MANDIANT

Lightweight Binary Similarity

YARA Using PE Features for Quick Wins

Connor McLaughlin

Manager, Operational Outcomes,
Insikt Group, Recorded Future

Greg Lesnewich

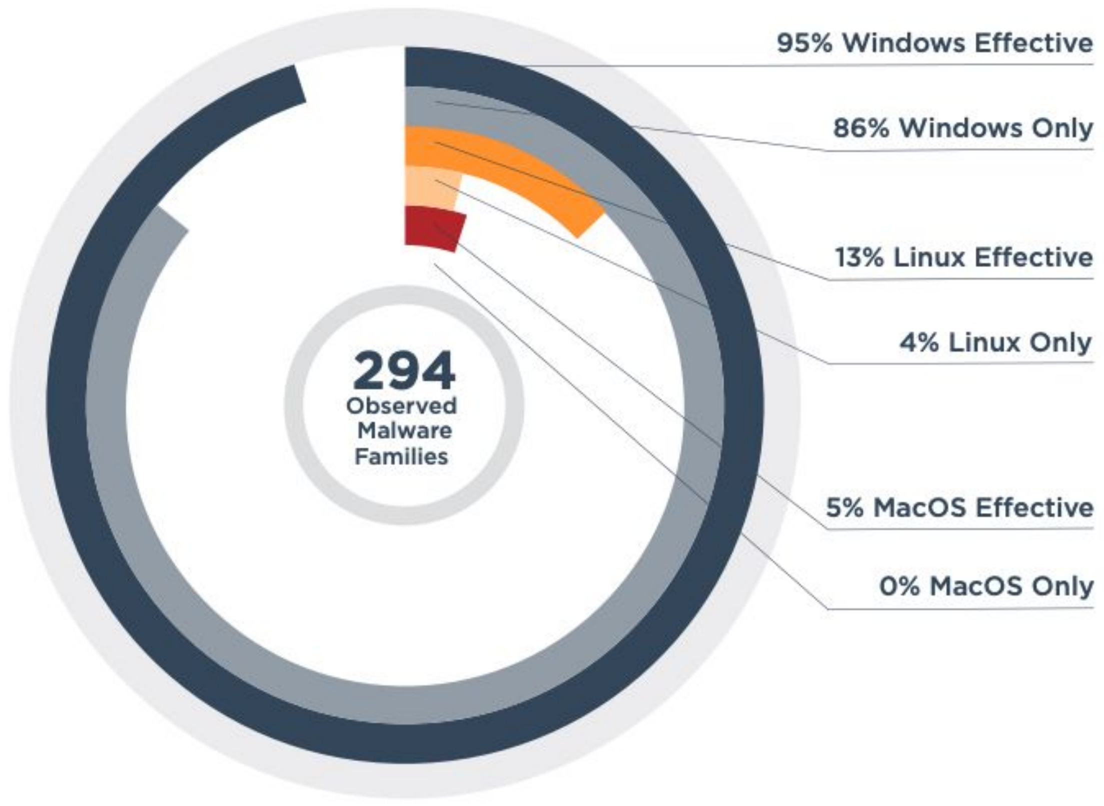
Analyst, Operational Outcomes, Insikt
Group, Recorded Future

Goals

- Speed up the YARA writing process
- Help those newer to YARA learn more ways to use the tool
- Assess a large dataset of known malware for insights

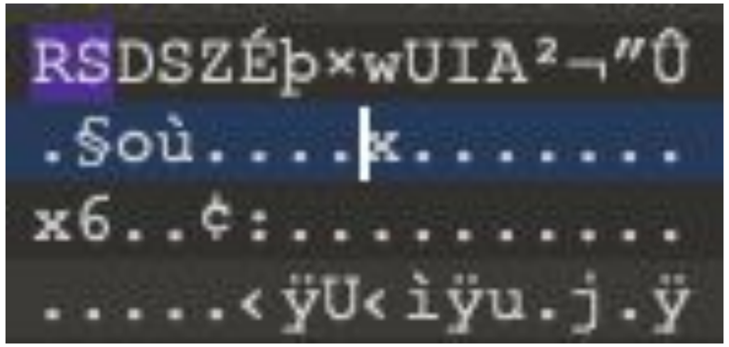
Why PEs?

EFFECTIVENESS OF OBSERVED MALWARE FAMILIES BY OPERATING SYSTEM, 2020

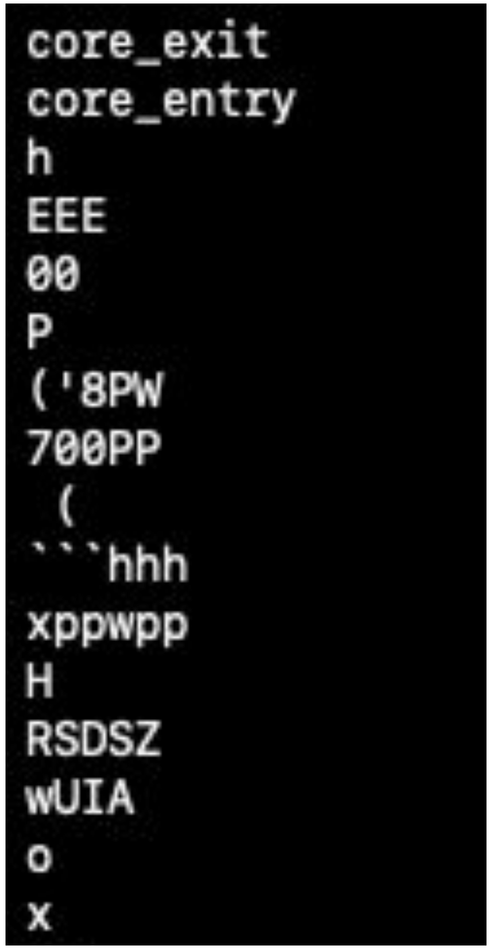


Position-Specific Artifacts

- Environment exhaust
- Seeing ‘BINARY’ or ‘EXE’ in a strings output doesn’t mean much
  - Vs. seeing it as a resource name string
- Seeing a benign DLL name in strings
  - vs. at the the start of IMAGE_DIRECTORY_ENTRY_EXPORT
- When is a string… more than a string?



```
RSDSZÉþ×wUIA²¬”Û
.§où....ĸ........
x6..¢:...........
.....‹ÿU‹ìÿu.j.ÿ
```



~~~
core_exit
core_entry
h
EEE
00
P
('8PW
700PP
 (
```hhh
xppwpp
H
RSDSZ
wUIA
o
x
~~~

Features with Developer Fingerprints

Reflect what the malware developer names their tools / functions

- DLL Name
- PDB Path
- Original Name

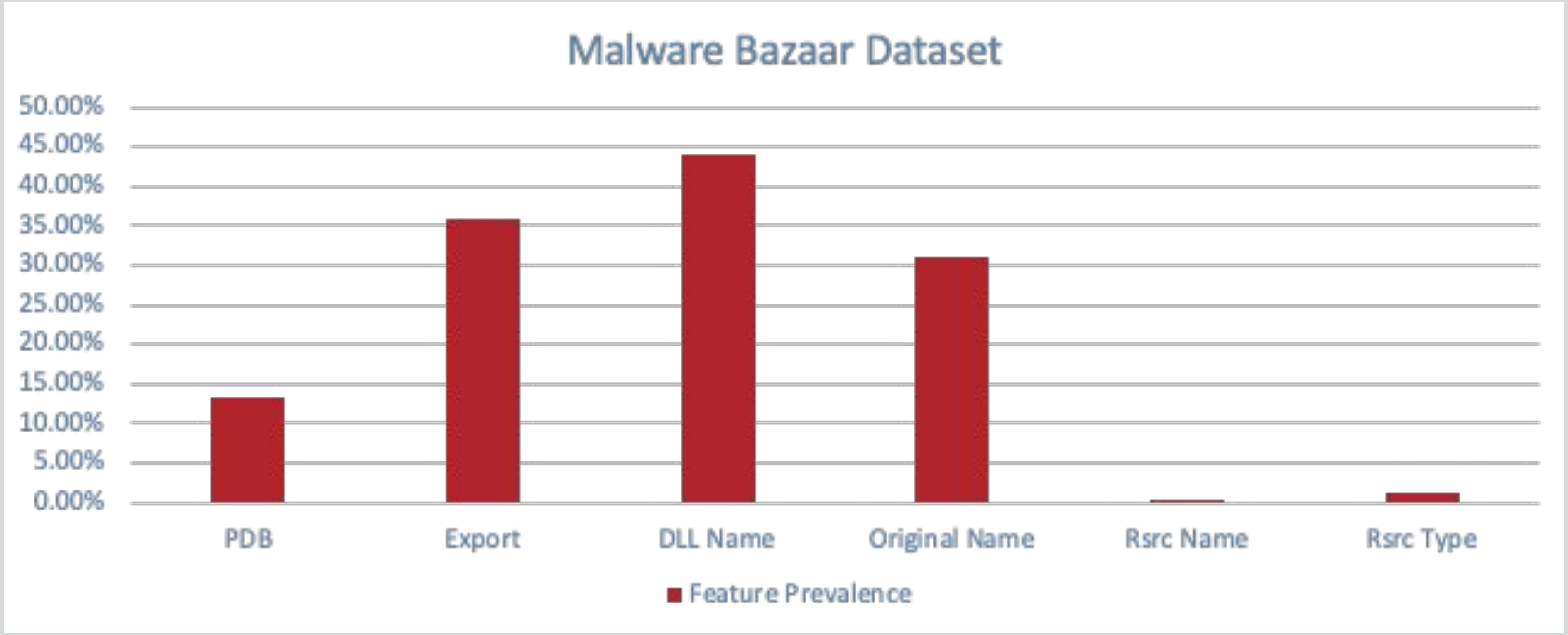
Show how a developer refers to, or attempts to hide, additional components

- Resource Name Strings
- Resource Type Strings

Choice of build environment

- Rich header data

How Common Are These Features?



Disclaimer

Overlap in features does NOT imply that the:

- Malware families are the same
- There is shared code among the samples

Presence of a feature != maliciousness

Searching for Artifacts with YARA

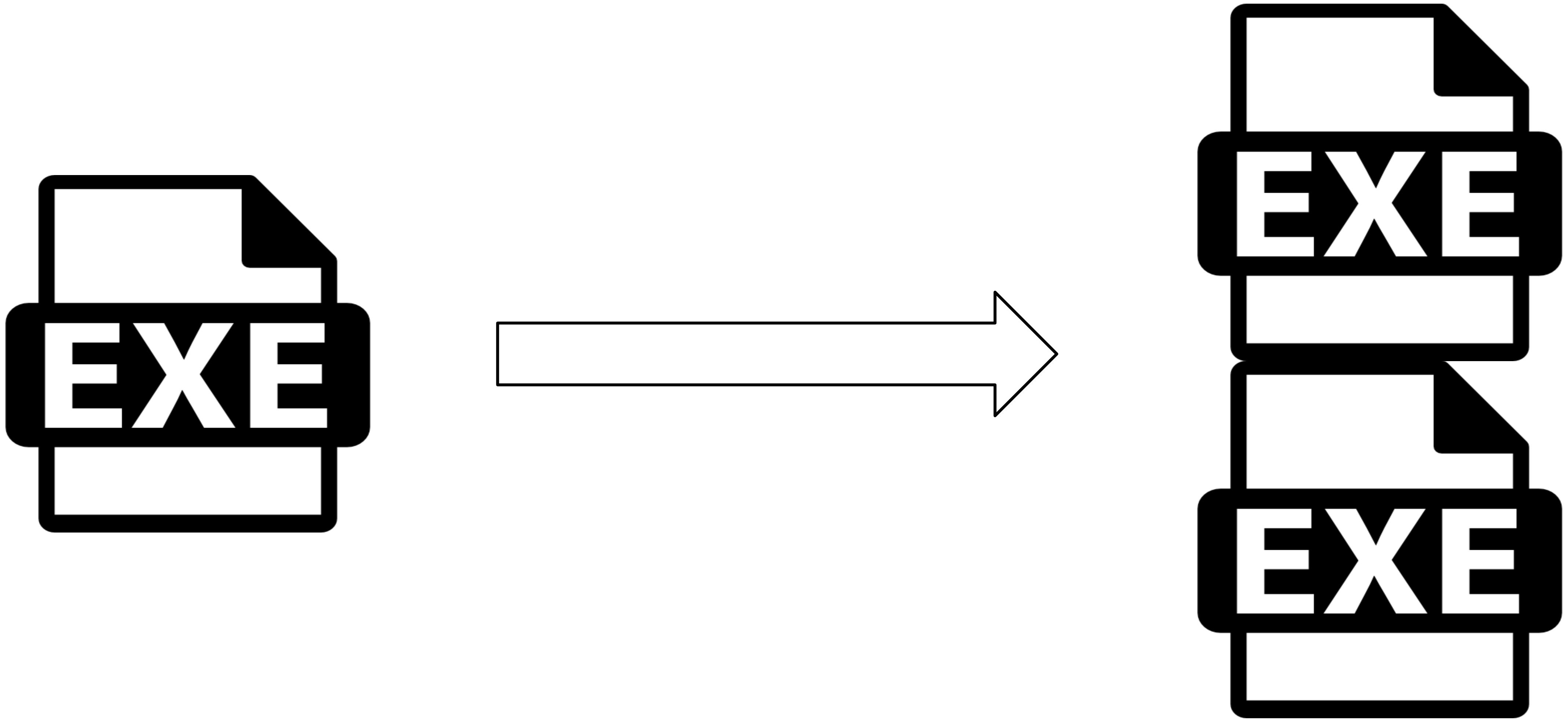
YARA

- Common language for pattern matching across malware repositories, incident response, and enterprise security tools like EDRs and email scanning

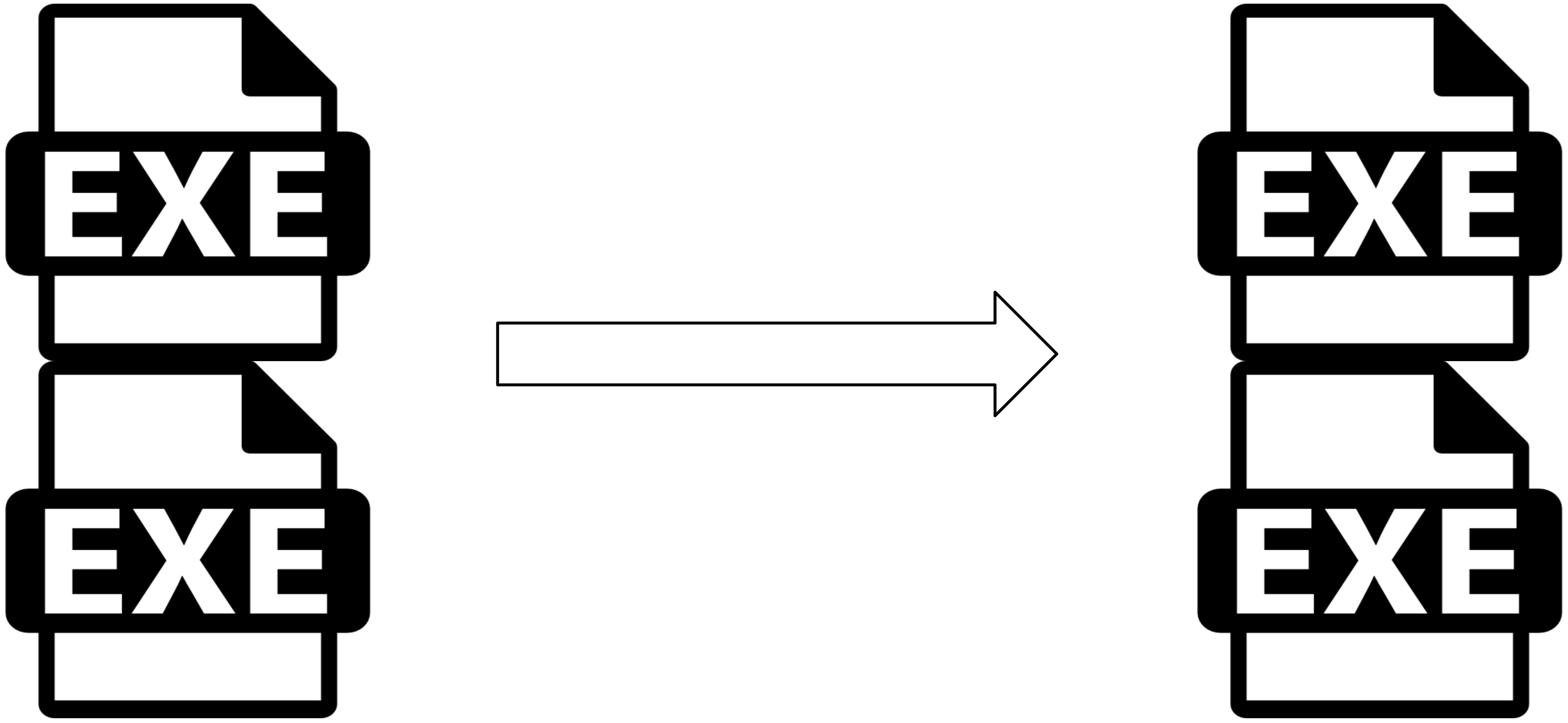
- YARA has modules to extend its capabilities:
  - “pe” Module
  - “dotnet” Module
  - “hash” Module

- YARA 4.0 or higher for some of these features

Incident Response YARA Workflow



Threat Intelligence YARA Workflow



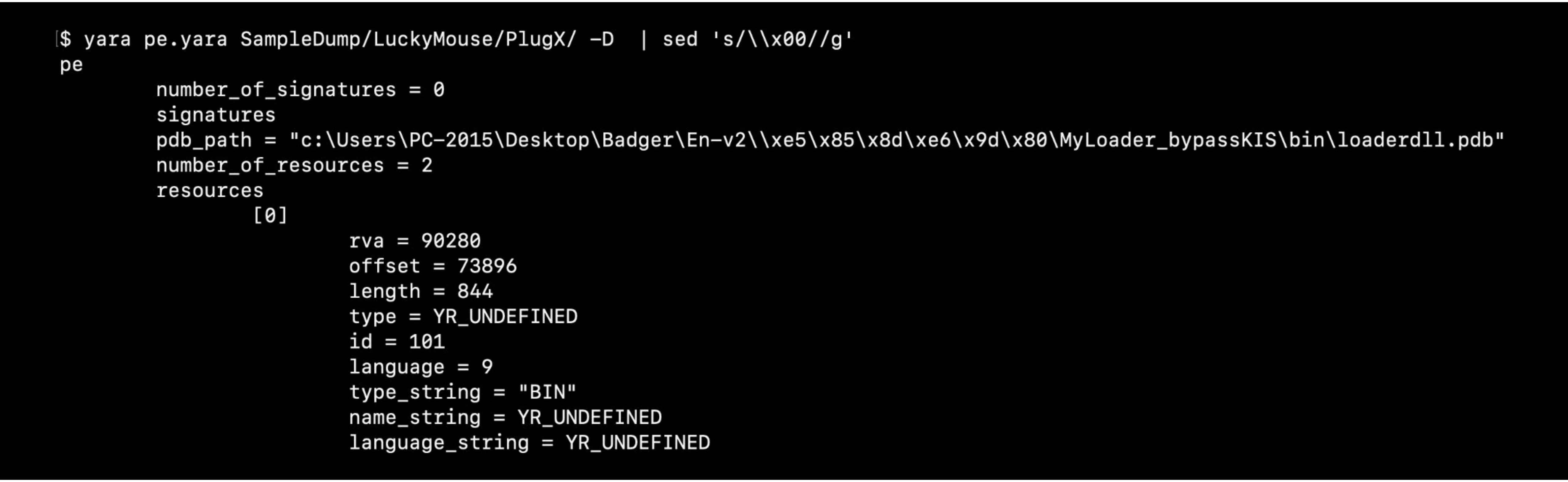
YARA as a Powerful Analysis Tool

- Producing YARA rules is often a goal of our analysis
- What if we used it as part of analysis & triage?
- YARA PE Module allows us to quick find overlaps in observed metadata

How-To?

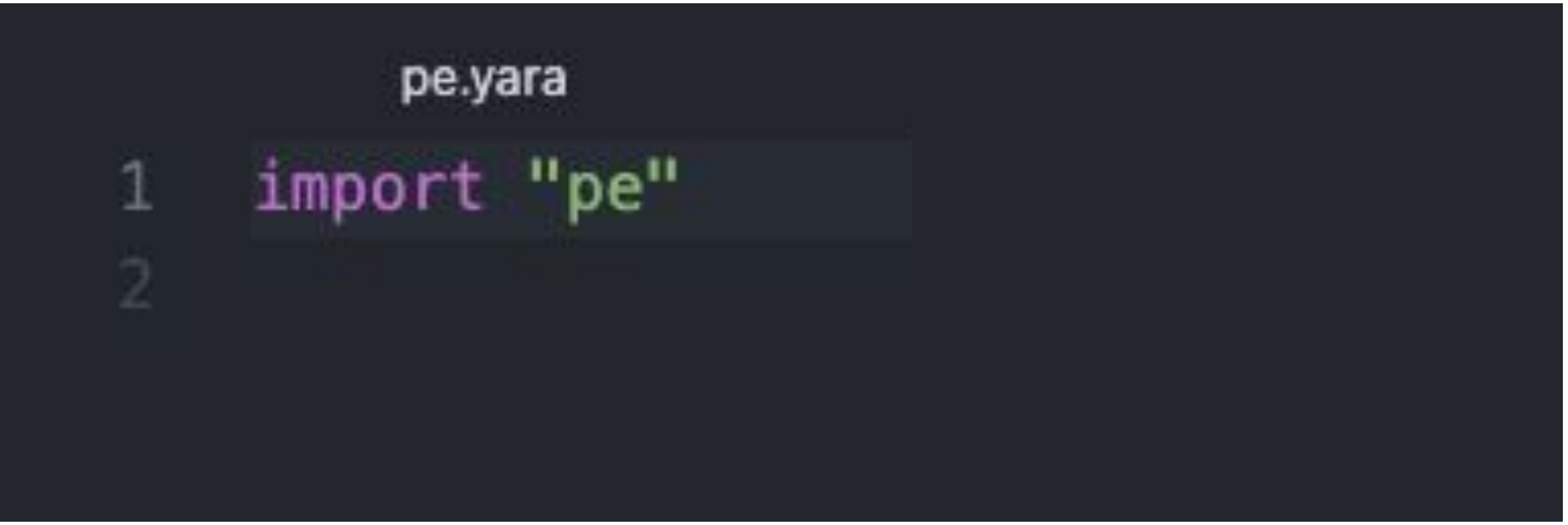
- YARA’s -D switch will already tells us how it sees a file
- Can also run it recursively (-r)
- Process the output using Bash utilities like grep, sort, and uniq

YARA’s –D Flag



```
[$ yara pe.yara SampleDump/LuckyMouse/PlugX/ -D  | sed 's/\\x00//g'
pe
        number_of_signatures = 0
        signatures
        pdb_path = "c:\Users\PC-2015\Desktop\Badger\En-v2\\xe5\x85\x8d\xe6\x9d\x80\MyLoader_bypassKIS\bin\loaderdll.pdb"
        number_of_resources = 2
        resources
                [0]
                        rva = 90280
                        offset = 73896
                        length = 844
                        type = YR_UNDEFINED
                        id = 101
                        language = 9
                        type_string = "BIN"
                        name_string = YR_UNDEFINED
                        language_string = YR_UNDEFINED
```

Start Simple


pe.yara

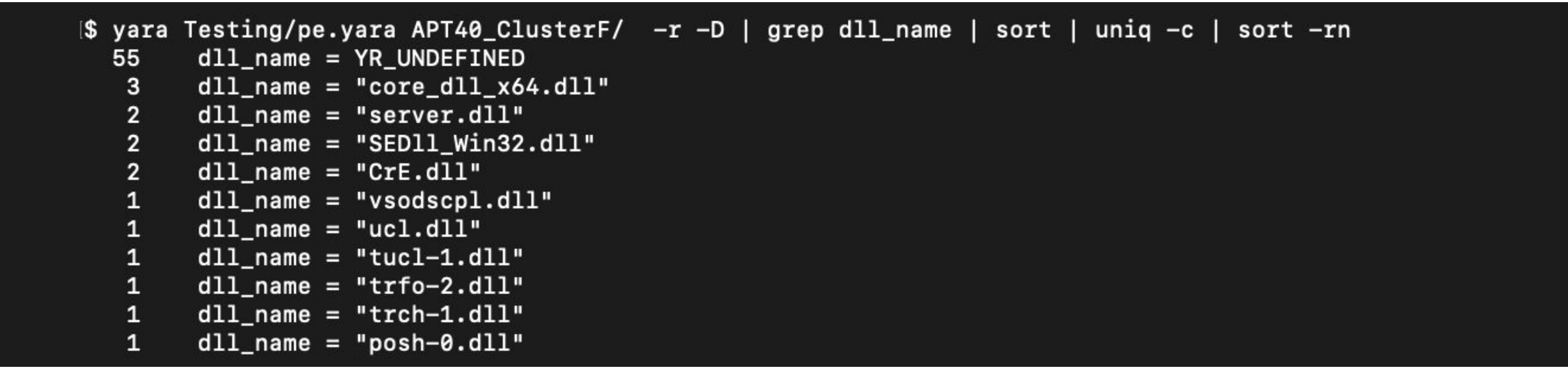
```
import "pe"

```

1. Grab a folder with samples of interest
2. Run YARA using the pe.yara file and -D output
3. Process the output: **grep <feature> | sort | uniq -c | sort -rn**

DLL Name

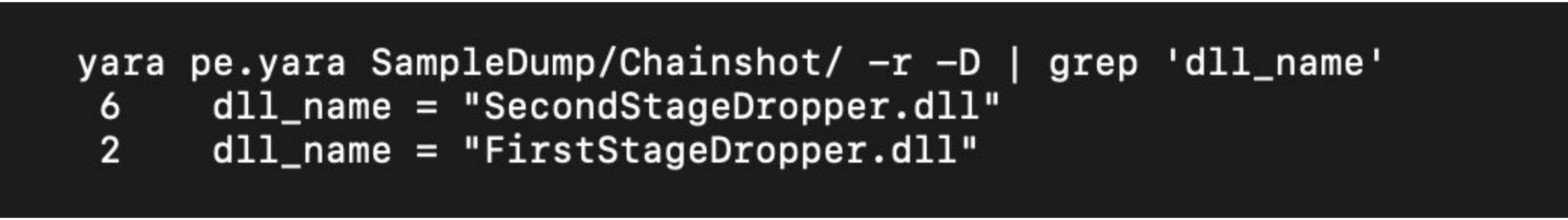
yara pe.yara APT40_ClusterF/ -r -D | grep **dll_name** | sort | uniq -c | sort -rn



```
[$ yara Testing/pe.yara APT40_ClusterF/  -r -D | grep dll_name | sort | uniq -c | sort -rn
  55     dll_name = YR_UNDEFINED
   3     dll_name = "core_dll_x64.dll"
   2     dll_name = "server.dll"
   2     dll_name = "SEDll_Win32.dll"
   2     dll_name = "CrE.dll"
   1     dll_name = "vsodscpl.dll"
   1     dll_name = "ucl.dll"
   1     dll_name = "tucl-1.dll"
   1     dll_name = "trfo-2.dll"
   1     dll_name = "trch-1.dll"
   1     dll_name = "posh-0.dll"
```

```
rule PE_DLL_Name {
condition: pe.dll_name == "EXAMPLE"
}
```

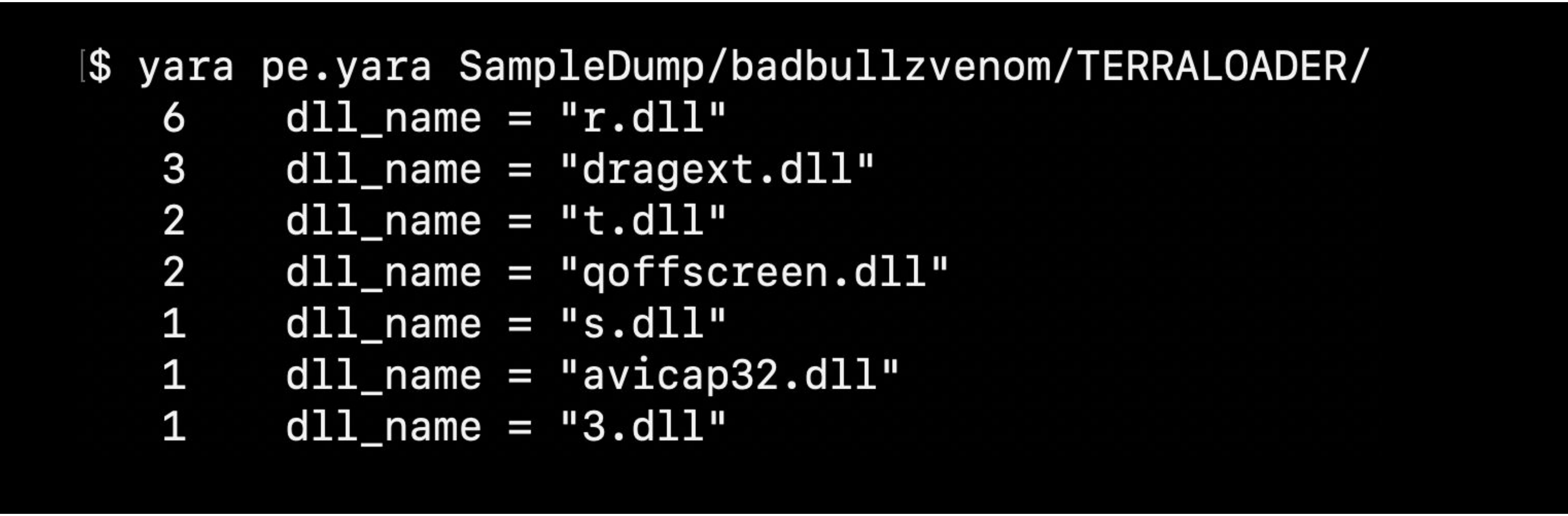
Chainshot



```
yara pe.yara SampleDump/Chainshot/ -r -D | grep 'dll_name'
 6     dll_name = "SecondStageDropper.dll"
 2     dll_name = "FirstStageDropper.dll"
```

TerraLoader

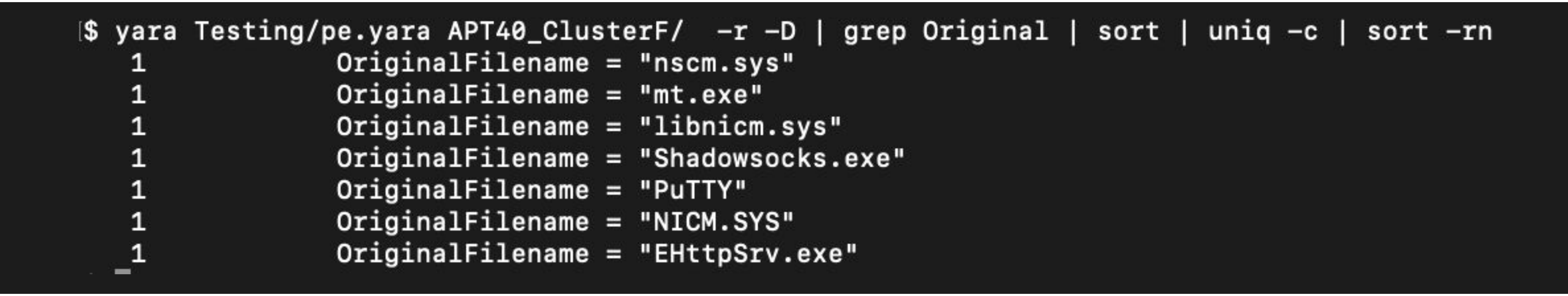
Odd scrubbing of artifacts



```
[$ yara pe.yara SampleDump/badbullzvenom/TERRALOADER/
   6     dll_name = "r.dll"
   3     dll_name = "dragext.dll"
   2     dll_name = "t.dll"
   2     dll_name = "qoffscreen.dll"
   1     dll_name = "s.dll"
   1     dll_name = "avicap32.dll"
   1     dll_name = "3.dll"
```

Original Name

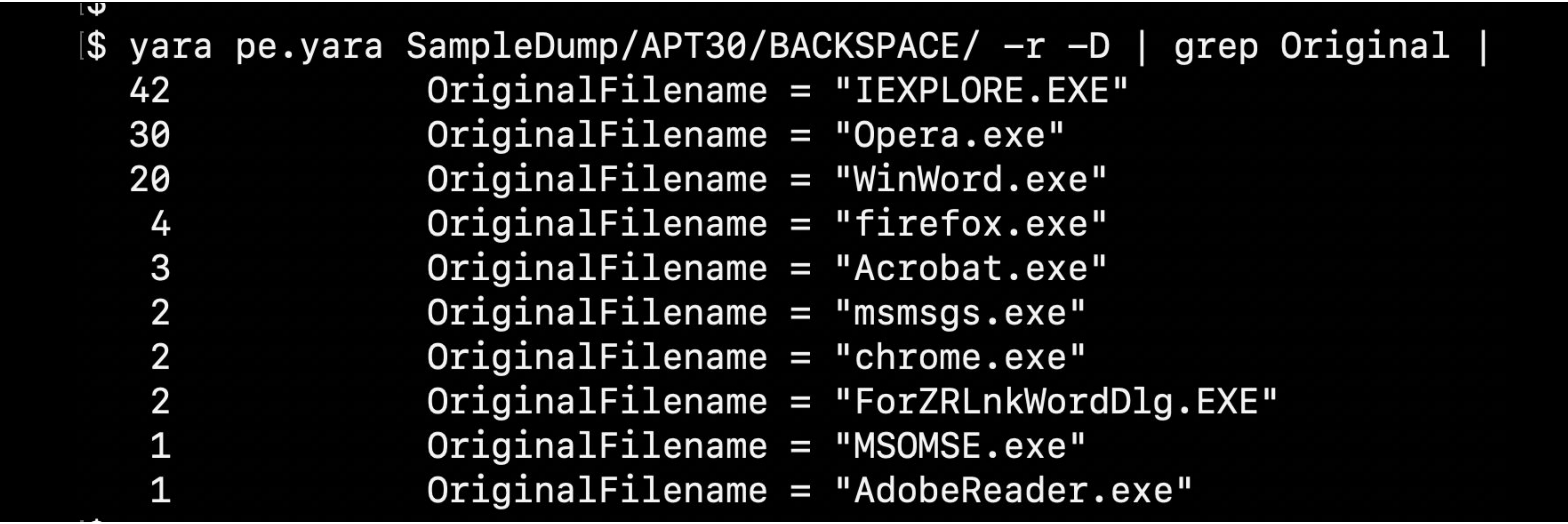
yara pe.yara APT40_ClusterF/ -r -D | grep **Original** | sort | uniq -c | sort -rn



```
$ yara Testing/pe.yara APT40_ClusterF/  -r -D | grep Original | sort | uniq -c | sort -rn
   1             OriginalFilename = "nscm.sys"
   1             OriginalFilename = "mt.exe"
   1             OriginalFilename = "libnicm.sys"
   1             OriginalFilename = "Shadowsocks.exe"
   1             OriginalFilename = "PuTTY"
   1             OriginalFilename = "NICM.SYS"
   1             OriginalFilename = "EHttpSrv.exe"
```

```
rule PE_Original_Filename  {
condition: pe.version_info["OriginalFilename"] == "EXAMPLE"
}
```

BACKSPACE Mimics Browsers



```
$ yara pe.yara SampleDump/APT30/BACKSPACE/ -r -D | grep Original |
  42             OriginalFilename = "IEXPLORE.EXE"
  30             OriginalFilename = "Opera.exe"
  20             OriginalFilename = "WinWord.exe"
   4             OriginalFilename = "firefox.exe"
   3             OriginalFilename = "Acrobat.exe"
   2             OriginalFilename = "msmsgs.exe"
   2             OriginalFilename = "chrome.exe"
   2             OriginalFilename = "ForZRLnkWordDlg.EXE"
   1             OriginalFilename = "MSOMSE.exe"
   1             OriginalFilename = "AdobeReader.exe"
```

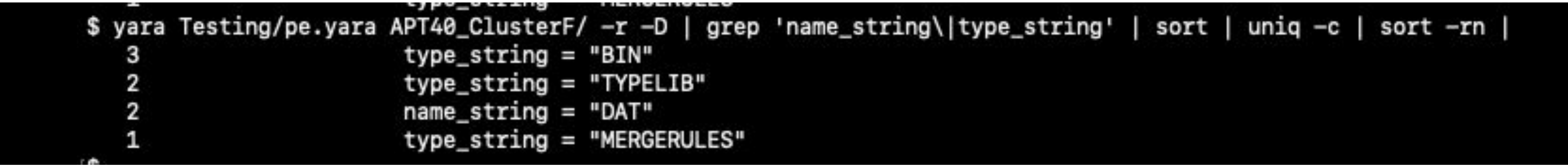
Careto / The Mask

Odd scrubbing of features:

- DLL Name: **InternalModuleNameDll**.dll
- Original Name: **OriginalFilename**.dll

Resource String Identifiers

yara pe.yara APT40_ClusterF/ -r -D | grep '**name_string\|type_string**' | sort | uniq -c | sort -rn | **sed 's/\\x00//g'**

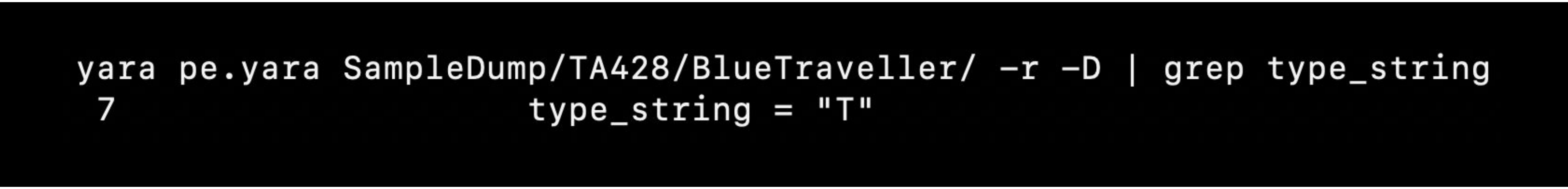


```
$ yara Testing/pe.yara APT40_ClusterF/ -r -D | grep 'name_string\|type_string' | sort | uniq -c | sort -rn |
   3                    type_string = "BIN"
   2                    type_string = "TYPELIB"
   2                    name_string = "DAT"
   1                    type_string = "MERGERULES"
```

```
rule PE_Resource_String_Identifier  {
  condition:
    for any resource in pe.resources:
      (resource.type_string == "E\x00X\x00A\x00M\x00P\x00L\x00E\x00")
}
```

Blue Traveller

Single char resource strings are suspect

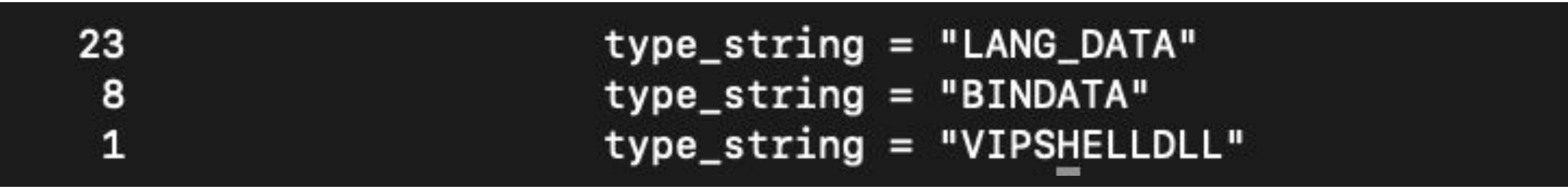


```
yara pe.yara SampleDump/TA428/BlueTraveller/ -r -D | grep type_string
 7                    type_string = "T"
```

Exforel Rootkit

OriginalFilename: "Adobe Update.exe“

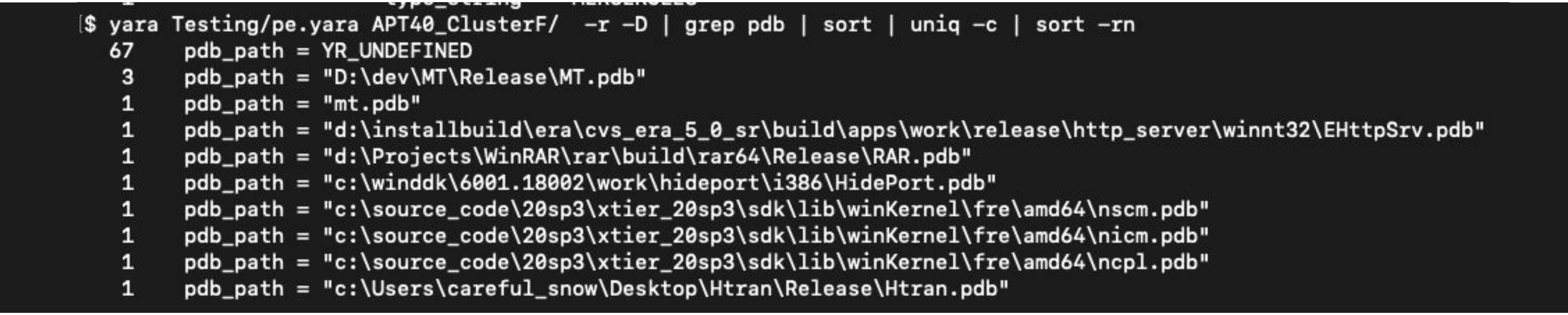
Odd resource strings:



```
23                    type_string = "LANG_DATA"
 8                    type_string = "BINDATA"
 1                    type_string = "VIPSHELLDLL"
```

Ye Olde PDB Path

yara pe.yara APT40_ClusterF/ -r -D | grep **'pdb_path'** | sort | uniq -c | sort -rn



```
[$ yara Testing/pe.yara APT40_ClusterF/  -r -D | grep pdb | sort | uniq -c | sort -rn
  67     pdb_path = YR_UNDEFINED
   3     pdb_path = "D:\dev\MT\Release\MT.pdb"
   1     pdb_path = "mt.pdb"
   1     pdb_path = "d:\installbuild\era\cvs_era_5_0_sr\build\apps\work\release\http_server\winnt32\EHttpSrv.pdb"
   1     pdb_path = "d:\Projects\WinRAR\rar\build\rar64\Release\RAR.pdb"
   1     pdb_path = "c:\winddk\6001.18002\work\hideport\i386\HidePort.pdb"
   1     pdb_path = "c:\source_code\20sp3\xtier_20sp3\sdk\lib\winKernel\fre\amd64\nscm.pdb"
   1     pdb_path = "c:\source_code\20sp3\xtier_20sp3\sdk\lib\winKernel\fre\amd64\nicm.pdb"
   1     pdb_path = "c:\source_code\20sp3\xtier_20sp3\sdk\lib\winKernel\fre\amd64\ncpl.pdb"
   1     pdb_path = "c:\Users\careful_snow\Desktop\Htran\Release\Htran.pdb"
```

rule PE_PDB_Path {

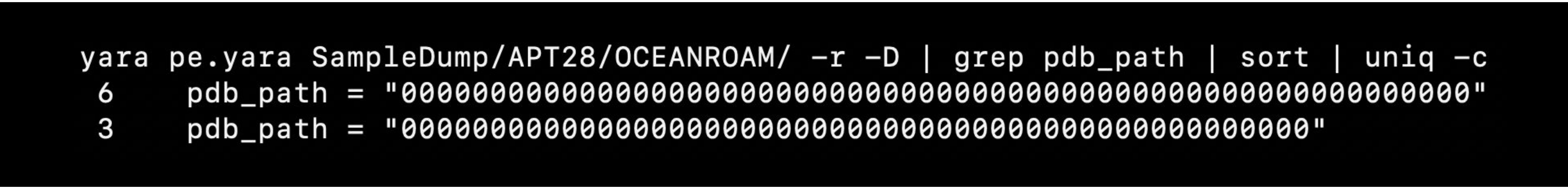
condition: pe.**pdb_path** == "EXAMPLE"

}

OCEANMAP & OCEANDRIVE

Oddly scrubbed PDB strings

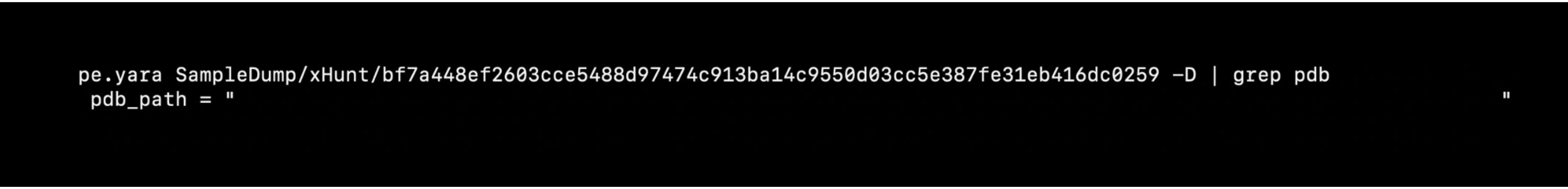
Easier to observe this in bulk:



```
yara pe.yara SampleDump/APT28/OCEANROAM/ -r -D | grep pdb_path | sort | uniq -c
 6    pdb_path = "000000000000000000000000000000000000000000000000000000000000"
 3    pdb_path = "000000000000000000000000000000000000000000000000000"
```

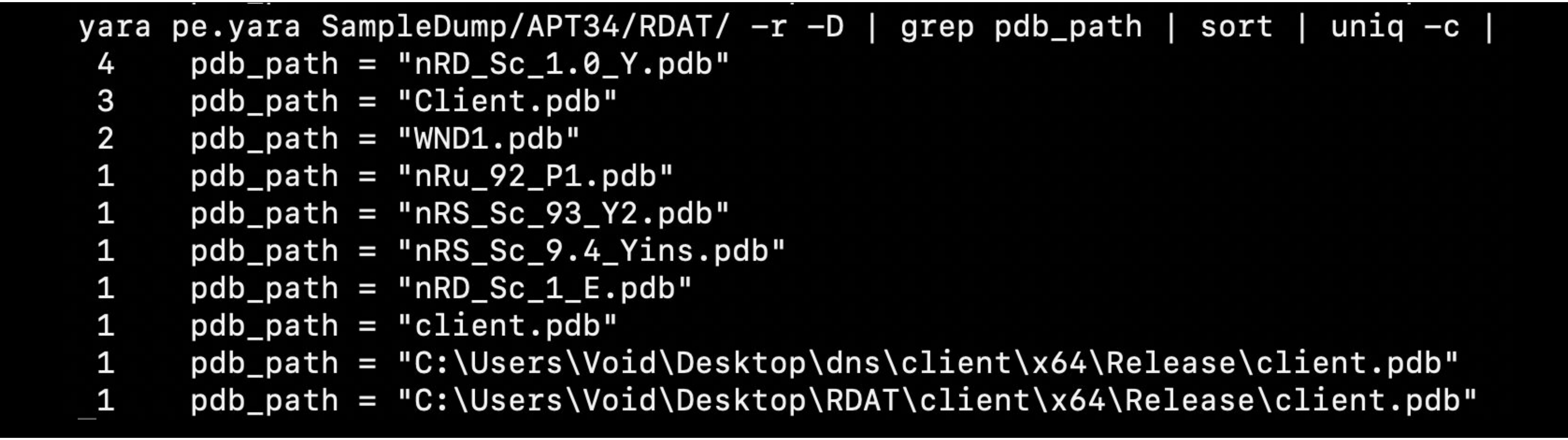
xHunt

More odd scrubbing



```
pe.yara SampleDump/xHunt/bf7a448ef2603cce5488d97474c913ba14c9550d03cc5e387fe31eb416dc0259 -D | grep pdb
 pdb_path = "                                                                                          "
```

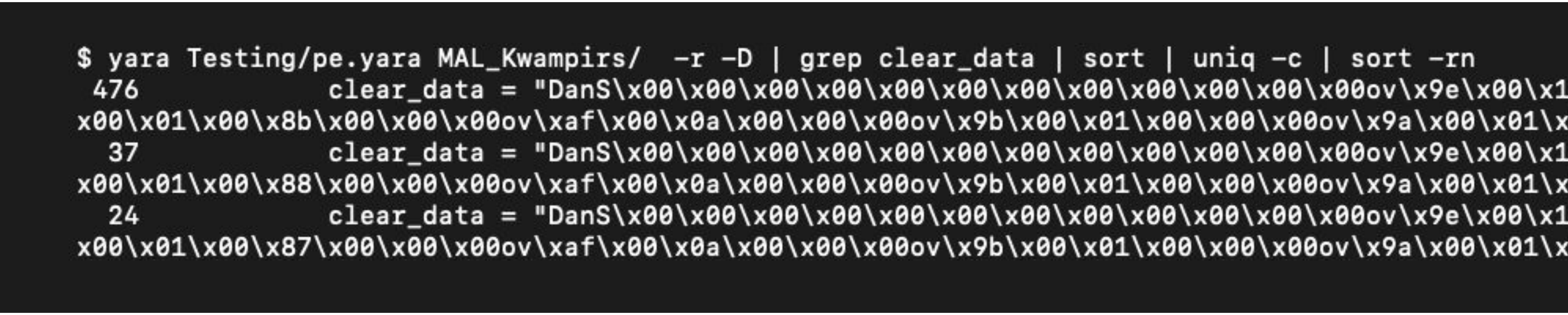
Everyone’s Favorite Group



```
yara pe.yara SampleDump/APT34/RDAT/ -r -D | grep pdb_path | sort | uniq -c |
 4     pdb_path = "nRD_Sc_1.0_Y.pdb"
 3     pdb_path = "Client.pdb"
 2     pdb_path = "WND1.pdb"
 1     pdb_path = "nRu_92_P1.pdb"
 1     pdb_path = "nRS_Sc_93_Y2.pdb"
 1     pdb_path = "nRS_Sc_9.4_Yins.pdb"
 1     pdb_path = "nRD_Sc_1_E.pdb"
 1     pdb_path = "client.pdb"
 1     pdb_path = "C:\Users\Void\Desktop\dns\client\x64\Release\client.pdb"
 1     pdb_path = "C:\Users\Void\Desktop\RDAT\client\x64\Release\client.pdb"
```

Rich Headers

yara pe.yara MAL_Kwampirs/ -r -D | grep **clear_data** | sort | uniq -c | sort -rn



```
$ yara Testing/pe.yara MAL_Kwampirs/  -r -D | grep clear_data | sort | uniq -c | sort -rn
 476            clear_data = "DanS\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00ov\x9e\x00\x1
x00\x01\x00\x8b\x00\x00\x00ov\xaf\x00\x0a\x00\x00\x00ov\x9b\x00\x01\x00\x00\x00ov\x9a\x00\x01\x
  37            clear_data = "DanS\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00ov\x9e\x00\x1
x00\x01\x00\x88\x00\x00\x00ov\xaf\x00\x0a\x00\x00\x00ov\x9b\x00\x01\x00\x00\x00ov\x9a\x00\x01\x
  24            clear_data = "DanS\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00ov\x9e\x00\x1
x00\x01\x00\x87\x00\x00\x00ov\xaf\x00\x0a\x00\x00\x00ov\x9b\x00\x01\x00\x00\x00ov\x9a\x00\x01\x
```

rule PE_Rich_Header_Hash {

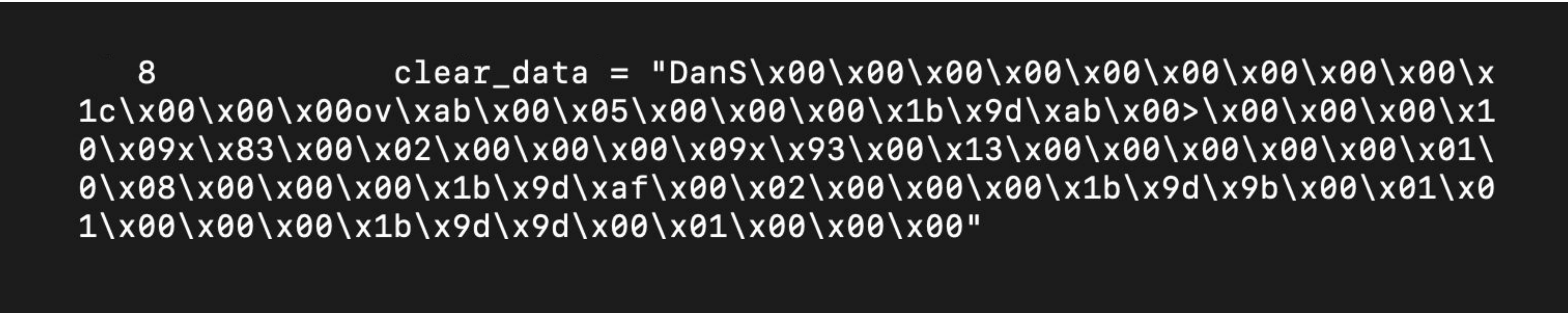
condition: pe.**rich_signature.clear_data** == "DanS\x00 .. "

}

WildPressure & Milum

Case Study for Rich Header Usage

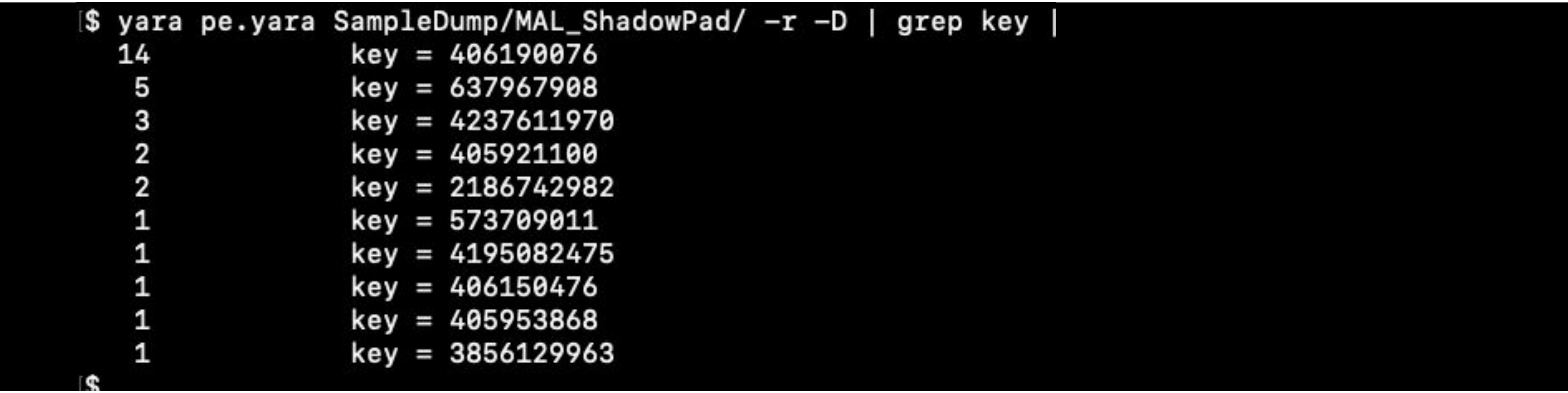
Impervious to use of UPX



```
                 clear_data = "DanS\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
1c\x00\x00\x00ov\xab\x00\x05\x00\x00\x00\x1b\x9d\xab\x00>\x00\x00\x00\x1
0\x09x\x83\x00\x02\x00\x00\x00\x09x\x93\x00\x13\x00\x00\x00\x00\x00\x01\
0\x08\x00\x00\x00\x1b\x9d\xaf\x00\x02\x00\x00\x00\x1b\x9d\x9b\x00\x01\x0
1\x00\x00\x00\x1b\x9d\x9d\x00\x01\x00\x00\x00"
```

Also: DLL Name == "Milum46_Win32.exe"

And the XOR Key!

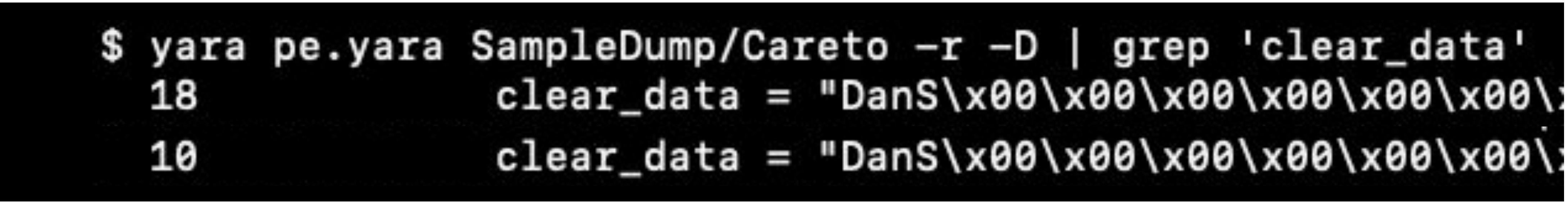


```
[$ yara pe.yara SampleDump/MAL_ShadowPad/ -r -D | grep key |
  14                key = 406190076
   5                key = 637967908
   3                key = 4237611970
   2                key = 405921100
   2                key = 2186742982
   1                key = 573709011
   1                key = 4195082475
   1                key = 406150476
   1                key = 405953868
   1                key = 3856129963
[$
```

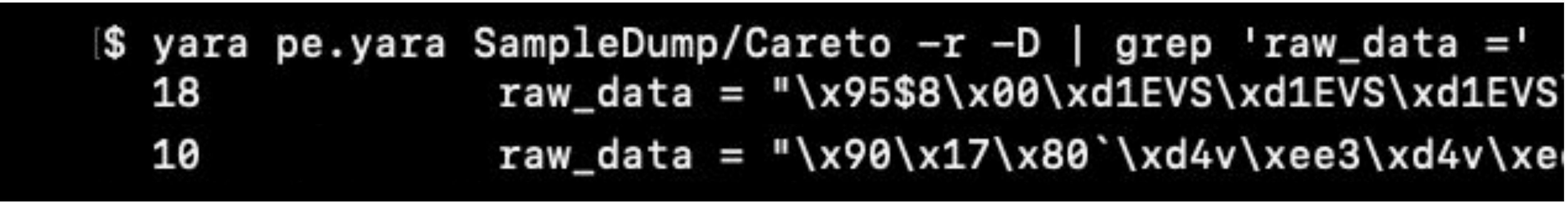
```
rule PE_Rich_Header_XOR_Key {
condition: pe.rich_signature.key == 871266004     // decimal notation
or pe.rich_signature.key == 0x33EE76D4        // hex notation
}
```

XOR Key != Rich Header Hash

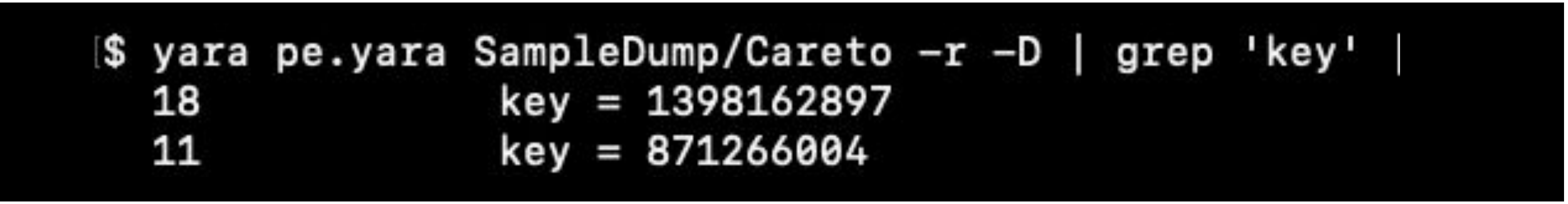
Useful to look at both



```
$ yara pe.yara SampleDump/Careto -r -D | grep 'clear_data'
  18               clear_data = "DanS\x00\x00\x00\x00\x00\x00\
  10               clear_data = "DanS\x00\x00\x00\x00\x00\x00\
```



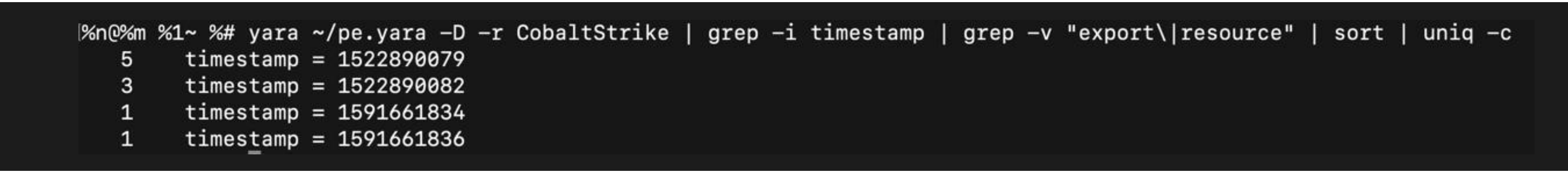
```
[$ yara pe.yara SampleDump/Careto -r -D | grep 'raw_data ='
  18               raw_data = "\x95$8\x00\xd1EVS\xd1EVS\xd1EVS
  10               raw_data = "\x90\x17\x80`\xd4v\xee3\xd4v\xe
```



```
[$ yara pe.yara SampleDump/Careto -r -D | grep 'key' |
  18               key = 1398162897
  11               key = 871266004
```

Compile Timestamp

yara pe.yara <path to samples> -D -r | grep -i timestamp | grep -v “export\|resource” | sort | uniq -c



```
%n@%m %1~ %# yara ~/pe.yara -D -r CobaltStrike | grep -i timestamp | grep -v "export\|resource" | sort | uniq -c
   5     timestamp = 1522890079
   3     timestamp = 1522890082
   1     timestamp = 1591661834
   1     timestamp = 1591661836
```

rule PE_Timestamp {

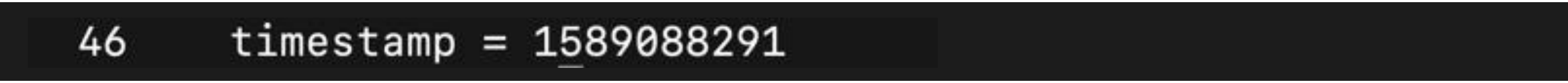
condition: pe.**timestamp** == 1522890079

}

AsyncRAT

Deterministic builds cause timestamps to be a hash of compilation inputs

yara pe.yara AsyncRAT/ -r -D | grep **timestamp** | grep -v “export\|resource” | sort | uniq -c | sort -rn

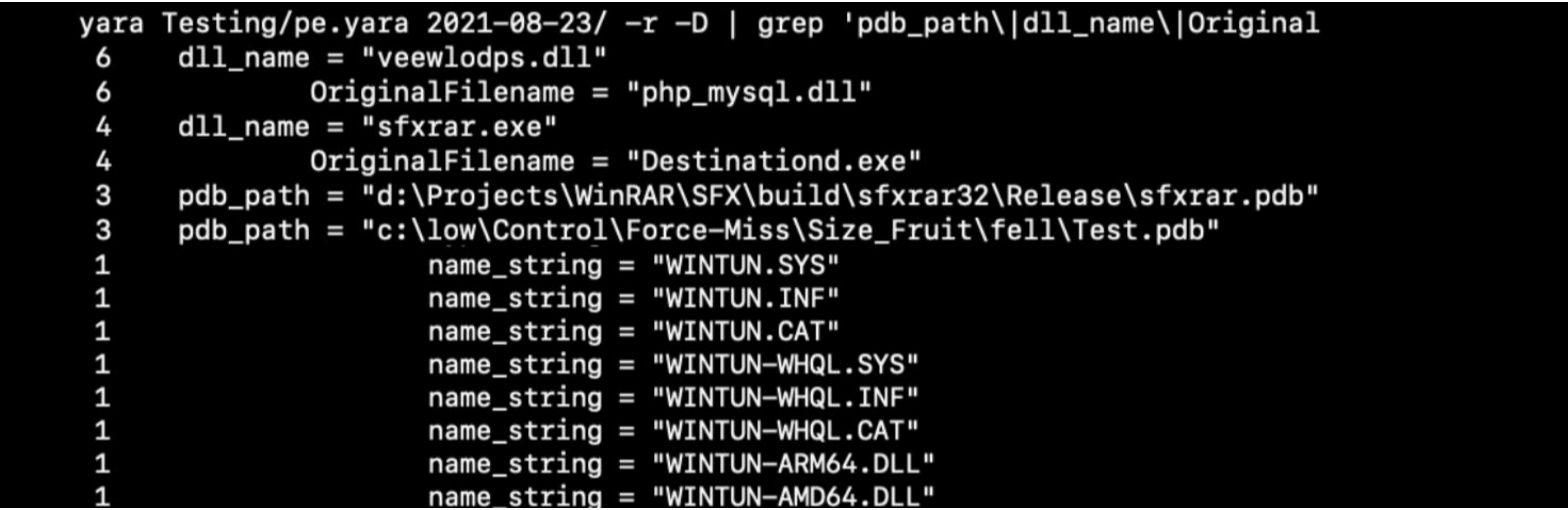


```
46      timestamp = 1589088291
```

Combining Features

Bulk Triage

Looking at a day of samples from MalwareBazaar



```
yara Testing/pe.yara 2021-08-23/ -r -D | grep 'pdb_path\|dll_name\|Original
 6     dll_name = "veewlodps.dll"
 6             OriginalFilename = "php_mysql.dll"
 4     dll_name = "sfxrar.exe"
 4             OriginalFilename = "Destinationd.exe"
 3     pdb_path = "d:\Projects\WinRAR\SFX\build\sfxrar32\Release\sfxrar.pdb"
 3     pdb_path = "c:\low\Control\Force-Miss\Size_Fruit\fell\Test.pdb"
 1                     name_string = "WINTUN.SYS"
 1                     name_string = "WINTUN.INF"
 1                     name_string = "WINTUN.CAT"
 1                     name_string = "WINTUN-WHQL.SYS"
 1                     name_string = "WINTUN-WHQL.INF"
 1                     name_string = "WINTUN-WHQL.CAT"
 1                     name_string = "WINTUN-ARM64.DLL"
 1                     name_string = "WINTUN-AMD64.DLL"
```

Bringing It Home

Be Kind & Do YARA

- Find things that a human eye wouldn’t find
- Find things that a yara eye wouldn’t find by itself
- Find more evil

Questions?

MANDIANT

YOUR CYBERSECURITY ADVANTAGE

Thank You.

References

PDB Research: https://www.fireeye.com/blog/threat-research/2019/08/definitive-dossier-of-devilish-debug-details-part-one-pdb-paths-malware.html

Rich Header Hunting: http://ropgadget.com/posts/richheader_hunting.html

Rich Headers: https://www.youtube.com/watch?v=ipPAFG8qtyg

Rich Headers: https://securelist.com/the-devils-in-the-rich-header/84348/

Rich Headers: https://www.virusbulletin.com/virusbulletin/2020/01/vb2019-paper-rich-headers-leveraging-mysterious-artifact-pe-format/

Rich Headers: https://infocon.hackingand.coffee/Hacktivity/Hacktivity%202016/Presentations/George_Webster-and-Julian-Kirsch.pdf

Appendix Some Artifacts Are Useless

- Resources prefixed with RT_ are Microsoft official name/types
  - https://www.hexacorn.com/blog/2020/04/24/re-sauce-part-1/
- BB* resource artifacts are from Delphi, not malware author
- SFX/RAR/WinRAR artifacts are very common in PDB, DLL Name, Resources:
  - PDB: D:\Projects\WinRAR\sfx\build\sfxrar32\Release\sfxrar.pdb
  - DLL Name: Sfxrar.exe
  - Resource Type Strings: PNG
  - Resource Name Strings: ASKNEXTVOL, GETPASSWORD1, LICENSEDLG, RENAMEDLG, REPLACEFILEDLG, STARTDLG

Appendix: Exported Functions

```
yara pe.yara APT40_ClusterF/  -r -D |
sed -n "/export_details/,/export_timestamp/p" |
grep name | grep -v 'forward_name'|
sort | uniq -c | sort -rn

rule PE_Export  {
condition: pe.exports(“EXAMPLE”)
}
```