

# Opstune.com

## How rare is a rare HTTP agent? Context-rich alerts because of math

### Known unknowns?

This time I start with another *cyber cliché*:

*There are known knowns. There are things we know we know. We also know there are known unknowns. That is to say, we know there are some things we do not know. But there are also unknown unknowns, the ones we don't know we don't know.*

Secretary of Defense Donald Rumsfeld, 2002

While some vendors claim they can spot the super badness “Unknowns Unknowns” using AI and Deep (really deep) Learning, I here present to you a simple way to spot “Known Unknowns” just to start your journey, hope you enjoy!

Despite being an atomic indicator, the mechanisms for inspecting the *HTTP user agent* are still a challenge in large scale environments. With the amount of data pouring into the SIEM from web proxies and internet facing web servers, detection engineers need to use creativity to win.

Not surprisingly, Splunk does provide commands such as [rare](https://docs.splunk.com/Documentation/Splunk/8.0.6/SearchReference/Rare) (<https://docs.splunk.com/Documentation/Splunk/8.0.6/SearchReference/Rare>) and [top](https://docs.splunk.com/Documentation/Splunk/8.0.6/SearchReference/Top) (<https://docs.splunk.com/Documentation/Splunk/8.0.6/SearchReference/Top>), which allow users to organize results based on least or most frequent values. But is it as easy as counting the number of occurrences of an agent string and firing an alarm?

### How rare ‘rare’ is?

Unless you want SOC analysts checking every single new agent string, which is not practical in most environments, math/stats can help you select better **alert qualifiers** with more context information.

What I am about to show is a simple unsupervised machine learning technique ✨ In case you haven't figured, the title is an allusion to a nice talk from a Brazilian friend, Alex Pinto: Secure because of Math (<https://www.youtube.com/watch?v=5NeOIhfkD0Q>).

For allowing you to play and fine-tune the searches for your environment, I'm going to use Splunk's Boss of the SOC (BOTS) v3 dataset in the following examples. In case you missed that, check it out (<https://github.com/splunk/botsv3>)!

However, since the dataset is not "accelerated", I'm going to cheat and backfill the Web data model (DM) with all events in the dataset containing a non-null `http_user_agent` field. If you are not sure how to do that, check the docs (<https://docs.splunk.com/Documentation/Splunk/8.0.5/Knowledge/Acceleratedatamodels>) or stop by Splunk's Slack channels (<https://splunk-usergroups.slack.com/team/U418E1BPB>), and say 'Hi!'

## How to find The rarest agent string from BOTS?

I tend to go with \*stats commands ([https://www.splunk.com/en\\_us/blog/tips-and-tricks/search-command-stats-eventstats-and-streamstats-2.html](https://www.splunk.com/en_us/blog/tips-and-tricks/search-command-stats-eventstats-and-streamstats-2.html)), as they provide more flexibility, but same can be achieved with *rare* and *top* as mentioned before. One simply needs to *sort* the results afterwards:

The screenshot shows a Splunk search interface with the following search query: `index=botsv3 earliest=0 http_user_agent=* | stats c by http_user_agent | sort 0 +num(c)`. The results are displayed in a table with 35 statistics. The table is sorted by the count of occurrences in descending order.

http_user_agent	Count
ClamAV/0.99.2 (OS: linux-gnu, ARCH: x86_64, CPU: x86_64)	1
Microsoft Office/16.0 (Windows NT 10.0; Microsoft Excel 16.0.10228; Pro)	1
Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/41.0.2228.0 Safari/537.36	1
Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/65.0.3325.181 Safari/537.36	1
Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; Win64; x64; Trident/5.0)	1
OneNote/16.0.10228.20134 (Windows/10.0; Desktop WOW64; en-US; Desktop app; VMware, Inc./VMware7,1)	1
Python-urllib/2.7	1
SEP/14.2.760.0000, MID/{B17CE05D-4C23-1A71-1056-F57493DF00EB}, SID/10	1
SEP/14.2.760.0000, MID/{B17CE05D-4C23-1A71-1056-F57493DF00EB}, SID/10 LUE/2.6.1.11 (Windows;10.0;SP0.0;X64;ENU)	1
SEP/14.2.760.0000, MID/{B17CE05D-4C23-1A71-1056-F57493DF00EB}, SID/10 SEQ/180725021	1
Alprazolam/2.0	2
Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_6) AppleWebKit/601.7.7 (KHTML, like Gecko) Version/9.1.2 Safari/601.7.7	2
Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/51.0.2704.103 Safari/537.36	2
Mozilla/5.0 (compatible; Googlebot/2.1; +http://www.google.com/bot.html)	3
Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/51.0.2704.103 Safari/537.36	4
Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/52.0.2743.116 Safari/537.36	4
urlgrabber/3.10 yum/3.4.3	4
Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/52.0.2743.116 Safari/537.36	5
Windows-Update-Agent/10.0.10011.16384 Client-Protocol/1.81	5
MICROSOFT_DEVICE_METADATA_RETRIEVAL_CLIENT	8
Microsoft Office/16.0 (Windows NT 10.0; Microsoft Outlook 16.0.10228; Pro)	10

Snapshot  
from  
HTTP  
User  
Agents  
available  
from  
raw  
events in  
the  
dataset

Of course, searching over raw events has a penalty since it may open multiple buckets (data directories) for retrieving data, but it already reveals some candidates for the *most rare* agent string observed in the dataset. Now, the same using the *Proxy* object from *Web* DM object:

<pre>  tstats summariesonly=1 c from datamodel=Web where nodename=Web.Proxy by Web.http_user_agent   sort 0 +num(c)</pre>		All time	
✓ 10,177 events (before 15/09/2020 17:27:51.000) No Event Sampling		Job	Verbose Mode
Events (10,177)	Patterns	Statistics (61)	Visualization
100 Per Page	Format	Preview	
Web.http_user_agent			c
ClamAV/0.99.2 (OS: linux-gnu, ARCH: x86_64, CPU: x86_64)			1
Hakai/2.0			1
Hello, World			1
Mayhem/5.0 (Windows NT 6.1; rv:52.0) Gecko/20100101 Firefox/52.0			1
Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/41.0.2228.0 Safari/537.36			1
Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/65.0.3325.181 Safari/537.36			1
Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; Win64; x64; Trident/5.0)			1
Mozilla/5.0 zgrab/0.x			1
OneNote/16.0.10228.20134 (Windows/10.0; Desktop WOW64; en-US; Desktop app; VMware, Inc./VMware7,1)			1
Python-urllib/2.7			1
SEP/14.2.760.0000, MID/{62C166A0-B273-C7E3-2B82-67E515CB53E6}, SID/11			1
SEP/14.2.760.0000, MID/{62C166A0-B273-C7E3-2B82-67E515CB53E6}, SID/11 LUE/2.6.1.11 (Windows;10.0;SP0.0;X64;ENU)			1
SEP/14.2.760.0000, MID/{62C166A0-B273-C7E3-2B82-67E515CB53E6}, SID/11 SEQ/180725021			1
SEP/14.2.760.0000, MID/{B17CE05D-4C23-1A71-1056-F57493DF00EB}, SID/10			1
SEP/14.2.760.0000, MID/{B17CE05D-4C23-1A71-1056-F57493DF00EB}, SID/10 LUE/2.6.1.11 (Windows;10.0;SP0.0;X64;ENU)			1
SEP/14.2.760.0000, MID/{B17CE05D-4C23-1A71-1056-F57493DF00EB}, SID/10 SEQ/180725021			1
SEP/14.2.760.0000, MID/{F9598250-CE4F-F317-56B8-3602D2E3393D}, SID/9			1
SEP/14.2.760.0000, MID/{F9598250-CE4F-F317-56B8-3602D2E3393D}, SID/9 LUE/2.6.1.11 (Windows;10.0;SP0.0;X64;ENU)			1
SEP/14.2.760.0000, MID/{F9598250-CE4F-F317-56B8-3602D2E3393D}, SID/9 SEQ/180725021			1
Alprazolam/2.0			2

Accelerated  
data model  
based  
search for  
unique  
HTTP USer

## Agent strings

This time it took **0.3s** and it reveals 61 distinct user agent strings. While that makes significant difference in my lab (**raw search** completes in almost a **minute**), in a large deployment, this makes a huge difference in use case design and search performance.

So why not alerting on every occurrence of a new, unique agent string? Simply because it would flood the SOC with many, many alerts depending on the environment, rendering the detection unusable in any enterprise environments given the amount of exceptions to be handled from harmless alerts.

The following would be a poor but valid embryonic detection prototype for spotting agents for the first time (in a day) within the last 30 days (daily run, earliest=-30d):

```
| tstats summariesonly=1 min(_time) AS first_time, max(_time) AS last_time
FROM datamodel=Web
WHERE nodename=Web.Proxy|
BY Web.http_user_agent
| eval first_time=strftime(first_time, "%F"), last_time=strftime(last_time, "%F")
| where first_time=last_time AND (strftime(now(), "%F")=last_time)
```

Group  
events  
by  
agent  
string  
when  
they  
happen  
in a  
single  
day  
(today)

Besides the alert volume problem, in case you monitor MTTD (Mean time to detect) and other metrics related to earlier detection, simply setting that query to run hourly would ruin the system performance.

## Splunk's cluster command

This is another overlooked command. It basically groups events based on how similar they are. Of course, if you are looking for a more elaborated version, [MLTK](https://splunkbase.splunk.com/app/2890/) (<https://splunkbase.splunk.com/app/2890/>) provides more options, but this one is available in most Splunk versions and does not require much effort as seen below:

The screenshot shows a Splunk search interface. The search bar contains the following query:

```
| tstats summariesonly=1 c
FROM datamodel=Web
WHERE nodename=Web.Proxy
BY Web.http_user_agent

| cluster field=Web.http_user_agent
```

Below the search bar, it indicates "10,177 events (before 15/09/2020 18:09:15.000)" and "No Event Sampling". The interface shows tabs for "Events (10,177)", "Patterns", "Statistics (46)", and "Visualization". The "Statistics" tab is active, showing a table of cluster results.

Web.http_user_agent	c	cluster_label
Alprazolam/2.0	2	1
ClamAV/0.99.2 (OS: linux-gnu, ARCH: x86_64, CPU: x86_64)	1	2
Debian APT-HTTP/1.3 (1.2.25)	2	3
ELB-HealthChecker/2.0	2385	4
Hakai/2.0	1	5
Hello, World	1	6
Install Service	10	7
MICROSOFT_DEVICE_METADATA_RETRIEVAL_CLIENT	8	8

Group  
similar  
agent  
strings  
in  
clusters

The `cluster` (<https://docs.splunk.com/Documentation/SplunkCloud/8.0.2007/SearchReference/Cluster>) command adds a **cluster\_label** field to each result row, indicating the cluster it is assigned to. It also takes `_raw` as a default target field, but in this case we pick DM's `Web.http_user_agent` field.

How to control what goes in and out of a cluster?

That's the `t` parameter (threshold) which controls the sensitivity. The closer it is to 1, the more similar events have to be for them to be considered in the same cluster. In short: lower value = less clusters.

Below is a more elaborated version of the previous command, now showing the cluster members and each cluster members count. Also, the default `t` value (0.8) is changed to 0.6:



```

| tstats summariesonly=1 c
  FROM datamodel=Web
  WHERE nodename=Web.Proxy
  BY Web.http_user_agent

| rename Web.* AS *
| cluster t=0.6 labelonly=1 field=http_user_agent
| stats count AS event_count, dc(http_user_agent) AS agent_count, |values(http_user_agent) AS agents by
  cluster_label

```

✓ 10,177 events (before 15/09/2020 18:24:59.000) No Event Sampling ▾
Job ▾ || ■ ↶ ↷ ⏏ ⏴ ⏵
Verbose Mode ▾

Events (10,177)
Patterns
Statistics (41)
Visualization

100 Per Page ▾
Format
Preview ▾

cluster_label	event_count	agent_count	agents
1	1	1	Alprazolam/2.0
10	1	1	Microsoft BITS/7.8
11	3	3	Microsoft Office/16.0 (Windows NT 10.0; Microsoft Excel 16.0.10228; Pro) Microsoft Office/16.0 (Windows NT 10.0; Microsoft Outlook 16.0.10228; Pro) Microsoft Office/16.0 (Windows NT 6.2; MAPI 16.0.10228; Pro)
12	1	1	Microsoft-CryptoAPI/10.0
13	1	1	Microsoft-Delivery-Optimization/10.0
14	1	1	Microsoft-WNS/10.0
15	2	2	Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 10.0; WOW64; Trident/7.0; .NET4.0C; .NET4.0E; Microsoft Outlook 16.0.10228; Microsoft Outlook 16.0.10228; ms-office; MSOffice 16) Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 10.0; WOW64; Trident/7.0; .NET4.0C; .NET4.0E; Microsoft Outlook 16.0.10228; ms-office; MSOffice 16)
16	1	1	Mozilla/5.0
17	1	1	Mozilla/5.0 (Linux; Android 8.1.0; Pixel XL Build/OPM4.171019.021.P1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/67.0.3396.87 Mobile Safari/537.36
18	1	1	Mozilla/5.0 (Macintosh; Intel Mac OS X 10.13; rv:61.0) Gecko/20100101 Firefox/61.0
19	5	5	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/67.0.3396.99 Safari/537.36 Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_6) AppleWebKit/601.7.7 (KHTML, like Gecko) Version/9.1.2 Safari/601.7.7 Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/67.0.3396.99 Safari/537.36 Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/67.0.3396.99 Safari/537.36 Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/68.0.3440.75 Safari/537.36
2	1	1	ClamAV/0.99.2 (OS: linux-gnu, ARCH: x86_64, CPU: x86_64)
20	4	4	Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/51.0.2704.103 Safari/537.36 Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/52.0.2743.116 Safari/537.36 Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/51.0.2704.103 Safari/537.36 Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/52.0.2743.116 Safari/537.36


Clustering  
agent  
strings  
based on  
threshold  
= 0.6

That's pretty cool, isn't it? 🥳

I'm sure some (detection engineers) readers are already wondering how many other potential use cases fit for this simple, yet powerful command.

## Now what?

Think about the following pseudo-code (I like having that when delivering detection documentation):

- 
1. Collect the stats on unique agents hourly (earliest=-1h);
  2. Append the profile of unique user agent strings seen over the last 30 days;
  3. Filter IN: agents from #1 which are NOT present in the agents profile;
  4. Cluster the agents profile data + new, unique agents from #3;
  5. Use the following enrichment points to determine an alert qualifier:
    - 5.1 Is the string matching any CTI's IOCs pattern? (easily done with lookups)
    - 5.2 Is the string related to a usual user/interactive agent (Mozilla)?
    - 5.3 Is the string related to a usual network/file transfer (\*ftp\*, \*scp\*)?
    - 5.4 (this list goes on and on, but I'm stopping here...)
  6. Use the following cluster data points to determine an alert qualifier:
    - 6.1 How many members are in the same cluster the new string is part of?
    - 6.2 What's the most prevalent agent string in the cluster?
    - 6.3 How many distinct daily users or clients are using this cluster?
  7. Determine a threshold (flag) based on alert qualifiers to fire an alert;
  7. Trigger an alert aggregated by agent string when threshold is true.

The **alert qualifiers** (hopefully more engineers/researchers adopt such term!) not only serve as input for determining which scenarios are to be alerted, but can also make it easier to dynamically set an alert severity/priority/urgency, which is by far the very first *indicator* analysts look at.

Can you imagine how easier for an analyst to determine how rare or 'interesting' a new agent is from such an alert? That's a complete different picture:

- One can easily determine if the agent is *super-rare* in case it's the only member of a cluster;
- One can easily determine if a new agent is likely related to a browser upgrade or in case the attacker has misspelled a character in it when the cluster accommodates a widespread browser (ex.: Chrome);
- Another hypothetical example: assuming most Python based agents strings starts with capital *P*, an analyst can better asses if that needs further investigation (escalation) or not.

## Agents profile search

Below is good start search for profiling the HTTP User Agent string, with that one running daily (early in the morning), it should deliver sufficient results while not hitting hard on performance:

```
| tstats summariesonly=1 dc(Web.src) AS daily_src_count,
  FROM datamodel=Web WHERE nodename=Web.Proxy
  BY Web.http_user_agent, _time span=1d

| rename Web.* AS *

| stats dc(_time) AS days_seen, median(daily_src_count) AS med_daily_src_count, min(_time) AS first_day, max(_time) AS last_day
  BY http_user_agent

| eval _comment="Here you can set a true/false flag for easier filtering/lookup - adjust to your environment"
| eval prevalent=if(days_seen>20 AND (med_daily_src_count>50), 1, 0)

| outputlookup override_if_empty=0 http_user_agent_profile.csv
```

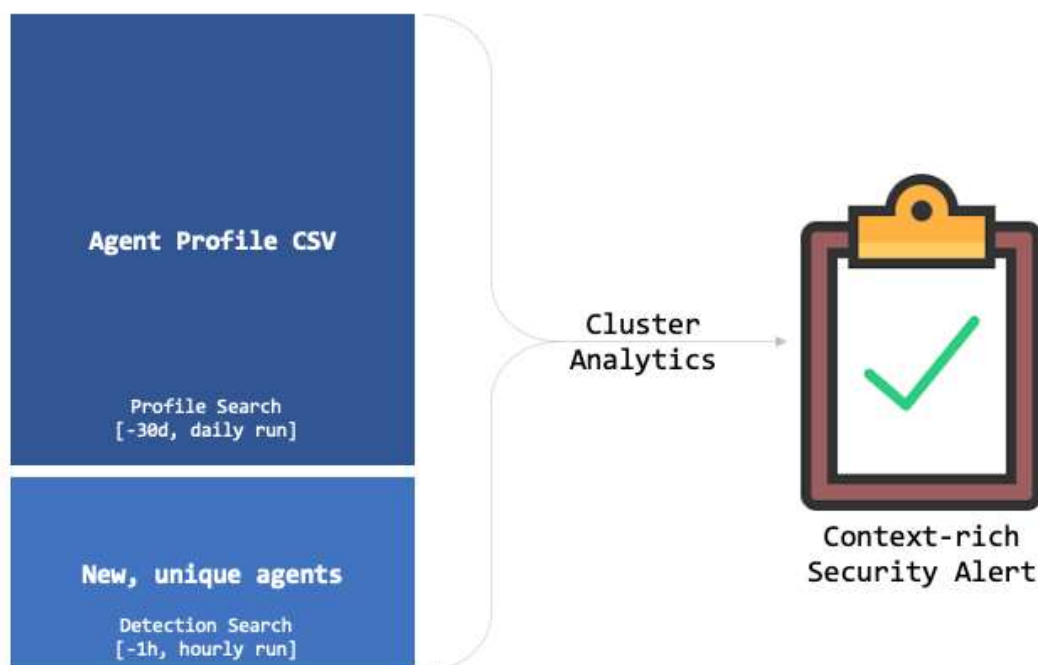
The following data points can be leveraged from that one:

- The number of days an agent was observed in the period (suggesting 30-45 days);
- Median daily number of clients (assuming *src*, but could be *user* field);
- First and last day seen and in case it's prevalent on not.

Of course, additional stats can be added later, just evaluate if the value is worth the performance cost. For instance: most accessed domain, latest URLs, etc.

## The actual detection search

To quickly recap, the idea is the following:



So the detection search will basically determine if an agent is unique, load the profile data (lookup) and perform the cluster analytics. The process will hopefully generate enough data points (alert qualifiers) to better determine if a scenario is worth to be alerted.



Here's a **prototype** ready to be used in an accelerated DM enabled environment, which applies to most *healthy* Splunk Enterprise Security (ES) deployments. Please note that rudimentary support for exception handling is implemented here, in case you want to deploy that in production.

```
| tstats summariesonly=1 count,
  values(Web.url) AS url,
  values(Web.src) AS src,
  FROM datamodel=Web
  WHERE nodename=Web.Proxy AND NOT [ | inputlookup exceptions.csv | rename * AS Web.* ]
  BY Web.http_user_agent

| rename Web.* AS *

| inputlookup append=1 http_user_agent_profile.csv WHERE NOT [ | inputlookup exceptions.csv ]

| eval _comment="Adjust t value to your environment"
| cluster t=0.4 labelonly=1 showcount=0 field=http_user_agent

| eventstats
  mode(http_user_agent) AS cluster_agent,
  dc(http_user_agent) AS cluster_members,
  avg(med_daily_src_count) AS cluster_clients,
  avg(days_seen) AS cluster_dseen
  BY cluster_label

| eval _comment="From here on, we only proceed for newly observed agents (no profile matches, hence days_seen = null)"
| lookup http_user_agent_profile.csv http_user_agent OUTPUT days_seen
| where isnull(days_seen)

| eval cluster_info=mvappend("Members count = ".cluster_members,
  "Prevalent agent = ".cluster_agent,
  "Daily clients (avg)= ".cluster_clients,
  "Days seen (avg) = ".cluster_dseen)

| eval _comment="Here's where you determine the alert qualifiers, list goes on and on..."
| eval qualifiers=if(cluster_members<10, mvappend(qualifiers, "- Likely NOT related to a usual, user/interactive browser"), qualifiers)
| eval qualifiers=if(match(http_user_agent, "(?i)(powershell|curl|nikto)"), mvappend(qualifiers, "- Automated or suspicious activity"), qualifiers)

| eval _comment="No qualifiers, no alert needed!"
| where mvcount(qualifiers)>0

| stats sum(count) AS result, values(url) AS url, values(src) AS src, values(cluster_info) AS cluster_info, values(qualifiers) AS qualifiers
  BY http_user_agent

| eval reason=mvappend("Alert qualifiers:", qualifiers)

| eval _comment="Provide more context to analyst combining info from the profile lookup and cluster analytics"
| eval note=mvappend("The new agent is part of a cluster with the following characteristics:", cluster_info)
| fields - cluster_info, qualifiers
```

The lookup file *exceptions.csv* is storing the agents or any other combination (AND operator) of fields. For instance, if the lookup contains *http\_user\_agent* and *url* fields, results matching those values are filtered out from the analytics or detection output.

## Some Alert output examples

To make sure we get “fresh” data, we cheat (again) by injecting some events at the index as follows:

```
| makeresults
| eval url="http://common.microsoft.com"
| eval http_user_agent="Microsoft Office/16.0 (Windows NT 10.0; Microsoft Outlook 17.0.10228; Pro)"
| eval src="host-2345"
| collect index=botsv3
```



Another one, this time a bit more suspicious:



```
| makeresults
| eval url="http://evilcorp.com"
| eval http_user_agent="Evil PowerShell v1.0"
| eval src="host-1234"
| collect index=botsv3
```

Since our detection defines as a threshold to alert on instances matching ANY alert qualifier, here's a sample output matching the events we just added to the index (after they are picked up by the DMA):

http_user_agent	result	url	src	note	reason
Evil PowerShell v1.0	3	http://evilcorp.com	host-1234	The new agent is part of a cluster with the following characteristics: Members count = 1 Prevalent agent = Evil PowerShell v1.0	Alert qualifiers: - Automated or suspicious activity - Likely NOT related to a usual, user/interactive browser
Microsoft Office/16.0 (Windows NT 10.0; Microsoft Excel 17.0.10228; Pro)	1	http://common.microsoft.com	host-2345	The new agent is part of a cluster with the following characteristics: Daily clients (avg)= 1 Days seen (avg) = 1 Members count = 5 Prevalent agent = Microsoft Office/16.0 (Windows NT 10.0; Microsoft Excel 16.0.10228; Pro)	Alert qualifiers: - Likely NOT related to a usual, user/interactive browser

Note that the all those fields are rendered in Splunk ES Incident Review page and that the *note* field shows the extra context gathered after the cluster/profile analysis.

The second result exemplifies how a simple “Excel Upgrade” scenario may look like, which is a potential benign instance. Check the cluster attributes (most prevalent cluster member) and spot the difference when comparing to the actual agent string alerted on.

Now, if the threshold is set to alert on scenarios only matching more than one alert qualifier, we would of course only get the first result (Evil PowerShell) as seen from *reason* field contents. This threshold would basically translate to the following SPL:



```
| where mvcount(qualifiers)>1
```

## Final thoughts



Again, those are all **prototype** (PoC) queries! Pretty much every customer will need a slight different approach, rendering sometimes significantly different searches, and here lies the beauty or challenge of content engineering.

A few things to consider before deploying to customers:

- First of all, evaluate the best options to **profile** agents from your environments. Should you consider all data sources feeding the Web DM? Should you consider 45 days instead of only 30?
- Make sure you define a good set of **alert qualifiers**, those are basically known as “features” in the ML/Stats domain. More alert qualifiers = more relevant alerts. For instance, besides the one

already mentioned: Microsoft Office related agents NOT accessing common MS domains might be one.

- Instead of clustering on all agents, you may want to cluster only on a **subset** of them (suspicious, matching extra constraints/qualifiers). This is an option for massive deployments.
- In case you haven't noticed, the cluster analytics section runs regardless of new agents observed or not, that's because **Splunk's engine does not provide ways to halt or break command execution** based on a condition like a procedural programming language does. However, there are some tricks to avoid that I am not including here for simplicity.
- A clear, **easy to triage** alert must include other fields in its output: drilldown to raw events, drilldown to triage dashboard, playbook/wiki url, exceptions lookup link, dynamic notable urgency based on alert qualifiers and much, much more (stay tuned).

## Where are my SPL Nuggets?

Here are the main queries showcased here:

### Profile Search

```
| tstats summariesonly=1 dc(Web.src) AS daily_src_count,
FROM datamodel=Web WHERE nodename=Web.Proxy
BY Web.http_user_agent, _time span=1d

| rename Web.* AS *

| stats dc(_time) AS days_seen, median(daily_src_count) AS
med_daily_src_count, min(_time) AS first_day, max(_time) AS last_day
BY http_user_agent

| eval _comment="Quick flag for easier lookup - adjust to your environment"
| eval prevalent=if(days_seen>20 AND (med_daily_src_count>50), 1, 0)

| outputlookup override_if_empty=0 http_user_agent_profile.csv
```



### Detection / Cluster Analytics Search



```

| tstats summariesonly=1 count,
values(Web.url) AS url,
values(Web.src) AS src,
FROM datamodel=Web
WHERE nodename=Web.Proxy AND NOT [ | inputlookup exceptions.csv | rename *
AS Web.* ]
BY Web.http_user_agent

| rename Web.* AS *

| inputlookup append=1 http_user_agent_profile.csv WHERE NOT [ | inputlooku
exceptions.csv ]

| eval _comment="Adjust t value to your environment"
| cluster t=0.4 labelonly=1 showcount=0 field=http_user_agent
| eventstats
mode(http_user_agent) AS cluster_agent,
dc(http_user_agent) AS cluster_members,
avg(med_daily_src_count) AS cluster_clients,
avg(days_seen) AS cluster_dseen
BY cluster_label

| eval _comment="From here on, we only proceed for newly observed agents (r
profile matches, hence days_seen = null)"
| lookup http_user_agent_profile.csv http_user_agent OUTPUT days_seen
| where isnull(days_seen)

| eval cluster_info=mvappend("Members count = ".cluster_members,
"Prevalent agent = ".cluster_agent,
"Daily clients (avg)= ".cluster_clients,
"Days seen (avg) = ".cluster_dseen)
| eval _comment="Here's where you determine the alert qualifiers, list goes
on and on..."
| eval qualifiers=if(cluster_members<10, mvappend(qualifiers, "- Likely NOT
related to a usual, user/interactive browser"), qualifiers)
| eval qualifiers=if(match(http_user_agent, "(?i)(powershell|curl|nikto)"),
mvappend(qualifiers, "- Automated or suspicious activity"), qualifiers)

| eval _comment="No qualifiers, no alert needed!"
| where mvcount(qualifiers)>0

| stats sum(count) AS result, values(url) AS url, values(src) AS src,
values(cluster_info) AS cluster_info, values(qualifiers) AS qualifiers
BY http_user_agent

| eval reason=mvappend("Alert qualifiers:", qualifiers)
| eval _comment="Provide more context to analyst combining info from the
profile lookup and cluster analytics"
| eval note=mvappend("The new agent is part of a cluster with the following
characteristics:", cluster_info)

```



| fields - cluster\_info, qualifiers

Website Built with WordPress.com.

