**Chronicle SIEM: Introduction & Single Event Rules**

**Overview**

Chronicle Security Operations empowers cloud-first, modern SecOps teams to protect their organizations confidently, with cloud-native architecture, petabyte scale, sub-second queries, and automated responses.

In this lab you will learn the basics for performing basic administrative tasks within the Chronicle platform.

Objectives

In this lab, you learn how to perform the following tasks:

* Get an overview of YARA-L.
* Understand single event rules.
* Duplicate a rule.
* Test a rule.
* Create a single event rule.

**YARA-L Overview**

When Chronicle was being built, the decision was made not to create another search language that users would need to learn to detect anomalies or threats in their data set. Instead, the Chronicle team leveraged YARA, which the Google VirusTotal team created and uses, but modified it to focus on logs and its associated telemetry.

* YARA-L mainly focuses on logs and associated telemetry.
* YARA-L is applicable for **real time detection of rule & historical rules**.
* YARA-L leverages UDM and its fields to create rules.
* For more information about YARA-L refer to the [whitepaper](https://go.chronicle.security/hubfs/YARA-L%20Overview%20White%20Paper.pdf).

Rule Construction

As you start digging into YARA-L rules, it is important to understand the manner in which the rules are constructed. There can be as many as six sections in a rule. However, for the most basic rules, there are only three required sections:

* meta
* events
* condition

As you add more capability to your rules, there are additional sections to add, but those extra sections also provide greater benefit in terms of fidelity and capability.

Rules can contain up to 6 sections:

1. Meta
   * Uses arbitrary key value pairs to describe the rule
   * Key values for "author" and "severity" are hard-coded to surface into the Rules Dashboard
2. Events
   * Event filter conditions, similar to a WHERE clause in a SQL statement
3. Match (optional)
   * Describes the time window for a match, required when running multi-event correlation
4. Outcome (optional)
   * Additional information to be extracted for each detection.
5. Condition
   * Describes the events and other conditions that must be true for a detection.
6. Options (optional)
   * Options to turn on or off while executing this rule

Variables

In YARA-L 2.0, all variables are represented as $variable\_name.

Variables can be defined within rules. They can be very important when comparing multiple events as well as temporarily storing specific values from events. The variables can then be used in performing actions such as counting the number of times an event occurs.

There are three different types of variables; Event, Placeholder and Match. All variables will be referenced with a dollar sign followed by the variable name. The hashtag variable will get introduced into the Match variable as well, but let’s not get ahead of ourselves.

Every single condition in the event section will have a variable in front of the UDM event or entity fields associated with it. The name of the variable can be arbitrary as you will see throughout this lab. The documentation will tell you that generally $u would indicate UDM events and $e would indicate UDM entity events, but this is not a hard and fast rule and in some examples you may see a variable like $event or $selection.

When you start getting into multi-event rules, you can describe events in a manner that makes it clear which events are which, i.e $failure or $success which might denote failed login events as compared to successful login events.

Event Variables

* These are either UDM Events or Entity fields
* UDM fields are assumed if graph (entity events) is not specified
* Fields are referenced as a chain - $event.target.user.userid
* Variable names can (and often should) be more descriptive than just event or entity or graph
* When working with multiple events, identifying failed logins as $failure.target.user.userid and successful logins as $success.target.user.userid is an easy way to keep their criteria separated and avoid confusion.

**Placeholder Variable**

Defined in the event section and used to associate events with one another or to compare them.

$event.target.user.userid = $targetUser

Placeholder variables can be used to associate events with one another. You will be using this in the multi-event lab. In the above example, the $targetUser serves as a place where the target.user.userid value will reside. It can then be used to match other events with that same value. These placeholders can also be used in the outcome section.

**Match Variable**

One or more variables can be used to group results. These variable must first be defined in the events section as a placeholder.

$targetUser over 5m

The match variables are similar to the placeholder variables in the sense that they have been described in the event section but when used in the match section, you are going to apply that variable against a time window. In this case, you will be taking all of the events with the target user meeting the event criteria and grouping by that value over a 5-minute window.  
It is similar to a SQL group-by section.

Additional Notes

* # is a special character in the condition section that can be used with event or placeholder variable names.
* # can be used in place of $ for an event or placeholder.
* Events or placeholders with # in place of $ will return a count of how many distinct events or placeholders were matched by the rule criteria.
* When setting up conditions, you can use event or placeholder variable names. If you use the hashtag (#), you are looking for the number of distinct events or values that satisfy a condition, so $var would be equivalent to #var > 0.
* You can also use the operator and if you have multiple criteria you need to meet before the rule fires. In the above example, you are looking for more than 5 distinct events that match your event criteria AND more than two values in targetUser. You will get into more examples of variables as we go.

Event Operators and Modifiers

* Another capability within YARA-L is the use of event operators and modifiers.
* and, or, and not can all be used within the event section. These, along with parenthesis, can be used to build out search criteria. For those who have worked with other tools that require these terms to be capitalized that is not the case here. and is assumed in the absence of other operators. In fact, if you define a list of events without specifying any operators it is assumed that and exists between each line.
* Parenthesis can be used to ensure order of precedence is maintained.
* nocase is a powerful modifier as well. This can be used to ignore case sensitivity by appending it to an event condition. In the below example, you are looking at a string, but this could apply to a regex as well. One important thing to note about nocase is that it cannot be applied to enumerated fields like [metadata.event\_type](https://cloud.google.com/chronicle/docs/reference/udm-field-list" \l "metadataeventtype" \t "_blank) or [network.ip\_protocol](https://cloud.google.com/chronicle/docs/reference/udm-field-list" \l "networkipprotocol" \t "_blank). The fields that are enumerated are called out in the UDM field list and are what you would think of as reference lists that Chronicle has built into the system.

Working with Repeated (Multi-Value) Fields

* Fields that can have more than one value in them are called Repeated (Multi-value) fields. Ie mac, IP, nat\_ip.
* Please refer to the UDM field list to get more information.
* any or all can be used to limit or broaden the search for values in fields that are labeled as repeated or multi-value.
* For example, a server could have multiple IP addresses associated with it. The IP field is a repeated field and has an entry for each IP address. To ease searching in the rules engine, any or all can be used in front of the event variable to search through the repeated field list for the string specified.

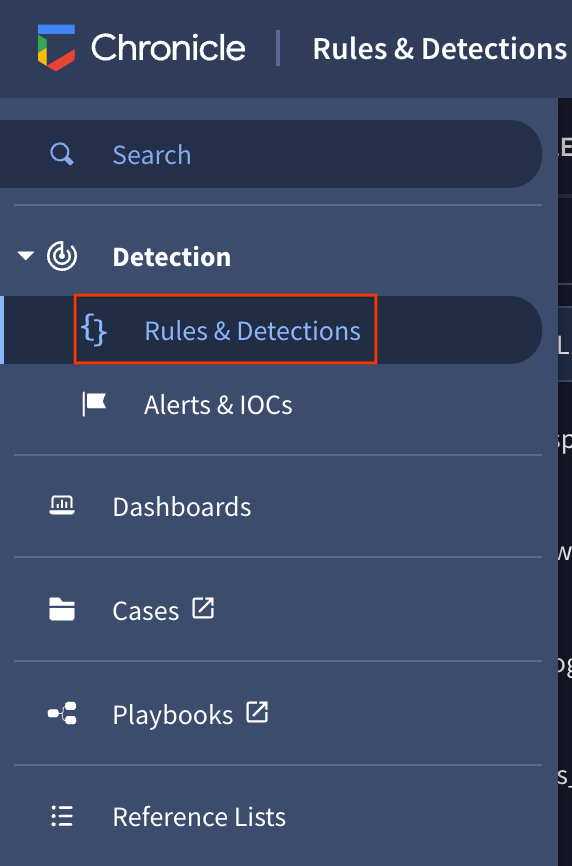
**Writing Rules**

With those basic concepts of YARA-L in place, let’s start looking at some rules. You are going to start with a single event rule and gradually add more complexity to it, so for some this may be review but for others it may be brand new.

Single Event Rules

\*\*As mentioned eariler, this is where you will use the additional browser tab or window to access the Chronicle Demo Environment[https://goo.gle/chroniclelab]

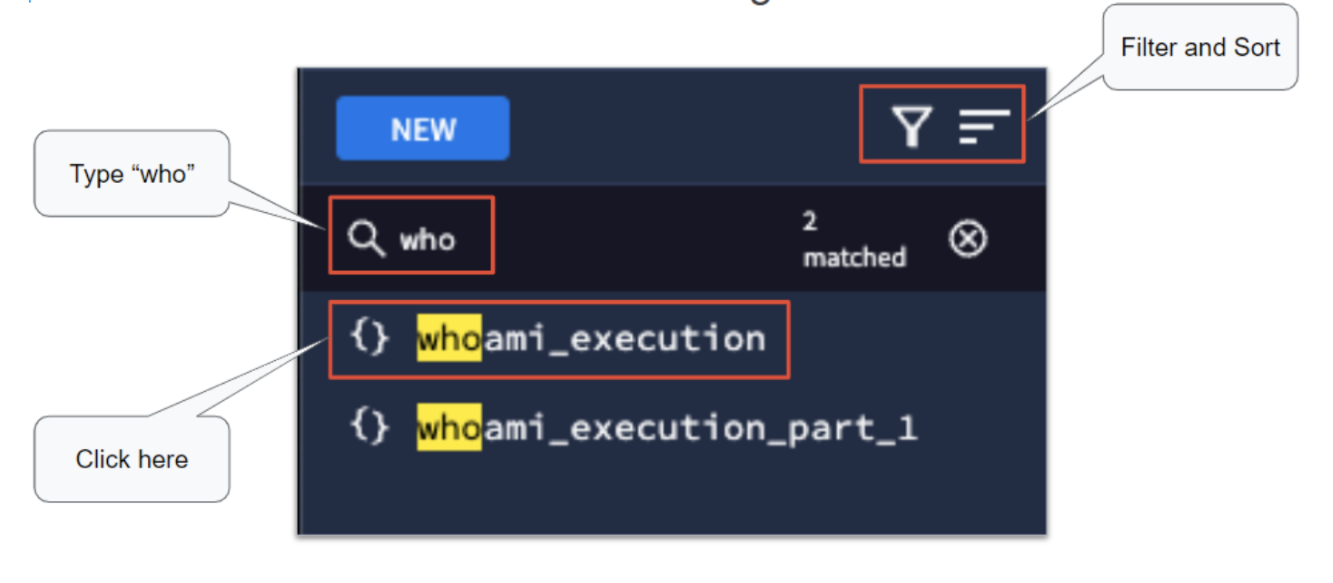
1. Start by navigating to the Rules section of the Chronicle instance. From left navigation menu, click **Detection** > **Rules & Detections**.



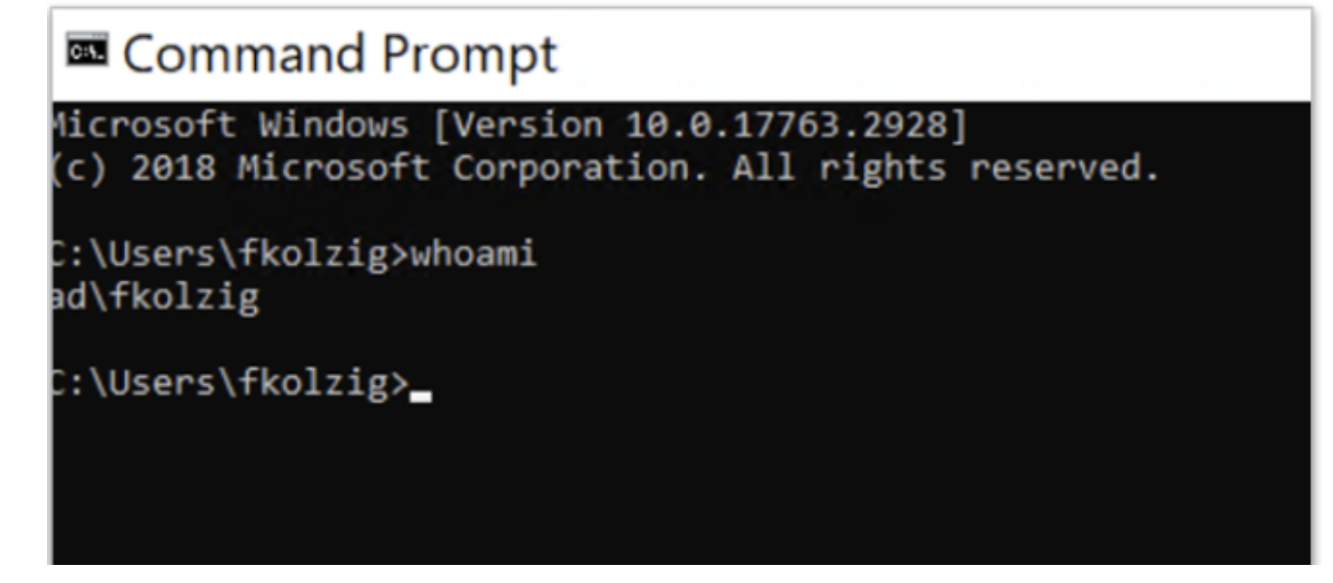
1. Select **Rules Dashboard** tab. The Rules Dashboard provides a breakdown of all the rules configured within the instance, as well as metrics around each of their performance. You can see when each of the rules last had a detection, metadata associated with the rules, how many retrohunts have been performed on each of the rules, the rule type, the frequency with which they are run, and whether they're live. Click on Rules Editor near the top of the window so you can view the logic within your rules.



1. On the left side of the screen, there is a list of all of the rules on the instance. This list of rules can be filtered to include active rules, archived rules, or both. Active rules does not imply they are Live and Alerting, just that they are available for you to work with. If rules are archived, they can not be edited until they are switched back to active. The sort allows you to sort the rules in alphabetical order, reverse alphabetical, or by last updated date. At the top, you can type a string in the text box and the rules that match that specified string will be shown. Type “who” into the text box and you will see the rules list shrink down to a smaller number. For this example, you want to look at the rule named whoami\_execution, so click on that rule.



1. This is a very basic rule that alerts every time someone executes the "whoami" command, it serves as a starting point to understand the rules engine and how rules are constructed.



* You will see the rule whoami\_execution. There is an abbreviated version of the rule here with some of the metadata and comments removed so it fits on the screen but yours should look similar.
* The events section is looking for values in two fields, the metadata.product\_event\_type and the target.process.command\_line. This rule happens to rely specifically on Microsoft Sysmon data and the product\_event\_type is 1 which denotes process creation. The command\_line is just looking for the string "whoami". Notice the $selection variable in front of each field.
* The last piece is the condition section which is just looking for the events, in this case called $selection. If you see an event that matches the two conditions in the event section, the rule fires.

1. On the far right side of the screen, you will see three dots. Click on the dots and you will have a listing of options that can be applied to this rule. Let’s walk through these options briefly.

RuleConfiguration

* The concept of a Live Rule is that the rule is actively being used to evaluate new events coming into Chronicle. If the user has built and tested a rule and wants to make it available for their analysts, the user would mark it as a live rule.
* Alerting is different than a live rule. Alerting is actually a triggering mechanism that creates an alert in the Alerts and IOCs screen when a rule fires. It is possible to have rules be live and not alert and vice versa. For example, a rule that is set to alert but is not live, could be used in a retrohunt over historical data and if it was set to alert, it could create a set of alerts based on the retrohunt that was conducted.ß
* Run frequency sets the frequency that a rule would run. The choices are 10 minutes, one hour, and 24 hours. Depending on the rule you have built, some of those options may not be available. For example, in a multi-event rule that has a match condition for 15 minutes, the 10 minute frequency would not be available because the time window of the match is greater than the frequency of the rule that it is running against.
* YARA-L retrohunt allows you to run a rule against a historical time range.
* Duplicate rule allows you to make a copy of a rule. You can then modify this copy to fit your needs without worry of modifying the original rule. This will be done during this lab so as to leave the original rule intact for others.
* Viewing rule detections allows you to see the previous detections that a specific rule triggered over a timeline, as well as a graph view.
* You can view previous versions of a rule to see what changed in your criteria and even save an additional version of the updated rule.
* Finally you can archive the rule which disables alerting and editing for that rule. As mentioned earlier, archived rules can be filtered out from the rules list (this is the default setting). If you want to see archived rules you will need to select them from the filter menu.

Testing a Rule

* At the bottom of the rule editor is the test rule capability. Using this is considered a best practice when building rules because it runs very quickly against a time window to give you feedback on the criteria of the rule you are creating or modifying. The other nice thing about this is that even if the alert flag is switched on, the test rule functionality will not create alerts. If running a retrohunt alerts would be createdd. The test rule functionality can only run against a maximum of a two week interval of data. A retrohunt, on the other hand, can hunt against a much wider timeframe.
* If you would like to test this rule, go ahead and set the time range to the past three days and click Run Test.
* On the left side of the screen, you get a listing of detections and to the right a bar chart with the date the events were seen. Notice in the detection section on the left side (the section in white), that you can click on the double-headed arrow to expand the section, you can wrap text, add columns and more. This capability isn’t unique to test rules, in fact you will see this elsewhere and it will used a bit later.

RegEx String Matching

Now let's move a little further into the event section and move away from just simple string matching across an entire field.

In the below example rule, you are looking at a MITRE ATTACK technique around establishing a windows admin share. The event criteria here is similar to the first example in the sense that you are looking for an event with a target.process.command\_line of some value. However in this case, you are going to have some variance that will need to be accounted for. You want your newly created rule to fire anytime a "net use" command is executed looking for c$, admin$ or ipc$. You could use "or" statements but the problem complexity grows when an adversary specifies a specific drive letter in this command, which would create a number of permutations when combined with the three different shares. That would create a pretty big "or" statement. To complicate matters, Windows commands like "net" can sometimes have extra spaces in between the commands and a string match would also need to account for those spaces. To address these challenges, you can use regex to handle this nicely, using the syntax below:

|  |
| --- |
| rule mitre\_attack\_T1021\_002\_windows\_admin\_share\_basic {  meta:  author = "Google Cloud Security"  description = "Net use commands for SMB/Windows admin shares"  reference = "https://attack.mitre.org/techniques/T1021/002/"  yara\_version = "YL2.0"  rule\_version = "1.0"  events:  $event.target.process.command\_line = /net.\*use.\*(C|ADMIN|IPC)\$/ nocase  condition:  $event  } |

* Using forward slashes denotes a regular expression. More details on syntax are available in the [documentation](https://cloud.google.com/chronicle/docs/detection/yara-l-2-0-syntax#regexp_functions). To address the variability in spaces and drive letters you place .\* between the commands. Because you are looking for either "C", "ADMIN", or "IPC", you can group those three variables and put a pipe (regular expression or statement) between each. Finally, you are looking for a dollar sign after one of those three terms but because the dollar sign has special meaning in regular expressions, you need to prepend it with an escape character, the backslash \</code> , so that Chronicle knows to treat the dollar sign as a literal dollar sign rather than denoting the end of a line. You can close the regular expression with a forward slash. Adversaries can try to be tricky by mixing capitalization with lower case, in order to account for that you add the nocase modifier to the end of your search.
* There are more mechanisms to work with regular expressions that will be introduced during the function section of this Lab. In fact, you may already be using the regex function in your rules.