# How To Write Yara Rules For Malware - Practical Examples

By Matthew

Published: 2023-10-04 · Archived: 2026-04-05 16:09:50 UTC

The purpose of this article is to highlight some practical examples of indicators that can be used for detection using Yara.

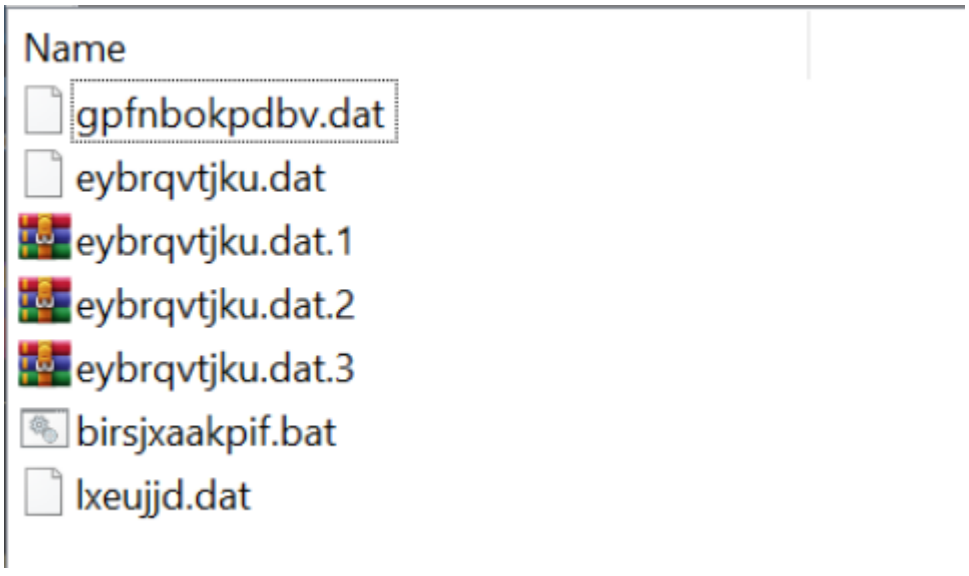
> The rules are not intended to be performance-optimized. Purely examples of indicators that can be used for detection. Here is a great link if you're interested in performance optimization.

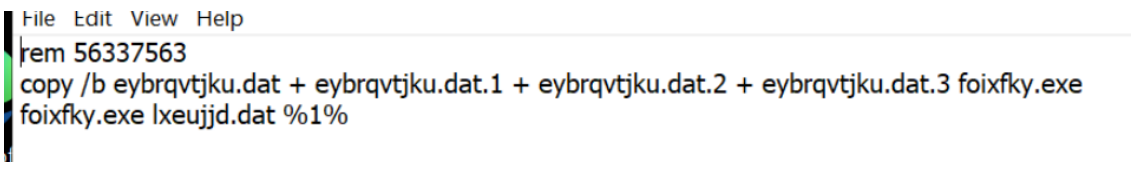
If you wish to try building or testing Yara rules for yourself, we recommend signing up for a free or boosted ($10USD) account on unpacme (which is what we personally use for testing). Unpacme has an excellent Yara hunting feature that allows you to test on a large collection of malware and legitimate samples.

### Lu0Bot SFX Archives

Some recent lu0bot samples are using self-extracting archives (essentially an `.exe` that unpacks a `.zip`). We found this sample using an any.run blog and Malware Bazaar.

Inside the sfx/zip file are multiple `.dat` files that are used to create an exe. The final exe is executed using the randomly named `.bat` file.



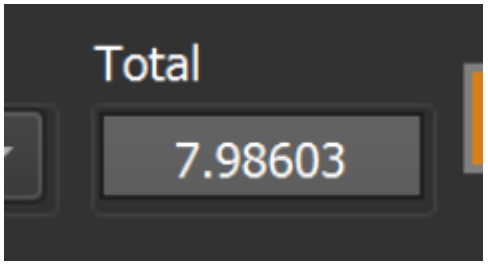

This introduces the following string artifacts inside the initial `.exe` file.

| | | | | |
|---|---|---|---|---|
| 877 | 0002286f | 00000005 | A | PMSCF |
| 878 | 000228ac | 0000000f | A | gpfnbokpdbv.dat |
| 879 | 000228cc | 0000000e | A | eybrqvtjku.dat |
| 880 | 000228eb | 00000010 | A | eybrqvtjku.dat.1 |
| 881 | 0002290c | 00000010 | A | eybrqvtjku.dat.2 |
| 882 | 0002292d | 00000010 | A | eybrqvtjku.dat.3 |
| 883 | 0002294e | 00000010 | A | birsjxaakpif.bat |
| 884 | 0002296f | 0000000b | A | lxeujjd.dat |

Which allows this signature to be created. Utilising the `.dat.1` `.dat.2` etc as well as the presence of the `.bat` file.

Since the file is an archive, it is mostly zipped and compressed. This results in an overall entropy of `7.98`, so we added an additional filter `math.entropy(0,filesize) > 7`. This provides more accuracy at the cost of additional compute resources.

> math.entropy is dependent on the math module and is compute intensive. So you are free to remove this jf you run into timeout issues.




We also noticed another sfx artifact of `"Win32 Cabinet Self-Extractor"`. We added this as a string in order to reduce false positives. This probably wasn't necessary, but something you can add to hone in on specific file types. (In this case, sfx files)

We ultimately used this rule to identify more samples. This is not necessarily the most efficient rule but it was able to find additional samples. The definition of "efficient" will depend on your exact situation and compute resources.

```
import "math"

rule win_lu0bot_sfx_packer_oct_2023
{
        meta:
                author = "Matthew @ Embee_Research"
                created = "2023/10/03"
                description = ""
                sha_256 = "9c84cd037b061c177ee10c45f1f87b3ea05744f1638ab3f348d6b9a3b1cbcfbf"
```

```
        strings:
                $s1 = ".dat" ascii
                $s2 = ".dat.1" ascii
                $s3 = ".dat.2" ascii
                $s4 = ".dat.3" ascii
                $s5 = ".bat" ascii


                $t1 = "Win32 Cabinet Self-Extractor" wide

        condition:
                        math.entropy(0,filesize) > 7
                and
                        (all of ($s*))
                and
                        $t1


}
```

Testing on unpacme returned 21 results. With 0 hits for goodware. All of the returned samples appear to be Lu0Bot.

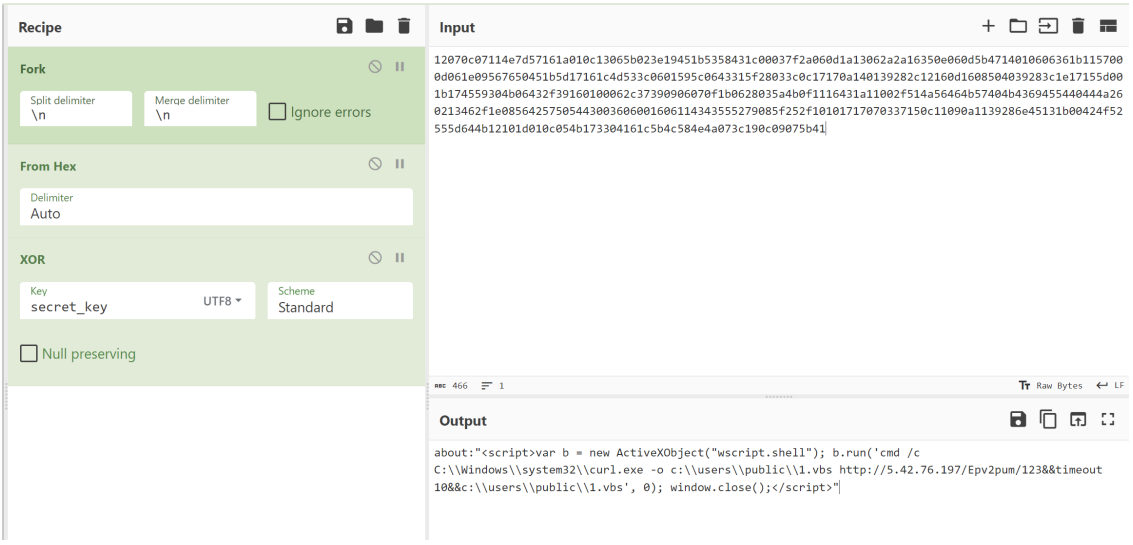
### DarkGate XLL Loader

Darkgate has recently utilised XLL (Excel Add-in) files as part of the infection process. An XLL is essentially a DLL file that can be executed by Microsoft Excel.

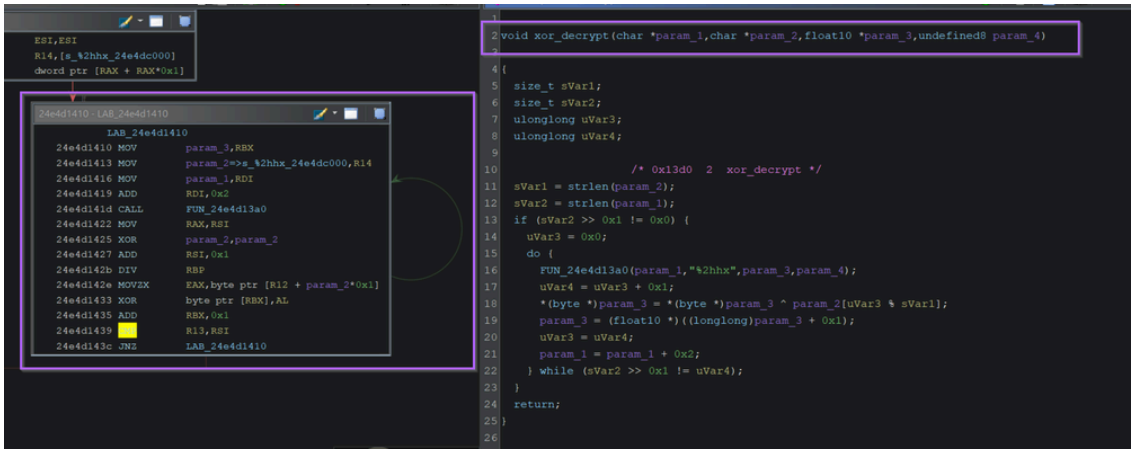
When opened, the XLL will automatically execute the `xlAutoOpen` export. Once this export is called, a blob of hex bytes is xor decoded to produce a Wscript command containing C2 information and filenames.

The `xlAutoOpen` and bytecodes associated with the hex decoding can be used as indicators for a yara rule.

```
undefined8 xlAutoOpen(void)

{
  float10 *_Source;
  undefined8 in_R9;
  undefined8 local_118;
  undefined8 local_110;
  undefined8 local_108;

                    /* 0x1450  1  xlAutoOpen */
  CopyFileW(L"C:\\Windows\\System32\\mshta.exe",L"C:\\Users\\Public\\me.exe",0x1);
  _Source = (float10 *)malloc(0xe9);
  *(undefined *)((longlong)_Source + 0xe8) = 0x0;
  xor_decrypt("12070c07114e7d57161a010c13065b023e19451b5358431c00037f2a060d1a13062a2a16350e060d5
  14010606361b1157000d061e09567650451b5d17161c4d533c0601595c0643315f28033c0c17170a140139282c121
  08504039283c1e17155d001b174559304b06432f39160100062c37390906070f1b0628035a4b0f1116431a11002f5
  4a514d40574c715a53405d5451414a432e2a4a4841564554111d320e0a0c0745524243523c5139250616060016280
  1b1f0c002e3945711d070a544943424c4f7f1c0c17170a145c0618301800515a5e5f5d16172d02150d4d47"
             ,"secret_key",_Source,in_R9);
  local_118 = 0x73726573555c3a43;
  local_110 = 0x5c63696c6275505c;
  local_108 = 0x206578652e656d;
  strcpy((char *)((longlong)&local_108 + 0x7),(char *)_Source);
  WinExec((LPCSTR)&local_118,0x1);
  free(_Source);
  MessageBoxA(NULL,"An internal error has occurred.","Internal Error",0x10);
  return 0x0;
}
```



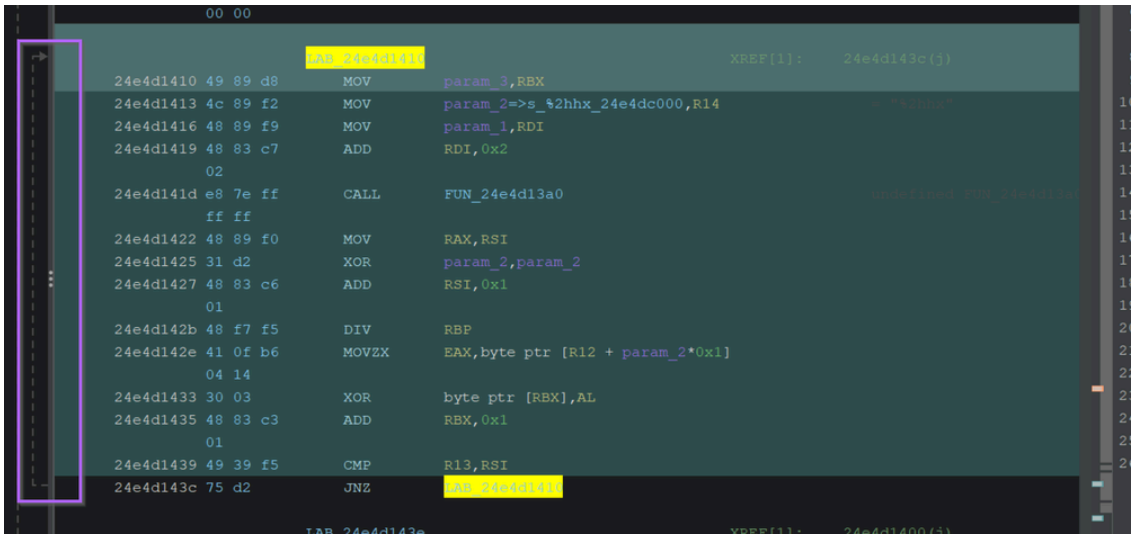

By jumping into the `xor_decrypt` function. We can observe the primary logic used for performing the decoding. This is usually easy to tell (for simple decoding functions) because the logic will be inside a loop (green arrow on left).



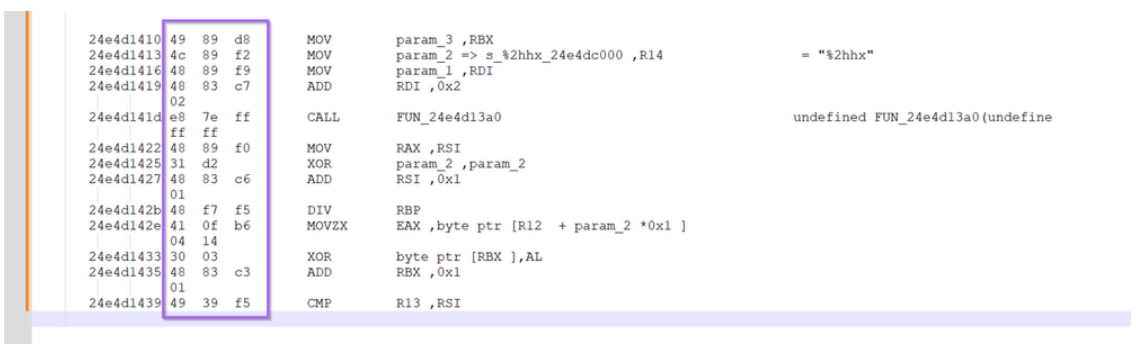

Since decoding logic is often re-used across similar malware (albeit with different decoding keys), we can use the logic itself as an indicator that can be signatured.

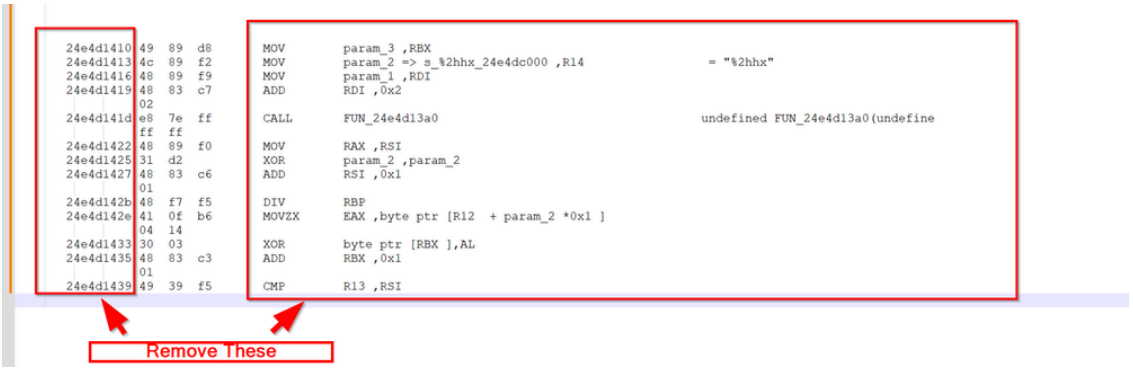
To achieve this, we can click the decoding loop and then browse back to the listing window. From here, we can highlight the same instructions that we observed in the previous screenshot.

Observe on the left that we are looking at the same loop.

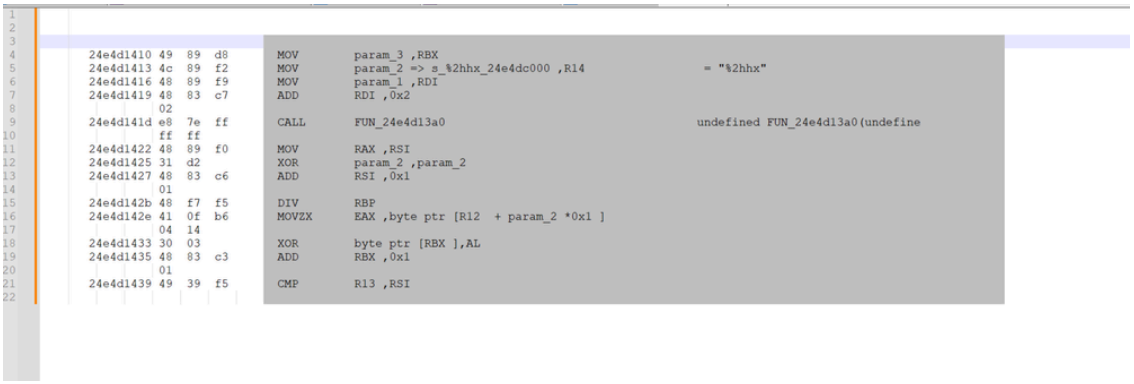



Copying the instructions into a text editor, gives the following. Noting that the highlighted bytecodes are what can be used as a signature.

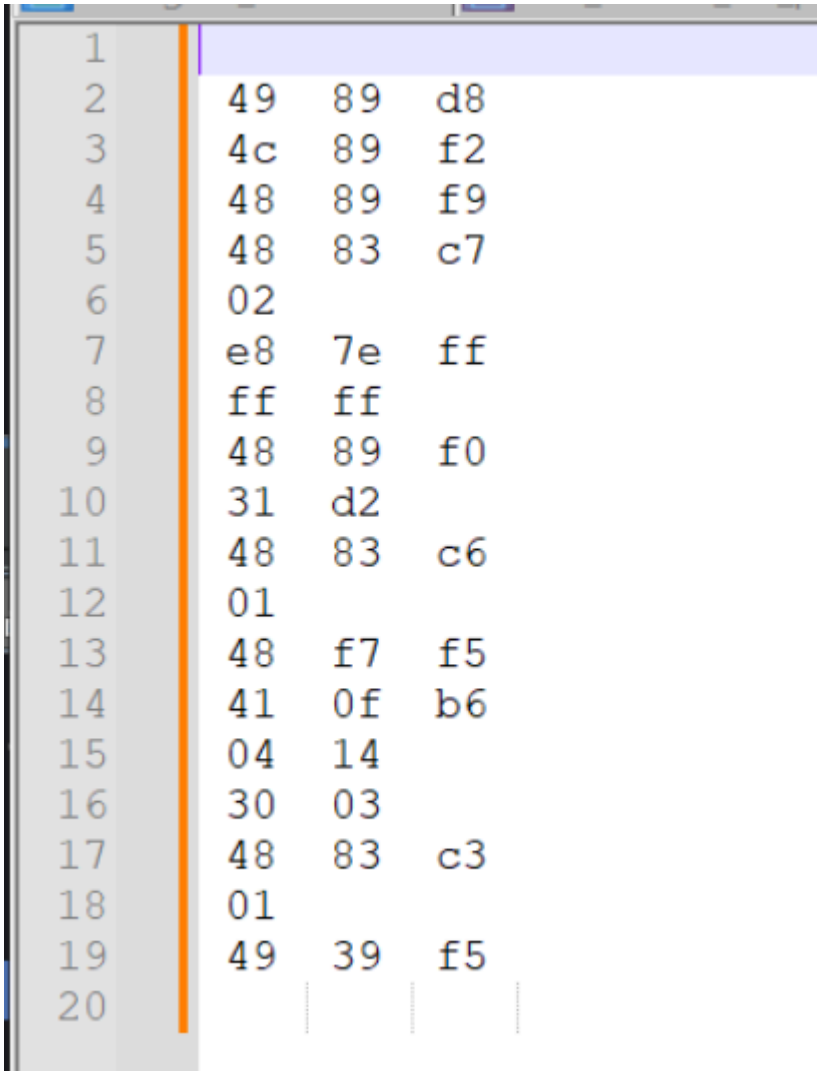




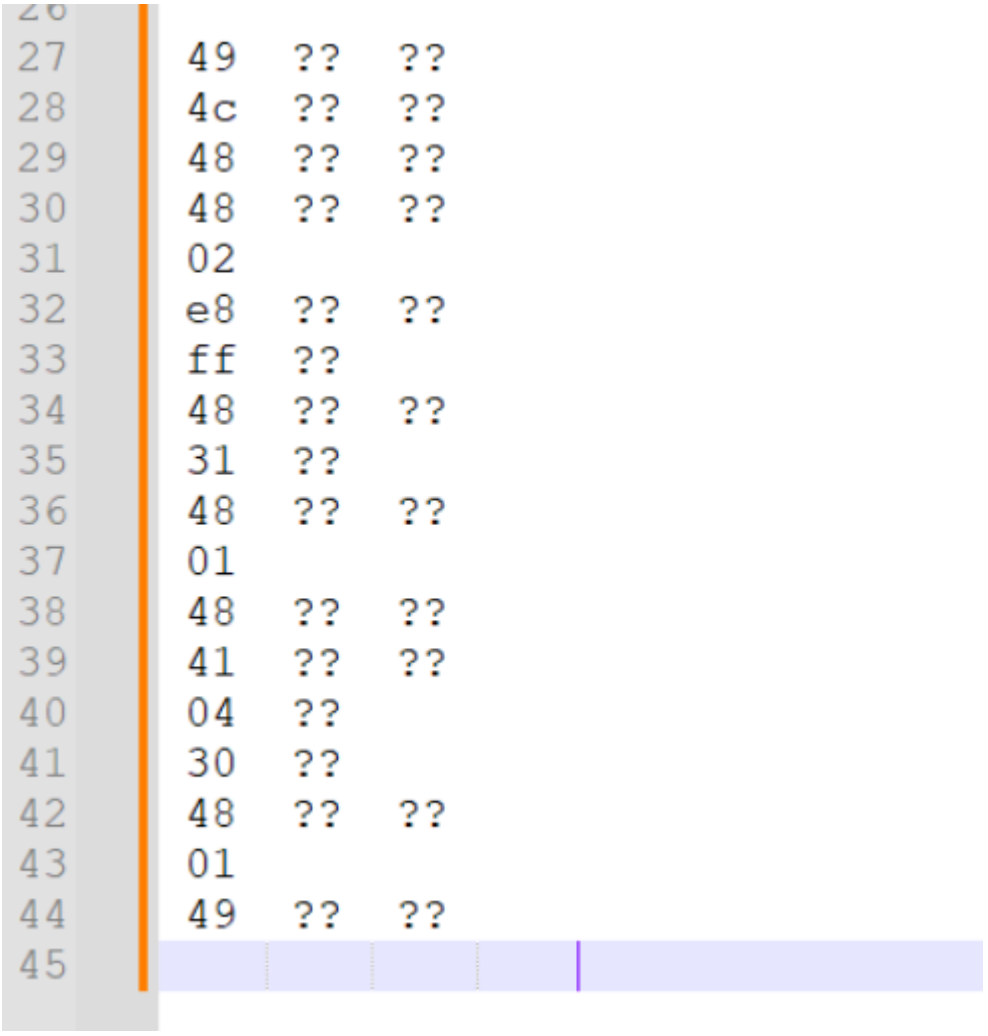

If you're using Notepad++, you can hold `alt` and select the right column and hit delete. The same can be done with the left column.




This leaves only the bytecodes.




Keeping only the opcodes and constant values, leaves this



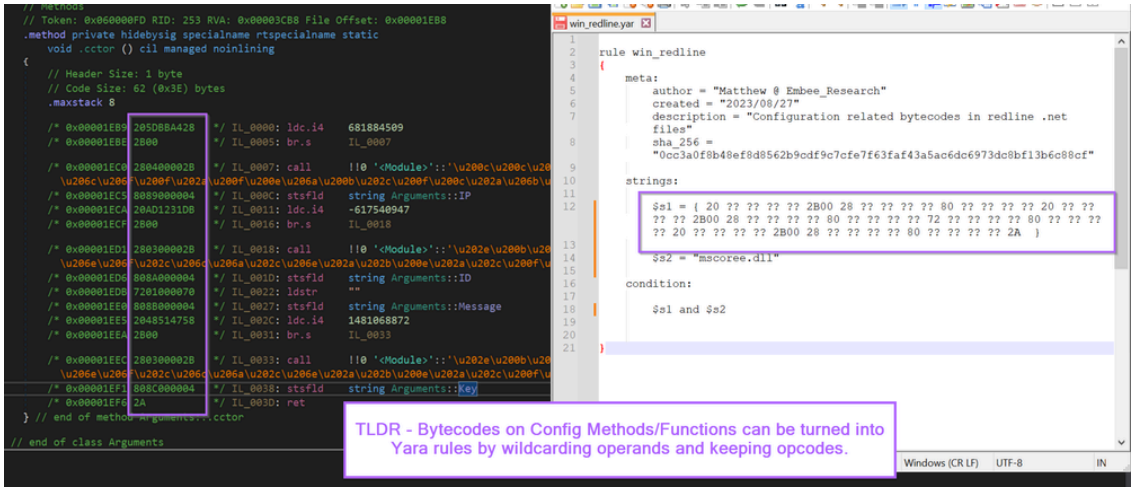

Which results in the following yara rule. we also added in `xlAutoOpen` to narrow the results down only (ideally) to XLL files.

```
 rule win_darkgate_xllloader_oct_2023
{
        meta:
                author = "Matthew @ Embee_Research"
                created = "2023/10/03"
                description = "Detects XLL Files Related to DarkGate"

        strings:
                $s1 = "xlAutoOpen" ascii
                $s2 = { 49 ?? ?? 4c ?? ?? 48 ?? ?? 48 ?? ?? 02 e8 ?? ?? ?? ?? 48 ?? ?? 31 ?? 48 ?? ?? 01 48 ??

        condition:
                        $s1 and $s2


}
```

Using unpacme, this resulted in 24 additional samples of the Darkgate XLL loader.

## Redline Stealer - Configuration/IL Bytecodes

Redline stealer samples have a relatively consistent pattern associated with the method that stores configuration settings. The bytecodes associated with this method can be used to create a yara rule that matches on similar samples.




---

Source: https://embee-research.ghost.io/practical-signatures-for-identifying-malware-with-yara/