SIEM Content Engineering

BEST PRACTICES

Writing Splunk Enterprise Security Correlation Searches

A few practical tips and ideas for Splunk Detection Engineers who are willing to standardize and improve detection code (SPL) based on hands-on, field experience.

Version 1.3 - Sep 2023

Why this?

This is an attempt to contribute back to the Splunk community with some simple, yet powerful ideas to help users and customers write more effective SPL detection code.

Splunk Enterprise Security (ES) App includes multiple Correlation Searches (rules, detections), but one hardly deploys them without some fine-tuning (customization).

This is particularly helpful for those continuously designing and delivering new detection content while also providing some directions for those starting in the field.

Happy Splunking!

Alex Teixeira

opstune

https://opstune.com/about

Aggregate, Summarize – always!

Motivation

This is by far the most intriguing behavior I spot in multiple projects/customers: there's simply **no event aggregation or summarization at all** before firing a notable towards analysts/SOC. This underutilizes Splunk analytics capabilities.

To make it clear, think about the following hypothetic rule, which detects when a field *severity_id* is greater than 2:

```
index=foo sourcetype=bar dest=* signature=* severity_id>2
| search NOT signature IN ("*EICAR*")
| table _time dest signature severity_id
```

That's a very valid rule prototype. The thing is, by default, Splunk/ES creates a notable for every single line (record) in the output. That means if within 1 minute there are 100 events matching that query's constraints, there will be at least 100 new alerts to be triggered! Imagine this approach for every single custom detection.

"But I want to know all of them! How can I go about it?" Answer: aggregate on an entity and describe what happens within the alert (pick as many IR fields you can to make it). Nearly every alert holds a subject, or a set of subjects in it, and they are either related to the target or the source of the "attack". For example:

- Source/Target Host (hostname, IP address, etc)
- Source/Target User Account or Email
- Source/Target Signature

How does it look like in practice?

The less entities you aggregate on (group by), the less alerts you tend to get in the end. Following up on above prototype:

```
index=foo sourcetype=bar dest=* signature=* severity_id>2
| search NOT signature IN ("*EICAR*")
| eval note=signature." (sev: ".severity_id.")"
| stats count AS result, values(signature) AS signature, values(note) AS note,
  earliest(_time) AS start_time, latest(_time) AS end_time BY dest
| eval note=mvappend("Signatures plus severity:", note)
| eval dd="index=foo sourcetype=bar dest=".dest
```

Aggregation happens here. All fields (green) are CIM compliant.

Depending on how much time is observed (ex.: -1h), regardless of how a target host (*dest*) is hit, there will be no more than 1 alert per host every time that query runs, while the analyst is still able to get a better, bigger picture; with all details available from the drilldown search, in case she/he needs it.

There are so many other tricks to share, but except for very few corner cases, always use ***stats/chart/dedup*** in your rules!

Add extra context via IR accepted fields

Motivation

Unless you perform alert triage in a distinct interface, the Incident Review (IR) dashboard is the analyst's first encounter with notable events in Splunk ES.

The key here is knowing that **only fields listed in the IR settings are properly rendered by ES**, therefore you should either use an existing field or add custom ones.

By following this advice, **alert normalization** is easily achieved and that's the first step towards integrating notables with another platform (SOAR, Case Management, Threat Intel and other systems).

This way, any system consuming notable events (security alerts) will know which fields/schema to expect.

How / Use Case

Go to the ES main menu under **Configure > Incident Management > Incident Review Settings**. The last panel displayed in that page is called "Incident Review - Event Attributes" (displayed below).

Provide as much information (context) as possible. Source and destination assets have their own subset. Others include *note*, *desc*, *signature*, *file_hash*, etc.

Pro Tip

Easily digestible alerts!

There's also a field called ***reason*** in which I usually provide a succinct message on why the the notable was fired. An example follows (SPL):

```
| eval reason="Max DNS (A) request rate from
this src exceeds max baseline by ".perc."%.
Observed rate: ".dns_a_rate." requests/sec."
```

Incident Review - Event Attributes (?)

dest

Filtering on destination related fields, same applies to source (*src*)

Field	Label	
dest	Destination	Edit | Remove
dest_threatlist_category	Destination Threat List Category	Edit | Remove
dest_threatlist_description	Destination Threat List Description	Edit | Remove
dest_threatlist_name	Destination Threat List Name	Edit | Remove
dest_bunit	Destination Business Unit	Edit | Remove
dest_category	Destination Category	Edit | Remove
dest_city	Destination City	Edit | Remove
dest_country	Destination Country	Edit | Remove
dest_dns	Destination DNS	Edit | Remove
dest_ip	Destination IP Address	Edit | Remove
dest_is_expected	Destination Expected	Edit | Remove
dest_lat	Destination Latitude	Edit | Remove
dest_long	Destination Longitude	Edit | Remove
dest_mac	Destination MAC Address	Edit | Remove
dest_nt_domain	Destination NT Domain	Edit | Remove
dest_nt_host	Destination NT Hostname	Edit | Remove
dest_owner	Destination Owner	Edit | Remove
dest_pci_domain	Destination PCI Domain	Edit | Remove
dest_port	Destination Port	Edit | Remove

Dynamic Drilldown search string

Motivation

After an alert is triggered, one of the first tasks a SOC analyst performs is to check the "contributing events" or the raw events linked to the given notable event.

The interface provides a small, simple input text area to define such "drilldown search string". That's perhapswhy it's often overlooked. My take on that:

"The triage and notable assessment tasks are an extension of the detection engineer's work."

Without being able to properly asses a new alert, chances are another alert will end up in the FP pile because the analyst had no means (or motivation) to actually investigate it deeper.

Every single alert must come with a easy, dynamic link (URL) that lands the user in a search query showing the contributing events. And results should quickly show up!

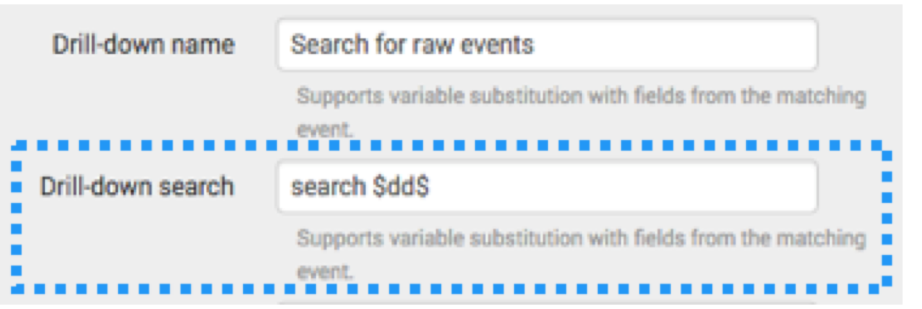
How / Use Case

This is a method I have been using for a long time and it's been explained in great detail in this blog post.

The idea is to set a field to hold the value for a whole, complete search, based on the output of a notable event, this way narrowing the scope to the exact raw events leading to the alert triggering.

After setting the field, you simply use it as a token in the correlation search drilldown input. This way, all rules might have the exact same value set in the interface: **dd**.

Again, the actual rule code holds all the data, not need to check multiple fields in the correlation search editor. Below is how it looks like in the **notable** section:




```
index=main sourcetype=mcafee:epo (severity=critical OR severity=high)
| stats values(event_description) AS desc, values(signature) AS signature, values(file_name) AS file_path, count AS result BY dest
| eval dd="index=main sourcetype=mcafee:epo (severity=critical OR severity=high) dest=".dest
```

All time

✓ 2,432 events (before 10/11/16 9:30:24.000 PM) No Event Sampling

Job Smart Mode

Events Patterns Statistics (15) Visualization

20 Per Page Format Preview

dest	desc	signature	file_path	result	dd
HOST-001	Infected file deleted. Scan Timed Out Scan found infected files. Unwanted program deleted. file infected. No cleaner available, file deleted successfully	EICAR test file Generic.dx!tmf Tool-HideWindow none	C:\Users\temp\file1.exe C:\Users\temp\file10.exe C:\Users\temp\file11.exe C:\Users\temp\file12.exe C:\Users\temp\file13.exe C:\Users\temp\file2.exe C:\Users\temp\file3.exe C:\Users\temp\file4.exe	152	index=main sourcetype=mcafee:epo (severity=critical OR severity=high) dest=HOST-001
HOST-002	Infected file deleted. Scan Timed Out Scan found infected files. Unwanted program deleted. file infected. No cleaner available, file deleted successfully	EICAR test file Generic.dx!tmf Tool-HideWindow none			) dest=HOST-002

Turn mv-fields into single-value ones

Assuming you want to prep the drilldown based on signature (sig, for short), you could use two constructions:

```
| eval dd="(sig=\"".mvjoin(sig, "\" OR sig=\")."\")
| eval dd=" sig IN (\"".mvjoin(sig, "\", \"")."\")"
```

Throttling on Notable Uniqueness

Motivation

There are many ways to safely reduce the alert volume, one of the ways is by leveraging the throttling mechanism available on Splunk/ES.

The challenge is finding the right balance so that analysts are not overwhelmed with alerts while no potential true-positive alerts are filtered out.

One path to achieve that is by controlling when an instance of an alert, that is, a new unique scenario for a certain detection is raised as a new notable.

What makes an alert unique? That can be a combination of different field values or even an unique ID already presented within the raw event.

How / Use Case

Let's assume the hypothetical detection prototype from screenshot below (Critical/High McAfee signatures) as our target to implement such throttling mechanism.

There a few options available:

1. Alert on every new infection event, regardless if it's a recurring infection or not (higher volume);
2. Aggregate for, let's say, one hour and group events by target (moderate volume);

What if we don't want to be alerted twice on the same scenario within a day? You *hash* the signature set and throttle based on that calculated value.

```
index=main sourcetype=mcafee:epo (severity=critical OR severity=high)
| stats values(event_description) AS desc, values(signature) AS signature, values(file_name) AS file_path, count AS result BY dest
| eval dd="index=main sourcetype=mcafee:epo (severity=critical OR severity=high) dest=".dest
```

All time

✓ 2,432 events (before 10/11/16 9:30:24.000 PM) No Event Sampling

Job Smart Mode

Events Patterns Statistics (15) Visualization

20 Per Page Format Preview

dest	desc	signature	file_path	result	dd
HOST-001	Infected file deleted. Scan Timed Out Scan found infected files. Unwanted program deleted. file infected. No cleaner available, file deleted successfully	EICAR test file Generic.dx!tmf Tool-HideWindow none	C:\Users\temp\file1.exe C:\Users\temp\file10.exe C:\Users\temp\file11.exe C:\Users\temp\file12.exe C:\Users\temp\file13.exe C:\Users\temp\file2.exe C:\Users\temp\file3.exe C:\Users\temp\file4.exe C:\Users\temp\file5.exe C:\Users\temp\file6.exe C:\Users\temp\file7.exe C:\Users\temp\file8.exe C:\Users\temp\file9.exe	152	index=main sourcetype=mcafee:epo (severity=critical OR severity=high) dest=HOST-001
HOST-002	Infected file deleted. Scan Timed Out Scan found infected files. Unwanted program deleted. file infected. No cleaner available, file deleted successfully	EICAR test file Generic.dx!tmf Tool-HideWindow none	C:\Users\temp\file1.exe C:\Users\temp\file10.exe C:\Users\temp\file11.exe C:\Users\temp\file12.exe C:\Users\temp\file13.exe C:\Users\temp\file2.exe C:\Users\temp\file3.exe C:\Users\temp\file4.exe	157	index=main sourcetype=mcafee:epo (severity=critical OR severity=high) dest=HOST-002

```
| eval _comment="Add to above query to enable better throttling control"
| eval notable_instance=dest.mvjoin(signature, "").mvjoin(file_path, "")
| eval notable_instance=md5(notable_instance)
```

Throttling

Window duration: 24 hour(s)

How much time to ignore other events that match the field values specified in Fields to group by.

Fields to group by: notable_instance ×

Type the fields to consider for matching events for throttling. Learn more

A few filtering, eval approaches

Motivation

Filtering events is pretty much part of every search, there are always constraints to apply (distinct scopes, unwanted events, etc). I'm sharing below a bit of my thought process when building the queries and a few obvious use cases for detection engineers and threat hunters.

Search or Where?

I tend to use *search* only when wildcard and/or case-insensitive filtering is enough:

```
| search signature IN ("*malware*", "botnet*")
| search signature="trojan *" severity_id>2
```

"Did you know?"
There's an implied *search* command at the beginning of any query not starting with "|"?

For pretty much all the rest, I go with *where* clause, which also covers pretty much all *search* command use cases:

```
| where match(signature, "(?i)(Botnet|Malware)")
| where match(signature, "^Trojan\s") AND severity_id>2
```

When filtering events out, always consider NOT instead of "!=" operator, unless you really know what you are doing or for performance reasons. Let's assume a dataset has 10 events, 5 have *signature* field set, 5 don't. The goal is to filter in all the events ***except*** when *signature*="EICAR". Which query would you pick? Answer: the 2nd query also catches NULL values.

```
index=foo sourcetype=bar signature!="EICAR"
index=foo sourcetype=bar NOT (signature="EICAR")
```

Regular Expressions

One page you should definitely have in your bookmarks is the common eval's functions. That's where you find lots of filtering and field/value/string manipulation ideas. I'm sharing a few examples using eval's *replace* function below:

```
| makeresults
| eval f1="This is a Short phrase"
| eval f2=replace(f1, "Short", "Long")
| eval f3=replace(f1, "(?i)^This is a (short|long) phrase$", "\1")
```

This one comes in very handy when manipulating command line and other user-generated data. Sharing an example:

```
| makeresults
| eval cmd="powershell -noP -nonI -Win hidden -c sc ftp.txt -val \"open\" -enc ascii"
| eval parameters=cmd
| makemv tokenizer="(-\S+)" parameters
```

Pro Tip

cmd	parameters
powershell -noP -nonI -Win hidden -c sc ftp.txt -val "open" -enc ascii	-noP -nonI -Win -c -val -enc

A few filtering, eval approaches (continued)

Permanent Exception Handling (filter-out)

ES suppressions allow users to filter events out by setting an expiry date and simple key-value pairs as part of the logic. There are always pros/cons on that, but many users default to lookup files as an alternative to deploy permanent filtering. Key here is mastering *format*, *return*, *inputlookup* and *lookup* commands usage.

terminal_servers.csv

ts_server	ts_user
viper	admin*
angra	andre
shaman	matos

Below are a 2 examples with same outcome (filtering out matching *dest* entries):

```
index=windows NOT ([
  | inputlookup terminal_servers.csv
  | rename ts_server AS dest | table dest ])
```

```
index=windows
| search NOT [ | inputlookup terminal_servers.csv | rename ts_server AS dest ] | format ]
```

One could also avoid the *subsearches* limits (more on that further) by using the *lookup* command instead:

```
| lookup terminal_servers.csv ts_server AS dest OUTPUT ts_server AS ts_host
| where isnull(ts_host)
```

There are many aspects to discuss here but let's say we also want to filter on *user* besides the *dest* field. Below is how the *format* command's output would look like when adding the *user* field to the mix:

```
| inputlookup terminal_servers.csv
| rename ts_server AS dest, ts_user AS user | format
```

search

```
( ( dest="viper" AND user="admin*" ) OR ( dest="angra" AND user="andre" ) OR ( dest="shaman" AND user="matos" ) )
```

Have you noticed the *search* field in the output with the perfect (expected) logic? That's piped into the search string when used within a *subsearch*. The *lookup* command would require a bit more effort (wildcard, case-insensitive lookup definition), so one needs to evaluate what's the best cost/benefit for each case.

Why not using macros instead?

That's another approach but **highly discouraged in case the users managing the exceptions are not fluent in SPL**. By managing exceptions via a lookup, it's harder to mess up with the code since a single misplaced comma is enough to ruin the entire logic. Remember: a syntax error may lead to a skipped query (no results until it's fixed).

Lookup File Editor x External interfaces

The **Lookup File Editor App** is a must-have. But I've seen implementation where the lookup is consumed from external sources and managed using distinct interfaces (GUI), I've also helped customers build their own Exception/Suppression policy editor app. Evaluate what fits better for you! A quick trick leveraging the Lookup App is shared on the next page.

Abuse Workflow Actions

Motivation

Imagine passing any notable event attribute as a parameter to another system or search – that's what this feature is meant for! Below a few use cases:

- **Playbook doc/wiki link:** based on rule name/id;
- **Alert Dashboard:** visualize the alert instead of only checking raw events (suitable for stats based alerts);
- **Exceptions handler:** set the link to the corresponding lookup entry enabling faster false-positive (FP) handling.

Properly implementing this will significantly speed up triage. It also allows **HTTP POSTs** (ex.: drilldown from IR to contributing events at the data source's web app).

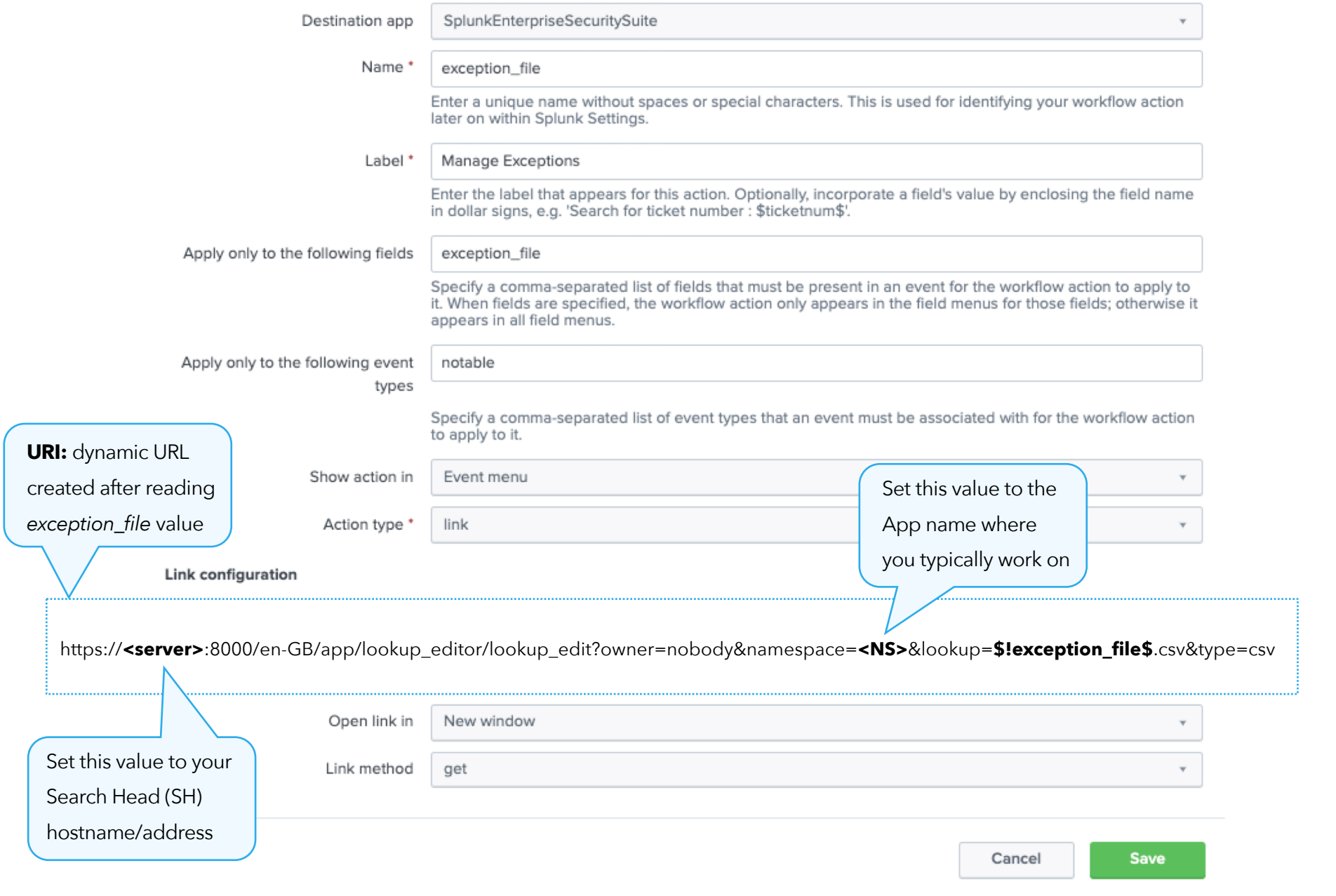
How / Use Case

Splunk/ES suppression mechanism is pretty limited (sadface). Therefore users come up with creative ways to address that (some develop their own Suppression Policy Apps, for instance). Here's a simple solution helping in this regards.

Assuming you have a rule that alerts on specific web traffic, and you want to filter out legit/FP cases based on ***url*** and ***user*** fields -- with wildcards supported. First, define a link within your rule's code:

```
| eval exception_file="rule_XYZ_exceptions"
```

Then, define a **Workflow Action** that reads the value from the notable when it's visualized at IR page. Follow the menu **Settings > Fields > Workflow actions** (details below).



Abuse Workflow Actions (continuation)

Create a lookup

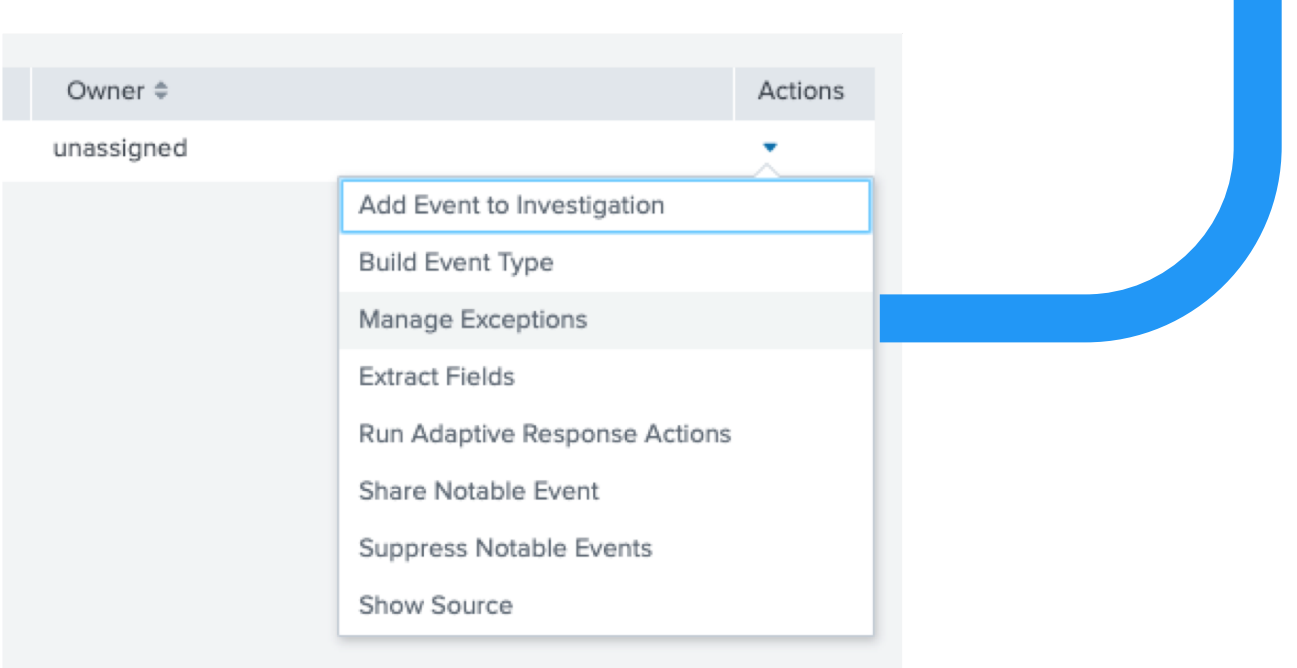
Assuming you have already installed the "Lookup File Editor" App https://splunkbase.splunk.com/app/1724/, you now need to create a lookup file (CSV) with the following columns:

- **url**: holding the value (accepting wildcards) that will be filtered out whenever the observed URL matches;
- **user**: same as above but applicable to the user value;
- **comment**: here's where the analyst document the change (example below).

	url	user	comment
1	http://updates.mcafee.com/*	svc_mcafee_usr	Charlie on 20.10.2020: McAfee AV updates

Test it out!

Now, fire a notable for a rule which instantiates ***exception_file*** field as described earlier and check if the following link is shown from the notable alert **Actions** context menu (right next to **Owner**, by default). Once that link is clicked, it should open the file defined as value for ***exception_file*** field.



Also, consider that the "Lookup File Editor" App **does provide backups** or a sort of simple versioning mechanism which comes in very handy, especially for multi-user SecOps teams.

Note: for heavy, long lookups, there's no better option than creating a lookup definition and using the *lookup* command.

More details on how to manage exceptions is available under "**Permanent Exception Handling (filter-out)**" section.

Avoid late events going under the radar

Motivation

Most TA developers extract ***_time*** from the log generation time. And that's the best practice since we all want to track the time set by the device or system generating the log.

However, when a rule is executed the time reference is the SH's clock, that means, the search head or the Splunk instance where the Enterprise Security (ES) App is installed.

A few risks introduced here, in a threat detection context:

1. If clocks are not using same TZ, or if that info is not accounted for when onboarding and scanning events, some time windows might go unchecked;
2. Same can happen when there's an issue with log transport/delivery, which might delay event index time;
3. There are also more *paranoid* scenarios: attacker modifies or tampers with target host clock or an infected machine is only plugged to the corporate network hours after an infection was registered.

In any case, your detection will fail in if it's not possible to reach for those events.

How / Use Case

How to deal with that? Besides designing monitoring use cases (detections) for these scenarios, a good practice is to consider using ***_indextime*** instead of regular ***_time*** to scope in your detection rules.

For more on the topic, please check the following blog posts I wrote some time ago where details are explained:

- It's about time to change your rule timing settings
- SIEM tricks: dealing with delayed events in Splunk

How does it look like in practice?

Below you can find a sample correlation search that leverages this approach. It also provides a dynamic drill down search query based exactly on the time boundaries used during the rule's execution time.

Consider you are stacking multiple critical NIDS signatures per target host every 5 minutes (interval) to raise an alert (notable event).

Time Settings

Earliest: -5h@h
Latest: +5h@h
Cron schedule (interval): */5 * * * *

```
index=foo sourcetype=bar severity=1 _index_earliest=-5min@min
| stats min(_indextime) AS imin,
  max(_indextime) AS imax,
  values(signature) AS signature
  BY host
| eval dd="index=foo sourcetype=bar severity=1 host=".host
| eval dd=dd." _indextime>=".imin." _indextime<=".imax
```

That would consider any matched event indexed within the last 5 minutes, allowing the event *_time* to be 5 hours off or "skewed" (positive/negative), as compared to the rule engine's clock (Splunk/ES search head).

Also, note the time boundaries are set by using ***_indextime*** instead of search modifiers ***_index_earliest*** and ***_index_latest***. Reason for that is because the latter is not inclusive, meaning events having the latest time within the boundaries will not match.

Set Notable attributes on the fly

SPL-centric approach

Besides the dynamic drilldown search string already shown, it's also possible to dynamically assign values to other notable attributes that can be later used as tokens or that are simply rendered in the Incident Review page by default.

Perhaps that's just a matter of preference, but this approach makes it easier to focus on the SPL search code only, not on multiple inputs scattered throughout the notable section in the Correlation Search editor.

Setting a Notable Title

When using the following construction within your rules, you can fill in all values for notable title using the same token:

```
| eval title="There were ".result." total suspicious AV events detected at ".src." host"
```

Dynamic Urgency

Sometimes it's just easier to control the *urgency* value from within your rule's logic. Usually, that's the first attribute an analyst look for in order to determine the next notable candidate for evaluation. This can also be integrated into another, custom alert framework (distinct threat scoring/rating algorithms, RBA approaches, etc).

```
| eval urgency=case(
  mvcount(signature)>1 AND severity_id>90, "critical",
  match(signature, "(?i)(trojan|exploit)"), "high",
  1=1, "medium")
```

Systematically referencing the Rule Name

This can be extended in many different ways, for instance, you can also manipulate the rule name to dynamically create external links (playbook, handling guidelines, wikis) based on other tokens (fields):

Pro Tip

```
| eval myrulename=replace("$name$", "^\S+ - (.+) - Rule$", "\1")
| eval myrulename=replace(myrulename, "\s", "_")
| eval playbook_link="https://wiki.local/playbook/".myrulename
```

That could be a good start on establishing certain rule code standards as part of a QA process. Also, don't forget to set a Workflow Action for allowing the link to be clicked from *Actions* notable context menu (as explained earlier).

Tstats & Data Models (DMs)

Should I master data models to write good SPL?

A clear answer is **no**. Simply count the number of *sourcetypes* in your environment not feeding data models (DMs) and how much content built around them. Imagine if you would need to build a custom DM for each of them.

However, ES heavily relies on DMs so there's no way out, especially if you aim towards standardizing the code and want to support multiple clients or environments where admins can manage to keep up with healthy accelerated DMs.

I'm not going to extend on this topic here since there's a complete blog post on that below where you will find lots of insights:

-> Should I date a model? Myths busted!

| tstats summariesonly=1

One takeaway here is this: that command prefix is the **only way** to access accelerated DM data. But what about *from*, *datamodel* *search* and *pivot* commands? Nope! They leave accelerated data intact when used. They will query on regular buckets.

Before creating a custom DM, check this out!

This time I'm going to put those statements pretty clear so that no one misses out:

- There's no need to build a (custom) DM to achieve event normalization (CIM compliance);
- There's no need to build a (custom) DM to achieve search speed.

Event normalization happens when you model extractions, field naming/aliasing, all that should happen outside DM configuration, at TA or search-time level. Same goes for speed, there are many other methods to achieve speedy results besides accelerated DMs. It all depends on the use case. Actually, **it all begins with use case design**.

Consider *selective* index-time field extraction

That's a feature many admins don't even know about, but super easy and handy in many, many small, medium and even some large environments. This is NOT suitable for all cases, but I highly encourage you to evaluate before considering anything else.

By using that you can set any field to be part of *tsidx* file! When you tell Splunk to extract a field at index-time, you can basically manipulate it afterwards via *tstats*, just like hosts, *sourcetype* and other internal fields. Think about that field you "group by" very often or that would significantly increase your critical searches if you could retrieve them via *tstats*.

If you feel like given it a try, please refer to this procedure here. After that, you will be able to manipulate those fields via ***tstats***.

Watch out for these traps!

More Freedom > More Responsibility

SPL is quite simple to learn, no advanced programming skills needed. However this comes at the price of a very "tolerant", less strict approach when it comes to variable types and definitions, for instance.

I'm listing a few common mistakes (and workarounds) I see customers and users doing, it's very easy to overlook those details but it might have a big impact when it comes to baselining (statistics, modeling) and reliable detection.

Multi-value (MV) fields

Any event containing a NULL value for ***dest*** field below will not be part of your results from example below:

```
<base search with events sometimes containing dest (hostname), and always containing dest_ip>
| stats count by dest
```

Use ***coalesce()*** or simply ***eval's if/else*** to make sure you always get a non-null value before aggregating.

```
| eval dest=coalesce(dest, dest_ip)
| eval dest=if(match(dest, "\S"), dest, "Unknown")
| stats count by dest
```

Now, what happens if the *group by* is done on a MV field? **There will be a new row (result) for each item of the MV field!** So carefully check if any group by field is actually a single-value field before performing any analytics on it.

Here's another tricky one. You can compare single value fields with MV ones, for instance:

```
| makeresults
| eval single="apple", multi=mvappend("banana", "apple", "watermelon")
| eval hasApple=if(single=multi, "Yes", "No")
```

The field *hasApple* is assigned the value "Yes", which might be an unexpected result depending on the use case. Also, what out for string concatenation as it doesn't work with multi-value fields!

```
| eval thisComesBackEmpty="I like ".multi.", but ".single." is my favorite!"
| eval thisComesBackOK="I like ".mvjoin(multi, ", ").", but ".single." is my favorite!"
```

MV-field truncation

Pro Tip

I often write rules that aggregate on fields like src/dest IP or usernames that might become a problem to render at the Incident Review page when they hold too many entries, so here's a simple solution:

```
| eval dest=if(mvcount(dest)>2, mvappend("Truncating at 2 of ".mvcount(dest)." total:", mvindex(dest, 0, 1)), dest)
```

That's useful when baselining or evaluating how a rule behaves for long periods of time while avoiding hanging the browser! Analyst will always have access to a drilldown search where all entries are available as last resort.

Watch out for these traps! (continuation)

Another tricky one: check for null values

Assuming *src* equals *space* – which is a non-NULL value, the first command below is going to assign a space value to *src* field, which is probably not the intended outcome. The latter will check if there's a non "\s" character (tab, space) anywhere in the field.

```
| eval src=coalesce(src, src_ip)
| eval src=if(match(src, "\S"), src, src_ip)
```

Usually, new users tend to leverage *isnull()* or *isnotnull()* eval's functions, which are not helping here either.

⚠️ Risky options to reconsider ⚠️

By default, *subsearches* return **a maximum of 10,000 results and have a maximum runtime of 60 seconds**. So unless you really know what you are doing, avoid using the following commands in *critical* code:

- Subsearches in general. Some exceptions may apply (small lookups via *inputlookup*, rest queries, etc);
- Subsearch based commands: ***map***, ***join***, ***append****. There's always another way to do it, but of course, there are some corner cases or scenarios in which those limits are acceptable, so using those commands is not a big deal.

Transaction Command

The transaction command is another beast, so use with care. (It deserves at least its own paragraph!). Unfortunately, new users rely on *transaction* command believing it's the holy grail of "correlation" when in fact, 90% of cases can be solved with some *eval*-foo and *stats* command.

It's very tempting to write a transaction based command but hard to validate unless you master some of the parameters or in case you can validate the results in another way (manual checks, peer-review, etc).

Other suggestions

A few other tips for safer, more standard code (again, not a must, just a suggestion!):

- ~~Consider~~ Always wrap OR and NOT sentences with parenthesis ();
- Consider eval's match() instead of like() for better control and more flexibility;
- Consider eval's trim() + lower() or upper() before grouping or manipulating "string" fields (ex.: Account Name);
- Consider strftime() or strptime() to manipulate datetime instead of *convert* or other commands;

Feedback and Discussions is another way to learn (and change perspectives)

I'm constantly sharing other quick tips via Blog or Twitter, so check those out and feel free to drop me an email or have a chat via Splunk's Slack channels. My contacts are listed here. Hope you enjoyed!