Hunting APTs with Yara like a GReAT Ninja

Costin G. Raiu,
Director, GReAT, Kaspersky
@craiu

About me - Costin G. Raiu

- Director, Global Research and Analysis Team (GReAT)
- 25 years in the AV industry, 19 at Kaspersky Lab
- Writing Yara rules for ~6 years
- Found: several zero-day's, APTs (eg: Miniduke)
- Worked on: Equation, Flame, Duqu, Regin, Remsec, Turla, Sofacy, etc...

About this webinar

- We’ve been spreading our love for Yara since 2016
- "Hunting APTs with YARA Like a GReAT Ninja" - SAS training
- First edition - February 2016, Tenerife
- ~~Next Yara training - SAS2020, Barcelona~~
- Meanwhile, what about online?

Yara - an introduction Software + Recommendations

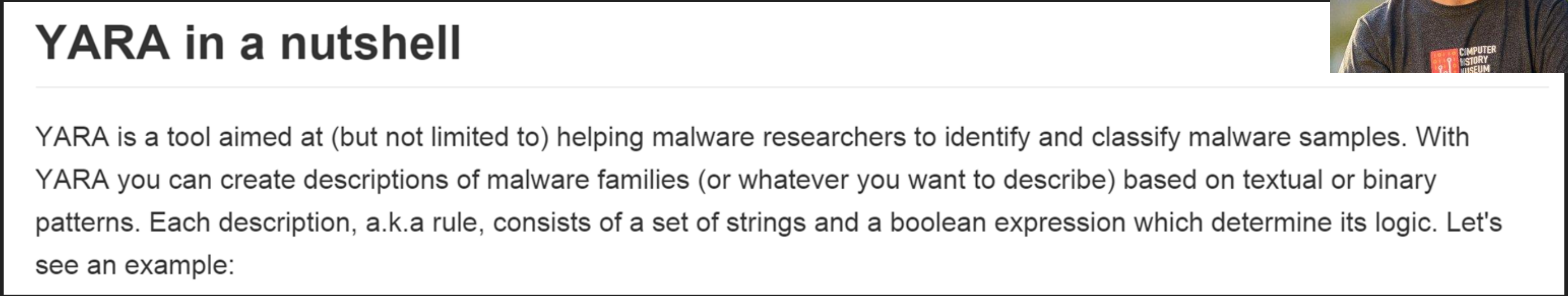
Recommended software when writing Yara rules

- Yara :-)
- String analyzer (Unix strings tool)
- PE file structure viewer
  - PE Studio (free, supports x64) <- excellent! (PEStudio is © Marc Ochsenmeier)
  - CFF Explorer, Cerbero Suite (commercial)
- Hex viewer
  - Hiew - (paid) <- highly recommended (Windows only)
  - FAR - free - highly recommended (Windows only)
  - Total Commander - free (Windows only)
  - Radare2 - free open-source (cross-platform)
- Binary diffing tool (eg. vbindiff, radiff2)
- (optional) IDA Pro / Ghidra

Recommended manuals / documentation

- Yara reference
  - https://yara.readthedocs.io/en/latest/
- File format memos (by Ange Albertini aka corkami):
  - https://github.com/corkami/pics
- Yara for VirusTotal:
  - https://www.virustotal.com/en/documentation/searching/

Yara - introduction



YARA in a nutshell

YARA is a tool aimed at (but not limited to) helping malware researchers to identify and classify malware samples. With YARA you can create descriptions of malware families (or whatever you want to describe) based on textual or binary patterns. Each description, a.k.a rule, consists of a set of strings and a boolean expression which determine its logic. Let's see an example:

https://github.com/VirusTotal/yara

Latest version: 3.11.0 (October 2019)
Precompiled Windows binaries; source code for Linux and Mac

Documentation: yara.readthedocs.org/en/latest/writingrules.html

In case you never saw a Yara rule

```
rule welcome_to_our_webinar {

meta:
        author = "Kaspersky Lab"
        description = "rule to find REAL friends"

strings:
        $a="kind"
        $b="funny"
        $c="imaginary"

condition:
        ($a and $b) and not($c)
}
```

Strings:

- $a="mystring"
- $a="mystring" fullword
- $a="mystring" wide
- $a="mystring" wide ascii
- $a="MyString" nocase
- $a="mystring" fullword ascii wide nocase
- $a={01 02 03 04 05}
- $a={01 ?? ?3 04 05}
- $a={01 [2-10] 04 05}
- $a={01 (02 03 | 03 04) 05}
- $a=/regexp .* blah/   wide ascii   <- PCRE subset

Conditions - basic

- Check headers (eg. PE file)
  - Good (fast): `uint16(0) == 0x5A4D`
  - Bad: `($mz at 0)`

Example:

```
//Mach-O or ELF
(uint32(0)==0x464c457f) or
(uint32(0) == 0xfeedfacf) or (uint32(0) == 0xcffaedfe) or
(uint32(0) == 0xfeedface) or (uint32(0) == 0xcefaedfe)
```

- String count: `(#a == 5) or (#b > 7)`
- filesize: `(filesize>512) and (filesize<5000000) or (filesize<5MB)`

Conditions - extended

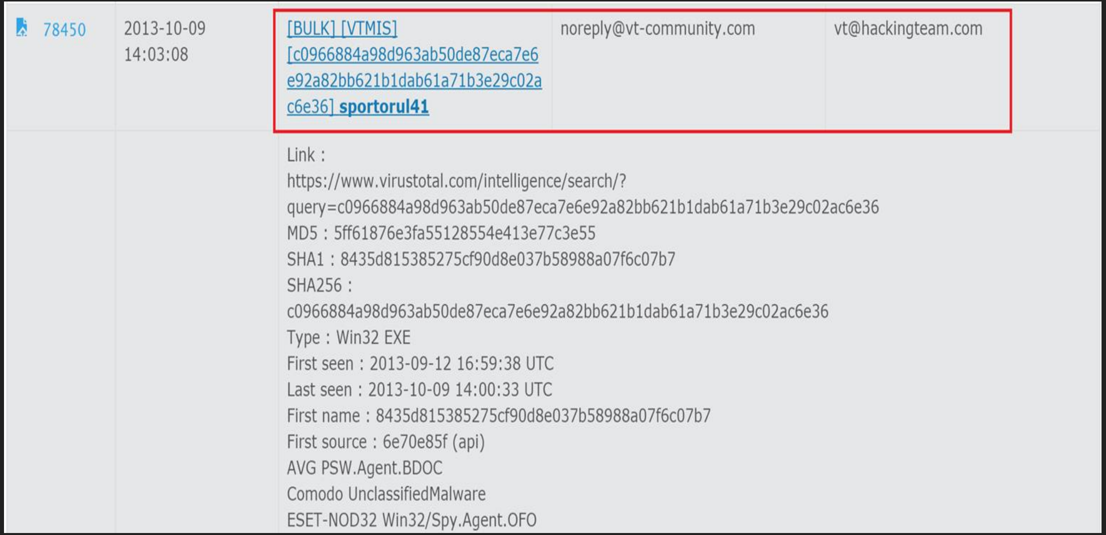
- ($a at 0) - string at offset
- ($a at pe.entry_point) - string at PE entrypoint
- $a at (@b+16) - string at where
  $b was found + 16
- $a in (0..100) and $b in (100..filesize) - ranges
- $a in (pe.entry_point .. pe.entry_point + 10)
- 2 of ($a,$b,$c)
- 2 of them
- 2 of ($a*)
- all of them
- any of them
- $a and not $b

What can we do with Yara?

- identify and classify malware
- find **new malware** samples based on family-specific features
- find **new exploits** and zero-days
- help speeding up **incident response**
- increase your defenses by deploying **custom rules** inside your organization
- **classification**: identify file formats, archives, packed files, known threats
- filter network traffic (why not)
- build your own private antivirus :-)
- ~~Presentations~~ (Vitaly - https://www.youtube.com/watch?v=fbidgtOXvc0)

Who uses Yara?

- Every serious security company
- Several appliance vendors (eg. FireEye)
- VirusTotal
- ClamAV 0.99b+ (June 2015)
- Fortune 500 companies
- Governmental agencies
- Spyware developers:
  - eg. HackingTeam



78450 2013-10-09 14:03:08 [BULK] [VTMIS] [c0966884a98d963ab50de87eca7e6e92a82bb621b1dab61a71b3e29c02ac6e36] **sportorul41** noreply@vt-community.com vt@hackingteam.com

Link :
https://www.virustotal.com/intelligence/search/?query=c0966884a98d963ab50de87eca7e6e92a82bb621b1dab61a71b3e29c02ac6e36
MD5 : 5ff61876e3fa55128554e413e77c3e55
SHA1 : 8435d815385275cf90d8e037b58988a07f6c07b7
SHA256 :
c0966884a98d963ab50de87eca7e6e92a82bb621b1dab61a71b3e29c02ac6e36
Type : Win32 EXE
First seen : 2013-09-12 16:59:38 UTC
Last seen : 2013-10-09 14:00:33 UTC
First name : 8435d815385275cf90d8e037b58988a07f6c07b7
First source : 6e70e85f (api)
AVG PSW.Agent.BDOC
Comodo UnclassifiedMalware
ESET-NOD32 Win32/Spy.Agent.OFO

Yara rules "design" tips

Design tips

- Think of Yara like a programming language, not enhanced regexp’s
- Indent the code, comment it nicely
- Quick "hack" rules can be okay now, but what will happen in 1-2 years when you need to reuse it?
- Other people might rely on your rule
- Some rules might be published. You may want to include contact info.
    - Name, e-mail address

Naming convention

```
rule apt_CN_Winnti_stolencert {...
```

APT,
exploit,
crimeware,
clean,
etc...

Country /
language
designator
(for APT's);
ZZ if unknown.

Threat actor name

Malware name
or specific
family

Example?

```
rule apt_CN_Winnti_stolencert {

meta:
           author = "Costin Raiu <craiu@kaspersky.ro>, Kaspersky Lab"
           type ="APT"
           filetype = "Win32 EXE"
           date = "2015-06-23"
           version = "1.0"
           reference = "https://securelist.com/blog/research/70991/games-are-over/"
           md5 = "8e61219b18d36748ce956099277cc29b"

strings:
           //serial - Asahi Kasei Microdevices Corporation
           $a1 = {7a bf 25 da 46 cc 27 48 ee dd 23 af 7b c8 54 d8}
           //thumb - Asahi Kasei Microdevices Corporation
           $a2 = {37 d3 66 74 d5 15 74 ed 5b f0 c8 28 60 da 8b ea 03 5a 1a 1c}

condition:
           ((uint16(0) == 0x5A4D)) and (filesize < 5000000) and ((any of ($a*)))

}
```

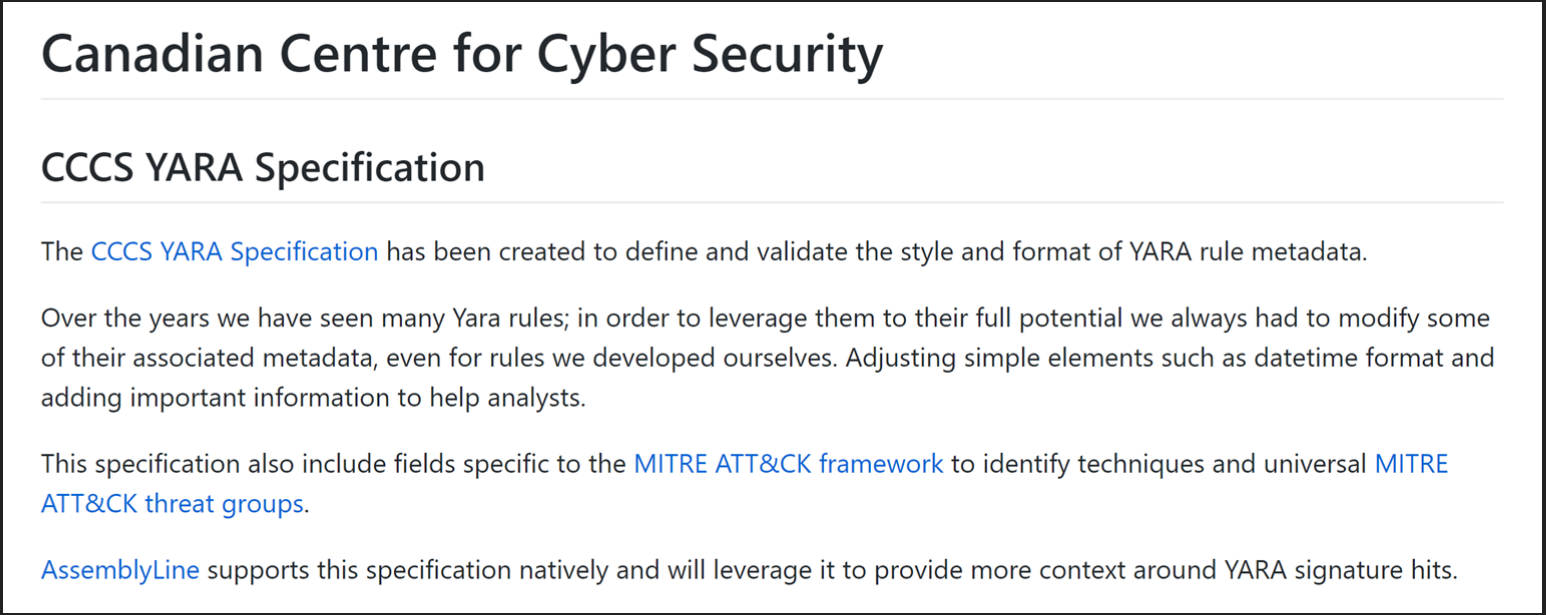
Metadata - recommended keywords

- author <e-mail>
- date
- version
- reference (url)
- description (brief)
- hash / md5 / sha1 / sha256

Also useful:

- last_modified
- copyright / TLP
- yara_versions - minimum Yara version required for the rule

More about metadata



Canadian Centre for Cyber Security

CCCS YARA Specification

The CCCS YARA Specification has been created to define and validate the style and format of YARA rule metadata.

Over the years we have seen many Yara rules; in order to leverage them to their full potential we always had to modify some of their associated metadata, even for rules we developed ourselves. Adjusting simple elements such as datetime format and adding important information to help analysts.

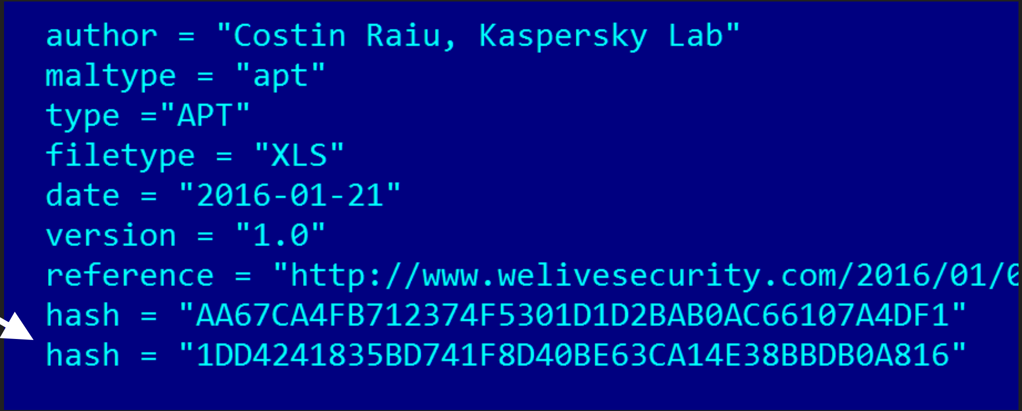
This specification also include fields specific to the MITRE ATT&CK framework to identify techniques and universal MITRE ATT&CK threat groups.

AssemblyLine supports this specification natively and will leverage it to provide more context around YARA signature hits.

https://github.com/CybercentreCanada/CCCS-Yara

A word about hashes...

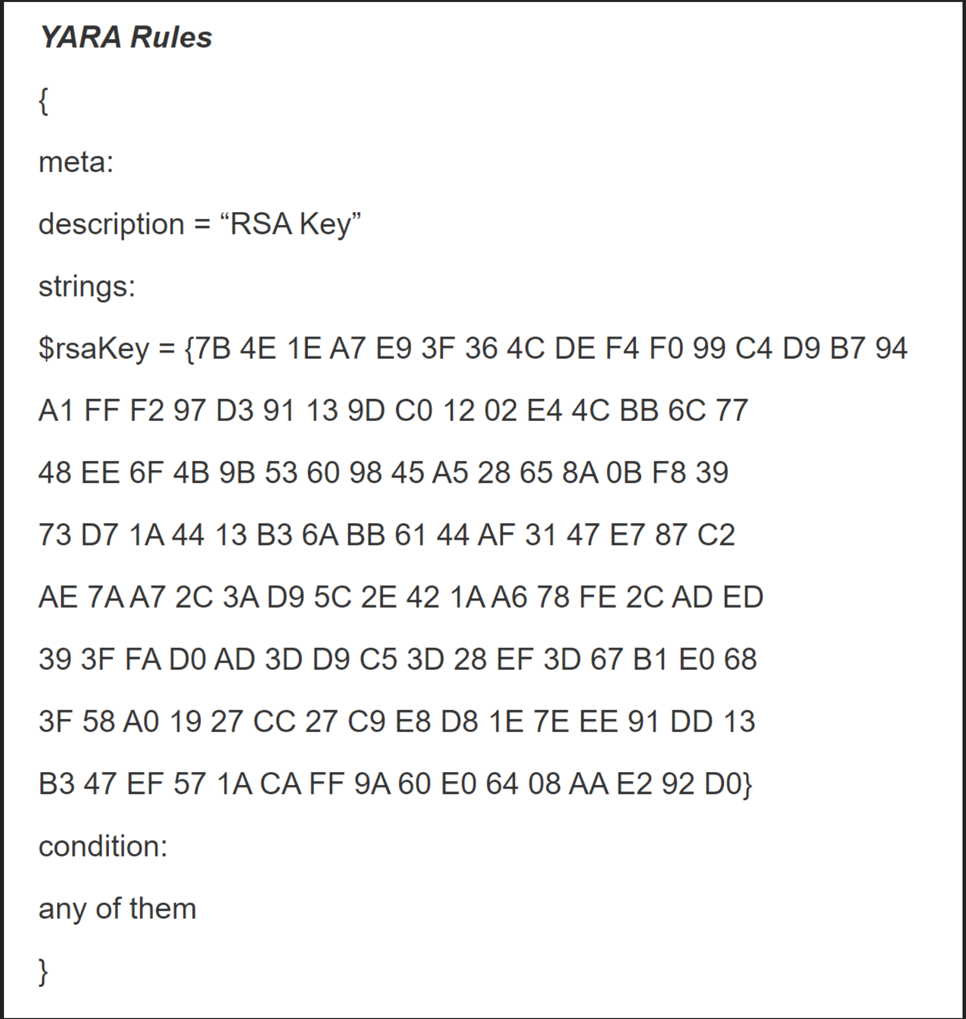
- Putting a hash in metadata helps other find the sample quickly or it helps you find the sample on which the rule was generated
- It’s okay to use "md5", "sha1", "sha256"
- Multiple hashes -> "md5_1", "md5_2"?
- Easiest solution: "hash" multiple times



```
author = "Costin Raiu, Kaspersky Lab"
maltype = "apt"
type ="APT"
filetype = "XLS"
date = "2016-01-21"
version = "1.0"
reference = "http://www.welivesecurity.com/2016/01/
hash = "AA67CA4FB712374F5301D1D2BAB0AC66107A4DF1"
hash = "1DD4241835BD741F8D40BE63CA14E38BBDB0A816"
```

The good, the bad and the ugly of Yara rules.

Is this a "good" rule?

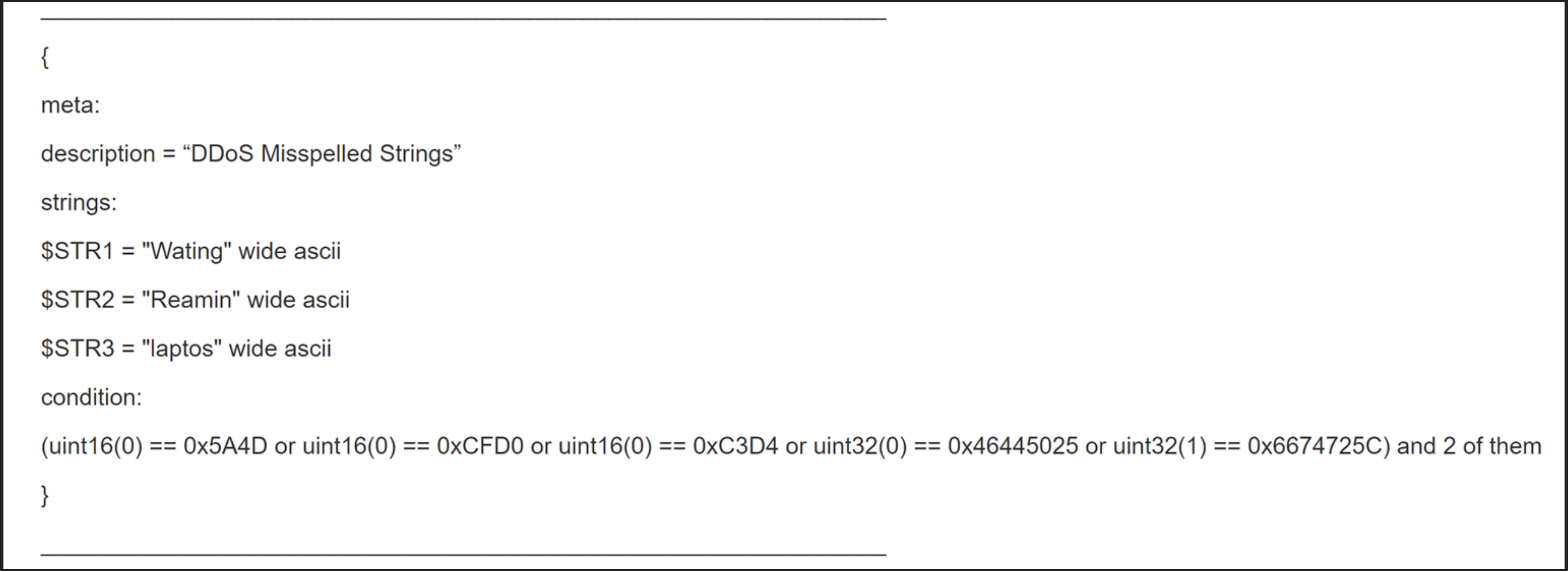


YARA Rules

```
{
meta:
description = "RSA Key"
strings:
$rsaKey = {7B 4E 1E A7 E9 3F 36 4C DE F4 F0 99 C4 D9 B7 94
A1 FF F2 97 D3 91 13 9D C0 12 02 E4 4C BB 6C 77
48 EE 6F 4B 9B 53 60 98 45 A5 28 65 8A 0B F8 39
73 D7 1A 44 13 B3 6A BB 61 44 AF 31 47 E7 87 C2
AE 7A A7 2C 3A D9 5C 2E 42 1A A6 78 FE 2C AD ED
39 3F FA D0 AD 3D D9 C5 3D 28 EF 3D 67 B1 E0 68
3F 58 A0 19 27 CC 27 C9 E8 D8 1E 7E EE 91 DD 13
B3 47 EF 57 1A CA FF 9A 60 E0 64 08 AA E2 92 D0}
condition:
any of them
}
```

Source:
https://www.us-cert.gov/ncas/alerts/TA17-164A

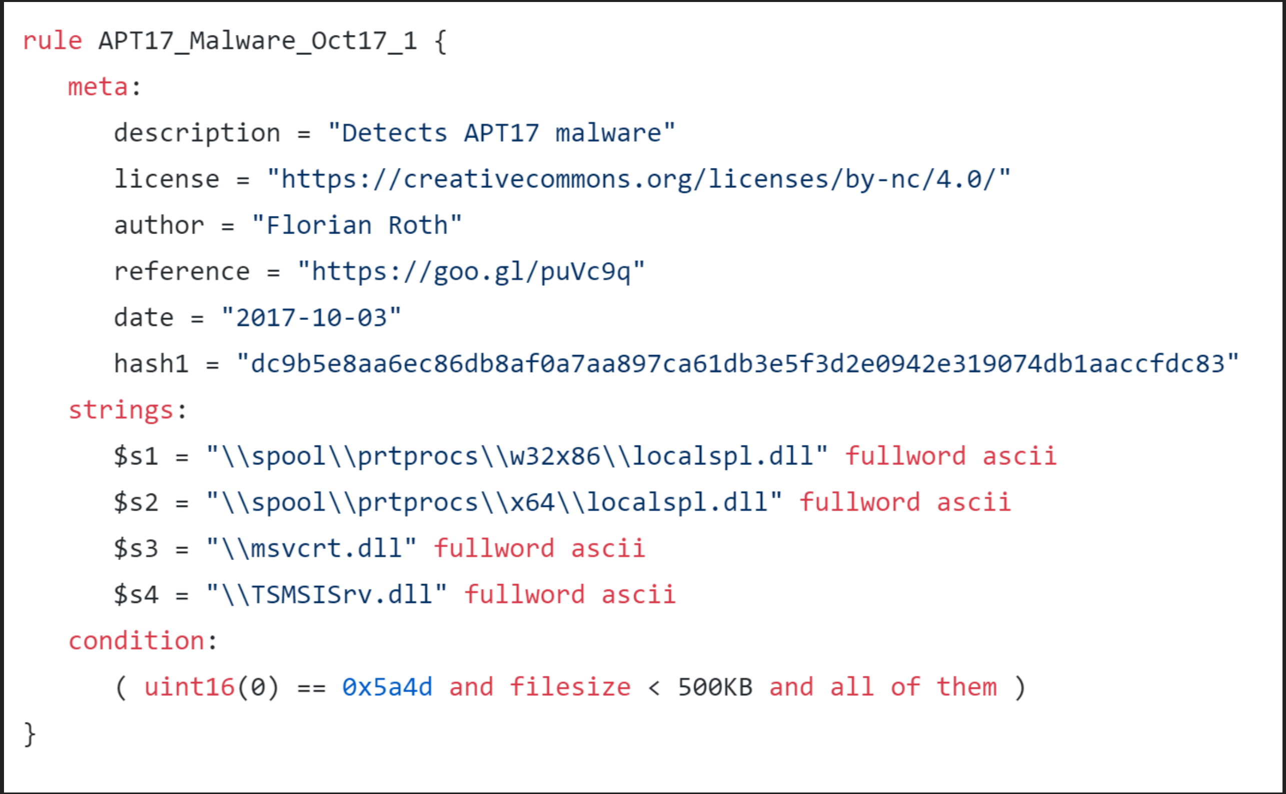
Is this a "good" rule?



```
{
meta:
description = “DDoS Misspelled Strings”
strings:
$STR1 = "Wating" wide ascii
$STR2 = "Reamin" wide ascii
$STR3 = "laptos" wide ascii
condition:
(uint16(0) == 0x5A4D or uint16(0) == 0xCFD0 or uint16(0) == 0xC3D4 or uint32(0) == 0x46445025 or uint32(1) == 0x6674725C) and 2 of them
}
```

Source:
https://www.us-cert.gov/ncas/alerts/TA17-164A

Good rule:



```
rule APT17_Malware_Oct17_1 {
   meta:
      description = "Detects APT17 malware"
      license = "https://creativecommons.org/licenses/by-nc/4.0/"
      author = "Florian Roth"
      reference = "https://goo.gl/puVc9q"
      date = "2017-10-03"
      hash1 = "dc9b5e8aa6ec86db8af0a7aa897ca61db3e5f3d2e0942e319074db1aaccfdc83"
   strings:
      $s1 = "\\spool\\prtprocs\\w32x86\\localspl.dll" fullword ascii
      $s2 = "\\spool\\prtprocs\\x64\\localspl.dll" fullword ascii
      $s3 = "\\msvcrt.dll" fullword ascii
      $s4 = "\\TSMSISrv.dll" fullword ascii
   condition:
      ( uint16(0) == 0x5a4d and filesize < 500KB and all of them )
}
```

Source:
https://github.com/Neo23x0/signature-base/blob/master/yara/apt_apt17_mal_sep17.yar

Rule detection quality

Common mistake: rule generates too many FP

- Test rule first (more details later)
  - Internally on some representative file set
  - Using a cloud solution: KLARA, VTMIS

- Use the "fullword" keyword when string is short!
- Try to not make rules based ONLY on popular imports/exports

- Remember to use filesize*
- Always check header*

* not for memory scanning

Is this a "good enough" rule?

```
rule c0c4_imports  {
strings:

    $a1 = "RemoveDirectoryW" fullword ascii
    $a2 = "GetTempFileNameW" fullword ascii
    $a3 = "GetTimeFormatW" fullword ascii
    $a4 = "MapViewOfFile" fullword ascii
    $a5 = "GetNumberFormatA" fullword ascii
    $a6 = "GetDiskFreeSpaceExW" fullword ascii
    $a7 = "SystemTimeToFileTime" fullword ascii
    $a8 = "CreateWindowExW" fullword ascii
    $a9 = "InvalidateRgn" fullword ascii
    $a10 = "SetDlgItemInt" fullword ascii
    $a11 = "CreateDialogIndirectParamW" fullword ascii
    $a12 = "LoadBitmapW" fullword ascii

condition:
    uint16(0) == 0x5A4D
    and all of ($a*)
    and filesize < 1000000
}
```

Common mistake: rule catches only one sample

- do not use runtime-generated strings
  - example coming up next
- do not put all possible criterias as a necessary condition
  - (5 of ...) instead of (all of ...)

Another "not quite the best" rule.

```
rule nordicransom_scr {
meta:
            hash = "2b8336130d449d035986863a8d796dc1"
strings:
            $string0 = "rE5Cj;"
            $string1 = "kP_J59"
            $string2 = "Dj,:xx5"
            $string3 = "HPI%TZ"
            $string4 = "{qUVO}"
            $string5 = "O]{O7:"
            $string6 = "<8z]_j6{"
            $string7 = "6hgSd-"
            $string8 = "HQH$AL]"
            $string9 = "6G3ne5"
            $string10 = "dcm(lI"
            $string11 = "aPM.uz"
            $string12 = ":{YE>D"
            $string13 = "HgbX)gO"
            $string14 = "d.$N0h1r"
            $string15 = "EKRPm$"
            $string16 = "q;C2E>"
            $string17 = "C-RC${["
            $string18 = "T5H{Kt"
condition:
            19 of them
}
```

Code-based strings garbage

No additional conditions like filesize or header

Better: use unique and specific (data) strings

- use the Unix strings tool to extract ASCII text strings*
  *to extract Unicode strings use "-e l" option

- look at the results manually (+Google):
  - mutex or event names (eg. "WerTyQ34C")
  - rare user-agents (eg. "NOKIAN95")
  - registry key / unique value names
  - typos (eg. SOFTWAE\MICROSOFT\WINDOWS\CURRENTVERSION\RUN\)
  - PDB paths (eg. C:\Users\client7\Desktop\chforce\Release\chforce.pdb)
  - GUID's
  - internal module names
  - encoded or encrypted configuration strings

Better: use different conditions

- when possible, use 2-3 groups of conditions:
  - based on unique artefacts found in the malware
  - based on a group of specific non-unique strings
  - based on file properties and structure (sections, entropy, timestamp etc)

Better rule

(removed meta: to fit screen)

```
rule apt_CN_BlueTraveller {
strings:

    $a1 = "http://%s/%02d%02d.htm"  fullword ascii
    $a2 = "%-25s %6s %6s %22s" fullword ascii

    $b1 = "WinHttpCrackUrl" fullword ascii
    $b2 = "UnRegisterTypeLib" fullword ascii
    $b3 = "GetProcessTimes" fullword ascii
    $b4 = "CharToOemBuffA" fullword ascii
    $b5 = "GetInputState" fullword ascii
    $b6 = "PostThreadMessageA" fullword ascii

    $c1 = "Crack url failed" fullword ascii
    $c2 = "Upload data ok!" fullword ascii
    $c3 = "Error! Set privilege failed.." fullword ascii
    $c4 = "Heap alloc when download failed" fullword ascii
    $c5 = "Fully-qualified distinguished name:" ascii
    $c6 = "%sCreate file %s failed" fullword ascii

condition:
    uint16(0) == 0x5A4D and
    (any of ($a*) or 5 of ($b*) or 2 of ($c*))
    and filesize < 100000
}
```

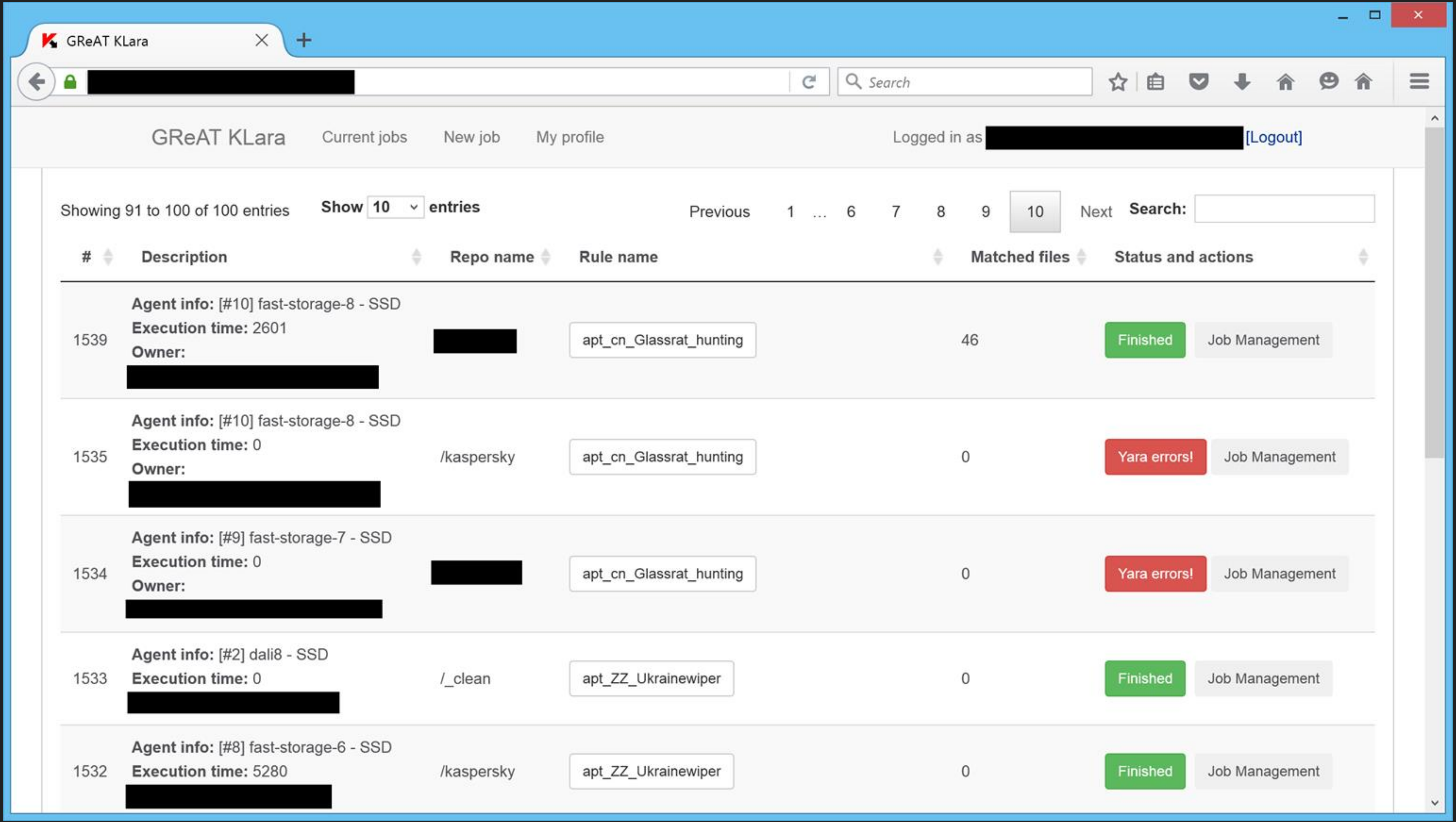
Sources of Yara rules

- Public rulesets
- Exchange groups
- APT reports
    - Directly published rules
    - Indirectly published rules (IOCs):
        - strings
        - file names
        - registry keys
        - mutexes
        - c2's
        - certificate s/n
        - ...
- Grab everything you find :-)
- Then test it
- ProTip: Organize your rules in different collections!

Kaspersky KLARA

KLARA - What is KLARA?

- KLARA is a Kaspersky-internal project designed to simplify scanning of collections using Yara rules
- WebUI
- It allows you to test your Yara rules on several file sets
- What you can do with KLARA:
    - test rules for false positives
    - fine-tune detections
    - hunt for interesting samples
- Quite similar to VT Retrohunt
- It's free open source software: https://github.com/KasperskyLab/klara


GReAT KLara

[redacted]

Search

GReAT KLara Current jobs New job My profile Logged in as [redacted] [Logout]

Showing 91 to 100 of 100 entries Show 10 entries Previous 1 … 6 7 8 9 10 Next Search:

#	Description	Repo name	Rule name	Matched files	Status and actions
1539	**Agent info:** [#10] fast-storage-8 - SSD **Execution time:** 2601 **Owner:** [redacted]	[redacted]	apt_cn_Glassrat_hunting	46	Finished Job Management
1535	**Agent info:** [#10] fast-storage-8 - SSD **Execution time:** 0 **Owner:** [redacted]	/kaspersky	apt_cn_Glassrat_hunting	0	Yara errors! Job Management
1534	**Agent info:** [#9] fast-storage-7 - SSD **Execution time:** 0 **Owner:** [redacted]	[redacted]	apt_cn_Glassrat_hunting	0	Yara errors! Job Management
1533	**Agent info:** [#2] dali8 - SSD **Execution time:** 0 [redacted]	/_clean	apt_ZZ_Ukrainewiper	0	Finished Job Management
1532	**Agent info:** [#8] fast-storage-6 - SSD **Execution time:** 5280 [redacted]	/kaspersky	apt_ZZ_Ukrainewiper	0	Finished Job Management

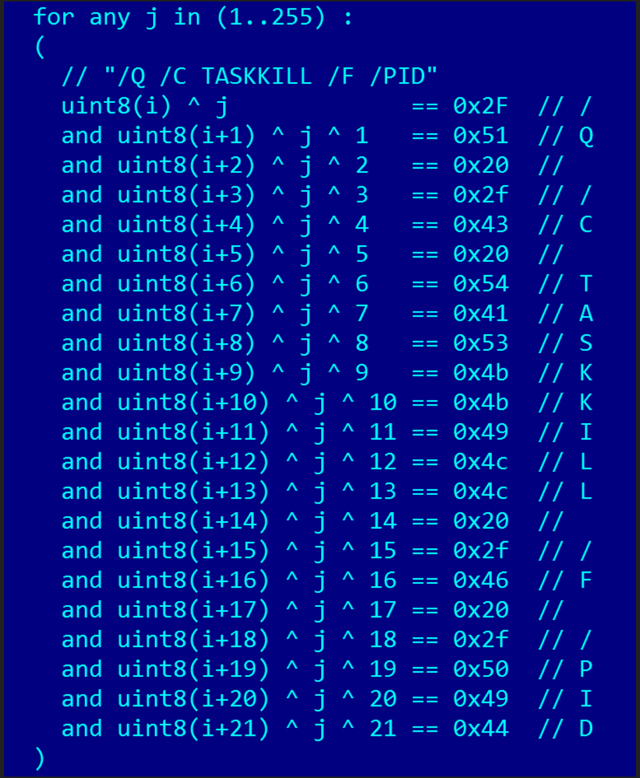
Important: Testing for false positives

- You can build your own clean set
  - Windows XP files 32-bit
  - A Linux installation
  - Windows 7/8 64-bit
  - OSX installation
  - iOS fs dump
- Public clean samples sets
  - https://www.microsoft.com/en-us/download
  - ftp://ftp.elf.stuba.sk/pub/pc/

Hints for Yara top performance!

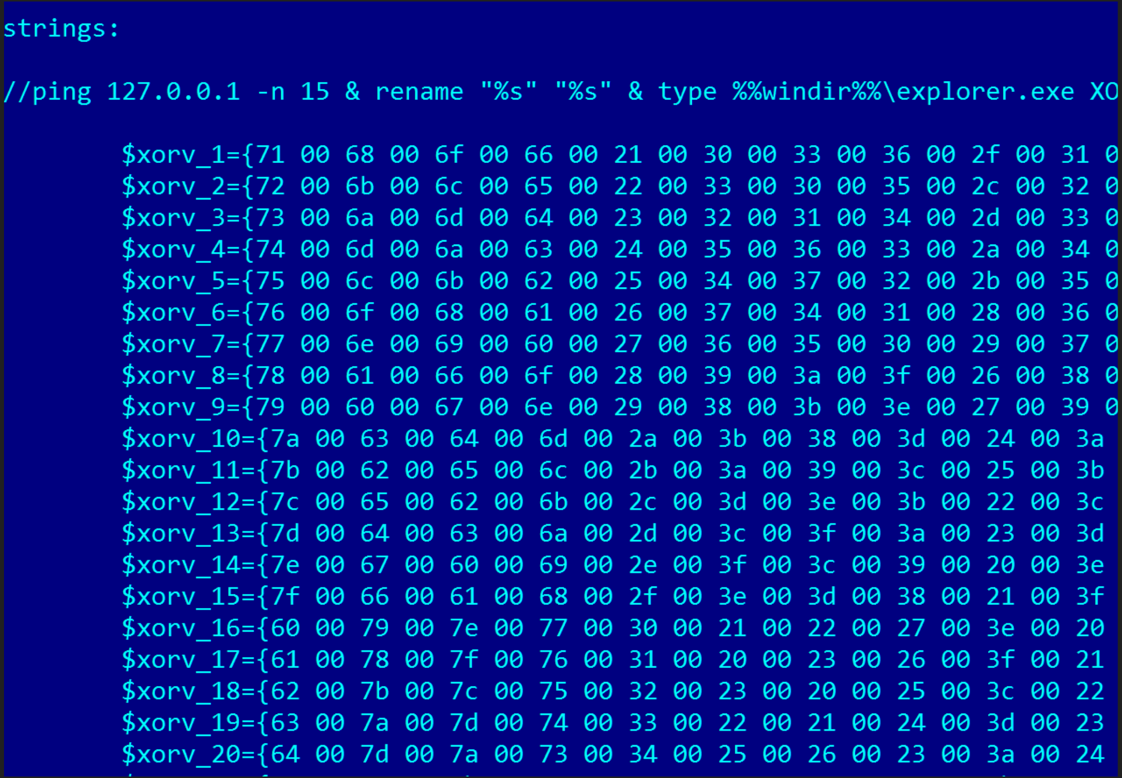
- If you have an SSD in your machine, it makes sense to use -p
- Common setting (SSD+i7 CPU): -p 4
- This can lead to 400% scan speed increase
- This is NOT true for classical HDD drives!
  - Still, you may get better performance out of -p 2
- The best performance: SSD's in RAID0
  - 4x 1TB Samsung 850 EVO disks in RAID0 will have a throughput of 2GB/s
  - You will need to run yara -p 32 to achieve 2GB/s Yara scan speed
  - Maximum value is 32 (you can't use more than 32 threads)
- SSD wear is not an issue for a YARA sample repository

Loop vs String list



```
  for any j in (1..255) :
  (
    // "/Q /C TASKKILL /F /PID"
    uint8(i) ^ j                == 0x2F   // /
    and uint8(i+1) ^ j ^ 1      == 0x51   // Q
    and uint8(i+2) ^ j ^ 2      == 0x20   //
    and uint8(i+3) ^ j ^ 3      == 0x2f   // /
    and uint8(i+4) ^ j ^ 4      == 0x43   // C
    and uint8(i+5) ^ j ^ 5      == 0x20   //
    and uint8(i+6) ^ j ^ 6      == 0x54   // T
    and uint8(i+7) ^ j ^ 7      == 0x41   // A
    and uint8(i+8) ^ j ^ 8      == 0x53   // S
    and uint8(i+9) ^ j ^ 9      == 0x4b   // K
    and uint8(i+10) ^ j ^ 10 == 0x4b   // K
    and uint8(i+11) ^ j ^ 11 == 0x49   // I
    and uint8(i+12) ^ j ^ 12 == 0x4c   // L
    and uint8(i+13) ^ j ^ 13 == 0x4c   // L
    and uint8(i+14) ^ j ^ 14 == 0x20   //
    and uint8(i+15) ^ j ^ 15 == 0x2f   // /
    and uint8(i+16) ^ j ^ 16 == 0x46   // F
    and uint8(i+17) ^ j ^ 17 == 0x20   //
    and uint8(i+18) ^ j ^ 18 == 0x2f   // /
    and uint8(i+19) ^ j ^ 19 == 0x50   // P
    and uint8(i+20) ^ j ^ 20 == 0x49   // I
    and uint8(i+21) ^ j ^ 21 == 0x44   // D
  )
```

Slower



```
strings:

//ping 127.0.0.1 -n 15 & rename "%s" "%s" & type %%windir%%\explorer.exe XO

        $xorv_1={71 00 68 00 6f 00 66 00 21 00 30 00 33 00 36 00 2f 00 31 0
        $xorv_2={72 00 6b 00 6c 00 65 00 22 00 33 00 30 00 35 00 2c 00 32 0
        $xorv_3={73 00 6a 00 6d 00 64 00 23 00 32 00 31 00 34 00 2d 00 33 0
        $xorv_4={74 00 6d 00 6a 00 63 00 24 00 35 00 36 00 33 00 2a 00 34 0
        $xorv_5={75 00 6c 00 6b 00 62 00 25 00 34 00 37 00 32 00 2b 00 35 0
        $xorv_6={76 00 6f 00 68 00 61 00 26 00 37 00 34 00 31 00 28 00 36 0
        $xorv_7={77 00 6e 00 69 00 60 00 27 00 36 00 35 00 30 00 29 00 37 0
        $xorv_8={78 00 61 00 66 00 6f 00 28 00 39 00 3a 00 3f 00 26 00 38 0
        $xorv_9={79 00 60 00 67 00 6e 00 29 00 38 00 3b 00 3e 00 27 00 39 0
        $xorv_10={7a 00 63 00 64 00 6d 00 2a 00 3b 00 38 00 3d 00 24 00 3a
        $xorv_11={7b 00 62 00 65 00 6c 00 2b 00 3a 00 39 00 3c 00 25 00 3b
        $xorv_12={7c 00 65 00 62 00 6b 00 2c 00 3d 00 3e 00 3b 00 22 00 3c
        $xorv_13={7d 00 64 00 63 00 6a 00 2d 00 3c 00 3f 00 3a 00 23 00 3d
        $xorv_14={7e 00 67 00 60 00 69 00 2e 00 3f 00 3c 00 39 00 20 00 3e
        $xorv_15={7f 00 66 00 61 00 68 00 2f 00 3e 00 3d 00 38 00 21 00 3f
        $xorv_16={60 00 79 00 7e 00 77 00 30 00 21 00 22 00 27 00 3e 00 20
        $xorv_17={61 00 78 00 7f 00 76 00 31 00 20 00 23 00 26 00 3f 00 21
        $xorv_18={62 00 7b 00 7c 00 75 00 32 00 23 00 20 00 25 00 3c 00 22
        $xorv_19={63 00 7a 00 7d 00 74 00 33 00 22 00 21 00 24 00 3d 00 23
        $xorv_20={64 00 7d 00 7a 00 73 00 34 00 25 00 26 00 23 00 3a 00 24
```

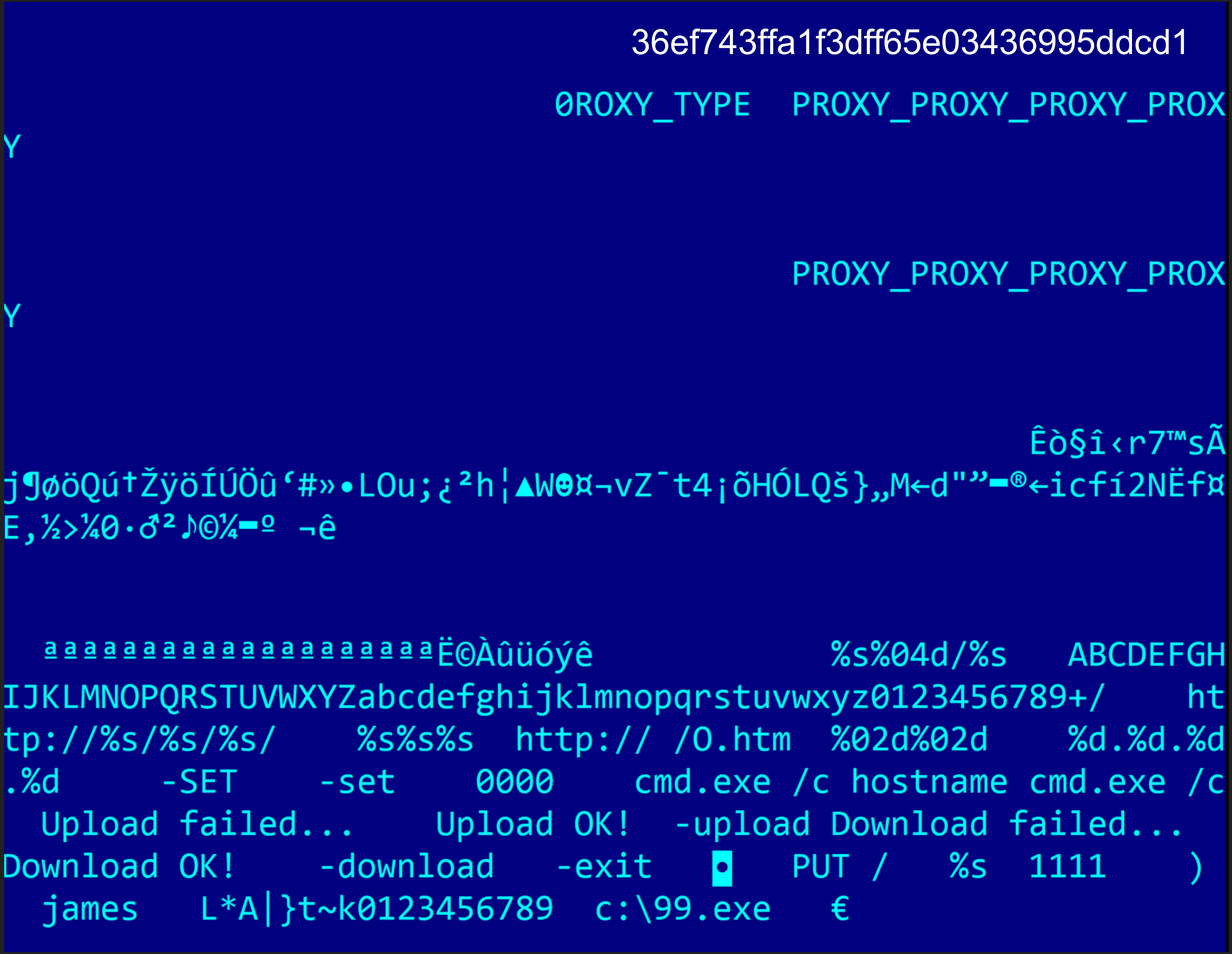
Faster

Examples

Example 1: BlueTraveller

- Context: an unknown APT group targets government entities
- We call them "BlueTraveller"
- We found 2 samples during IR
  - 36ef743ffa1f3dff65e03436995ddcd1
  - f77c69a1810c8316d2b5e5d3775b3786
- We want to find as many other samples as possible

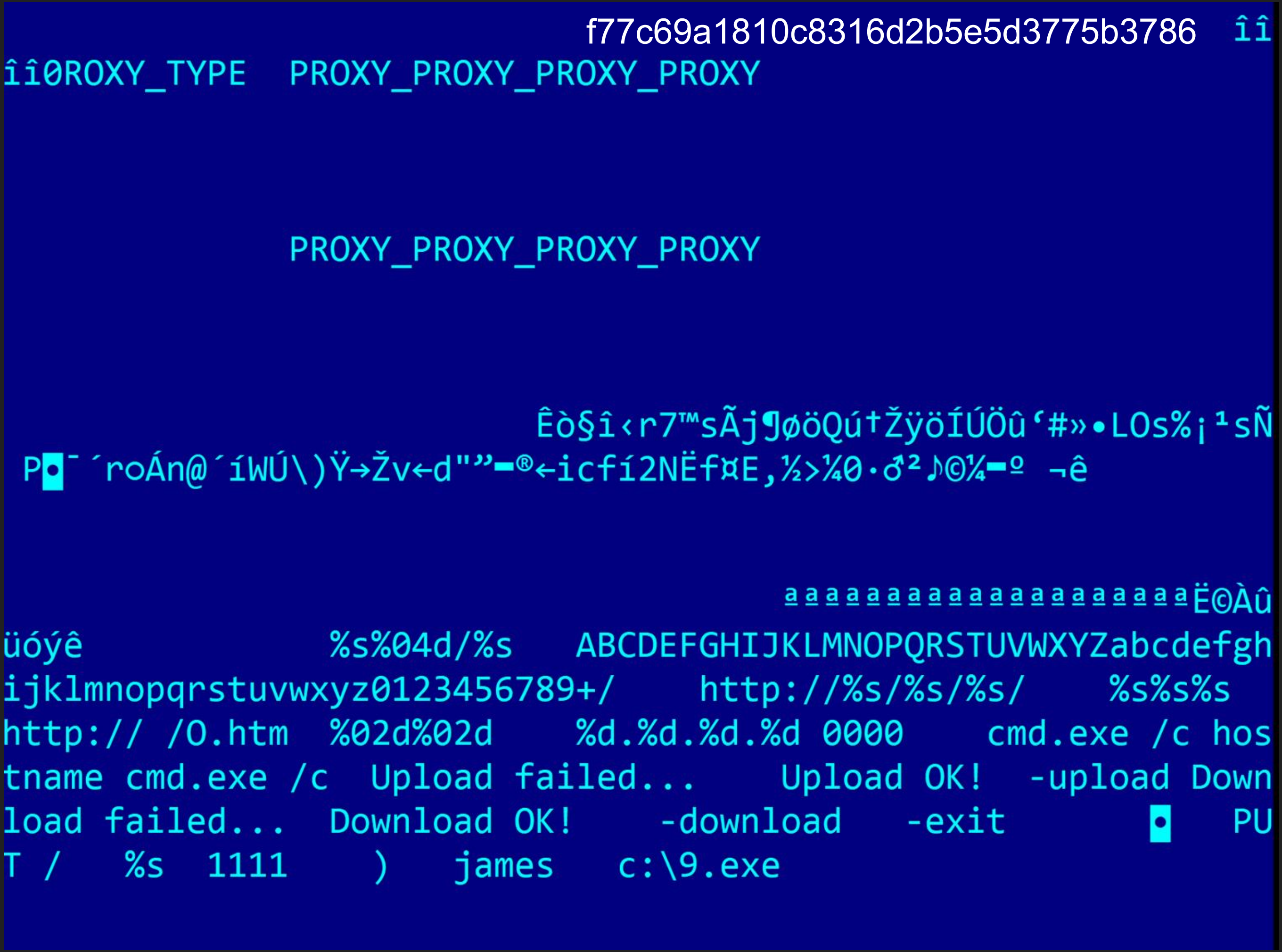
First sample:



36ef743ffa1f3dff65e03436995ddcd1

```
                                  0ROXY_TYPE  PROXY_PROXY_PROXY_PROX
Y


                                              PROXY_PROXY_PROXY_PROX
Y



                                                          Êò§î‹r7™sÃ
j¶øöQú†ŽÿöÍÚÖû‘#»•LOu;¿²h¦▲W☻¤¬vZ¯t4¡õHÓLQš},,M←d"”▬®←icfí2NËf¤
E,½>¼0·♂²♪©¼▬º ¬ê


  ªªªªªªªªªªªªªªªªªªªªË©Àûüóýê            %s%04d/%s   ABCDEFGH
IJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/     ht
tp://%s/%s/%s/    %s%s%s  http:// /O.htm  %02d%02d    %d.%d.%d
.%d     -SET    -set    0000    cmd.exe /c hostname cmd.exe /c
  Upload failed...    Upload OK!  -upload Download failed...
Download OK!    -download   -exit   ◘   PUT /   %s  1111    )
  james   L*A|}t~k0123456789  c:\99.exe   €
```

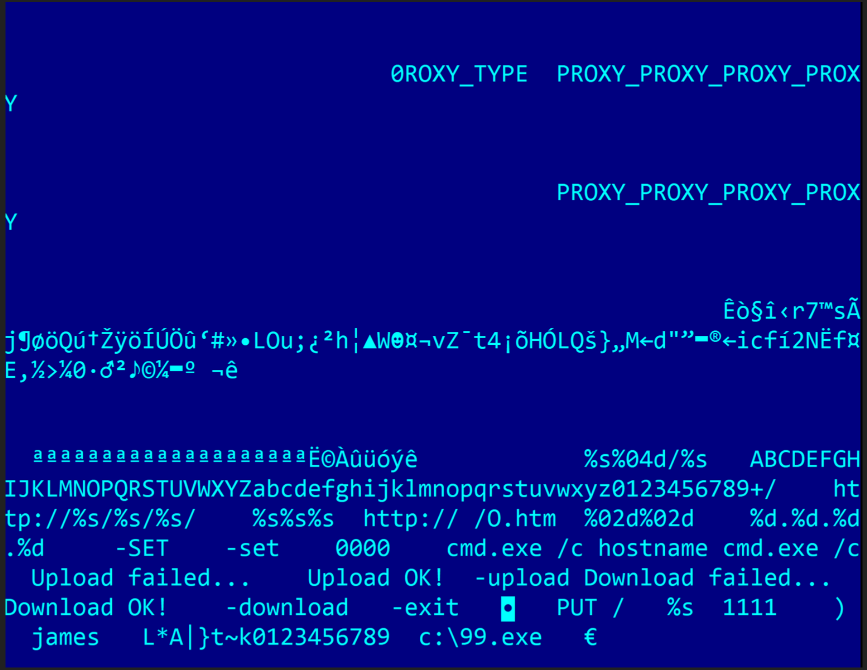
Second sample:



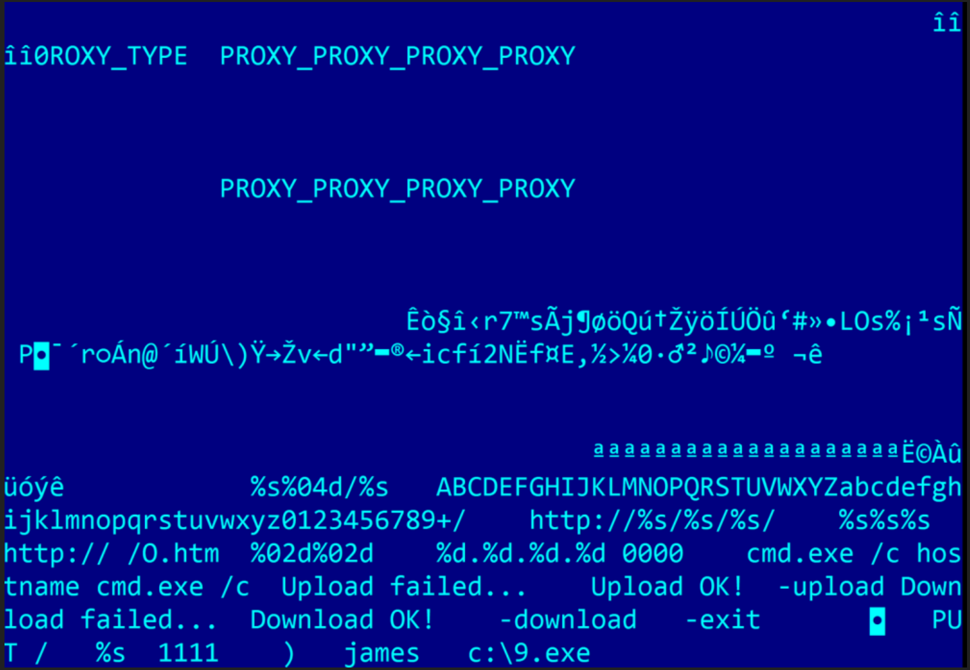
f77c69a1810c8316d2b5e5d3775b3786

```
                                                          îî
îî0ROXY_TYPE  PROXY_PROXY_PROXY_PROXY



              PROXY_PROXY_PROXY_PROXY




                         Êò§î‹r7™sÃj¶øöQú†ŽÿöÍÚÖû‘#»•LOs%¡¹sÑ
 P•¯´roÁn@´íWÚ\)Ÿ→Žv←d"”▬®←icfí2NËf¤E,½>¼0·♂²♪©¼▬º ¬ê


                                       ªªªªªªªªªªªªªªªªªªªªË©Àû
üóýê            %s%04d/%s   ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefgh
ijklmnopqrstuvwxyz0123456789+/    http://%s/%s/%s/    %s%s%s
http:// /O.htm  %02d%02d    %d.%d.%d.%d 0000    cmd.exe /c hos
tname cmd.exe /c  Upload failed...    Upload OK!  -upload Down
load failed...  Download OK!    -download   -exit       •   PU
T /   %s  1111     )   james   c:\9.exe
```

Side by side:



```
                                        0ROXY_TYPE   PROXY_PROXY_PROXY_PROX
Y


                                                     PROXY_PROXY_PROXY_PROX
Y


                                                                  Êò§î‹r7™sÃ
j¶øöQú†ŽÿöÍÚÖû‘#»•LOu;¿²h¦▲W☻¤¬vZ¯t4¡õHÓLQš},„M←d"”▬®←icfí2NËf¤
E,½>¼0·♂²♪©¼▬º ¬ê


  ªªªªªªªªªªªªªªªªªªªªË©Àûüóýê            %s%04d/%s    ABCDEFGH
IJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/    ht
tp://%s/%s/%s/    %s%s%s  http:// /0.htm  %02d%02d    %d.%d.%d
.%d     -SET    -set    0000    cmd.exe /c hostname cmd.exe /c
  Upload failed...    Upload OK!  -upload Download failed...
Download OK!    -download   -exit   ◘   PUT /   %s  1111    )
  james   L*A|}t~k0123456789  c:\99.exe   €
```



```
                                                                        îî
îî0ROXY_TYPE   PROXY_PROXY_PROXY_PROXY




               PROXY_PROXY_PROXY_PROXY



                           Êò§î‹r7™sÃj¶øöQú†ŽÿöÍÚÖû‘#»•LOs%¡¹sÑ
 P◘¯´roÁn@´íWÚ\)Ÿ→Žv←d"”▬®←icfí2NËf¤E,½>¼0·♂²♪©¼▬º ¬ê


                                        ªªªªªªªªªªªªªªªªªªªªËÀû
üóýê           %s%04d/%s   ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefgh
ijklmnopqrstuvwxyz0123456789+/    http://%s/%s/%s/    %s%s%s
http:// /0.htm  %02d%02d    %d.%d.%d.%d 0000    cmd.exe /c hos
tname cmd.exe /c  Upload failed...    Upload OK!  -upload Down
load failed...  Download OK!    -download   -exit       ◘   PU
T /   %s  1111    )   james   c:\9.exe
```

Exercise: BlueTraveller (solution, pitfalls)

- We will use strings for detection
- Some strings (eg. "james" are either too short or too common)
  - They can’t be good by themselves, but could help when used together
- Some strings appear to vary from sample to sample
  - c:\9.exe, c:\99.exe
- These stand out:
  - PROXY_PROXY_PROXY_PROXY and
  - OROXY

BlueTraveller a solution:

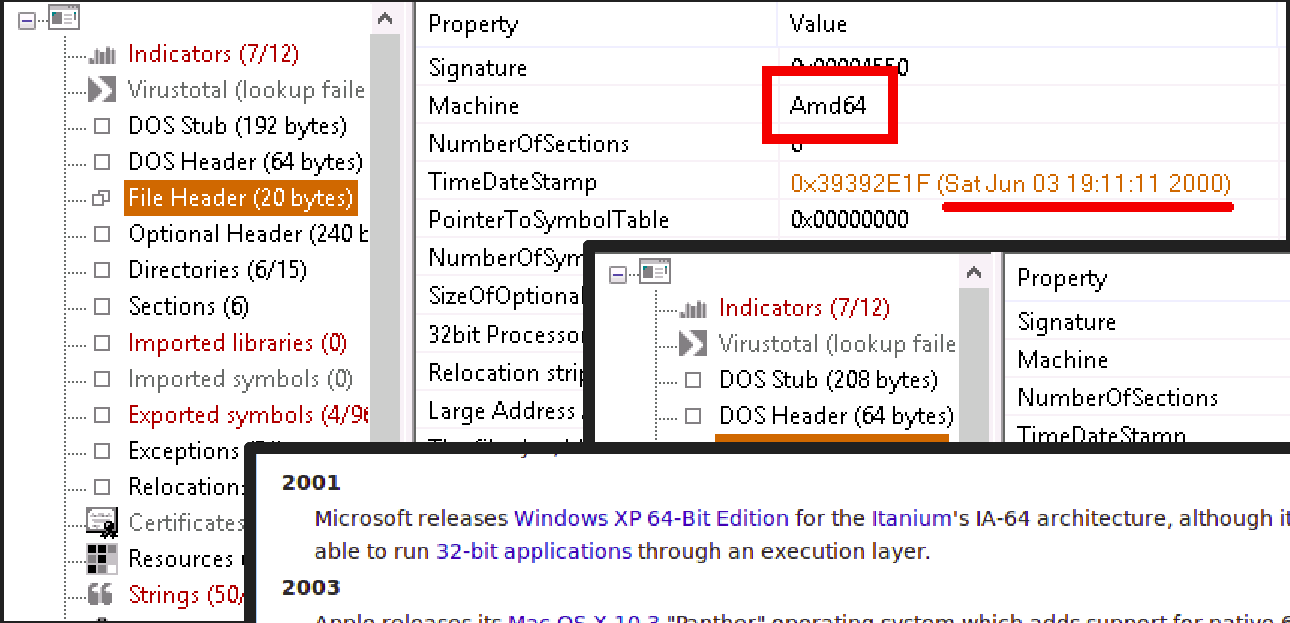
(meta: removed to fit screen)

```
rule apt_CN_BlueTraveller {

strings:

    $a1 = "PROXY_PROXY_PROXY_PROXY"  fullword ascii
    $a2 = "0ROXY_TYPE" fullword ascii

    $b1 = "cmd.exe /c hostname" fullword ascii
    $b2 = "/O.htm" fullword ascii
    $b3 = "%s%04d/%s" fullword ascii
    $b4 = "http://%s/%s/%s/" fullword ascii

    $c1 = "cmd.exe /c" fullword ascii
    $c2 = "Upload failed..." fullword ascii
    $c3 = "Download OK!" fullword ascii
    $c4 = "-download" fullword ascii
    $c5 = "-exit" fullword ascii
    $c6 = "james" fullword ascii

condition:
    uint16(0) == 0x5A4D and
    (any of ($a*) or 2 of ($b*) or 4 of ($c*))
    and filesize < 400000
}
```

Magic for creative minds

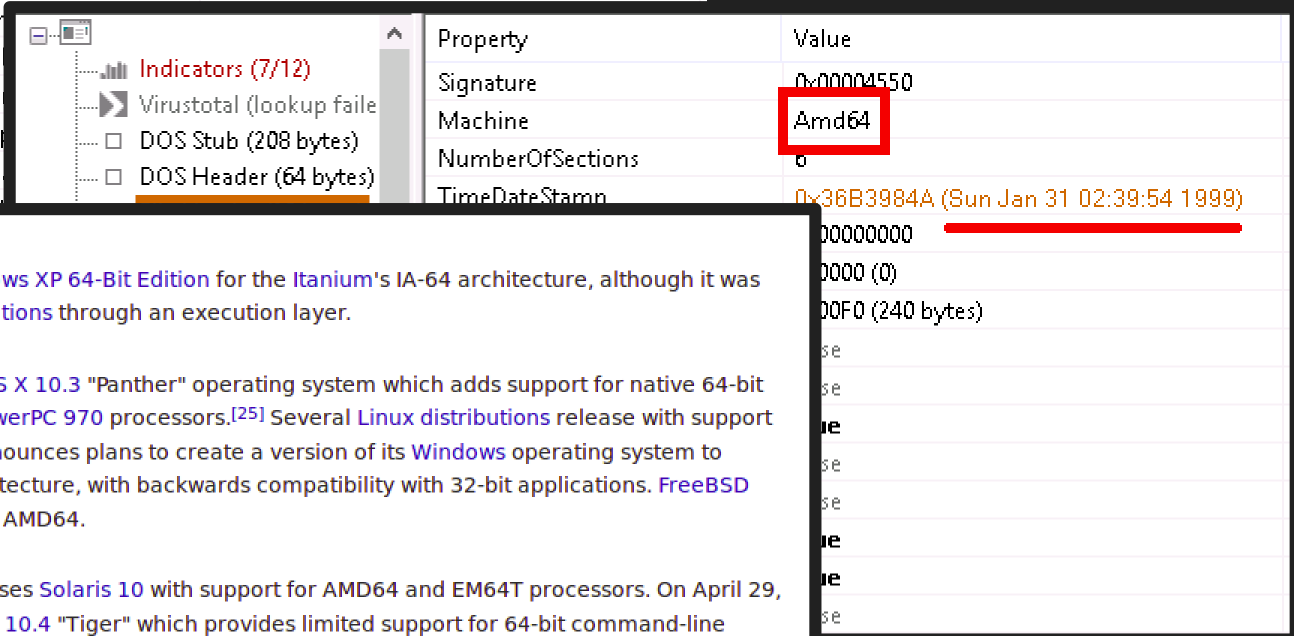
Example 2: Equation’s TripleFantasy

- We have two 64-bit samples
  - 79fc0dd4004f015c28b98d1c2bbf4612
  - ab100f3bc6708f576976e8dc6221acef
- We need to find something very unique / unusual with them and make a rule to catch it
- We WILL NOT reverse engineer the samples
- We look for logical semantics anomaly



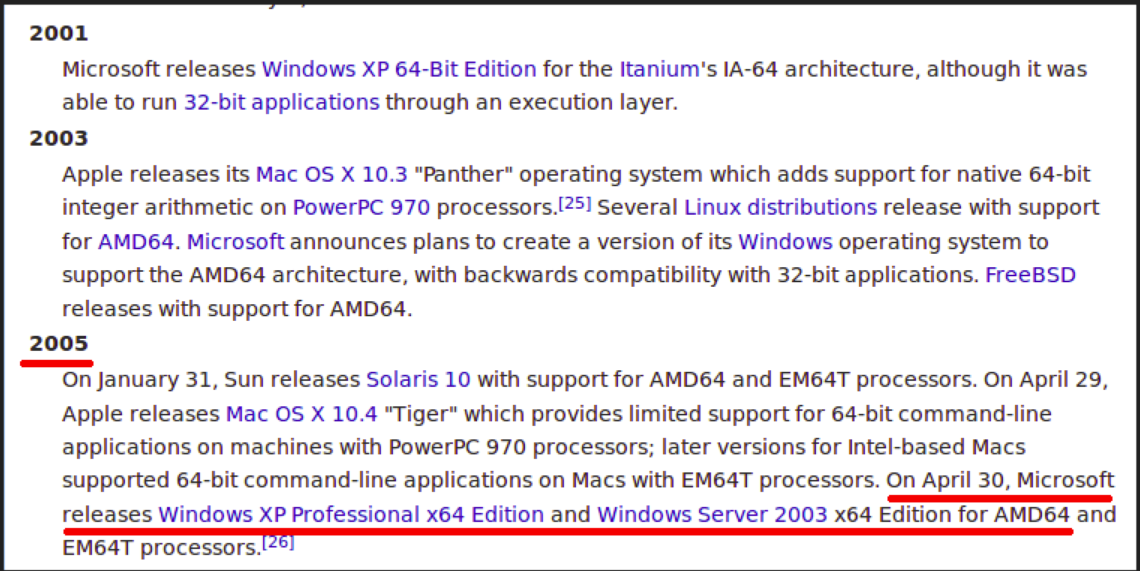
Indicators (7/12)
Virustotal (lookup faile
DOS Stub (192 bytes)
DOS Header (64 bytes)
File Header (20 bytes)
Optional Header (240 b
Directories (6/15)
Sections (6)
Imported libraries (0)
Imported symbols (0)
Exported symbols (4/9
Exceptions
Relocation
Certificates
Resources
Strings (50

Property	Value
Signature	0x00004550
Machine	Amd64
NumberOfSections	6
TimeDateStamp	0x39392E1F (Sat Jun 03 19:11:11 2000)
PointerToSymbolTable	0x00000000
NumberOfSym	
SizeOfOptiona	
32bit Processo	
Relocation stri	
Large Address	



Indicators (7/12)
Virustotal (lookup faile
DOS Stub (208 bytes)
DOS Header (64 bytes)

Property	Value
Signature	0x00004550
Machine	Amd64
NumberOfSections	6
TimeDateStamp	0x36B3984A (Sun Jan 31 02:39:54 1999)
	00000000
	0000 (0)
	00F0 (240 bytes)
	se
	se
	ue
	se
	se
	ue
	ue
	se



2001

Microsoft releases Windows XP 64-Bit Edition for the Itanium's IA-64 architecture, although it was able to run 32-bit applications through an execution layer.

2003

Apple releases its Mac OS X 10.3 "Panther" operating system which adds support for native 64-bit integer arithmetic on PowerPC 970 processors.[25] Several Linux distributions release with support for AMD64. Microsoft announces plans to create a version of its Windows operating system to support the AMD64 architecture, with backwards compatibility with 32-bit applications. FreeBSD releases with support for AMD64.

2005

On January 31, Sun releases Solaris 10 with support for AMD64 and EM64T processors. On April 29, Apple releases Mac OS X 10.4 "Tiger" which provides limited support for 64-bit command-line applications on machines with PowerPC 970 processors; later versions for Intel-based Macs supported 64-bit command-line applications on Macs with EM64T processors. On April 30, Microsoft releases Windows XP Professional x64 Edition and Windows Server 2003 x64 Edition for AMD64 and EM64T processors.[26]

TripleFantasy

A solution:

```
import "pe"

rule susp_too_old_x64_PE_samples {

condition:
     uint16(0) == 0x5A4D
          and (pe.machine == pe.MACHINE_AMD64
          or pe.machine == pe.MACHINE_IA64)
     and pe.timestamp > 631155661    // 1990-01-01
     and pe.timestamp < 1072915200 // 2004-01-01
          and filesize < 2000000
}
```

Example 3: find fake MS-signed Chinese samples

- let’s try to make a rule that generically finds:
  - Chinese language samples
  - See: https://msdn.microsoft.com/en-us/library/windows/desktop/dd318693%28v=vs.85%29.aspx
  - which are signed
  - ...but not by Microsoft
  - and mimic legit Microsoft files by re-using their PE metainfo

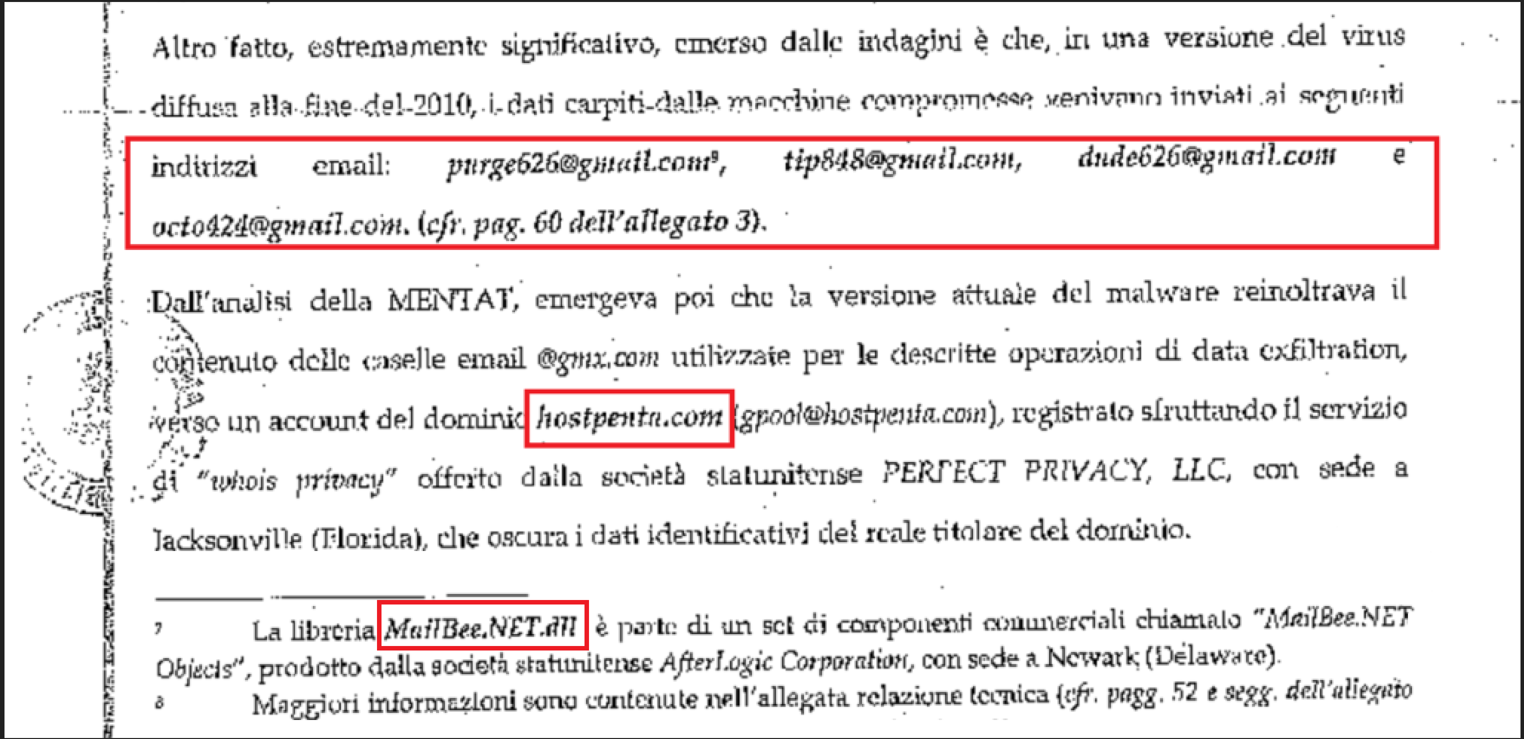
Signed Chinese samples

A solution:

```
import "pe"

rule susp_signed_CN_fake_MS_metainfo {

condition:

    uint16(0) == 0x5A4D
    and filesize < 1000000

    and pe.version_info["CompanyName"] contains "Microsoft"
    and pe.number_of_signatures > 0
    and not for all i in (0..pe.number_of_signatures - 1):
    (
            pe.signatures[i].issuer contains "Microsoft" or
            pe.signatures[i].issuer contains "VeriSign"
    )
    and pe.language(0x04) // LANG_CHINESE
}
```

Note: pe.language is broken in Yara 3.11.0

Example: EyePyramid



Altro fatto, estremamente significativo, emerso dalle indagini è che, in una versione del virus diffusa alla fine del 2010, i dati carpiti dalle macchine compromesse venivano inviati ai seguenti indirizzi email: *purge626@gmail.com*[8], *tip848@gmail.com*, *dude626@gmail.com* e *octo424@gmail.com*. (*cfr. pag. 60 dell'allegato 3*).

Dall'analisi della MENTAT, emergeva poi che la versione attuale del malware reinoltrava il contenuto delle caselle email *@gmx.com* utilizzate per le descritte operazioni di data exfiltration, verso un account del dominio *hostpenta.com* (*gpool@hostpenta.com*), registrato sfruttando il servizio di *"whois privacy"* offerto dalla società statunitense PERFECT PRIVACY, LLC, con sede a Jacksonville (Florida), che oscura i dati identificativi del reale titolare del dominio.

7 La libreria *MailBee.NET.dll* è parte di un set di componenti commerciali chiamato *"MailBee.NET Objects"*, prodotto dalla società statunitense *AfterLogic Corporation*, con sede a Newark (Delaware).

8 Maggiori informazioni sono contenute nell'allegata relazione tecnica (*cfr. pagg. 52 e segg. dell'allegato*

Example: EyePyramid

E-mail's used for exfiltration

- gpool@hostpenta[.]com
- hanger@hostpenta[.]com
- hostpenta@hostpenta[.]com
- purge626@gmail[.]com
- tip848@gmail[.]com
- dude626@gmail[.]com
- octo424@gmail[.]com
- tim11235@gmail[.]com
- plars575@gmail[.]com

Command-and-Control Servers

- eyepyramid[.]com
- hostpenta[.]com
- ayexisfitness[.]com
- enasrl[.]com
- eurecoove[.]com
- marashen[.]com
- millertaylor[.]com
- occhionero[.]com
- occhionero[.]info
- wallserv[.]com
- westlands[.]com

Excerpt from the Italian court order:
http://www.agi.it/pictures/pdf/agi/agi/2017/01/10/132733992-5cec4d88-49a1-4a00-8a01-dde65baa5a68.pdf

Case study: EyePyramid

```
rule apt_ZZ_EyePyramid {
strings:
        $a0="eyepyramid.com" ascii wide nocase fullword
        $a1="hostpenta.com" ascii wide nocase fullword
        $a2="ayexisfitness.com" ascii wide nocase fullword
        $a3="enasrl.com" ascii wide nocase fullword
        $a4="eurecoove.com" ascii wide nocase fullword
        $a5="marashen.com" ascii wide nocase fullword
        $a6="millertaylor.com" ascii wide nocase fullword
        $a7="occhionero.com" ascii wide nocase fullword
        $a8="occhionero.info" ascii wide nocase fullword
        $a9="wallserv.com" ascii wide nocase fullword
        $a10="westlands.com" ascii wide nocase fullword
        $a11="217.115.113.181” ascii wide nocase fullword
        $a12="216.176.180.188” ascii wide nocase fullword
        $a13="65.98.88.29” ascii wide nocase fullword
        $a14="199.15.251.75” ascii wide nocase fullword
        $a15="216.176.180.181” ascii wide nocase fullword
        $b0="purge626@gmail.com" ascii wide fullword
        $b1="tip848@gmail.com" ascii wide fullword
        $b2="dude626@gmail.com" ascii wide fullword
        $b3="octo424@gmail.com" ascii wide fullword
        $b4="antoniaf@poste.it" ascii wide fullword
        $b5="mmarcucci@virgilio.it" ascii wide fullword
        $b6="i.julia@blu.it" ascii wide fullword
        $b7="g.simeoni@inwind.it" ascii wide fullword
        $b8="g.latagliata@live.com" ascii wide fullword
        $b9="rita.p@blu.it" ascii wide fullword
        $b10="b.gaetani@live.com" ascii wide fullword
        $b11="gpierpaolo@tin.it" ascii wide fullword
        $b12="e.barbara@poste.it" ascii wide fullword
        $b13="stoccod@libero.it" ascii wide fullword
        $b14="g.capezzone@virgilio.it" ascii wide fullword
condition:
        ((uint16(0) == 0x5A4D)) and (filesize < 10MB) and
        ((any of ($a*)) or (any of ($b*)) )
}
```

Example 4: EyePyramid

- Using the **known emails and CnC names** we found 3 samples:
  - 778d103face6ad7186596fb0ba2399f2
  - 47bea4236184c21e89bd1c1af3e52c86
  - 4ef78f1c020b47719f7f41e367a06739
- How to make an effective rule to find more?
- Hint: let's use pe.imphash (see PEStudio)

EyePyramid Solution:

```
import "pe"

rule crime_ZZ_EyePyramid_imphash {
meta:
         description = "Yara rule to detect EyePyramid samples"
         author = "Kaspersky"

strings:

       $a = "MailBee.NET" ascii wide fullword

condition:

         uint16(0) == 0x5A4D and
         $a and
         filesize < 10MB and
         pe.imphash() == "f34d5f2d4577ed6d9ceec516c1f5a744"
}
```

- The imphash value should be written in lowercase
- pe.imphash() == "F34D5F2D4577ED6D9CEEC516C1F5A744" will NOT work

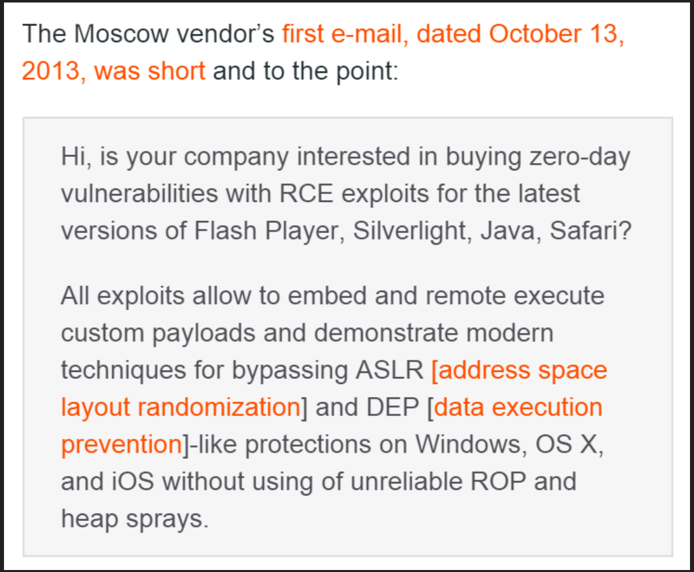
Example 5: Vitaly Toropov's Silverlight 0-day

A Silverlight zero-day exploit is on the loose. How can we find it?

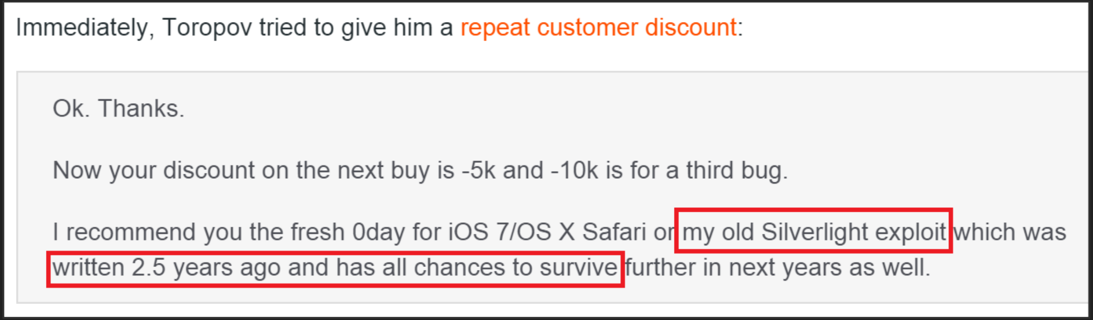
Vitaly Toropov's Silverlight 0-day

- July 2015
- How a Russian hacker made $45,000 selling a 0-day Flash exploit to Hacking Team
- http://arstechnica.com/security/2015/07/how-a-russian-hacker-made-45000-selling-a-zero-day-flash-exploit-to-hacking-team/

Vitaly Toropov's Silverlight 0-day


The Moscow vendor's first e-mail, dated October 13, 2013, was short and to the point:

> Hi, is your company interested in buying zero-day vulnerabilities with RCE exploits for the latest versions of Flash Player, Silverlight, Java, Safari?
>
> All exploits allow to embed and remote execute custom payloads and demonstrate modern techniques for bypassing ASLR [address space layout randomization] and DEP [data execution prevention]-like protections on Windows, OS X, and iOS without using of unreliable ROP and heap sprays.

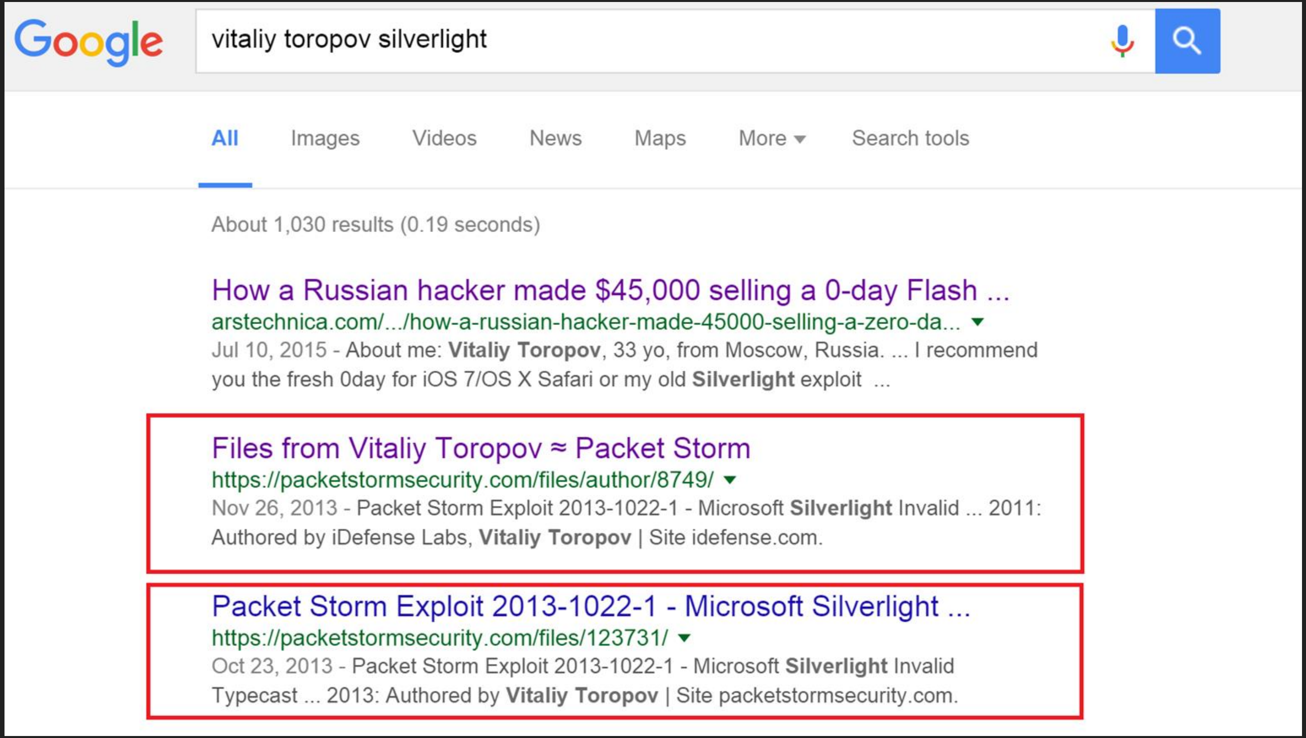

Immediately, Toropov tried to give him a repeat customer discount:

> Ok. Thanks.
>
> Now your discount on the next buy is -5k and -10k is for a third bug.
>
> I recommend you the fresh 0day for iOS 7/OS X Safari or my old Silverlight exploit which was written 2.5 years ago and has all chances to survive further in next years as well.

"my old Silverlight exploit which was written 2.5 years ago"

OMG! Silverlight zero-day? Let's find it! But how?

Vitaly Toropov's Silverlight 0-day



vitaliy toropov silverlight

All Images Videos News Maps More Search tools

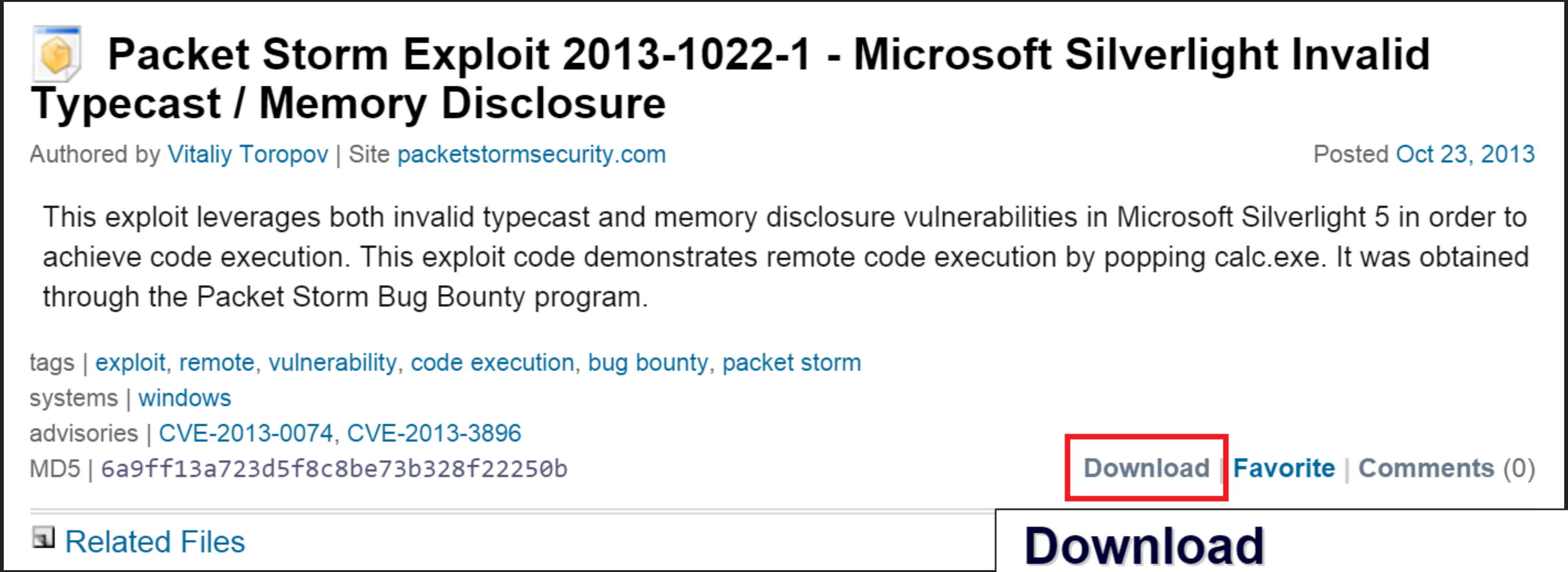
About 1,030 results (0.19 seconds)

How a Russian hacker made $45,000 selling a 0-day Flash ...
arstechnica.com/.../how-a-russian-hacker-made-45000-selling-a-zero-da...
Jul 10, 2015 - About me: **Vitaliy Toropov**, 33 yo, from Moscow, Russia. ... I recommend you the fresh 0day for iOS 7/OS X Safari or my old **Silverlight** exploit ...

Files from Vitaliy Toropov ≈ Packet Storm
https://packetstormsecurity.com/files/author/8749/
Nov 26, 2013 - Packet Storm Exploit 2013-1022-1 - Microsoft **Silverlight** Invalid ... 2011: Authored by iDefense Labs, **Vitaliy Toropov** | Site idefense.com.

Packet Storm Exploit 2013-1022-1 - Microsoft Silverlight ...
https://packetstormsecurity.com/files/123731/
Oct 23, 2013 - Packet Storm Exploit 2013-1022-1 - Microsoft **Silverlight** Invalid Typecast ... 2013: Authored by **Vitaliy Toropov** | Site packetstormsecurity.com.

Vitaly Toropov's Silverlight 0-day



Packet Storm Exploit 2013-1022-1 - Microsoft Silverlight Invalid Typecast / Memory Disclosure

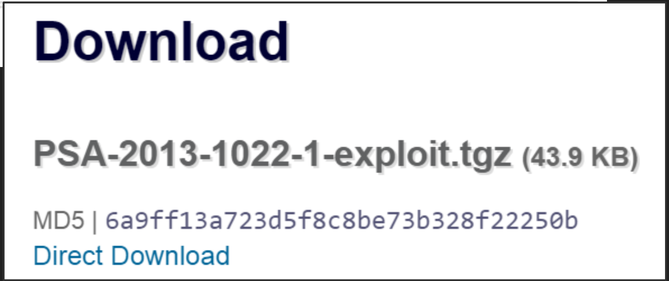
Authored by Vitaliy Toropov | Site packetstormsecurity.com Posted Oct 23, 2013

This exploit leverages both invalid typecast and memory disclosure vulnerabilities in Microsoft Silverlight 5 in order to achieve code execution. This exploit code demonstrates remote code execution by popping calc.exe. It was obtained through the Packet Storm Bug Bounty program.

tags | exploit, remote, vulnerability, code execution, bug bounty, packet storm
systems | windows
advisories | CVE-2013-0074, CVE-2013-3896
MD5 | `6a9ff13a723d5f8c8be73b328f22250b`

Download | Favorite | Comments (0)

Related Files



Download

PSA-2013-1022-1-exploit.tgz (43.9 KB)

MD5 | `6a9ff13a723d5f8c8be73b328f22250b`
Direct Download

Vitaly Toropov's Silverlight 0-day

- Starting point: 6a9ff13a723d5f8c8be73b328f22250b (tgz file)
- Can be found:

  https://dl.packetstormsecurity.net/1310-exploits/PSA-2013-1022-1-exploit.tgz

Let's write an effective Yara rule for it!

Solution:

```
rule exploit_Silverlight_Toropov_Generic_XAP {

meta:
            author = "Costin Raiu, Kaspersky Lab"
            maltype = "apt"
            type ="APT"
            filetype = "Win32 EXE"
            date = "2015-06-23"
            version = "1.0"

strings:
            $b2="Can't find Payload() address"  ascii wide
            $b3="/SilverApp1;component/App.xaml" ascii wide
            $b4="Can't allocate ums after buf[]" ascii wide
            $b5="------------ START ------------" ascii wide

condition:
            ((uint16(0) == 0x5A4D)) and (filesize < 5000000)
and
            ( (2 of ($b*)) )

}
```

https://securelist.com/the-mysterious-case-of-cve-2016-0034-the-hunt-for-a-microsoft-silverlight-0-day/73255/

Bonus: Vitaly Toropov’s other zero-day’s

- Special homework
- Write Yara rules for Toropov’s OSX and iOS Safari exploits

Using automatic rules generators

Automatic Yara rule generators

- yarGen - by Florian Roth
  - Latest update: February 2017
  - Best Yara generator ← in our opinion
- yara-generator.net from JoeSandbox
  - Requires registration and approval (24 hours)
- XenOphOn by Chris Clark, 2013
  - basic CLI python yara generator
- autorule by Joxean Koret, 2012
  - experimental python code

yarGen by Florian Roth

Platform: Python (with dependencies)
Available on github: https://github.com/Neo23x0/yarGen

Weak sides:

- Memory exhaustive (requires up to 4Gb of RAM)

Strong sides:

- The tool is very mature and follows best practices (implements automatic size limit, header recognition, binary pattern detection, string whitelisting)
- A life-saver on a large set of files
- Automatically generated rules can be quickly fine-tuned

Innovations:

- Inverse-mode to detect system files anomalies

"Do-It-Yourself" Yara generator

- Download StringAnalyzer tool by Mark Russinovich (SysInternals)
- Extract all strings from popular clean samples
- Put into a SQL DB
- Calculate strings popularity
- Store all relations of samples and strings

Now you can:

- find unique strings inside of a sample (not appearing in any popular samples)
- find common and unique strings specific for a malware family
- find common set of non-unique strings specific for a malware family

Example of auto-generated rule based on unique strings:

```
rule apt_ZZ_Sofacy_linkinfo_uniques {

strings:

 $a1 = "unsuccess&nbsp:&nbsp" fullword wide                        // 2 of 5
 $a2 = "ReleaseMutex error!" fullword ascii                        // 2 of 5
 $a3 = "<3tm<4ti<5te<6ta" fullword ascii                           // 2 of 5
 $a4 = "Active Acceessibility Resource File" fullword wide    // 1 of 5
 $a5 = "11.0.4621.4331splm.dll" fullword wide                      // 1 of 5
 $a6 = "13.5.6765.37769" fullword wide                             // 1 of 5
 $a7 = "11.0.4621.4331" fullword wide                              // 1 of 5
 $a8 = "5.1.2600.2185" fullword wide                               // 1 of 5

condition:
 uint16(0) == 0x5A4D and
 filesize < 1000000 and
 2 of them  // any of these strings is unique but better to use 2
}
```

Example of auto-generated rule based on non-unique strings:

```
rule apt_ZZ_Sofacy_linkinfo_heur {
strings:
 $a1 = "CreateLinkInfo" fullword ascii
 $a2 = "CreateLinkInfoA" fullword ascii
 $a3 = "ResolveLinkInfo" fullword ascii
 $a4 = "ResolveLinkInfoA" fullword ascii
 $a5 = "GetCanonicalPathInfo" fullword ascii
 $a6 = "GetCanonicalPathInfoA" fullword ascii
 $a7 = "DisconnectLinkInfo" fullword ascii
 $a8 = "GetCanonicalPathInfoW" fullword ascii
 $a9 = "CompareLinkInfoVolumes" fullword ascii
 $a10 = "CompareLinkInfoReferents" fullword ascii
 $a11 = "ResolveLinkInfoW" fullword ascii
 $a12 = "GetLinkInfoData" fullword ascii
 $a13 = "linkinfo.dll" fullword ascii
 $a14 = "CreateLinkInfoW" fullword ascii
 $a15 = "DestroyLinkInfo" fullword ascii
 $a16 = "IsValidLinkInfo" fullword ascii
 $a17 = "HttpSendRequestExW" fullword ascii
 $a18 = ".?AV_Ref_count_base@tr1@std@@" fullword
 $a19 = "InternetQueryDataAvailable" fullword ascii
 $a20 = "HttpOpenRequestA" fullword ascii
 $a21 = "InternetConnectW" fullword ascii
 $a22 = "InternetOpenW" fullword ascii
 $a23 = "InternetReadFile" fullword ascii
 $a24 = "GetUserNameW" fullword ascii
 $a25 = "GetVolumeInformationW" fullword ascii
 $a26 = "SystemFunction036" fullword ascii
 $a27 = "lstrcatW" fullword ascii
 $a28 = "WININET.dll" fullword ascii
 $a29 = "ADVAPI32.DLL" fullword wide
 $a30 = "GetEnvironmentVariableW" fullword ascii
 $a31 = "GetExitCodeThread" fullword ascii
 $a32 = "TerminateThread" fullword ascii
condition:

 uint16(0) == 0x5A4D and
 filesize < 1000000 and
 28 of them  // you can safely remove 4 garbage strings

}
```

Hunting with Yara like a GReAT Ninja

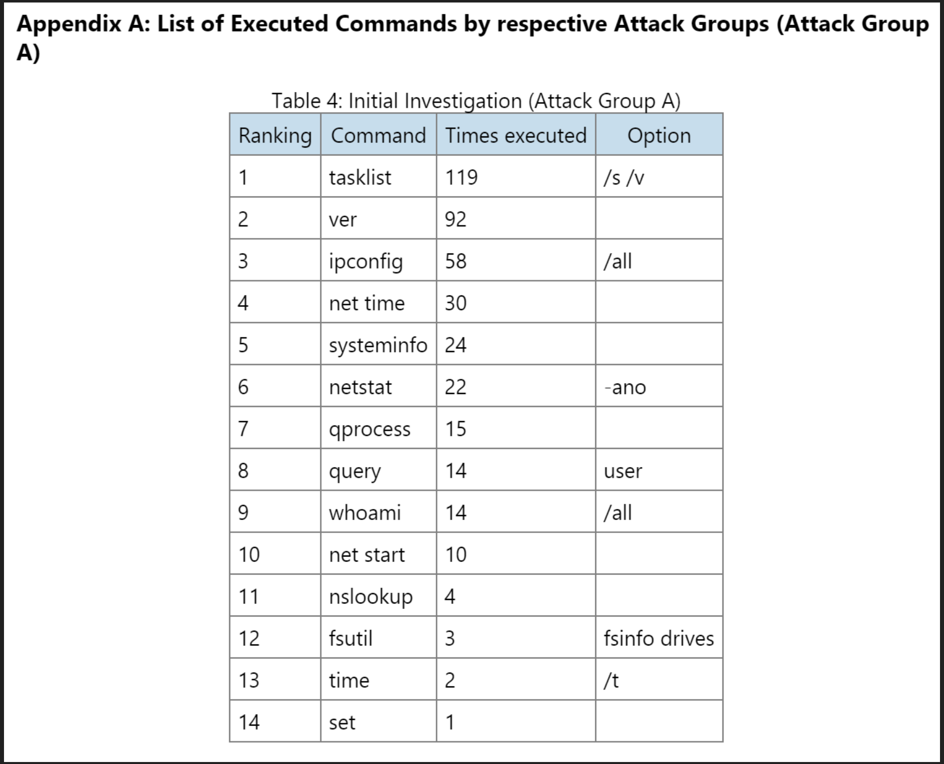
Search of anomalies

- Fake timestamp
- Fake signature
- Connections to known bad Dynamic DNS domains
- Mimics legitimate files (Adobe or MS or Skype etc)
  - Using legit PE info but not signed/trusted
  - Using legit resources (Skype icon for example)
  - Using "legit" metadata and wrong entropy / code page / exports
- By lateral movement patterns
- By crypto signatures
- By atypical actions for this file type

```
rule susp_WinRAR_DownloadToFile {

strings:

    $a1 = "WinRAR\\shell\\open\\command"
fullword nocase ascii

    $a2 = "URLDownloadToFile" nocase ascii

condition:

    uint16(0) == 0x5A4D

    and all of ($a*)

    and not kaspersky

    and filesize < 1000000

}
```

```
rule susp_localIP {

strings:

    $a1 = "cmd /c " nocase ascii wide

    $a2 = "SELECT * FROM AntiVirusProduct"
nocase ascii wide

    $a3 = "recycle.bin\\" nocase

    $b1 = "192.168." nocase ascii wide

condition:

    uint16(0) == 0x5A4D

    and any of ($a*) and $b1

}
```

Windows commands used for lateral movement



Appendix A: List of Executed Commands by respective Attack Groups (Attack Group A)

Table 4: Initial Investigation (Attack Group A)

Ranking	Command	Times executed	Option
1	tasklist	119	/s /v
2	ver	92	
3	ipconfig	58	/all
4	net time	30	
5	systeminfo	24	
6	netstat	22	-ano
7	qprocess	15	
8	query	14	user
9	whoami	14	/all
10	net start	10	
11	nslookup	4	
12	fsutil	3	fsinfo drives
13	time	2	/t
14	set	1	

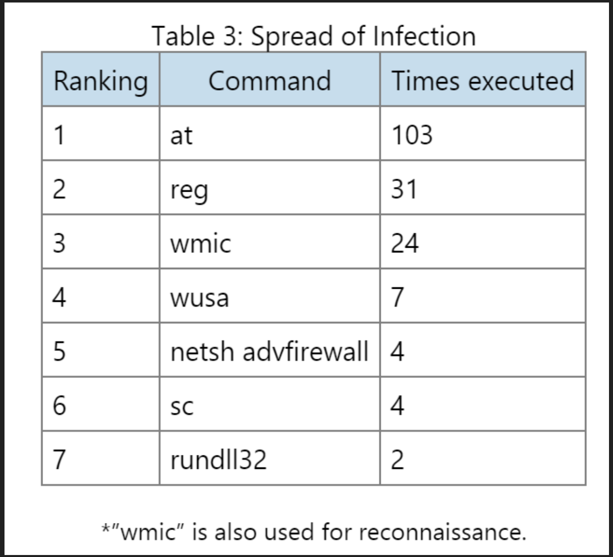
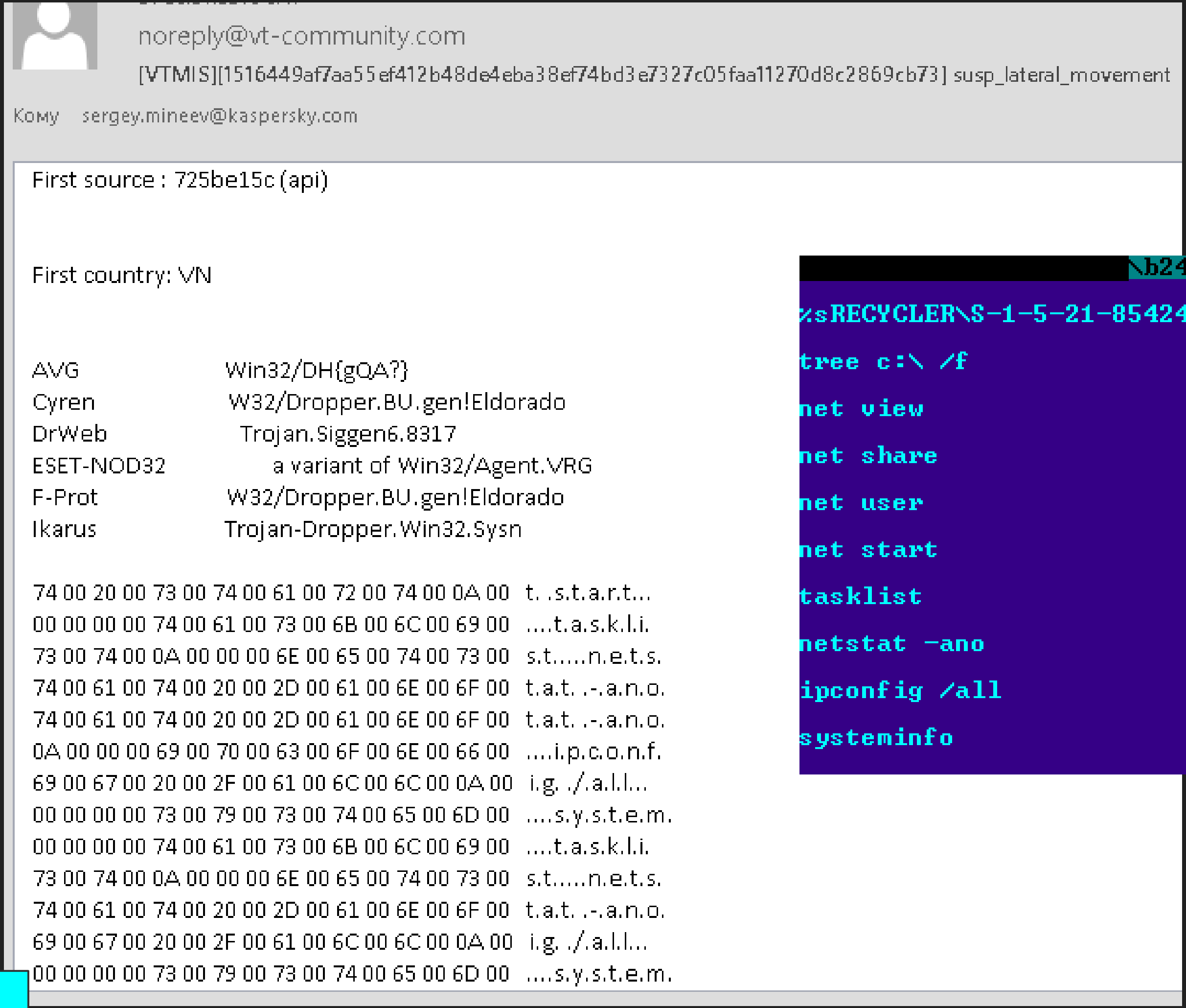


Table 3: Spread of Infection

Ranking	Command	Times executed
1	at	103
2	reg	31
3	wmic	24
4	wusa	7
5	netsh advfirewall	4
6	sc	4
7	rundll32	2

*"wmic" is also used for reconnaissance.

JPCERT CC® Official Blog
Japan Computer Emergency Response Team Coordination Center



noreply@vt-community.com

[VTMIS][1516449af7aa55ef412b48de4eba38ef74bd3e7327c05faa11270d8c2869cb73] susp_lateral_movement

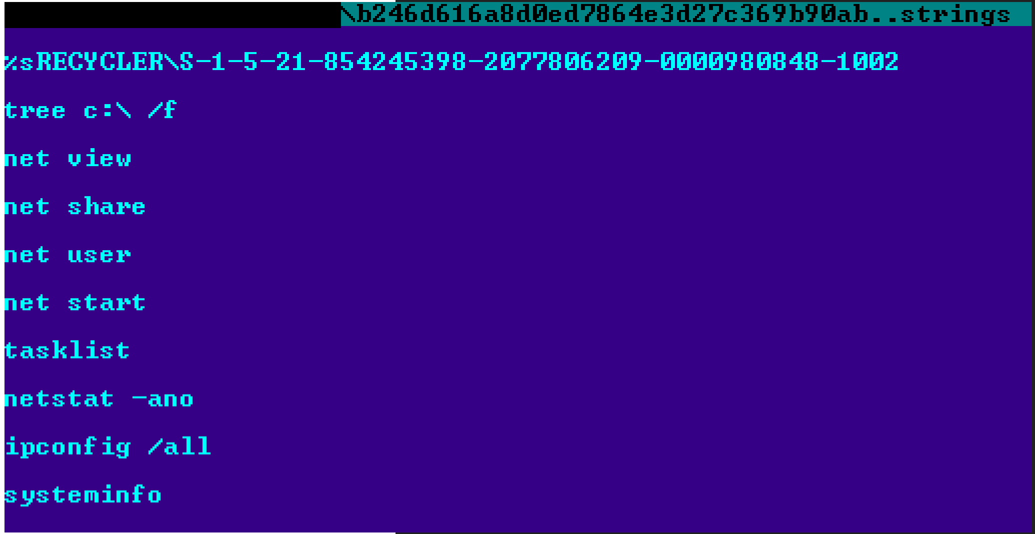
Кому sergey.mineev@kaspersky.com

First source : 725be15c (api)

First country: VN

AVG	Win32/DH{gQA?}
Cyren	W32/Dropper.BU.gen!Eldorado
DrWeb	Trojan.Siggen6.8317
ESET-NOD32	a variant of Win32/Agent.VRG
F-Prot	W32/Dropper.BU.gen!Eldorado
Ikarus	Trojan-Dropper.Win32.Sysn

```
74 00 20 00 73 00 74 00 61 00 72 00 74 00 0A 00  t. .s.t.a.r.t...
00 00 00 00 74 00 61 00 73 00 6B 00 6C 00 69 00  ....t.a.s.k.l.i.
73 00 74 00 0A 00 00 00 6E 00 65 00 74 00 73 00  s.t.....n.e.t.s.
74 00 61 00 74 00 20 00 2D 00 61 00 6E 00 6F 00  t.a.t. .-.a.n.o.
74 00 61 00 74 00 20 00 2D 00 61 00 6E 00 6F 00  t.a.t. .-.a.n.o.
0A 00 00 00 69 00 70 00 63 00 6F 00 6E 00 66 00  ....i.p.c.o.n.f.
69 00 67 00 20 00 2F 00 61 00 6C 00 6C 00 0A 00  i.g. ./.a.l.l...
00 00 00 00 73 00 79 00 73 00 74 00 65 00 6D 00  ....s.y.s.t.e.m.
00 00 00 00 74 00 61 00 73 00 6B 00 6C 00 69 00  ....t.a.s.k.l.i.
73 00 74 00 0A 00 00 00 6E 00 65 00 74 00 73 00  s.t.....n.e.t.s.
74 00 61 00 74 00 20 00 2D 00 61 00 6E 00 6F 00  t.a.t. .-.a.n.o.
69 00 67 00 20 00 2F 00 61 00 6C 00 6C 00 0A 00  i.g. ./.a.l.l...
00 00 00 00 73 00 79 00 73 00 74 00 65 00 6D 00  ....s.y.s.t.e.m.
```



```
[redacted]\b246d616a8d0ed7864e3d27c369b90ab..strings

%sRECYCLER\S-1-5-21-854245398-2077806209-0000980848-1002
tree c:\ /f
net view
net share
net user
net start
tasklist
netstat -ano
ipconfig /all
systeminfo
```

```
rule susp_exe_23lldnur {

strings:

     $a1 = "exe.23lldnur" nocase ascii wide

     $a2 = "lld.23lenrek" nocase ascii wide

     $a3 = "lld.23llehs" nocase ascii wide

     $a4 = "lld.23resu" nocase ascii wide

     $a5 = "lld.teniniw" nocase ascii wide

condition:

     uint16(0) == 0x5A4D

     and any of ($a*) and filesize < 1000000

}
```

```
rule susp_security_tricks {

strings:

     $a1 = "InitializeAcl" nocase

     $a2 = "InitializeSecurityDescriptor"
nocase

     $a3 = "AllocateAndInitializeSid" nocase

     $b1 = "socket" nocase ascii wide

     $b2 = "namedpipe" nocase ascii wide

condition:

     uint16(0) == 0x5A4D

     and all of ($a*)  and any of ($b*)

}
```

Hungry for more?

- Our full Yara training is 2 days long
    - 21 exercises
    - More complex detection cases
    - More APTs
- Consider starting to write your own rules
    - Look at available examples from reputable sources (eg. Florian Roth)
    - Test your own rules
- Deploy KLARA
    - Build a clean test set
    - Test rules before deployment
- Let us know if you'd be interested in more online trainings!

Thank you!

Happy APT hunting!

Questions, ideas, suggestions:

yarawebinar@kaspersky.com