34 Stosowanie urządzeń i narzędzia zabezpieczające dostęp

Opracuj wykaz kolejnych czynności, z zastosowaniem urządzeń i narzędzia zabezpieczających dostęp w PowerShell tak aby zabezpieczyć dostęp do urządzenia PC.

Wykorzystaj do tego celu skrypty znajdujące się w folderze z tym zadaniem:

Zadanie 1

ReverseMarshal.ps1 - przyjmuje obiekt PSCredential, określa, czy wewnętrzne poświadczenie jest nazwą użytkownika/hasłem, czy kartą inteligentną. Jeśli jest to karta inteligentna, rozpakowujemy poświadczenia i lokalizujemy poprawny certyfikat ze sklepu CurrentUser.

a) Zapoznaj się z działaniem tego skryptu PowerShell służącego do zarządzania poświadczeniami karty inteligentnej. Oto krok po kroku jego działanie:

1. Definicja kodu źródłowego C#: Początek skryptu definiuje kod C# zawierający klasy i metody potrzebne do zarządzania poświadczeniami karty inteligentnej. Kluczowe elementy to:

- **SmartcardCredManager**: Klasa statyczna zawierająca metody do zarządzania poświadczeniami.
- **CERT_CREDENTIAL_INFO**: Struktura reprezentująca informacje o poświadczeniu certyfikatu.
- **CredUnmarshalCredential**: Metoda do deserializacji poświadczenia z ciągu znaków.

Deserializacja odnosi się do procesu konwersji danych w formacie, który jest przeznaczony do transportu lub przechowywania (na przykład jako ciąg znaków lub binarny strumień danych), z powrotem do ich pierwotnej postaci obiektowej. Proces ten jest szczególnie istotny w kontekście komunikacji międzyprogramowej lub przechowywania danych w bazach danych lub plikach.

W przypadku tego skryptu PowerShell, deserializacja jest wykonywana na poświadczeniach karty inteligentnej. Poświadczenia te są pierwotnie w formie obiektów w pamięci systemu operacyjnego lub jako dane binarne, a następnie są zamieniane na ciąg znaków za pomocą metody **Marshal.StringToHGlobalUni**, która konwertuje obiekt do reprezentacji ciągu znaków Unicode. Następnie ten ciąg znaków jest przekazywany do metody **CredUnmarshalCredential**, która deserializuje go z powrotem do postaci struktury danych **CERT_CREDENTIAL_INFO**.
Ta struktura zawiera informacje o poświadczeniu, takie jak skrót certyfikatu.
Deserializacja pozwala aplikacji na odczytanie i wykorzystanie danych zawartych w poświadczeniach w bardziej zrozumiały sposób.

- **IsCertificateCredential**: Metoda sprawdzająca, czy poświadczenie jest poświadczeniem certyfikatu.
- **ReverseMarshal**: Metoda do odwrotnego przetwarzania poświadczenia na ciąg znaków szesnastkowych reprezentujących skrót certyfikatu.

2. Dodanie typów: Skrypt dodaje typy z zestawów System, System.Runtime.InteropServices, System.Security i System.Management.Automation. Jest to potrzebne do kompilacji kodu źródłowego C#.

Dodanie typów z zestawów System, System.Runtime.InteropServices, System.Security i System.Management.Automation oznacza, że skrypt PowerShell importuje biblioteki zawierające zdefiniowane typy (klasy, interfejsy, struktury, etc.) z tych zestawów do bieżącego kontekstu skryptu. Oto krótkie objaśnienie każdego z tych zestawów:

1. **System**: Zestaw System zawiera podstawowe typy i funkcje dotyczące obsługi podstawowych operacji systemowych, takich jak zarządzanie pamięcią, obsługa wyjątków, operacje wejścia/wyjścia itp.

2. **System.Runtime.InteropServices**: Ten zestaw zawiera typy i funkcje, które umożliwiają interakcję z kodem zarządzanym i niezarządzanym. Umożliwia to na przykład wywoływanie funkcji z bibliotek DLL napisanych w językach takich jak C++ z poziomu kodu zarządzanego (np. kodu C# lub PowerShell).

3. **System.Security**: Zestaw ten zawiera typy i funkcje związane z mechanizmami bezpieczeństwa w platformie .NET. Obejmuje to m.in. kryptografię, zarządzanie certyfikatami, kontrole dostępu itp.

4. **System.Management.Automation**: Ten zestaw jest specyficzny dla PowerShell i zawiera typy i funkcje związane z obsługą poleceń PowerShell, zarządzaniem sesją, analizą skryptów itp.

Dodanie tych zestawów umożliwia wykorzystanie funkcji i typów zdefiniowanych w nich w obrębie skryptu PowerShell, co zwiększa jego funkcjonalność i umożliwia wykonywanie bardziej zaawansowanych operacji, takich jak interakcja z systemem operacyjnym, zarządzanie bezpieczeństwem, czy manipulacja danymi.

3. Pobranie poświadczenia: Skrypt wywołuje Get-Credential, aby zebrać poświadczenie od użytkownika. Jest to standardowy sposób w PowerShellu na pobranie poświadczeń.

4. Sprawdzenie typu poświadczenia: Wywoływana jest metoda **IsCertificateCredential** z klasy **SmartcardCredManager**, aby sprawdzić, czy poświadczenie jest poświadczeniem certyfikatu.

5. Jeśli poświadczenie jest poświadczeniem certyfikatu:

- Skrypt wywołuje **ReverseMarshal** z klasy **SmartcardCredManager**, aby przekształcić nazwę użytkownika (zawierającą skrót certyfikatu) na skrót certyfikatu w postaci ciągu szesnastkowego.

- Następnie skrypt używa **Get-ChildItem** w celu znalezienia certyfikatu w magazynie certyfikatów użytkownika na podstawie jego skrótu.

- Jeśli certyfikat zostanie odnaleziony, jego szczegóły (takie jak nazwa podmiotu i rozszerzenia) są wyświetlane w konsoli.

6. Jeśli poświadczenie nie jest poświadczeniem certyfikatu:

- Skrypt wyświetla komunikat informujący użytkownika, że to poświadczenie nie jest certyfikatem.

- Można dodać odpowiednie instrukcje, aby obsłużyć poświadczenia typu nazwa użytkownika/hasło.

Definicja kodu źródłowego C#: Skrypt zawiera definicję klasy SmartcardCredManager w języku C#, która zarządza poświadczeniami karty inteligentnej.

```
[string] $SourceCode = @"

using System;

using System.Runtime.InteropServices;

using System.Security;

using System.Management.Automation;

namespace Bongiovi.SmartcardLogon

{

    public static class SmartcardCredManager

    {

        #region Enumeracje i stałe

        // Definicja enumeracji CRED_MARSHAL_TYPE określającej typ poświadczenia

        public enum CRED_MARSHAL_TYPE

        {

            CertCredential = 1,    // Poświadczenie certyfikatu

            UsernameTargetCredential    // Poświadczenie nazwy użytkownika

        }

        // Stała określająca długość skrótu certyfikatu

        public const int CERT_HASH_LENGTH = 20;

        #endregion

        #region Struktury i importy funkcji zewnętrznych

        // Definicja struktury CERT_CREDENTIAL_INFO przechowującej dane o certyfikacie

        [StructLayout(LayoutKind.Sequential)]

        internal struct CERT_CREDENTIAL_INFO

        {

            public uint cbSize;    // Rozmiar struktury
```

```
            [MarshalAs(UnmanagedType.ByValArray, SizeConst = 20)]
            public byte[] rgbHashOfCert;    // Skrót certyfikatu
        }
        // Import funkcji zewnętrznej CredUnmarshalCredential z advapi32.dll, która deserializuje poświadczenie
        [DllImport("advapi32.dll", CharSet = CharSet.Unicode, SetLastError = true)]
        public static extern bool CredUnmarshalCredential(
            IntPtr MarshaledCredential,
            out CRED_MARSHAL_TYPE CredType,
            out IntPtr Credential
        );
        // Import funkcji zewnętrznej CredFree z advapi32.dll, która zwalnia zaalokowaną pamięć dla poświadczenia
        [DllImport("advapi32.dll", SetLastError = true)]
        public static extern bool CredFree([In] IntPtr buffer);
        #endregion
        #region Metody
        // Metoda ReverseMarshal dokonuje odwrotnej deserializacji ciągu znaków na skrót certyfikatu
        public static string ReverseMarshal(string data)
        {
            IntPtr credData = IntPtr.Zero;    // Uchwyt na dane poświadczenia
            IntPtr credInfo = IntPtr.Zero;    // Uchwyt na informacje o poświadczeniu
            CRED_MARSHAL_TYPE credType = 0;   // Typ poświadczenia
            try
            {
                // Przekształcenie ciągu znaków w obiekcie data na ciąg znaków Unicode i przypisanie go do credData
                credData = Marshal.StringToHGlobalUni(data);
```

```
            // Deserializacja poświadczenia
            bool success = CredUnmarshalCredential(credData, out credType, out credInfo);
            if (success)
            {
                // Konwersja struktury credInfo na CERT_CREDENTIAL_INFO
                CERT_CREDENTIAL_INFO certStruct =
(CERT_CREDENTIAL_INFO)(Marshal.PtrToStructure(credInfo,
typeof(CERT_CREDENTIAL_INFO)));
                // Pobranie skrótu certyfikatu i przekształcenie go na ciąg szesnastkowy
                byte[] data2 = certStruct.rgbHashOfCert;
                string hex = BitConverter.ToString(data2).Replace("-", string.Empty);
                return hex;    // Zwrócenie ciągu szesnastkowego
            }
        }
        catch(Exception e)
        {
            // Obsługa wyjątku przez wyświetlenie komunikatu
            Console.WriteLine("An error occured: " + e.Message + e.StackTrace);
        }
        finally
        {
            // Zwolnienie zaalokowanej pamięci
            Marshal.FreeHGlobal(credData);
            Marshal.FreeHGlobal(credInfo);
        }
        return null;    // Zwrócenie null w przypadku błędu
    }
    // Metoda IsCertificateCredential sprawdza, czy poświadczenie jest poświadczeniem certyfikatu
```

```
        public static bool IsCertificateCredential(string data)
        {
            IntPtr credData = IntPtr.Zero;    // Uchwyt na dane poświadczenia
            IntPtr credInfo = IntPtr.Zero;    // Uchwyt na informacje o poświadczeniu
            try
            {
                // Przekształcenie ciągu znaków w obiekcie data na ciąg znaków Unicode i przypisanie go do credData
                credData = Marshal.StringToHGlobalUni(data);
                credInfo = IntPtr.Zero;
                CRED_MARSHAL_TYPE credType = 0;
                // Deserializacja poświadczenia
                bool success = CredUnmarshalCredential(credData, out credType, out credInfo);
                if (!success)
                {
                    int error = Marshal.GetLastWin32Error();
                    // Rozważ zapisanie komunikatu o błędzie do logu
                }
                // Zwrócenie informacji o tym, czy poświadczenie jest poświadczeniem certyfikatu
                return (success && credType == CRED_MARSHAL_TYPE.CertCredential);
            }
            catch(Exception)
            {
                // Rozważ zapisanie komunikatu o błędzie do logu
                return false;
            }
            finally
```

```
        {
            // Zwolnienie zaalokowanej pamięci
            CredFree(credInfo);
            Marshal.FreeHGlobal(credData);
        }
    }
    #endregion
  }
}
"@
# Dodanie typów: Importuje typy zdefiniowane w zestawach System, System.Runtime.InteropServices, System.Security i System.Management.Automation
add-type -AssemblyName System;
add-type -AssemblyName System.Runtime.InteropServices;
add-type -AssemblyName System.Security;
add-type -AssemblyName System.Management.Automation;
# Kompilacja kodu źródłowego C# i dodanie go do bieżącej sesji PowerShell
add-type -TypeDefinition $SourceCode -Language CSharp
# Pobranie poświadczenia od użytkownika
$smartcardCred = Get-Credential
# Sprawdzenie, czy poświadczenie jest poświadczeniem certyfikatu
$isCertCred = [Bongiovi.SmartcardLogon.SmartcardCredManager]::IsCertificateCredential($
```

b) Przedstaw opracowany wykaz kolejnych czynności do wykonania w PowerShell, aby zastosować urządzenia i narzędzia zabezpieczające dostęp tak aby zabezpieczyć dostęp do urządzenia PC.

c) Wykonaj notatkę zawierającą wykaz kolejnych czynności do wykonania w PowerShell, aby zastosować urządzenia i narzędzia zabezpieczające dostęp tak aby zabezpieczyć dostęp do urządzenia PC.

d) Przedstaw opracowany skrypt w PowerShell, z zastosowaniem urządzenia i narzędzia zabezpieczającego dostęp tak aby zabezpieczyć dostęp do urządzenia PC.

e) Wykonaj opracowany skrypt w PowerShell, z zastosowaniem urządzenia i narzędzia zabezpieczającego dostęp tak aby zabezpieczyć dostęp do urządzenia PC i zademonstruj jego działanie.

f) Wykonaj notatkę zawierającą działanie powyższego skrypt w PowerShell.

Zadanie 2

ReadFromAnySmartcard.ps1 - lokalizuje wszystkie certyfikaty karty inteligentnej z magazynu CurrentUser, przedstawia użytkownikowi listę certyfikatów karty inteligentnej do wyboru, zbiera numer PIN użytkownika, a następnie generuje PSCredential z wybranego certyfikatu karty inteligentnej. (Uznanie dla Joshua Chase za ten kod)

a) Zapoznaj się z działaniem tej funkcji PowerShell **Get-SmartCardCred** służącej do pobierania poświadczenia certyfikatu od użytkownika przy użyciu karty inteligentnej. Oto krok po kroku jej działanie:

1. **Definicja funkcji**: Funkcja jest zdefiniowana za pomocą słowa kluczowego **Function** oraz nazwy **Get-SmartCardCred**. Jest to funkcja, która ma za zadanie zwrócić poświadczenie w postaci obiektu **PSCredential**.

2. **Polecenie param** : **[cmdletbinding()] param()**: Definiuje atrybuty funkcji, które umożliwiają korzystanie z zaawansowanych funkcji przetwarzania w obrębie funkcji, takich jak potok, a także przyjmowanie parametrów.

3. **Kod C# do obsługi karty inteligentnej**: Kod C#, który jest przypisany do zmiennej **$SmartCardCode**, jest odpowiedzialny za obsługę karty inteligentnej oraz odzyskiwanie certyfikatów z magazynu certyfikatów użytkownika. Wykorzystuje on interop do advapi32.dll, co umożliwia wywoływanie funkcji systemowych z poziomu C#.

Użycie interop do advapi32.dll w skrypcie PowerShell oznacza korzystanie z interfejsu API (Application Programming Interface) systemu Windows, który jest dostarczany przez bibliotekę advapi32.dll. Jest to biblioteka dynamiczna zawierająca funkcje związane z bezpieczeństwem oraz uwierzytelnianiem w systemie Windows.

W skrypcie PowerShell, który wykorzystuje interop do advapi32.dll, najprawdopodobniej używa się tych funkcji, aby uzyskać dostęp do zaawansowanych funkcji związanych z bezpieczeństwem, na przykład:

1. **Czytanie i zapisywanie poświadczeń**: Można użyć funkcji z advapi32.dll do odczytywania i zapisywania poświadczeń (credentials) w magazynie systemowym.

2. **Uwierzytelnianie**: Można użyć funkcji z advapi32.dll do wykonywania operacji uwierzytelniania użytkowników.

3. **Zarządzanie certyfikatami**: Funkcje z advapi32.dll mogą być wykorzystywane do operacji na certyfikatach, takich jak dodawanie, usuwanie czy wyszukiwanie certyfikatów w magazynach certyfikatów.

W skrypcie PowerShell, który korzysta z advapi32.dll, można znaleźć deklaracje funkcji z biblioteki advapi32.dll za pomocą atrybutu **[DllImport("advapi32.dll")]**, co pozwala na wywoływanie tych funkcji bezpośrednio z poziomu skryptu PowerShell.

W przypadku opisanego skryptu, wykorzystanie interop do advapi32.dll występuje w kontekście obsługi karty inteligentnej i operacji związanych z certyfikatami, takimi jak serializacja i deserializacja certyfikatów oraz zarządzanie poświadczeniami.

4. **Dodanie typów i kodu C# do sesji PowerShell**: Funkcja korzysta z **Add-Type** do dodania kodu C# do bieżącej sesji PowerShell, aby umożliwić wywołanie kodu C# w obrębie funkcji PowerShell.

5. **Pobranie certyfikatu od użytkownika**: Funkcja korzysta z **Get-ChildItem** w celu pobrania listy certyfikatów z magazynu certyfikatów użytkownika.
Następnie korzysta z **X509Certificate2UI.SelectFromCollection** w celu wyświetlenia okna dialogowego, które umożliwia użytkownikowi wybór certyfikatu.

6. **Pobranie PIN od użytkownika**: Funkcja prosi użytkownika o wprowadzenie PINu przy użyciu **Read-Host -AsSecureString**, który jest potrzebny do uzyskania dostępu do prywatnego klucza certyfikatu.

7. **Marshalowanie certyfikatu i zwrócenie obiektu PSCredential**: Funkcja wywołuje metodę **MarshalFlow** z klasy **SmartCardLogon.Certificate** (zdefiniowanej w kodzie C#), aby uzyskać poświadczenie certyfikatu w postaci obiektu **PSCredential**. W przypadku powodzenia, funkcja zwraca ten obiekt jako wynik.

Dzięki temu procesowi, użytkownik jest w stanie uzyskać poświadczenie na podstawie certyfikatu przechowywanego na karcie inteligentnej oraz odpowiedniego PINu.

```
Function Get-SmartCardCred{

<#

.SYNOPSIS

Get certificate credentials from the user's certificate store.

.DESCRIPTION

Returns a PSCredential object of the user's selected certificate.

.EXAMPLE

Get-SmartCardCred
```

```
UserName                                 Password
--------                                 --------
@@BVkEYkWiqJgd2d9xz3-5BiHs1cAN System.Security.SecureString
.EXAMPLE
$Cred = Get-SmartCardCred
.OUTPUTS
[System.Management.Automation.PSCredential]
.NOTES
Author: Joshua Chase
Last Modified: 01 August 2018
C# code used from https://github.com/bongiovimatthew-microsoft/pscredentialWithCert
#>
[cmdletbinding()]
param()
    $SmartCardCode = @"
// Kod C# do obsługi karty inteligentnej i pobierania certyfikatu
"@
    # Dodanie kodu C# do bieżącej sesji PowerShell
    Add-Type -TypeDefinition $SmartCardCode -Language CSharp
    Add-Type -AssemblyName System.Security
    # Pobranie listy certyfikatów z magazynu certyfikatów użytkownika
    $ValidCerts = [System.Security.Cryptography.X509Certificates.X509Certificate2[]](Get-ChildItem 'Cert:\CurrentUser\My')
    # Wybór certyfikatu z wyświetlonym oknem dialogowym
    $Cert = [System.Security.Cryptography.X509Certificates.X509Certificate2UI]::SelectFromCollection($ValidCerts, 'Choose a certificate', 'Choose a certificate', 0)
    # Pobranie PINu od użytkownika
```

```
    $Pin = Read-Host "Enter your PIN: " -AsSecureString

    # Wywołanie metody MarshalFlow z klasy Certificate w celu uzyskania poświadczenia

    Write-Output ([SmartCardLogon.Certificate]::MarshalFlow($Cert.Thumbprint, $Pin))

}
```

Komentarze wyjaśniające działanie:

.SYNOPSIS: Krótkie streszczenie funkcji, które opisuje jej przeznaczenie.

.DESCRIPTION: Szczegółowy opis funkcji, który informuje użytkownika, że funkcja zwraca obiekt PSCredential wybranego przez użytkownika certyfikatu.

.EXAMPLE: Przykładowe użycie funkcji, które pokazuje wynikowy obiekt PSCredential z wybranym certyfikatem.

.OUTPUTS: Informacja o typie zwracanym przez funkcję.

.NOTES: Dodatkowe informacje o autorze i dacie ostatniej modyfikacji funkcji oraz źródło kodu C#, z którego korzysta funkcja.

Parametry i kod C#: Definicja parametrów funkcji oraz kod C# do obsługi karty inteligentnej, który jest dodawany do bieżącej sesji PowerShell.

Pobranie certyfikatu i PINu: Funkcja pobiera listę certyfikatów z magazynu użytkownika, pozwala użytkownikowi na wybór certyfikatu, a następnie prosi o wprowadzenie PINu.

Wywołanie metody MarshalFlow: Wywołuje metodę MarshalFlow z klasy Certificate (zdefiniowanej w kodzie C#), aby uzyskać poświadczenie certyfikatu na podstawie wybranego certyfikatu i PINu.

b) Przedstaw opracowany wykaz kolejnych czynności do wykonania w PowerShell, aby zastosować urządzenia i narzędzia zabezpieczające dostęp tak aby zabezpieczyć dostęp do urządzenia PC.

c) Wykonaj notatkę zawierającą wykaz kolejnych czynności do wykonania w PowerShell, aby zastosować urządzenia i narzędzia zabezpieczające dostęp tak aby zabezpieczyć dostęp do urządzenia PC.

d) Przedstaw opracowany funkcje w PowerShell, z zastosowaniem urządzenia i narzędzia zabezpieczającego dostęp tak aby zabezpieczyć dostęp do urządzenia PC.

e) Wykonaj opracowany funkcje w PowerShell, z zastosowaniem urządzenia i narzędzia zabezpieczającego dostęp tak aby zabezpieczyć dostęp do urządzenia PC i zademonstruj jego działanie.

f) Wykonaj notatkę zawierającą działanie powyższej funkcji w PowerShell.