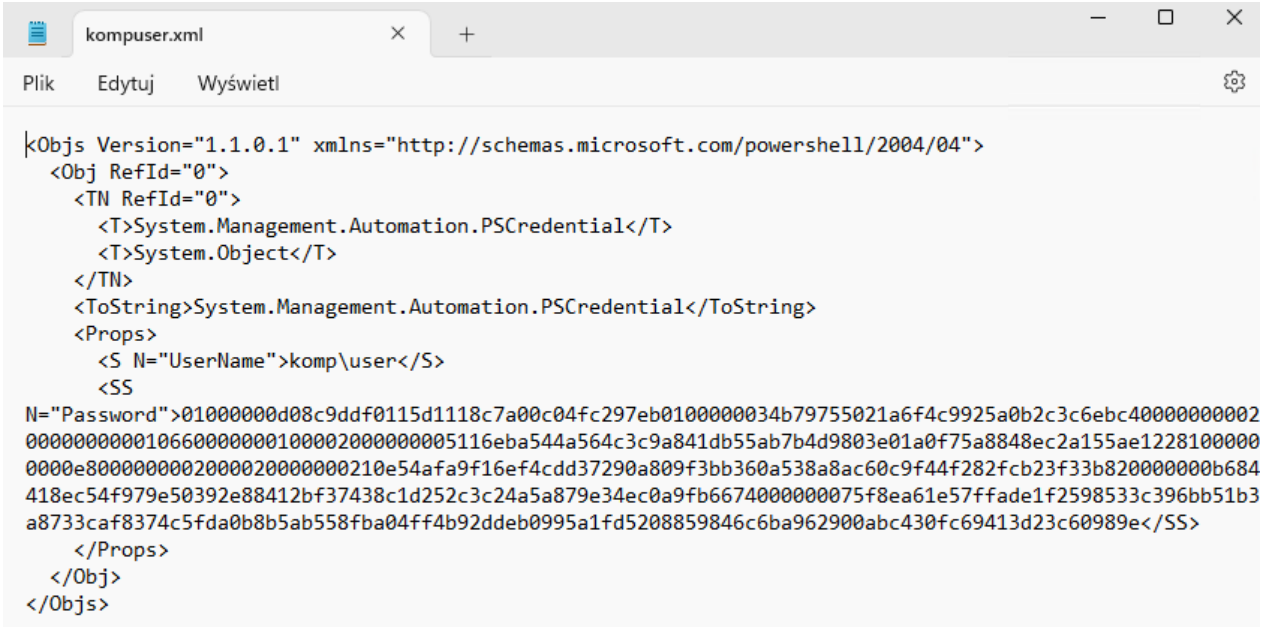
$Secure | Export-CliXml -Path \\$env:COMPUTERNAME\C$\kompuser.xml

```
PS C:\Windows\system32> $Secure | Export-CliXml -Path \\$env:COMPUTERNAME\C$\kompuser.xml
```

Otwórz plik XML w Notatniku

Invoke-Item \\$env:COMPUTERNAME\C$\kompuser.xml

```
PS C:\Windows\system32> Invoke-Item "\\$env:COMPUTERNAME\C$\kompuser.xml"
```



Jeśli Export-Clixml jest używany do zapisania tej zmiennej do pliku (tu clientid.xml), zapisze ją tak, jak pokazano powyżej. Zauważysz, że pole Hasło jest zaszyfrowane. To tutaj przechowywane są bezpieczne informacje, co jest świetne, ponieważ są teraz szyfrowane na dysku.

Kolejna rzecz w używaniu Export-Clixml to:

Export-Clixml cmdlet szyfruje poświadczeń obiektów za pomocą API ochrony danych systemu Windows. Szyfrowanie zapewnia, że tylko konto użytkownika na tym komputerze może odszyfrować zawartość obiektu poświadczeń. Wyeksportowanego pliku CLIXML nie można używać na innym komputerze ani przez innego użytkownika.

C. Bezpieczne załadowanie poświadczeń z pliku z powrotem do zmiennej.

1. Aby przeprowadzić próbę załadowania poświadczenia z pliku z powrotem do zmiennej:

Poniższe czynności wykonaj na komputerze, gdzie dane były szyfrowane z innego konta, na którym dane nie były szyfrowane (np. Administrator)

a. ładując plik z komputera lokalnie

```
PS C:\Windows\system32> $Secure | Import-CliXml -Path .\kompuser.xml
Import-CliXml : The input object cannot be bound to any parameters for the command either because the command does not take pipeline input or the i
nput and its properties do not match any of the parameters that take pipeline input.
At line:1 char:11
+ $Secure | Import-CliXml -Path .\kompuser.xml
+           ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : InvalidArgument: (System.Manageme...on.PSCredential:PSObject) [Import-Clixml], ParameterBindingException
    + FullyQualifiedErrorId : InputObjectNotBound,Microsoft.PowerShell.Commands.ImportClixmlCommand
```

b. ładując plik z komputera w sieci

```
PS C:\Windows\system32> $Secure | Import-CliXml -Path \\$env:COMPUTERNAME\C$\kompuser.xml
Import-CliXml : The input object cannot be bound to any parameters for the command either because the command does not take pipeline input or the i
nput and its properties do not match any of the parameters that take pipeline input.
At line:1 char:11
+ $Secure | Import-CliXml -Path \\$env:COMPUTERNAME\C$\kompuser.xml
+           ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : InvalidArgument: (System.Manageme...on.PSCredential:PSObject) [Import-Clixml], ParameterBindingException
    + FullyQualifiedErrorId : InputObjectNotBound,Microsoft.PowerShell.Commands.ImportClixmlCommand
```

Jeśli plik z zapisanymi i zaszyfrowanymi informacjami zostanie skopiowany i użyty przez kolejne logowanie na tym samym lub innym komputerze, otrzymasz powyższy wynik. Nie można go odszyfrować. Nie jest to idealne, ale oznacza to, że po zapisaniu informacji przy użyciu powyższej techniki jedynym sposobem na ich odszyfrowanie jest to samo logowanie na tym samym komputerze.

2. Aby załadować poświadczenia z pliku z powrotem do zmiennej:

Poniższe czynności wykonaj na komputerze, gdzie dane były szyfrowane z konta, na którym były szyfrowane (admin)

a. ładując plik z komputera lokalnie

```
PS C:\Windows\system32> $Secure = Import-Clixml -Path ".\kompuser.xml"

PS C:\Windows\system32> |
```

lub

```
PS C:\Windows\system32> $Secure

UserName                      Password
--------                      --------
komp\user System.Security.SecureString
```

b. ładując plik z komputera w sieci

```
PS C:\Windows\system32> $Secure = Import-CliXml -Path "\\$env:COMPUTERNAME\C$\kompuser.xml"

PS C:\Windows\system32> $Secure

UserName                       Password
--------                       --------
komp\user System.Security.SecureString
```

Oznacza to, że nie musisz mieć bezpiecznych zmiennych zapisanych jako zwykły tekst w skryptach lub w niezabezpieczonych plikach na dysku, które można skopiować i pracować w dowolnym miejscu.

Dzięki tej technice możesz zapewnić, że informacje zapisane w pliku są zaszyfrowane i nie mogą być używane przez żadnego innego użytkownika ani przez żadną inną maszynę. W związku z tym, jeśli ktoś zdobędzie plik, informacje nie będą mogły zostać wyświetlone ani odszyfrowane, a zatem dostęp zostanie odmówiony.

D. Bezpieczne zapisywanie wielu poświadczeń za pomocą PowerShell

Załóż konto poleceniem:

```
net user User zaq1@WSX /add
```

Dla konta Administrator ustaw hasło:

```
net user Administrator zaq1@WSX
```

Nie jesteś ograniczony do przechowywania jednego zestawu poświadczeń w ten sposób, możesz użyć dowolnej liczby kont.

Możesz użyć dowolnej liczby kont, na przykład poniższy przykład wyświetli monit o 3 różne zestawy i przechowa je w tabeli skrótów. Można go następnie wyeksportować/zaimportować w podobny sposób: Sprawdź:

a) dla kont nie istniejących w systemie

```
$Hash = @{
   'Admin'      = Get-Credential -Message 'Please enter administrative credentials'
   'Zdalny' = Get-Credential -Message 'Please enter remote user credentials'
```

'User' = Get-Credential -Message 'Please enter user credentials'

}

$Hash | Export-Clixml -Path "${env:\userprofile}\Hash.Cred"

$Hash = Import-CliXml -Path "${env:\userprofile}\Hash.Cred"

$Hash.Admin

$Hash.Zdalny

$Hash.User

```
$Hash = @{
    'Admin'  = Get-Credential -Message 'Please enter administrative credentials'
    'Zdalny' = Get-Credential -Message 'Please enter remote user credentials'
    'User'   = Get-Credential -Message 'Please enter user credentials'
}
$Hash | Export-Clixml -Path "${env:\userprofile}\Hash.Cred"
$Hash = Import-CliXml -Path "${env:\userprofile}\Hash.Cred"
$Hash.Admin
$Hash.Zdalny
$Hash.User
```

```
PS C:\WINDOWS\system32> $Hash = @{
    'Admin'  = Get-Credential -Message 'Please enter administrative credentials'
    'Zdalny' = Get-Credential -Message 'Please enter remote user credentials'
    'User'   = Get-Credential -Message 'Please enter user credentials'
}
$Hash | Export-Clixml -Path "${env:\userprofile}\Hash.Cred"
$Hash = Import-CliXml -Path "${env:\userprofile}\Hash.Cred"
$Hash.Admin
$Hash.Zdalny
$Hash.User


UserName                     Password
--------                     --------
Admin    System.Security.SecureString
Zdalny   System.Security.SecureString
User     System.Security.SecureString
```

b) dla kont istniejących w systemie

```
$Hash = @{
    'Admin'      = Get-Credential -Message 'Please enter admin credentials'
    'Administrator' = Get-Credential -Message 'Please enter administrative credentials'
    'User'       = Get-Credential -Message 'Please enter user credentials'
}
$Hash | Export-Clixml -Path "${env:\userprofile}\Hash.Cred"
$Hash = Import-CliXml -Path "${env:\userprofile}\Hash.Cred"
Invoke-Command -ComputerName "$env:COMPUTERNAME" -Credential $Hash.Admin -ScriptBlock {whoami}
Invoke-Command -ComputerName "$env:COMPUTERNAME" -Credential $Hash.Administrator -ScriptBlock {whoami}
Invoke-Command -ComputerName "$env:COMPUTERNAME" -Credential $Hash.User -ScriptBlock {whoami}
```