Boolean Operators

Boolean Operators are used to connect and define the relationship between your EQL queries. These Boolean Operators can be utilized to either narrow or broaden your queries.

and	Utilized to combine two queries.	`process where process_name == “powershell.exe” and command_line == “*download*”`
or	Used to broaden your query criteria.	`process where process_name == “powershell.exe” or process_name == “wscript.exe”`
not	Used to exclude criteria from your query.	`process where process_name == “powershell.exe” and not user_name == “jsmith”`

Comparative Operators

Comparative Operators are utilized to compare fields/value pairs to narrow or broaden your queries.

<	Less Than	`network where total_in_bytes < 4000`
<=	Less than or equal to	`network where total_in_bytes <= 4000`
==	Equal to	`process where process_name == “nmap.exe”`
!=	Not equal to	`process where process_name != “nmap.exe”`
>	Greater than	`network where total_out_bytes > 4000`
>=	Greater than or equal to	`network where total_out_bytes >= 400`

Process Lineage

Process Lineage generates an Endgame Resolver view to show a timeline of related parent/child process activity.

Process lineage on wscript.exe with a PID of 452 on 192.168.1.239	`process lineage on wscript.exe PID 452 on 192.168.1.239`

- Process lineage must specify an endpoint or IP address to investigate.
- It is recommended to always specify the PID of the target process.
- The process lineage tree will only show process creation events

Lookups

Lookups allow for queries to be built that search for values specified in a list format. These lists can be dynamic (specify other fields) or a static list.

Query for a list of users (static).	`user_name in ("Administrator", "SYSTEM", "NETWORK SERVICE")`
Query for a list of processes (dynamic).	`process_name in ("cmd.exe", parent_process_name)`

Event Relationships

Event Relationships can be utilized for stateful tracking within the respective query. If a related event exists that matches the criteria, it is then evaluated by the query. Relationships can be arbitrarily nested, allow for complex behavior and state to be tracked.

child of	Grandchild of WMI Provider Host.	`process where child of [process where parent_process_name == "wmiprvse.exe"]`
event of	Text file modifications by command line redirection.	`file where file_name == "*.txt" and event of [process where process_name == "cmd.exe" and command_line == "* > *"]`
descendent of	Network Activity for PowerShell processes not spawned from explorer.exe	`network where process_name == "powershell.exe" and not descendant of [process where process_name == "explorer.exe"]`

Sequences

Sequences are utilized to query for a specified pattern of events within a defined amount of time or until a certain event has occurred.

Search for all instances of bad.exe utilizing port 443 within 1 second.	`sequence with maxspan=1s`
	`[process where process_name == bad.exe]`
	`[network where destination_port == 443]`

Search for all instances of bad.exe utilizing until an inbound connection is received on port 443.	`sequence by unique_pid`
	`[process where process_name == bad.exe]`
	`until`
	`[network where destination_port == 443]`

- When specifying “maxspan”, seconds (s), minutes (m) and hours (h) can be utilized.
- Sequences can be constrained by matching fields, so that the specified events must share the same source host. This is accomplished by using “sequence by”.

Wildcard Function

The Wildcard Function allows for a list of wildcarded values to be searched against a specified field.

Search for encoded PowerShell commands.	`process where process_name == “powershell.exe” and wildcard(command_line, “*-ec*”, “*-en*”, “*-enc*”)`

Join

The join statement allows for multiple unordered event types to be linked. This functions similar to sequences, but lacks the time constraints.

Join by source_ip and destination_ip for RDP, RPC and SMB connection events.	`join by source_ip, destination_ip`
	`[network where destination_port == 3389]`
	`[network where destination_port == 135]`
	`[network where destination_port == 445]`

Join and query for process, network and file events until a process termination event.	`join`
	`[process where true]`
	`[network where true]`
	`[file where true]`
	`until [process where event_subtype_full == "termination_event"]`

- With join, events can happen in any order and when all events match, the join is complete.
- Like sequences, events can be joined until an expiration event is met.

Pipes

Pipes are utilized for the post-processing of events and can be leveraged for data enrichment, aggregations, statistics and filtering.

sort	Sort results by the specified field.	`file where true | sort file_name`
unique	Filter results to show events with unique occurrences of the specified field.	`process where true | unique process_name, command_line`
filter	Filter will output events that only match the specified criteria.	`network where true` `| filter timestamp_utc >= "2018-05-01"`

head	Display the first *N* number of query results.	`process where process_name == "powershell.exe"`
		`| unique command_line`
		`| head 50`

tail	Display the last *N* number of query results.	`security where event_id == 4624` `| tail 10`

rare	By using multiple pipes, we can find the least frequent values of a specified field.	`process where process_name == "powershell.exe"`
		`| unique_count parent_process_name, command_line`
		`| sort count`
		`| head 5`

Sort can be intensive if the number of results is not bound using head or tail. Multiple pipes can be combined to enrich the presentation of results (ex. rare).

Opcodes

Opcodes are a short hand operator that map directly to the event_subtye_full values. Below is a list of each opcode and the respective event_subtype_full value.

Process Event Subtypes	
1	creation_event
2	termination_event
3	already_running
4	still_running
5	exec_event
6	uid_change
7	session_id_change
9	fork_event
1016	lookup_failure
3008	request_event

Image Load Subtypes	
0	image_load_event
1	driver_load_event

DNS Event Subtypes	
1016	lookup_failure
3008	request_event

File Event Subtypes	
0	file_create_event
1	file_modify_event
2	file_delete_event
3	file_rename_event
4	file_overwrite_event
6	file_exchange_event

Network Event Subtypes	
12	`ipv4_connection_attempt_event`
13	`ipv4_connection_accept_event`
15	`ipv4_disconnect_received_event`
16	`ipv4_reconnect_attempt_event`
17	`ipv4_http_request_event`
55	`ipv6_connection_attempt_event`
56	`ipv6_disconnect_received_event`
57	`ipv6_connection_accept_event`
58	`ipv6_reconnect_attempt_event`
59	`ipv6_http_request_event`

Registry Event Subtypes	
0	`registry_create_event`
1	`registry_modify_event`
2	`registry_delete_event`

Security Event Subtypes	
4624	`Logon Successful`
4625	`Logon Failure`
4634	`Logoff`
4648	`Explicit Logon Attempt`
4672	`Admin Logon`
4800	`Workstation Locked`
4801	`Workstation Unlocked`

- To leverage Opcodes, simply use the following syntax:
  `process where opcode == 1.`