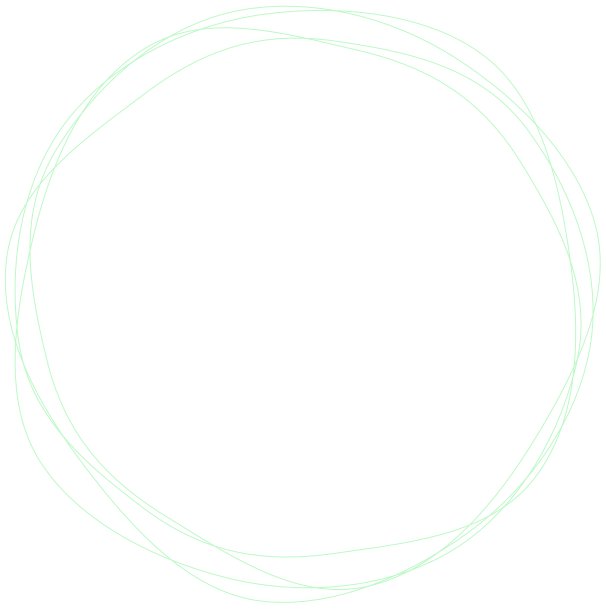
Stairwell

Threat research blog • February 28, 2022

Quick n' dirty detection research: Building a labeled malware corpus for YARA testing



By Steve Miller, Threat Researcher

One of the big hurdles to using YARA fruitfully is having a decent corpus of files on which to test your ideas. Whether you are using YARA rules for threat detection, intelligence clustering and analysis, incident response and investigation, feature measurement and extraction, simple file enrichment, or some nuanced combination thereof, you need a reasonably large and detailed corpus of labeled files to help you understand if your rules satisfy your intent with the precision or fidelity that you desire. Counting rule matches alone won't cut it. You need the context of what is matching and what all of those matching files represent.

If you're without comprehensive tooling and data to develop and test YARA rules, you are stuck sitting at a lonely command prompt and wondering what you can do without building a huge toolkit. The least fancy approach is to get lots of malware files and use your terminal to run YARA at the command line to get matches. In this blog, we walk through an example of building a test corpus of malware with at-a-glance™ intelligence context. The combination of fast CLI tooling and simply labeled malware will help you in your work with YARA rules.

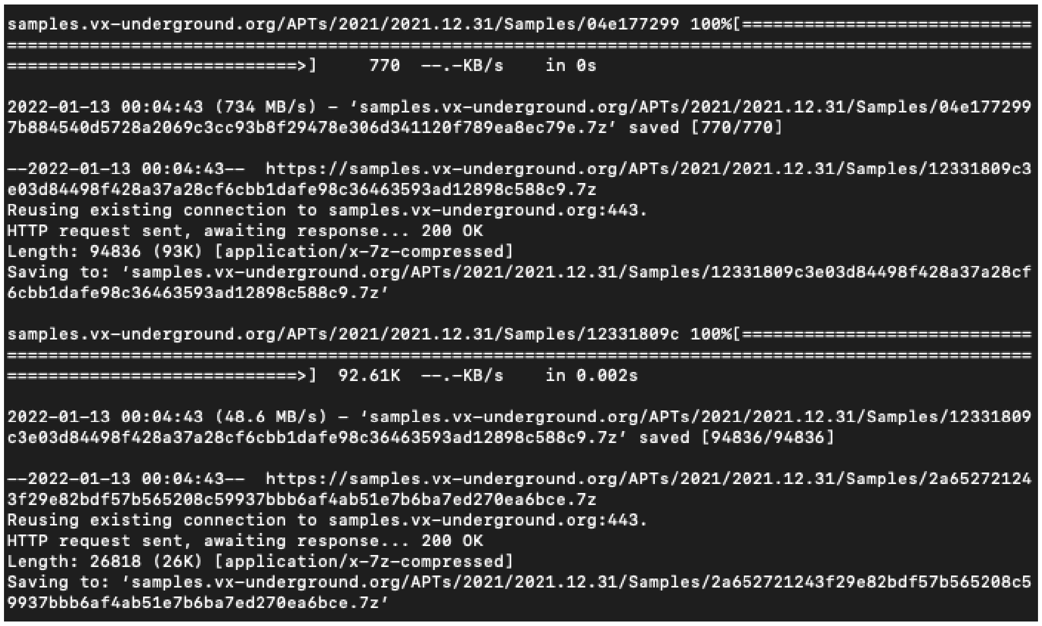
Step 1: Downloading the malware corpus From vx-underground

The challenge of building a malware corpus is not only getting the files, but getting them in an organized way that is married up to a whisper of context. Thankfully, our friends at vx-underground organize and maintain a collection of malware organized by family and by association with open source APT reports published by security organizations. This data is organized in a directory structure that helps you understand what the samples are.

We start by using wget (or something similar) to download a set of malware from vx-underground.

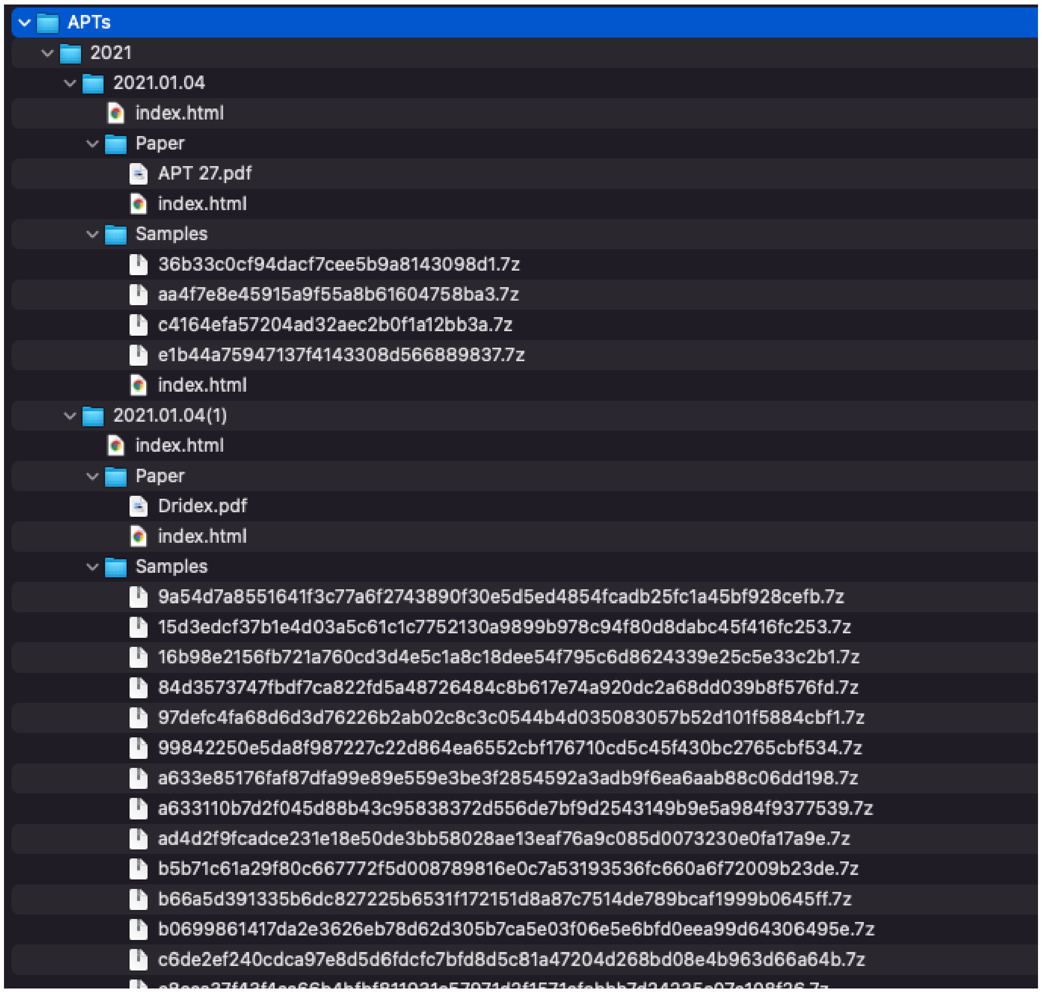
```
wget -r --no-parent --reject "index.html*" https://samples.vx-
underground.org/APTs/
```

This will obviously take some time, and we may not know how long or how many files are actually in the corpus, or what the overall size will be like. Last we checked, there were about 30K samples in /APTs/ and this is probably something like 20-30GB. This might take an hour or so, depending on connection speed and the internet tube flux and all that.



```
samples.vx-underground.org/APTs/2021/2021.12.31/Samples/04e177299 100%[============================
===================================================================================================
============================>]     770  --.-KB/s    in 0s

2022-01-13 00:04:43 (734 MB/s) - 'samples.vx-underground.org/APTs/2021/2021.12.31/Samples/04e177299
7b884540d5728a2069c3cc93b8f29478e306d341120f789ea8ec79e.7z' saved [770/770]

--2022-01-13 00:04:43--  https://samples.vx-underground.org/APTs/2021/2021.12.31/Samples/12331809c3
e03d84498f428a37a28cf6cbb1dafe98c36463593ad12898c588c9.7z
Reusing existing connection to samples.vx-underground.org:443.
HTTP request sent, awaiting response... 200 OK
Length: 94836 (93K) [application/x-7z-compressed]
Saving to: 'samples.vx-underground.org/APTs/2021/2021.12.31/Samples/12331809c3e03d84498f428a37a28cf
6cbb1dafe98c36463593ad12898c588c9.7z'

samples.vx-underground.org/APTs/2021/2021.12.31/Samples/12331809c 100%[============================
===================================================================================================
============================>]  92.61K  --.-KB/s    in 0.002s

2022-01-13 00:04:43 (48.6 MB/s) - 'samples.vx-underground.org/APTs/2021/2021.12.31/Samples/12331809
c3e03d84498f428a37a28cf6cbb1dafe98c36463593ad12898c588c9.7z' saved [94836/94836]

--2022-01-13 00:04:43--  https://samples.vx-underground.org/APTs/2021/2021.12.31/Samples/2a65272124
3f29e82bdf57b565208c59937bbb6af4ab51e7b6ba7ed270ea6bce.7z
Reusing existing connection to samples.vx-underground.org:443.
HTTP request sent, awaiting response... 200 OK
Length: 26818 (26K) [application/x-7z-compressed]
Saving to: 'samples.vx-underground.org/APTs/2021/2021.12.31/Samples/2a652721243f29e82bdf57b565208c5
9937bbb6af4ab51e7b6ba7ed270ea6bce.7z'
```

When our wget downloading is all done, we are left with a directory tree of folders and .7z compressed malware files.



```
APTs
  2021
    2021.01.04
      index.html
      Paper
        APT 27.pdf
        index.html
      Samples
        36b33c0cf94dacf7cee5b9a8143098d1.7z
        aa4f7e8e45915a9f55a8b61604758ba3.7z
        c4164efa57204ad32aec2b0f1a12bb3a.7z
        e1b44a75947137f4143308d566889837.7z
        index.html
    2021.01.04(1)
      index.html
      Paper
        Dridex.pdf
        index.html
      Samples
        9a54d7a8551641f3c77a6f2743890f30e5d5ed4854fcadb25fc1a45bf928cefb.7z
        15d3edcf37b1e4d03a5c61c1c7752130a9899b978c94f80d8dabc45f416fc253.7z
        16b98e2156fb721a760cd3d4e5c1a8c18dee54f795c6d8624339e25c5e33c2b1.7z
        84d3573747fbdf7ca822fd5a48726484c8b617e74a920dc2a68dd039b8f576fd.7z
        97defc4fa68d6d3d76226b2ab02c8c3c0544b4d035083057b52d101f5884cbf1.7z
        99842250e5da8f987227c22d864ea6552cbf176710cd5c45f430bc2765cbf534.7z
        a633e85176faf87dfa99e89e559e3be3f2854592a3adb9f6ea6aab88c06dd198.7z
        a633110b7d2f045d88b43c95838372d556de7bf9d2543149b9e5a984f9377539.7z
        ad4d2f9fcadce231e18e50de3bb58028ae13eaf76a9c085d0073230e0fa17a9e.7z
        b5b71c61a29f80c667772f5d008789816e0c7a53193536fc660a6f72009b23de.7z
        b66a5d391335b6dc827225b6531f172151d8a87c7514de789bcaf1999b0645ff.7z
        b0699861417da2e3626eb78d62d305b7ca5e03f06e5e6bfd0eea99d64306495e.7z
        c6de2ef240cdca97e8d5d6fdcfc7bfd8d5c81a47204d268bd08e4b963d66a64b.7z
```

Step 2: Extracting and tidying up the files

Since we know we will be doing YARA scans across these files, we know that we will have to extract the raw malware samples from the archive files. To start, we find all the 7z files and pipe them to a loop to extract the malware with the password "infected" and place it into the same directory.

```
find * -name "*.7z" | while read filename; do 7z e -aos $filename -pinfected -o"`dirname "$filename"`"; done;
```

Last, we can delete all the remaining 7z files since we don't need them anymore, and we should probably reclaim all that disk space.

```
find * -name '*.7z' -delete
```

Now we are left with a somewhat tidy directory tree that looks something like this.

To run a quick test of YARA rules at the command line over this corpus, we can take a trivial rule just to see that matching works and what the results look like. We take a basic rule to check for the MZ header at offset 0 and run that against the directory.

```
steve@CFO-MBP ~ % cat pe.yar
import "pe"
rule is_mz { condition: uint16be(0) == 0x4d5a }
steve@CSO-MBP ~ % yara -r ~/pe.yar /vx-underground.org/APTs
is_mz /vx-
underground.org/APTs/2021/2021.01.06(1)/Samples/c9038e31f798119d9e93e
7eafbdd3e0f215e24ee2200fcd2a3ba460d549894ab
is_mz /vx-
underground.org/APTs/2021/2021.01.06(1)/Samples/93ec3c23149c3d5245adf
5d8a38c85e32cda24e23f8c4df2e19e1423739908b7
is_mz /vx-
underground.org/APTs/2021/2021.01.06(1)/Samples/6b53ba14172f0094a00ed
```

```
fef96887aab01e8b1c49bdc6b1f34d7f2e32f88d172
is_mz /vx-
underground.org/APTs/2021/2021.01.12(1)/Samples/3A65745DEE2AFBFFE0056
9C83572723FD8C04E76
is_mz /vx-
underground.org/APTs/2021/2021.01.12(1)/Samples/3751D00639C255EE53002
CA1DCCABD185094BFB4
is_mz /vx-
underground.org/APTs/2021/2021.01.12(1)/Samples/2E5E628F2CE5AEB2235B7
FBB155B13BE2B432FFC
is_mz /vx-
underground.org/APTs/2021/2021.01.12(1)/Samples/0A4742BE00AF2B0E26987
E5E3F37B9784BDEA826
is_mz /vx-
underground.org/APTs/2021/2021.01.12(1)/Samples/13A5C261C2B59FC416AC4
B4AF004A858E272DF2F
```

Step 3: Massaging the directory tree

OK, we're making progress. We can see our matches with respect to the year in which they were published or referenced by a public report, and somewhere in those directories, there is a PDF copy of the report for us to go take a look at. Sweet.

While it is nice we have the hash upfront, because of the naming convention, we can't actually see the context for what this malware is, and the report PDF is in a different directory altogether. We could navigate to each of the parent directories to see what it is, but that's no fun. We want a faster, lighter peek at what the results are associated with.

To automatically apply the most and easiest context possible, we decided to take the name of the PDF report and append that to the YYYY.MM.DD directory. So with a report such as 2021.01.04(1)/Paper/APT 27.PDF, we would rename the directory /2021.01.04(1)/ to /2021.01.04(1) APT 27/. This seems dumb at first, but hang in there. It's worth it.

Using a hacky, terrible Python script, we run through the directory tree, grab the PDF name, strip it of any special characters, and append it to the name of its grandparent directory.

```
#!/usr/bin/env python3
import os,sys
from os import path
import re
def get_all_files(treeroot):
   for dir,subdirs,files in os.walk(treeroot):
       for f in files:
           if f in __file__: continue
           if f.lower().endswith('.pdf'):
               fullpath = os.path.realpath( os.path.join(dir,f) )
               no_ext = os.path.basename(f).split('.pdf')[0]
               parent = os.path.dirname(fullpath)
               file_count = len(files)
               grandparent = os.path.dirname(parent)
               sanitized_no_ext = re.sub(r"[^a-zA-Z0-9-\s_]", "",
no_ext)
               sanitized_newnewpath = grandparent + " " +
sanitized_no_ext
               print("New name: " + sanitized_newnewpath)
               print("\n")
               #do renaming
               if file_count == 1:
                  os.rename(grandparent,sanitized_newnewpath)
def main():
   top_dir="."
   get_all_files(top_dir)
if __name__ == '__main__' : main()
```

This is an imperfect script that does some things poorly, but you get the picture. We run it to rename the directory something more informative than simply the date, and finally, we are looking at something like this.

When we return to our YARA query, we now have an at-a-glance view of the association of each of the matches. This is a cheap and easy way to infer context about the matches.

```
steve@CMO-MBP 2021 % yara -r ~/pe.yar /vx-underground.org/APTs
is_mz /vx-underground.org/APTs/2021/2021.01.11 Sunburst
Kazuar/Samples/1F70BEF5D79EFBDAC63C9935AA353955
is_mz /vx-underground.org/APTs/2021/2021.01.11 Sunburst
Kazuar/Samples/150D0ADDF65B6524EB92B9762DB6F074
is_mz /vx-underground.org/APTs/2021/2021.01.11 Sunburst
Kazuar/Samples/053DDB3B6E38F9BDBC5FB51FDD44D3AC
is_mz /vx-underground.org/APTs/2021/2021.01.04 APT
27/Samples/c4164efa57204ad32aec2b0f1a12bb3a
is_mz /vx-underground.org/APTs/2021/2021.01.04 APT
27/Samples/36b33c0cf94dacf7cee5b9a8143098d1
is_mz /vx-underground.org/APTs/2021/2021.01.11 Sunburst
Kazuar/Samples/2C4A910A1299CDAE2A4E55988A2F102E
is_mz /vx-underground.org/APTs/2021/2021.01.04 APT
27/Samples/e1b44a75947137f4143308d566889837
is_mz /vx-underground.org/APTs/2021/2021.01.12(1) Operation
Spalax/Samples/466D5DF1F085689D4DD305B4B4F7B88095C6F0DB
is_mz /vx-underground.org/APTs/2021/2021.01.12(1) Operation
Spalax/Samples/642EC136B72B76EBAC5D6312B6DFA6600220403B
is_mz /vx-underground.org/APTs/2021/2021.01.12(1) Operation
Spalax/Samples/3751D00639C255EE53002CA1DCCABD185094BFB4
is_mz /vx-underground.org/APTs/2021/2021.01.12(1) Operation
Spalax/Samples/3C97CF4091233D2C2FC6A692208AE99EAF5EE9A6
is_mz /vx-underground.org/APTs/2021/2021.01.04(1)
Dridex/Samples/c6de2ef240cdca97e8d5d6fdcfc7bfd8d5c81a47204d268bd08e4b
```

```
963d66a64b
is_mz /vx-underground.org/APTs/2021/2021.01.12(1) Operation
Spalax/Samples/185664DF6E1547C8E695E6018A53124E522612A6
is_mz /vx-underground.org/APTs/2021/2021.01.04(1)
Dridex/Samples/c8cca37f43f4aa66b4bfbf811931c57971d2f1571cfebbb7d24235
c07e108f26
is_mz /vx-underground.org/APTs/2021/2021.01.04(1)
Dridex/Samples/fa61c3c9e2089deb3f2b40333f5ee0860177692c436c50b07eef85
993a1dbfa9
…
```

Step 4: Taking stock of the corpus

Now that we have our little collection tidied up and somewhat organized, it is important that we take some basic measurements of the corpus. That way, when we test YARA rules and count results for a given rule, we can put these numbers into the perspective of the overall collection. The numerator matters most when it's accompanied by a denominator.

How many files are in this corpus?

As of Jan 13, 2022, we downloaded approximately 32,300 files from the /APTs/ directory, which is about 36.54GB,

What are the major file types?
Using YARA rules, we can take some basic inventory by evaluating the file magic. We threw together a list of rules to do header checks with hex evals at offset 0. This is obviously imperfect and not meant to capture all things as we know there are other file types, and often malware authors mangle the headers to evade this exact type of check anyway. However, this basic measurement can help us get an impression of the data we've got.

```
rule head_gz { condition: uint16be(0) == 0x1f8b }
rule head_7z { condition: uint16be(0) == 0x377a }
rule head_mz { condition: uint16be(0) == 0x4d5a }
rule head_pkzip { condition: uint16be(0) == 0x504b }
rule head_html { condition: uint32be(0) == 0x3c68746d }
rule head_gif { condition: uint32be(0) == 0x47494638 }
rule head_kwaj { condition: uint32be(0) == 0x4b57414a }
rule head_mscf { condition: uint32be(0) == 0x4d534346 }
rule head_rar { condition: uint32be(0) == 0x52617221 }
rule head_szdd { condition: uint32be(0) == 0x535a4444 }
rule head_rtf { condition: uint32be(0) == 0x7b5c7274 }
rule head_elf { condition: uint32be(0) == 0x7f454c46 }
rule head_png { condition: uint32be(0) == 0x89504e47 }
rule head_doc { condition: uint32be(0) == 0xd0cf11e0 }
rule head_jpeg { condition: uint32be(0) == 0xffd8ffe0 }
rule head_dex { condition: uint32be(0) == 0x6465780a }
rule head_macho { condition: uint32(0) == 0xfeedface or uint32(0) ==
0xcefaedfe or uint32(0) == 0xfeedfacf or uint32(0) == 0xcffaedfe or
uint32(0) == 0xcafebabe or uint32(0) == 0xbebafeca }
```

Taking this list of rules in a single rules file, we can jam them all against our corpus, search for the match results and then sort, unique, and count the number of matches.

```
steve@CTO-MBP ~ % yara -r head_tests.yar /vx-underground.org/APTs |
awk '{print $1}' | sort | uniq -c
  13 head_7z
  11 head_dex
1596 head_doc
 347 head_elf
  41 head_gif
  50 head_gz
  18 head_html
  31 head_jpeg
  50 head_macho
   7 head_mscf
20368 head_mz
2925 head_pkzip
  32 head_png
  89 head_rar
 405 head_rtf
```

Boom, OK. Now we've got a good sense that there's like 20K PEs in here and plenty of other stuff to fiddle around with. Having an idea of the overall numbers is valuable when you are trying to determine if your match count is appropriate for the intention of your rule, and this approach helps determine if the thing you are looking for is notable, unique, or significant.

Step 5: Putting it into action

Now you've got a relatively contextualized, labeled corpus of malware. What do you do next?

Well, suppose you're interested in researching signed malware and Authenticode anomalies, and you want to search the corpus you have for samples that you can look into further. We jump into a text editor and create a new rule called signed_pe.yar and run that against the lot.

```
import "pe"
rule signed_pe { condition: pe.number_of_signatures != 0 }
```

First, we do a test for the count.

```
steve@FBO-MBP ~ % yara -r signed_pe.yar /vx-underground.org/APTs |
awk '{print $1}' | sort | uniq -c
2334 signed_pe
```

Oh, sweet, looks like we've got *lots* of samples, so now we can cherry-pick ones we think are interesting. We've got the year they were reported, an admittedly rough name for how they were reported, and the file name is a hash so we can jump and query that as we like too. And of course, we've got the file bytes, and the original PDF of the report in each directory, so we can reference the original work and dive as deep as we please.

```
signed_pe /vx-underground.org/APTs/2013/2013.05.20 Miniduke and Operation
Hangover/Samples/3a89f05c09425f03fe74b2242b119cce
signed_pe /vx-underground.org/APTs/2013/2013.01.14 Red
October/Samples/797541f87e2e3a9a0754a097772f3192
signed_pe /vx-underground.org/APTs/2013/2013.05.16 Targeted
information stealing attacks in South Asia use email signed
binaries/Samples/0D610F3F51750EADCF426E10E6DE5313605400FA
signed_pe /vx-underground.org/APTs/2013/2013.05.16 Targeted
information stealing attacks in South Asia use email signed
binaries/Samples/09D4ECA67B1D071E57C5951D97FE9DD9C62F1580
```

```
signed_pe /vx-underground.org/APTs/2014/2014.01.21 h12756-wp-shell-crew/Samples/bf3c20d4ba3b954a17a20da1fd11c3124203f5510ddf53d293f180afc81502fa
signed_pe /vx-underground.org/APTs/2014/2014.12.19 Alert TA14-353A/Samples/6AEAC618E29980B69721158044C2E544
signed_pe /vx-underground.org/APTs/2019/2019.08.29(1) Heatstroke Campaign/Samples/744d44cb8b6f4d9ea547553e8915282762909070 1bc56c386a36264125ed81da
signed_pe /vx-underground.org/APTs/2019/2019.12.11(1) Dropping Anchor/Samples/23e5e8d180d8d7bfaf9cf469aa104ab0ce6a5ee6
signed_pe /vx-underground.org/APTs/2021/2021.08.03(2) APT31 new dropper/Samples/78cc364e761701455bdc4bce100c2836566e662b87b5c28251c178eba2e9ce7e
signed_pe /vx-underground.org/APTs/2021/2021.01.04 APT 27/Samples/e1b44a75947137f4143308d566889837
…
```

Say you're interested in a nuanced API hashing technique and you want to find if any historical APT malware has used something similar. Maybe you borrow a rule or set of rules from the internet as a starting point, such as @tbarabosch research into API hashing.

With a quick check to the corpus, we can see a variety of malware families, operations, campaigns that have used API hashing with crc32, and we can pick and choose which ones we examine for more specific bit and byte details.

```
steve@CEO-MBP ~ % yara -r apihashing_crc32.yar /vx-underground.org/APTs
Methodology_APIHashing_crc32 /vx-underground.org/APTs/2015/2015.10.15 Fin Fishers/Samples/e2ecf89a49c125e0b4292645a41b5e97c0f7bf15d418faeac0d592205f083119
Methodology_APIHashing_crc32 /vx-underground.org/APTs/2015/2015.10.15 Fin Fishers/Samples/94abf6df38f26530da2864d80e1a0b7cdfce63fd27b142993b89c52b3cee0389
Methodology_APIHashing_crc32 /vx-underground.org/APTs/2017/2017.10.16 BlackOasis/Samples/4a49135d2ecc07085a8b7c5925a36c0a
Methodology_APIHashing_crc32 /vx-underground.org/APTs/2019/2019.01.2
```

```
GandCrab and Ursnif
Campaign/Samples/c2464e1aa533ead8d2c2d27a388ae2617616cf17c2c4936b7299
1e33f0c2ccff
Methodology_APIHashing_crc32 /vx-underground.org/APTs/2019/2019.01.24
GandCrab and Ursnif
Campaign/Samples/9cf10f8b2444eaa0c2fbbf199d5fe583923ed14cb92ddd5445b3
e6ffcc86152a
Methodology_APIHashing_crc32 /vx-underground.org/APTs/2019/2019.01.24
GandCrab and Ursnif
Campaign/Samples/ca1381e1e5a506e5fc69278c6989249c5571dc59df872af69bdd
4720a3219e93
Methodology_APIHashing_crc32 /vx-underground.org/APTs/2019/2019.01.24
GandCrab and Ursnif
Campaign/Samples/0b28b28ab38a34c86501ac7eac77464ac77ce36f0cbde16b818d
c8723032cd6b
Methodology_APIHashing_crc32 /vx-underground.org/APTs/2021/2021.01.28
Lebanese Cedar APT/Samples/8ac64a171736252b81c4a559df1f9bae
Methodology_APIHashing_crc32 /vx-underground.org/APTs/2021/2021.01.28
Lebanese Cedar APT/Samples/09a0970bfc1bc8acec1ec609d8d98fda
Methodology_APIHashing_crc32 /vx-underground.org/APTs/2020/2020.07.20
APT attacks on state institutions in Kazakhstan and
Kyrgyzstan/Samples/3e1d66ea09b7c4dbe3c6ffe58262713806564c17
```

Perhaps you've discovered a piece of malware that does some type of string obfuscation shenanigans, and it pushes the name of a DLL like advapi32dll to the stack in four-byte chunks. You're curious to see if this precise technique has been used before, so you use a script to input a list of common DLL names and generate a list of mutated YARA terms into stack string push form:

```
import "pe"
rule TTP_Mutation_StackPush_Windows_DLLs {
 meta:
   author = "Stairwell"
   description = "Searching for PE files with mutations of odd, rare,
or interesting string equities. Here we look for strings from common
PE strings, DLLs and functions in pseudo stack strings form, where
the string pushed onto the stack 4 bytes at a time using PUSH 0x68,
appearing in reverse four byte chunk order, where the PUSH which
shows up as an ASCII letter h."
 strings:
   $a0_kernel32dll = "h.dllhel32hkern" ascii nocase
   $a1_ws2_32dll = "hllh32.dhws2_" ascii nocase
   $a2_msvcrtdll = "hllhrt.dhmsvc" ascii nocase
   $a3_KernelBasedll = "hllhse.dhelBahKern" ascii nocase
   $a4_advapi32dll = "h.dllhpi32hadva" ascii nocase
   $a5_advapires32dll = "hdllhs32.hpirehadva" ascii nocase
   $a6_gdi32dll = "hlh2.dlhgdi3" ascii nocase
   $a7_gdiplusdll = "hdllhlus.hgdip" ascii nocase
   $a8_win32ksys = "hysh2k.shwin3" ascii nocase
   $a9_user32dll = "hllh32.dhuser" ascii nocase
   $a10_comctl32dll = "h.dllhtl32hcomc" ascii nocase
   $a11_commdlgdll = "hdllhdlg.hcomm" ascii nocase
   $a12_comdlg32dll = "h.dllhlg32hcomd" ascii nocase
   $a13_commctrldll = "h.dllhctrlhcomm" ascii nocase
   $a14_shelldll = "hlhl.dlhshel" ascii nocase
   $a15_shell32dll = "hdllhl32.hshel" ascii nocase
   $a16_shlwapidll = "hdllhapi.hshlw" ascii nocase
   $a17_netapi32dll = "h.dllhpi32hneta" ascii nocase
   $a18_shdocvwdll = "hdllhcvw.hshdo" ascii nocase
   $a19_mshtmldll = "hllhml.dhmsht" ascii nocase
   $a20_urlmondll = "hllhon.dhurlm" ascii nocase
   $a21_iphlpapidll = "h.dllhpapihiphl" ascii nocase
   $a22_httpapidll = "hdllhapi.hhttp" ascii nocase
   $a23_msvbvm60dll = "h.dllhvm60hmsvb" ascii nocase
   $a24_shfolderdll = "h.dllhlderhshfo" ascii nocase
   $a25_OLE32DLL = "hLh2.DLhOLE3" ascii nocase
   $a26_wininetdll = "hdllhnet.hwini" ascii nocase
   $a27_wsock32dll = "hdllhk32.hwsoc" ascii nocase
 condition:
   filesize < 15MB
   and uint16be(0) == 0x4d5a
   and 1 of them
}
```

With your handy new rule, you can smash this rule against your vx-underground corpus and boom, you’ve got a couple of leads and a couple of samples you can look at further.

```
steve@LFO-MBP ~ % yara -r -s pipe.yar /vx-underground.org/
Methodology_Mutation_Stack_Type2_Windows_DLLs /vx-
underground.org//APTs/2015/2015.11.09 Rocket
Kitten/Samples/f89a4d4ae5cca6d69a5256c96111e707
0x7571:$a4_advapi32dll: h.dllhpi32hadva
Methodology_Mutation_Stack_Type2_Windows_DLLs /vx-
underground.org//APTs/2015/2015.11.09 Rocket
Kitten/Samples/e8dbcde49c7f760165ebb0cb3452e4f1c24981f5
0x7571:$a4_advapi32dll: h.dllhpi32hadva
Methodology_Mutation_Stack_Type2_Windows_DLLs
/vx-underground.org//APTs/2015/2015.03.19 Goldfish
Phishing/Samples/e8dbcde49c7f760165ebb0cb3452e4f1c24981f5
0x7571:$a4_advapi32dll: h.dllhpi32hadva
Methodology_Mutation_Stack_Type2_Windows_DLLs /vx-
underground.org//APTs/2019/2019.03.04 APT
40/Samples/ab37ed19d7300e673e66dbd4fec990b1
0x777:$a0_kernel32dll: h.dllhel32hkern
```

Afterthoughts

We gathered and tidied up a quick and dirty malware corpus, and we’re using YARA rules to search through all of it to help understand and contextualize our ideas across a decade of notable malware files from public reporting. We took this approach with YARA and a hacked-out directory tree for a visual context clue out of necessity. We love YARA because it satisfies the gaps and shortcomings in existing security tooling, and we like the vx-underground malware set because it is organized, relatively labeled, and open to the world. We would not be taking this approach if there were better, easier “bulk analysis” solutions available for free.

We understand, accept, and admit that there are plenty of flaws and biases in this approach. Here, when we scan our corpus of mostly “Big APT” malware, we may be prone to thinking that something is more rare, targeted, and elite than it truly is. What we search for using YARA may be present in tons of malware but not actually be malicious at all, and the results obtained from our curated APT collection do not really give us that perspective.

On the other hand, it may still be helpful to know that a so-called advanced adversary has used a particular <thing> before, and with a quick YARA search, we can find samples in which a given <thing> is used and take that as a prompt for further analysis. With clever YARA rules for any given technique, value, feature, or idea, we believe these quick litmus tests on the labeled corpus hold promise in speeding up detection and threat intelligence research.

Q&A

My goodness, smiller, won't this set my poor laptop on fire?

It might, but *probably* not. On a newish MacBook, scans with single, intensive rules (such as the API hashing rules) take 30-40 seconds to complete, which is not awful. Mileage may vary.

Wouldn't it be so much better to use a more elegant solution involving a small database and a web app or something?

Probably, maybe. This is a quick and dirty approach that uses the least amount of tooling possible, and accordingly, it is #basic in all the best and worst ways. We are inclined to believe that fancier, more technically nuanced implementations of more richly labeled data will indeed provide a far superior analysis experience, yet it is also important to have an approach that works for people who do not have the resources to invest in a grander technical solution.

This is all well and good for malware, but how can I do this with a goodware corpus?

Great question. We're so glad you asked. As you experiment with YARA rules against a malware corpus, you will no doubt want to balance your research across a goodware corpus too. This is especially important if you intend to study file anomalies. There are a few ways that you can build a goodware corpus. You may consider creating fresh virtual machine installations of Windows 10, Windows Server, and Ubuntu, then zipping up everything into a folder structure that you can test against alongside your vx-underground corpus. It's kind of a pain, but again, helpful for litmus testing your ideas.

—

Steve is a researcher and intelligence analyst focused on understanding and detecting adversary tradecraft.

Stairwell

Any Questions?

CONTACT US

Subscribe

Enter email

Stay connected

Privacy Policy / Terms