<Instructor>

32 WPS/DOA | Nellis AFB, Nevada

**<Weapon/Defensive> System Training - <#>**

Lab: <Content>

Contents

[Lab: Threat Hunting with Event Logs and Wireshark 2](#_Toc145852561)

[Overview 2](#_Toc145852562)

[Procedures 3](#_Toc145852563)

[Step 1. Configure Custom Index Patterns 3](#_Toc145852564)

[Step 2. Explore Event 4688 Fields 6](#_Toc145852565)

[Step 3. Review the Index Mapping 10](#_Toc145852566)

[Step 4. Create a Threat Hunting Dashboard 13](#_Toc145852567)

[Step 5. Create Visualization of Child Processes 15](#_Toc145852568)

[Step 6. Create Visualization of Parent Processes 21](#_Toc145852569)

[Step 7. Create a Lens Table Detailed View 24](#_Toc145852570)

[Step 8. Experiment with Filters and Understand the Impact 28](#_Toc145852571)

[Summary 31](#_Toc145852572)

[Appendix 33](#_Toc145852573)

[Appendix A: Event Id 4688 Fields 33](#_Toc145852574)

[Appendix B: ElasticSearch Fields 35](#_Toc145852575)

|  |  |  |
| --- | --- | --- |
| **Symbols Table** | |  |
| **Symbol** | **Name** | **Meaning** |
| ✅ | **Note** | Detailed information that is required to fully understanding the concept or to be able to execute a procedure but is not necessarily related to a key learning objective. |
| **💡** | **Learning Point** | Information related to key learning objectives. |
| ⚠️ | **Warning** | Important information related to safety and security. |
| ✋ | **Raise Hand** | Raise your hand for instructor assistance. This is often used at critical points to validate your understanding of the material. |

# Lab: Threat Hunting with Event Logs and Wireshark

## Overview

**Summary:** The purpose of this lab is to acquire the necessary knowledge and skills to effectively navigate event logs and Wireshark queries to detect and investigate malicious cyber activity.

**Outcomes:** By the end of the lab, you will be able to perform the following:

* Discover events and fields extracted from provided data sources.
* Create Windows event log queries using PowerShell and Command Line.
* Create PCAP queries using Wireshark.
* Create analytics for detecting or highlighting malicious activity.

## Procedures

### Configure Custom Index Patterns

Index Patterns are used in ElasticSearch queries to identify the data that you want to explore. This allows the operator to query multiple indices or only one specific index depending upon how specific the pattern is. For example, you may want to run a query against both Sysmon logs and Windows Event Logs searching for process execution because Sysmon isn’t fully deployed in the environment. The two datasets likely reside in different indices because they contain a different schema. Another reason to use index patterns is because your indices are named based upon the rollover date. Instead of having one massive index with every log ever created in the environment, you may have multiple indices such as so-beats-2022.04.03 where the data at the end increments every X days or when the index reaches a certain size. You could create an index pattern such as so-beats-\* to query all of the logs that have ever existed, or you could query so-beats-2022-04-\* to only get logs from April 2022, thus reducing the amount of time it takes to return the query compared to querying every index.

1. Navigate to SecurityOnion by going to the following URL:
   1. **URL:** <https://192.168.10.4>
   2. **Username:** soadmin@lab.net
   3. **Password:** !QAZ2wsx
2. Go to **Tools → Kibana → Login**
3. Login with the following credentials:
   1. **Username:** soadmin@lab.net
   2. **Password:** !QAZ2wsx
4. Click on the three horizontal bars in the top, left-hand corner of the screen to open the main menu, then navigate to **Stack Management → Kibana → Index Patterns**.

A screenshot of a computer

Description automatically generated with medium confidence

There are several pre-built index patterns in SecurityOnion. The primary index pattern is the \*:so-\* pattern which cover multiple indices, but not the ones we’re interested in. We are primarily interested in indices that match the pattern “so-\*” such as:

|  |  |
| --- | --- |
| Index | Purpose |
| so-zeek-yyyy.mm.dd | The default location for zeek logs. |
| so-beats-yyyy.mm.dd | The default location for beat data, including Winlogbeat. Data collected from Sysinternals Sysmon is normalized to a different index. |
| so-ElasticSearch-yyyy.mm.dd | The default location for generic logs residing in ElasticSearch. |
| so-ossec-yyyy.mm.dd | The default location for logs from the Wazuh agent. |
| so-osquery-yyyy.mm.dd | The default location for logs from the osquery agent. |
| so-zeek\_dns-yyyy.mm.dd | The default location for zeek DNS logs. |
| so-ids-yyyy.mm.dd | The default location for logs originating from the Intrusion Detection System (IDS) which is typically Suricata or Snort. |

For this lab, we will focus on the so-beats-yyyy.mm.dd indices which is where all of our Windows Event log data is being stored after being processed by Logstash. The default index pattern \*:so-\* does not capture what we want, so we are going to create a new index pattern to limit our queries to just the Windows Event logs.

1. From the “Index patterns” page, click on the “Create index pattern” button in the top, right-hand corner of the screen. On the “Create index pattern” screen, type “\*so-beats-\*”. The filter should return only indices related to beats.

A screenshot of a computer

Description automatically generated with medium confidence

Next, we need to specify the default Timestamp field that will be used with the time range filter in ElasticSearch queries and the Kibana dashboard.

1. Click on the “Timestamp field” dropdown menu and select the @timestamp. This field is the one generated by Logstash, so we should use this one.

Graphical user interface, text, application

Description automatically generated

1. Next, click on the “Create index” button at the bottom of the screen. Now that we have created our beats Index Pattern, we will use that pattern to explore our dataset.
2. Navigate to **Main Menu → Analytics → Discover**.
3. Select our newly created \*so-beats-\* index pattern from the drop-down menu in the top, left-hand corner of the screen underneath the search bar.

Graphical user interface, text

Description automatically generated

✅**NOTE: The logs are currently a combination of Windows Event logs as well as logs related to Winlogbeat agent connections. We want to explore just the Windows Events, so we need to filter out other events.**

In order to show only Windows Event Log events, we can create a filter to only show logs where the field winlog.event\_id exists.

1. Build the following query in the filter bar and hit enter to apply:

|  |
| --- |
| winlog.event\_id:\* |

You should see the number of events decrease, reflecting the fact that non-winlog events have now been filtered out.

### Explore Event 4688 Fields

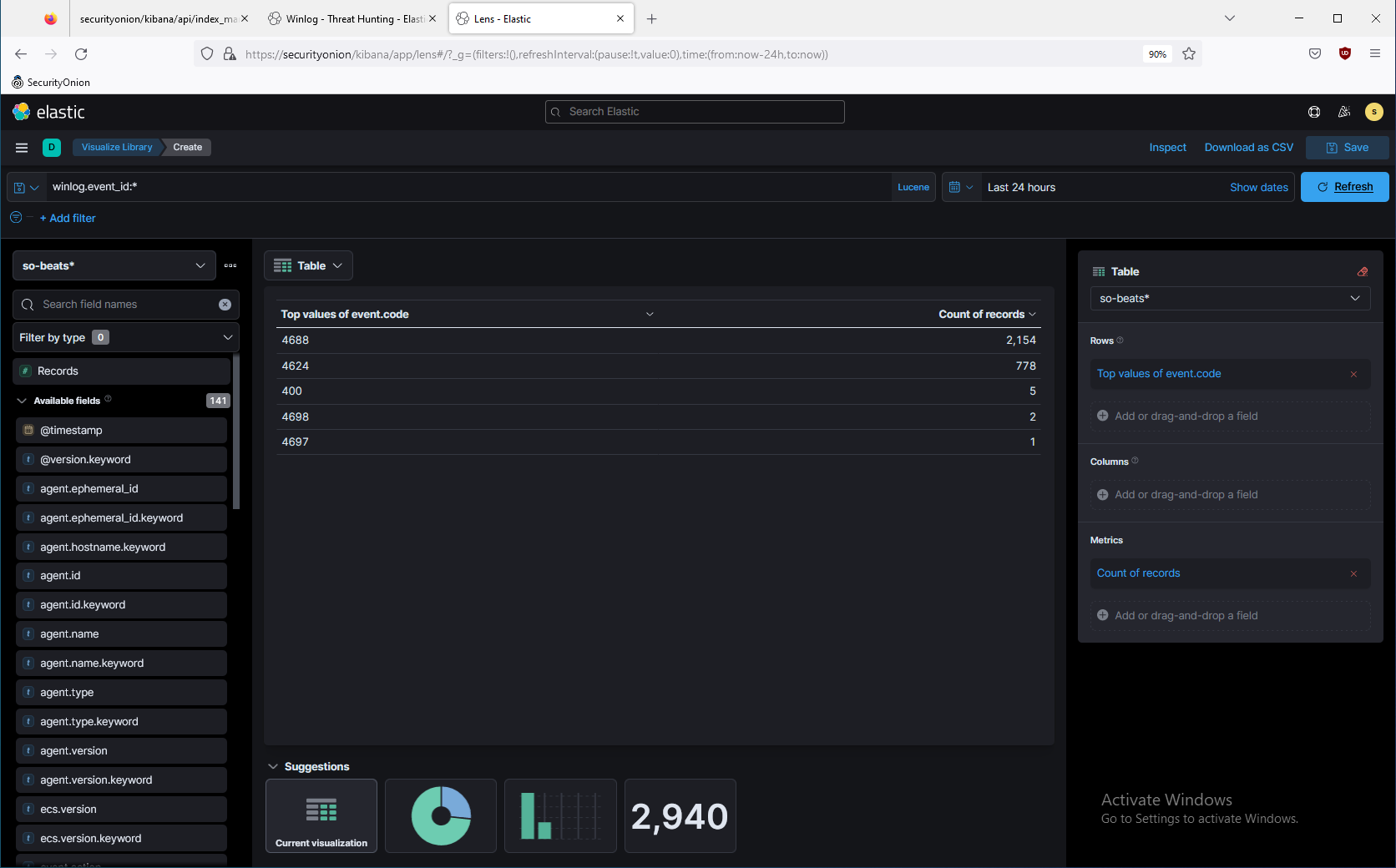
Before we can start building visualizations, dashboards, and analytics we need to take some time to understand the events and the semantic meaning of each of the fields. There are several nuances with the fields and what the information contained within them represents. We also need to evaluate the event to see if there is even any useful data that we could use for threat hunting. To do that, we are going to open a few events in the Discovery page and look over what is available.

1. From the Discovery page, find the “Available fields” list and click on the “event.code” field. If it is not easily found, you can search for it be typing “code” in the “Search field names” text box above the list. The event.code field contains the Event Id for the Windows Event log. Click on the field to bring up a summary of the values contained in this field. We are going to be particularly interested in Event Id 4688 – Process Creation because that event is really useful for a variety of analytics covering multiple categories of attacker TTPs.

A screenshot of a computer

Description automatically generated with medium confidence

1. Click on the blue “Visualize” button at the bottom of the “event.code” preview popup. This will take you to the Lens visualization creator screen. We are just going to use this to explore the data a little more.
2. On the far, right-hand corner of the screen, click on the event.code field and configure the following values:
   1. **Display count:** 100
3. Click the “X” button at the top, right-hand corner of the screen to close the settings for the event.code field.
4. Change the Visualization type to “Table” by selecting that value from the drop-down menu located at the top, left-hand corner of the Lens visualization screen. You should see a view similar to the one below:



1. Next, find the event.action.keyword field and add it to the workspace by clicking the + button or dragging the event to the “Rows” section in the right-hand panel. You should now see human readable names of each Windows Log event.

A screenshot of a computer

Description automatically generated

**Add event to table button**

1. Click the back button in the browser to navigate back to the Discovery page.
2. Next, add a filter to only show process creation events. Click on the “+Add filter” button in the top, left-hand corner of the Discovery page and configure the following options:
   1. **Field:** event.code
   2. **Operator:** is
   3. **Value:** 4688
3. We want to explore the data we get for these events and discover what fields might be interesting to develop dashboards and analytics against. To do that, expand one of the events by clicking on the “>” symbol next to the event and then scroll down to view the fields that were extracted from the Event Log.

A screenshot of a computer

Description automatically generated with medium confidence

1. Review the descriptions of Event Id 4688 located in the table in Appendix A. Additionally, there are several warnings and learning points located at the bottom of that appendix that are worth paying attention to.

### Review the Index Mapping

Every Index contains a field mapping that describes the structure of the documents loaded into that particular index to include field names and types. This is important to reference when you are trying to analyze data. Different field types support different types of filters, searches, aggregations, and mathematical operations. For example, keyword fields are not ideal for wildcard queries, but the wildcard or text types support those operations. In this step, we are going to show you where you can find the index mapping for a given index and how to find and interpret the mappings for specific fields.

1. Start by clicking on the three bars in the top, left-hand corner of the screen and select **Management → Stack Management**.
2. Click on **Data → Index Management**.
3. Click on one of the “so-beats-\*” indices in the list shown below:

Graphical user interface, application

Description automatically generated

1. Click on the **Mappings** tab in the top of the screen. You should see a JSON document that describes the structure of the index similar to the snippet shown below.

Text

Description automatically generated

1. Copy the JSON out to notepad++ to make it easier to view. Then save the document with the .json extension in order to get syntax highlighting and collapsers on the left-hand side of the screen that make this large json blob easier to work with.
2. Type CTRL+F and search for “process” and then look at the first subfield called “args”.

|  |
| --- |
| "args": {  "type": "keyword",  "ignore\_above": 1024,  "fields": {  "security": {  "type": "text",  "analyzer": "es\_security\_analyzer"  }  }  } |

The first result from the top should bring you to process field structure definition shown above. Take a look at the definition for the “args” property which is designed to capture the commandline arguments of a process creation event split by token (i.e. this field is an array of strings).

**💡LEARNING POINT: Strings longer than the ignore\_above setting will not be indexed or stored. For arrays of strings, ignore\_above will be applied for each array element separately and string elements longer than ignore\_above will not be indexed or stored. This means that if an adversary executed powershell.exe with a 16K character Base64 encoded script, that element of the commandline arguments would NOT get stored in this particular field. That means that if you search for Base64 strings in Kibana based on this field, you wouldn’t see the attack! It is absolutely critical that you understand these mappings so that you don’t make false assumptions about the data.**

1. Next, search for the “command\_line” field. You should see the configuration shown below:

|  |
| --- |
| "command\_line": {  "type": "wildcard",  "fields": {  "keyword": {  "type": "keyword"  },  "security": {  "type": "text",  "analyzer": "es\_security\_analyzer"  }  }  } |

This field is of type “wildcard” which is perfect for this field type because it is optimized for wildcard and regular expression searches. There is no configured ignore\_above field, so this should capture the entire commandline argument regardless of how large it is (the max commandline length allowable on Windows is 32KB, so all of them *should be* less than or equal to that limit).

There are also two fields nested underneath called “keyword” and “security”. These fields are the same as the parent, but are stored as different types so that you can select the field type that is most appropriate for the filter, search, or aggregation that you are trying to build.

Reference Appendix B for a description of different field types and the differences between them.

✅**NOTE: The index mapping document is approximately 430K characters and 12K lines long, so we won’t (and realistically cannot) review the entire document. It is just important to keep around and reference for the specific fields you need to build filters, searches, aggregations, and visualizations from.**

So now we know the structure and type definitions for fields in the index, but where do those fields come from? The fields come from one of a few places: (1) the original event itself, (2) the JavaScript processor running in winlogbeat, or (3) a Logststash pipeline. To speed things up, the last two fields came from the JavaScript processor. Let’s take a look at the script that generates these fields.

1. Open up the winlogbeat-security-custom.js file located at "C:\Users\Assessor\Desktop\WST2\winlogbeat-security-custom.js" in notepad++.
2. Search for “process.args” and continue until you find the function shown below.

|  |
| --- |
| **function(**evt**)** **{**  **var** cl **=** evt**.**Get**(**"winlog.event\_data.CommandLine"**);**  **if** **(!**cl**)** **{**  **return;**  **}**  evt**.**Put**(**"process.args"**,** windows**.**splitCommandLine**(**cl**));**  evt**.**Put**(**"process.command\_line"**,** cl**);**  **}** |

The function above creates the “process.args” field we reviewed in the index mapping from the last step. It uses the windows.splitCommandLine function to create an array of strings that will be stored in this field. The documentation is unclear as to how this function parses the field; however, I have noticed that it is not always tokenized in the manner I would expect.

### Create a Threat Hunting Dashboard

Ultimately, we want a threat hunting dashboard that can be used to grok the data we receive from the Event Id 4688 and display it in a form that is useful for filtering out false positives so that we can drill down to suspicious events. To do that, we are going to start by creating the dashboard itself. Once that is created, we will begin adding visualizations to the dashboard to help understand the data.

1. Start by navigating to **Main Menu → Analytics → Dashboard.** In SecurityOnion, this will drop you into the main **Security Onion – Home** dashboard. To create a new dashboard, you may need to navigate back to the “Dashboard” page by clicking on the “Dashboard” link in the top, left-hand corner of the screen.

A screenshot of a video game

Description automatically generated

You should end up on a page that looks like the one below:

A screenshot of a computer screen

Description automatically generated with medium confidence

1. Click on the bright blue “Create dashboard” button. You should end up on a page like the one below:

A screenshot of a computer screen

Description automatically generated with medium confidence

1. Start by saving the Dashboard so that we can find it and return to it later. Click on the “Save” button in the top, right-hand corner of the screen. Name the new dashboard “Winlog – Threat Hunting (<last name>)”. If multiple students are using the same range, add your last name to the dashboard name to make it unique. Give it a short description, then click the “Save” button at the bottom of the screen.

A screenshot of a phone

Description automatically generated with medium confidence

### Create Visualization of Child Processes

The first analytic we are going to tackle is Frequency Analysis of parent and child processes. One way to identify anomalous activity is through frequency analysis of a particular field. In general, legitimate process paths are going to be executed more frequently and on more systems than the paths to malicious executable files. Let’s assume that a malicious executable is dropped to disk and executed and that the file is not masquerading the full path to a legitimate executable such as C:\Windows\System32\svchost.exe. That means that, in this situation, the malicious binary will stand out by having a unique path.

We are also going to assume that all of the systems in are environment are fairly homologous (i.e. they all have relatively similar configurations and executables located in standard paths). Given those assumptions, we can create a visualization to show the least frequent binaries based on the number of systems that have observed processes created from those paths.

**💡LEARNING POINT: We are going to count each unique path by the number of unique computer names a path has been observed on. This is better for our use case than pure frequency analysis because a malicious process executed 1000 times may blend into the background, but a malicious process on only one computer in the environment will still show up no matter how many times the process executes.**

In order to create this visualization, follow the steps below:

1. Start by navigating to **Main Menu → Analytics → Visualize Library**.

A screenshot of a computer screen

Description automatically generated with medium confidence

1. Next, click on the bright blue “Create visualization” button. You should see the window below.

Graphical user interface, application

Description automatically generated

1. We are going to create an “Aggregation based” visualization, so click on the “Aggregation based” button.

**💡LEARNING POINT: We are explicitly avoiding the “Lens” visualization editor because it limits our options for creating the type of visualization that we want to create. Additionally, the amount of rows returned is capped at 1000 where as “Aggregation based” visualizations are capped at the ElasticSearch global setting search.max\_buckets, which defaults to 65535 values.**

Graphical user interface, application

Description automatically generated

1. Next, select “Data table” from the list of Visualization types. This should bring you to the data source selection menu shown below:

Graphical user interface, text

Description automatically generated

1. Finally, select the \*so-beats-\* index pattern we created earlier. You should now be on the Visualization editor that should look like the image below:

Graphical user interface, application

Description automatically generated

1. First, we are going to configure a filter so that this aggregation is only applied to Event Id 4688 events.
   1. Click on the blue “+ Add filter” button in the top, left-hand corner of the screen.
   2. **Field:** winlog.event\_id
   3. **Operator:** is
   4. **Value:** 4688

A screenshot of a computer screen

Description automatically generated with medium confidence

* 1. Then click the “Save” button.

✅**NOTE: This field will only apply to this visualization and not the entire Dashboard. If an operator configures a filter at the dashboard level to only show Event Id 4624 events, then nothing will show in this visualization.**

1. Configure the “Metrics” option with the following settings:
   1. **Aggregation:** Unique Count
   2. **Field:** winlog.computer\_name
   3. **Custom label:** Count
   4. Click on the blue “Update” button to get a visual for what the table will look like.

A screenshot of a computer

Description automatically generated with medium confidence

**💡LEARNING POINT: This aggregation will create a column in the Data table that is the unique count of all computer names for each given row in the Data table.**

1. Next, we need to determine which filed to use to split the rows of our Data Table visualization. This is where the discover panel and index mapping comes in useful. We need the full path to the executable, preferably as a “keyword” type as those are optimized for aggregations. Searching through the index mapping, we see the “winlog.event\_data.NewProcessName” field configured below:

|  |
| --- |
| "NewProcessName": {  "type": "keyword",  "ignore\_above": 1024,  "fields": {  "security": {  "type": "text",  "analyzer": "es\_security\_analyzer"  }  }  } |

✅**NOTE: You can find this index mapping by searching through the json document we previously saved for the term “NewProcessName”.**

✅**NOTE: This field limits the length to 1024 characters, which *should* be fine for most scenarios as *most* Windows API’s limit file paths to 260 characters or less; however, it is theoretically possible to hide from this visualization by executing a process stored in a path greater than 1024 characters. Index Mapping types are important #footstomp.**

1. Next, we are going to split the rows in the Data table by the unique NewProcessName (case-insensitive since these are Windows systems). Configure the “Buckets” option with the following settings:
   1. Split rows
   2. **Aggregation:** Terms
   3. **Field:** winlog.event\_data.NewProcessName
   4. **Order by:** Metric: Count
   5. **Order:** Ascending
   6. **Size:** 1000
   7. **Group other values in separate bucket:** Checked
   8. **Show missing values:** Checked
   9. **Custom label:** NewProcessName

A screenshot of a computer

Description automatically generated with medium confidence

**💡LEARNING POINT: This bucket option will create a new row for each unique NewProcessName (case-insensitive). We order the results in ascending order to show the most anomalous entries at the top. Given that the path to the malware in our hypothesis is unique, it should show up near the top of this visualization.**

**💡LEARNING POINT: When this bucket configuration is combined with the previously created metric, you get the number of hosts were a NewProcessName value was found. For example, a row containing NewProcessName = C:\Windows\System32\BackgroundTransferHost.exe and a Unique count of winlog.computer\_name value = 1 means that this particular process path has only be observed from one system in the given timeframe.**

**💡LEARNING POINT: The “Show missing values” option is helpful to highlight events that didn’t have a value for this field, which is an indicator that something is wrong (e.g. a processor threw an error before creating this field).**

1. The default number of rows displayed per panel is 10, which is probably too low for this visualization. To change this value, click on the “Options” tab in the visualization editor. The configure the following settings:
   1. **Max rows per page:** 100
   2. **Show toolbar:** true (gives us an export button)

A screenshot of a computer

Description automatically generated with medium confidence

1. Next, click the blue “Update” button in the bottom, right-hand corner of the page. You should see results similar to the following:

Graphical user interface, text, application

Description automatically generated

✋**RAISE HAND: Explain what this visualization displays and how it works. Be sure to include an explanation of how Unique Count works compared to Count.**

This visualization is looking pretty good. It’s **NOT** going to find everything for us, especially on larger networks, but it will give us valuable insight and context when combined with other visualizations.

1. Save the visualization by clicking on the “Save” button in the top, right-hand corner of the screen. Fill out the fields listed blow and then click on the “Save to library” button.
   1. **Title:** Winlog - 4688 - NewProcessName Summary (<last name>)
   2. **Add to dashboard:** Winlog – Threat Hunting (or whatever name you used before)
   3. Click the blue “Save and go to dashboard” button.

Graphical user interface, text, application

Description automatically generated

1. Stretch and position the visualization so that the final dashboard looks like the one below:

A picture containing table

Description automatically generated

### Create Visualization of Parent Processes

We are going to create another visualization to show a unique count of computer\_name by parent process path. Essentially, we will create the same visualization we made in Step 5, but use the ParentProcessName field instead of the NewProcessName field.

1. Ensure the Winlog – Threat Hunting dashboard is in “edit” mode by viewing the top, right-hand corner of the screen. You should see the “Switch to view mode” button if the dashboard is already in edit mode. If instead you see a button called “Edit”, click on that button to switch to edit mode.

A screenshot of a computer

Description automatically generated with medium confidence

1. Click on the “gear” icon in the top, right-hand corner of the “Winlog – 4688 NewProcessName (<last name>)” visualization and then click “Clone panel”.

A screenshot of a computer

Description automatically generated with medium confidence

You should now have a new visualization titled “Winlog – 4688 NewProcessName (copy 1).

1. Click the “gear” icon in the top, right-hand corner of the cloned panel, and then click “Edit Visualization”.
2. Change the Buckets configuration to reflect the settings below:
   1. **Field:** winlog.event\_data.ParentProcessName.keyword
   2. **Custom label:** ParentProcessName

A screenshot of a computer

Description automatically generated

1. Click on the blue “Update” button in the bottom, right-hand corner of the screen. Verify that the results look similar to the image below:

Text

Description automatically generated

1. Next, save the Data Table to the Visualizations Library by clicking on the blue “Save to library” button in the top, right-hand corner of the screen.
   1. **Title:** Winlog – 4688 ParentProcessName (<last name>)
   2. **Update panel on Dashboard:** Checked
   3. **Save as new visualization:** Unchecked

A screenshot of a phone

Description automatically generated with medium confidence

1. Click the blue “Save and return” button.
2. Click on the “gear” icon at the top of the visualization, and then click “Edit panel title”.
3. Change the title to “4688 – ParentProcessName (<last name>)”.
4. Position the visualization so that the dashboard looks like the image below.

A screenshot of a computer

Description automatically generated with medium confidence

1. Click the blue “Save” button in the top, right-hand corner of the screen.

### Create a Lens Table Detailed View

The next thing we are going to add to our dashboard is a detailed table of events with specific properties that provide context to the events, given the current filters. We will also use this table to create Drill Down events that pass arguments to an investigation dashboard. We want to be able to pass multiple fields from this table on the next dashboard, so we have to use a Lens Table as opposed to an Aggregation Table that we used in the previous steps.

**💡LEARNING POINT: Aggregation Tables only support the “Single click” and “Context menu” Drill Down events while limiting the arguments that can be passed to the URL to only the filters currently applied and a single table row value.**

1. From the Winlog – Threat Hunting dashboard, click on the blue “Create visualization” button in the top, left-hand corner of the screen. This will take you to the Lens visualization page.
2. First, we are going to configure a filter so that this aggregation is only applied to Event Id 4688 events.
   1. Click on the blue “+ Add filter” button in the top, left-hand corner of the screen.
   2. **Field:** winlog.event\_id
   3. **Operator:** is
   4. **Value:** 4688

A screenshot of a computer screen

Description automatically generated with medium confidence

* 1. Then click the “Save” button.

1. Change the visualization type to “Table” by selecting it from the drop down menu to the right of the index pattern selector.

Graphical user interface

Description automatically generated

1. Use the “Search field names” text box to search for the process.child.command\_line field. Drag the field onto the visualization.
2. Under the “Rows” section in the panel on the right-hand side of the screen, click on the blue field name an apply the following settings:
   1. **Select a field:** process.child.command\_line.keyword
   2. **Number of values:** 100
   3. **Rank by:** Count of records
   4. **Rank direction:** Ascending
   5. **Group other values as “Other”:** Checked
   6. Click on the “X” in the top, right-hand corner of this panel to apply the settings.
3. Use the “Search field names” text box to search for the process.pid field. Drag the field onto the visualization.
4. Under the “Rows” section in the panel on the right-hand side of the screen, click on the blue field name an apply the following settings:
   1. **Select a field:** process.pid
   2. **Number of values:** 100
   3. **Rank by:** Count of records
   4. **Rank direction:** Ascending
   5. **Group other values as “Other”:** Checked
   6. Click on the “X” in the top, right-hand corner of this panel to apply the settings.
5. Use the “Search field names” text box to search for the user.name field. Drag the field onto the visualization.
6. Under the “Rows” section in the panel on the right-hand side of the screen, click on the blue field name an apply the following settings:
   1. **Select a field:** user.name.keyword
   2. **Number of values:** 100
   3. **Rank by:** Count of records
   4. **Rank direction:** Ascending
   5. **Group other values as “Other”:** Checked
   6. Click on the “X” in the top, right-hand corner of this panel to apply the settings.
7. Use the “Search field names” text box to search for the winlog.computer\_name field. Drag the field onto the visualization.
8. Under the “Rows” section in the panel on the right-hand side of the screen, click on the blue field name an apply the following settings:
   1. **Select a field:** winlog.computer\_name
   2. **Number of values:** 100
   3. **Rank by:** Count of records
   4. **Rank direction:** Ascending
   5. **Group other values as “Other”:** Checked
   6. Click on the “X” in the top, right-hand corner of this panel to apply the settings.
9. Use the “Search field names” text box to search for the @timestamp field. Drag the field onto the visualization.
10. Under the “Rows” section in the panel on the right-hand side of the screen, click on the blue field name an apply the following settings:
    1. **Select a field:** @timestamp
    2. **Customize time interval:** Checked
    3. **Minimum interval:** 30 seconds

✅**NOTE: This interval only configures the *minimum* interval. Kibana will automatically determine the interval for you based on the current time filter and histogram:maxBars global setting. Normally, the range is somewhere between 1 minute to 10 minutes. I wish you had more control over this setting, but you don’t. Welcome to Lens visualizations.**

1. Click on the “Refresh” button. Arrange the rows so that the visualization looks like the image below:

A screenshot of a computer

Description automatically generated with medium confidence

1. Click on the blue “Save to library” button in the top, right-hand corner of the screen to save the visualization to the library.
   1. **Title:** Winlog – 4688 Lens Table Detailed (<last name>)
   2. **Add to Dashboard after saving:** Checked
   3. **Save the new Lens visualization:** Checked
   4. Then click the blue “Save and return” button.

Graphical user interface, application

Description automatically generated

1. Position the new table below the other visualizations we’ve already created so that the dashboard looks like the image below:

Graphical user interface, text

Description automatically generated

1. Save the updates we’ve made to the dashboard by clicking on the blue “Save” button in the top, right-hand corner of the screen.

### Experiment with Filters and Understand the Impact

We now have the core components of our Threat Hunting dashboard complete. It doesn’t look like much, but it gives you a lot of power to investigate the data. Let’s explore the data by creating a couple of basic filters and explain the impact those filters have on the rest of the dashboard.

One of the most prevalent attack vectors over the last 20 years has been Microsoft Office products, specifically winword.exe, powerpnt.exe, and excel.exe among others. One way to understand your environment is to review the parent and child processes of those applications to see if there are any processes that stand out. Understanding the parent and child processes.

1. Verify that the dashboard is in “view” mode.
2. In the search bar, type in “winword.exe”. You should see results similar to the following image:

A screenshot of a computer

Description automatically generated with medium confidence

**💡LEARNING POINT: Because no field was provided in this query, the multi\_match query defaults to the index.query.default\_field index settings, which in turn defaults to \* . \* and extracts all fields in the mapping that are eligible to term queries and filters the metadata fields. All extracted fields are then combined to build a query.**

The “4688 – ParentProcessName” visualization is going to show the winlog.event\_data.ParentProcessName field for all events where winlog.event\_data.ParentProcessName contains the string “winword.exe” **AND** all events where any other field contains winword.exe. The line where “winword.exe” is listed as parent process does not necessarily mean that “winword.exe” was ever a parent process of “winword.exe”, though this is often the case in the real world. This will be the same case for the “4688 – NewProcessName” visualization as well.

Let’s take a look at another query.

1. Type the query below into the search bar. You should see something similar to the image below:

**Query:**

|  |
| --- |
| winlog.event\_data.ParentProcessName:winword.exe |

**Output:**

A screenshot of a computer

Description automatically generated

✋**RAISE HAND: Explain to an instructor why the explorer.exe ParentProcessName observed in the previous query is no longer visible under the current query.**

1. Type the query below into the search bar. What do you expect to see? What was the difference between this query and the previous query?

**Query:**

|  |
| --- |
| winlog.event\_data.ParentProcessName.keyword:winword.exe |

**💡LEARNING POINT: Because we applied the filter to the winlog.event\_data.ParentProcessName.keyword field, we received no results because there is no ParentProcessName that contains only the text “winword.exe” by itself. Remember, keyword fields are not searchable via substrings. #footstomp**

A quick review of the index mapping for the ParentProcessName field is shown below. Because the ParentProcessName field is of type “text” it is searchable via wildcards and regular expressions whereas the ParentProcessName.keyword field is of type keyword and is not searchable via wildcards or substrings.

|  |
| --- |
| "ParentProcessName": {  "type": "text",  "fields": {  "keyword": {  "type": "keyword",  "ignore\_above": 32765  },  "security": {  "type": "text",  "analyzer": "es\_security\_analyzer"  }  }  } |

1. Run the following query to capture all events where the parent process is a Microsoft Office product:

**Query:**

|  |
| --- |
| winlog.event\_data.ParentProcessName:"C:\Program Files\Microsoft Office\" |

**Output:**

A screenshot of a computer screen

Description automatically generated

On a larger network, this dashboard with this query applied to it will give you a good understanding of what processes are spawned by Microsoft Office products. Because the “4688 – NewProcessName” implements frequency analysis, the least common child processes of Microsoft Office products will be shown towards the topic. Common child processes such as other Microsoft Office products, adobe reader, and web browsers (for when people click on embedded links inside of the documents they are using) will tend towards the bottom of the visualization.

This view has made it very obvious that powershell.exe stands out as an odd child process. In fact, the only way that powershell.exe is going to be a child process of Microsoft Office products is if a Visual Basic for Applications (VBA) script was executed from the parent process. The Lens Table at the bottom gives us the relevant information about this specific event. Namely:

1. Timestamp of when the event occurred.
2. FQDN of the system the event occurred.
3. Username (unfortunately without the domain, so we don’t know if this is a local or domain user)
4. PID
5. Commandline arguments

There was a Base64 encoded command passed to powershell.exe. As a next step, we could copy that value out to CyberChef and decode it to see what was run. Unfortunately, it is difficult to see all of the properties of this event from the threat hunting dashboard. Additionally, it would be nice to have some context about what else was happening on the system around this time. I would be interested to know what events were created by that suspicious powershell.exe process. Did it have any child processes? Did it generate any other events? We need a way to take a deep dive and investigate a single event.

## Summary

In this lab, we explored the data collected via Winlogbeat as well as the structure of the data we’ve ingested. We used that information to build a basic threat hunting dashboard that helps us understand the events taking place in our environment. We implemented the concept of frequency analysis on both parent and child process paths. We then used those fields to build queries that can be used to hunt for malicious activity. This led us to finding suspicious child processes of Microsoft Word.

While this dashboard we created is good for finding suspicious events when combined with good filters and searches, it doesn’t help us to investigate the events taking place around a single suspicious event. The purpose of the threat hunting dashboard is to analyze the entire network as a whole and find single events that are worth taking a closer look at, but it does not help us validate whether those events are actually related to malicious activity.

In the next lab, we will create an investigation dashboard that will allow us to drill down into an event in order to see the other events that were taking place on that one system in and around the time of a suspicious event.

# Appendix

## Appendix A: Event Id 4688 Fields

The table below summarizes the import fields of the Security Event Id 4688.

|  |  |  |
| --- | --- | --- |
| **Field** | **Example** | **Description** |
| winlog.event\_id | 4688 | The Windows Event Log identifier. In this case the 4688 – Process Creation event. |
| @timestamp | Jun 16, 2022 @ 13:13:11.635 | The time the event took place. |
| event.created | Jun 16, 2022 @ 13:13:13.540 | The time the EventLog entry was created. |
| metadata.ip\_address | 192.168.1.103 | The IP address of the computer at the time of the event. |
| process.pid | 652 | The ProcessId of the parent process. This field is normalized across all of the event types specified in the Winlogbeat YAML file so that it is easy for an analyst to correlate events related to the same process. Additionally, this is the **parent Process Id**, not the child, because it is the parent taking the action. In general, the process.pid field will always refer to the process that is taking the action (in the case of 4688, that action is to create a child process). |
| process.executable | C:\Windows\System32\svchost.exe | The full path to the parent process. This field is normalized across all of the types specified in the Winlogbeat YAML file that provide the full path to the parent executable. Just like with process.pid, this field can be used across multiple event types. |
| process.name | svchost.exe | The name of the parent process without the full path. This field is normalized across all of the types specified in the Winlogbeat YAML file that provide the parent executable name. The shortened version is nice because it can be used in some aggregations and displays to refer to the same family of executables that may be located in different paths (e.g. winlogbeat.exe is located in a folder that contains the version string which creates unique values for process.executable, but would have the same value for this field despite being located in different directories. |
| winlog.event\_data.NewProcessId | 0x1a60 | The ProcessId for the process being created. This is a useful field to correlate with other events. For example, we could have a separate search or visualization for this same event type where the ProcessId = 0x1a60 to find child process of this process. We can also use the NewProcessId field to correlate other events in an investigation dashboard. |
| winlog.event\_data.NewProcessName | C:\Windows\System32\wermgr.exe | The full path to the Portable Executable file that is the main module of the process. |
| winlog.event\_data.CommandLine | C:\Windows\system32\wermgr.exe -upload | The commandline arguments for the process. This typically includes the path to the process itself as the first argument. |
| winlog.event\_data.ParentProcessName | C:\Windows\System32\svchost.exe | The full path to the parent process (i.e. the process that started a child process which generated this event). |
| winlog.event\_data.ProcessId | 0x4dc | The parent ProcessId (because that’s not confusing). |
| winlog.event\_data.TargetUserName | WKST-001$ | The username for the child process (this may be different from the parent process). |
| winlog.computer\_name | WKST-001.lab.net | The fully qualified domain name of the computer that generated the record. When using Windows event forwarding, this name can differ from agent.hostname which is the fully qualified domain name of the system the event was collected from. |
| winlog.process.pid | 4 | The ProcessId of the Client Server Runtime Process (because we aren’t confused about PIDs enough already). |
| winlog.generic\_message | **Event 4688 – created-process**  (7360) C:\Windows\explorer.exe  **Event 4624 – logged-in**  User NT AUTHORITY\SYSTEM local logon success type Service login. | This field is uniquely formatted to each Event Id in order to only show the relevant information to that event so that this field can be used in a table with multiple Event Ids. |

⚠️**WARNING: The ProcessId field refers to the parent process while the NewProcessId refers to the child process. This will be important for correlations and analysis later on.**

⚠️**WARNING: The ProcessIds can be reused by the operating system, so there is no way to guarantee that two events are related by simply comparing PIDs between events. This will work most of the time for events that take place withing a short timespan, but will not work in all situations.**

**💡LEARNING POINT: The three most important fields of Event 4688 are the**  **winlog.event\_data.CommandLine field,** **winlog.event\_data.NewProcessName, and**  **winlog.event\_data.ParentProcessName fields. These fields can be used with a variety of general analytics to detect multiple types of adversary actions including initial access, enumeration, and lateral movement.**

## Appendix B: ElasticSearch Fields

The following table lists some common field types in ElasticSearch.

|  |  |
| --- | --- |
| **Type** | **Description** |
| keyword | This type is a string that is tokenized before being added to the inverted index. The tokens themselves are left “as-is”. For example, if a field has the text “The quick brown fox”, then you could search for documents where that field contains the word “fox” and Elasticsearch would return the document. Avoid using keyword fields for full-text search.  Not all numeric data should be mapped as a numeric field data type. Elasticsearch optimizes numeric fields, such as integer or long, for range queries. However, keyword fields are better for term and other term-level queries.  **Optimized for the following Operations:**   * Filtering * Sorting * Aggregations   **Example Data:**   * Email addresses * Hostnames * Status codes * Tags   **Query Examples:**  http.status:NotFound  winlog.action:create.process  process.executable:\*setup.exe  process.executable:\*powershell\*  **Does not work:**  process.executable:”C:\Windows\\*” |
| text | The text datatype is a string format that gets analyzed before being added to the index. The standard analyzer is the default analyzer which is used if none is specified. It provides grammar based tokenization (based on the Unicode Text Segmentation algorithm, as specified in Unicode Standard Annex #29) and works well for most languages.  The analysis process allows Elasticsearch to search for individual words within each full text field. Text fields are not used for sorting and seldom used for aggregations. text fields are best suited for unstructured but human-readable content.  **Query Examples:**  process.executable:”C:\Windows\\*”  process.executable:\*powershell\* |
| wildcard | Version 7.9 of Elasticsearch introduced a new field type called the “wildcard” field. Driven largely by requirements from security applications, this field is optimized for matching any part of string values using wildcards or regular expressions. This type supports a lot of the same operations as the text field; however, it is optimized for wildcard and regex searches and may be faster in some cases than text fields.  **Query Examples:**  process.child.command\_line:"C:\Windows\\*"  process.child.command\_line:\*powershell\* |
| numbers | These types store a numeric value and can be integers such as long or floating-point values such as double. This field type is good for scenarios where you want to perform some type of mathematical operation on the data or you need to search or group by ranges of values.  **Consider mapping a numeric identifier as a keyword if:**   1. You don’t plan to search for the identifier data using range queries. 2. Fast retrieval is important. term query searches on keyword fields are often faster than term searches on numeric fields. |
| date | Stores date and time information. Internally, dates are converted to UTC (if the time-zone is specified) and stored as a long number representing milliseconds-since-the-epoch. Values for milliseconds-since-the-epoch must be non-negative. Use a formatted date to represent dates before 1970.  Dates will always be rendered as strings, even if they were initially supplied as a long in the JSON document. |
| ip | An IPv4 or IPv6 address that enables searching via CIDR notation.  **Example Query:**  ip\_addr:192.168.0.0/24 |