Now that we've got Packetbeat properly configured, we should see the network data populating Kibana:

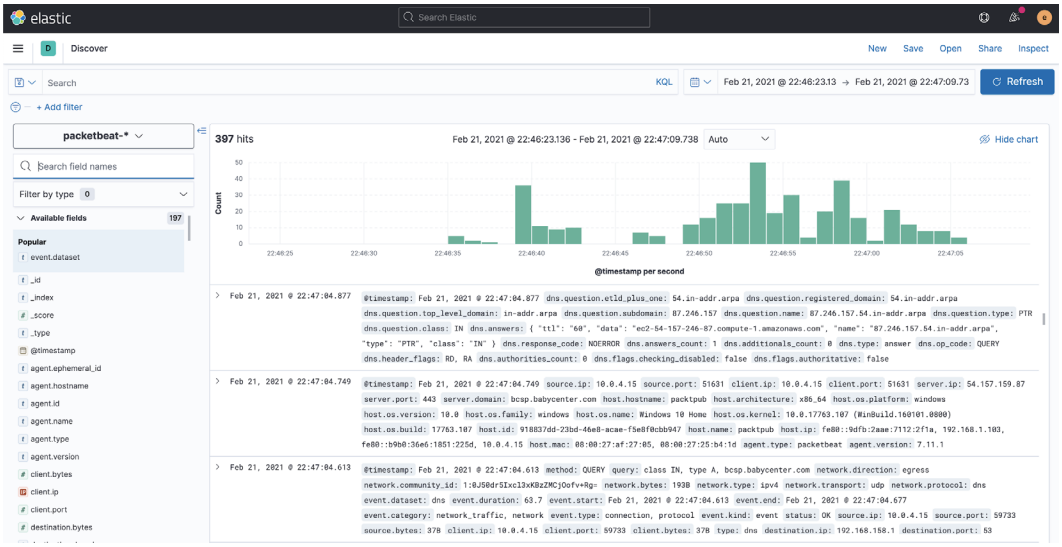

Figure 6.4 – Viewing Packetbeat logs in Kibana

Next, let's install Packetbeat as a service and then reboot:

```
PowerShell.exe -ExecutionPolicy UnRestricted -File .\install-
service-packetbeat.ps1
```

Just like with Winlogbeat, there is one final step – we need to move the KeyStore that we created into the hidden `ProgramData` directory. In an administrative terminal, copy the file there:

```
copy "C:\Program Files\packetbeat-{version}-windows-x86_64\
data\winlogbeat.keystore" c:\ProgramData\winlogbeat\
```

Now let's reboot your VM and ensure that the services are starting as expected. They are both set to a delayed start, so once the reboot is complete, you can check on the services by typing `services.msc` in the Windows search bar and then searching for the `packetbeat` and `winlogbeat` services. You will likely notice they have not started. Give it a few minutes and click the *refresh* button in the **Services** window or you can simply click the **Start** link.

Finally, check in Kibana to ensure that data is being populated in the `winlogbeat-*` and `packetbeat-*` index patterns.

In this section, we configured and installed two Beats to collect data on the Windows VM. These two Beats will provide both Windows event and application-type network data for analysis.

In the next section, we'll deploy and configure Sysmon to provide additional and detailed events for collection.

Configuring Sysmon for endpoint collection

System Monitor (**Sysmon**) is a Windows service that collects detailed events on Windows processes, services, operations, and so on. Sysmon is part of Microsoft's Sysinternals project.

Let's download Sysmon, apply a configuration, and run it as a service. First, we need to collect the Sysmon binary, so from the Windows VM, do the following:

- Download Sysmon: `https://docs.microsoft.com/en-us/sysinternals/downloads/sysmon`.
- Download SwiftOnSecurity's Sysmon config:

```
curl -OL https://raw.githubusercontent.com/
SwiftOnSecurity/sysmon-config/master/sysmonconfig-export.
xml
```

Find the `sysmon.zip` file that was downloaded, right-click it, and extract it to the `c:\Program Files` directory.

Open up a terminal window and install Sysmon as a service with the `SwiftOnSecurity` configuration. Remember you need administrator privileges, so right-click and select **Run as administrator** when you open Command Prompt (`cmd.exe`), and then type the following:

```
c:\Program Files\Sysmon64.exe -i
c:\Users\packtpub\Downoads\sysmonconfig-export.xml
```

Accept the EULA and you should have Sysmon running as a service. Just like checking Winlogbeat and Packetbeat, you can open `services.msc` to validate that `Sysmon64` is running.

Let's check to make sure that the data is getting into the Elastic Stack. Over in Kibana, go to the **Discover** tab and ensure you're on the Winlogbeat index pattern. You can scroll down the different fields or simply type `module` in the **Field Name** search bar and then click **event.module**, and you'll see the `sysmon` module is reporting to Elasticsearch:

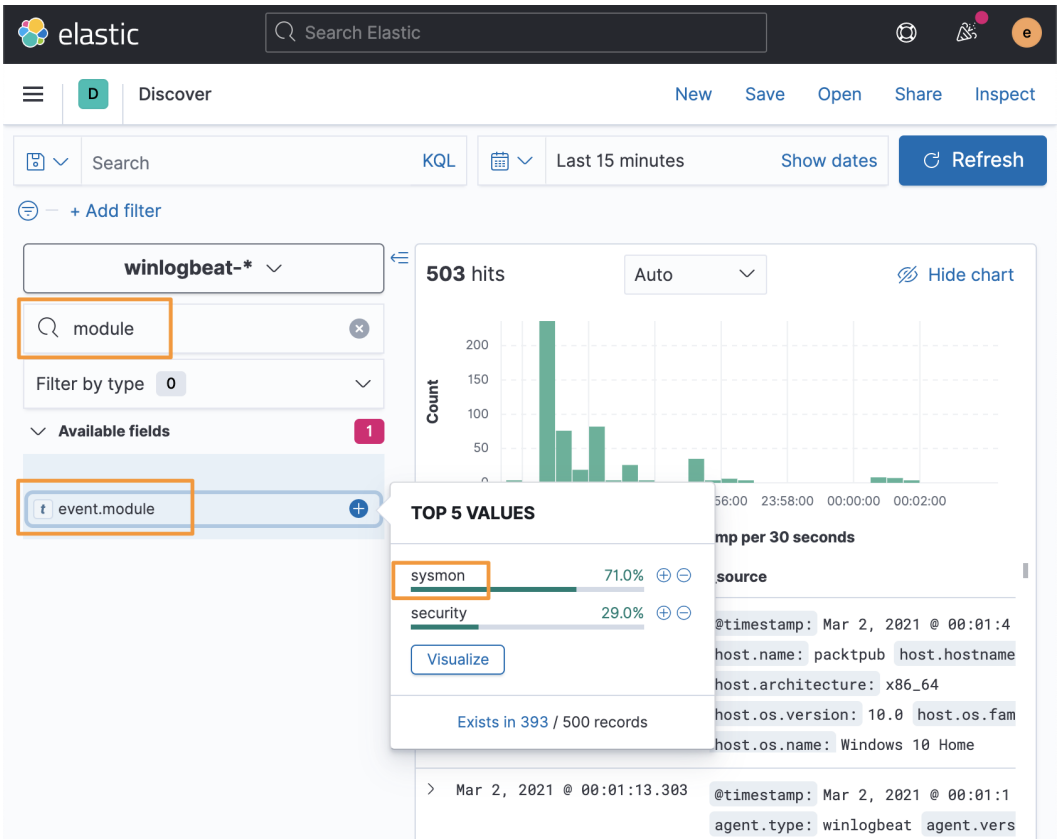

Figure 6.5 – Viewing Sysmon data in Kibana

In this section, we downloaded, installed, and configured Microsoft's Sysmon program to gain deep insights into Windows processes and services. This data is collected by Winlogbeat and shipped directly to Elasticsearch.

Next, we'll use the Fleet app to deploy and configure Elastic Agent with multiple integrations and collection policies.

Configuring Elastic Agent

For this section, we'll be spending time in the Fleet app in Kibana. As mentioned in *Chapter 3*, *Introduction to the Elastic Stack*, **Fleet** is a central management framework for Elastic Agent. Elastic Agent will manage integrations into the endpoint to collect targeted data, most specifically, security data.

A big benefit of Fleet is that as additional integrations are released by Elastic or the community, you can use Fleet to roll out these new features or make adjustments to existing ones.

To get to Fleet, click on the hamburger menu and then scroll down to **Fleet**:

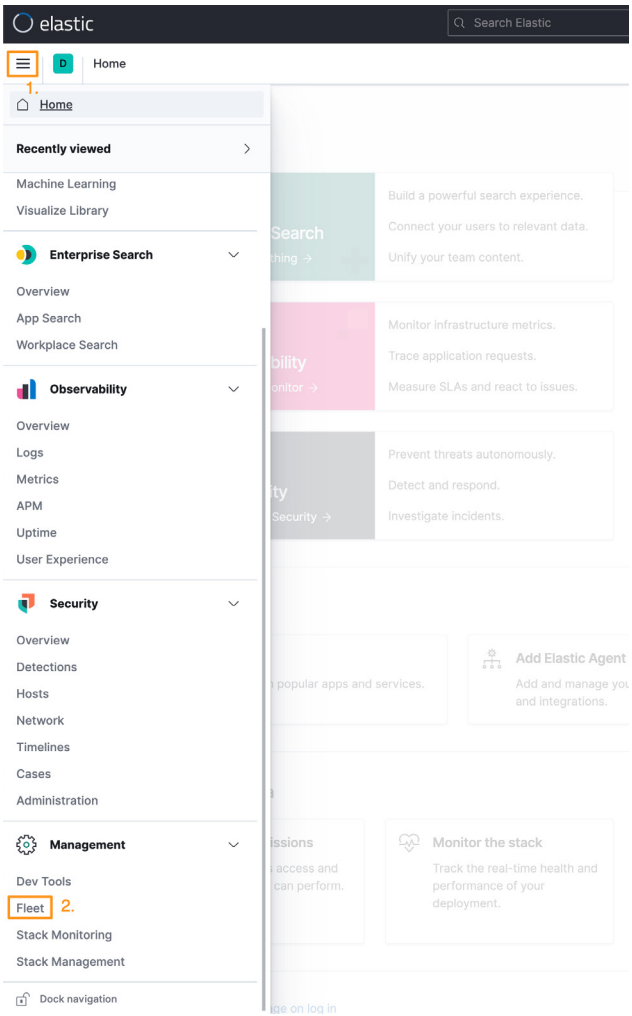

Figure 6.6 – Accessing Fleet

Once we've clicked on the **Fleet** menu option, we'll be dropped onto the Fleet **Overview** landing page:

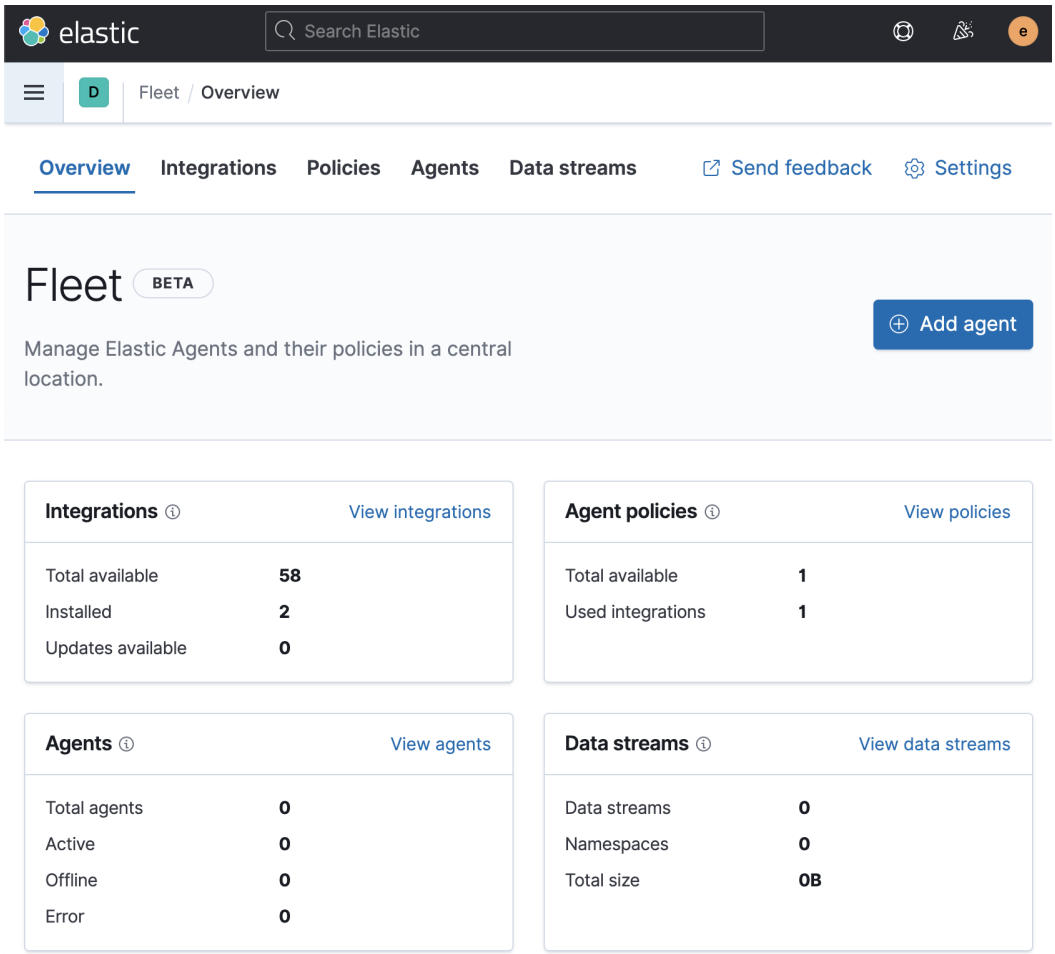

Figure 6.7 – Fleet Overview page

Click on **Policies** and then the blue **Create agent policy** button on the right. Name this policy `Windows` (or something similar) – you can also give it a description, although it isn't necessary. Uncheck **Collect system metrics**, **Collect agent logs**, and **Collect agent metrics**. These metrics and logs are unuseful, but for what we're going to be focused on, it will just be more noise. Click **Create agent policy**:

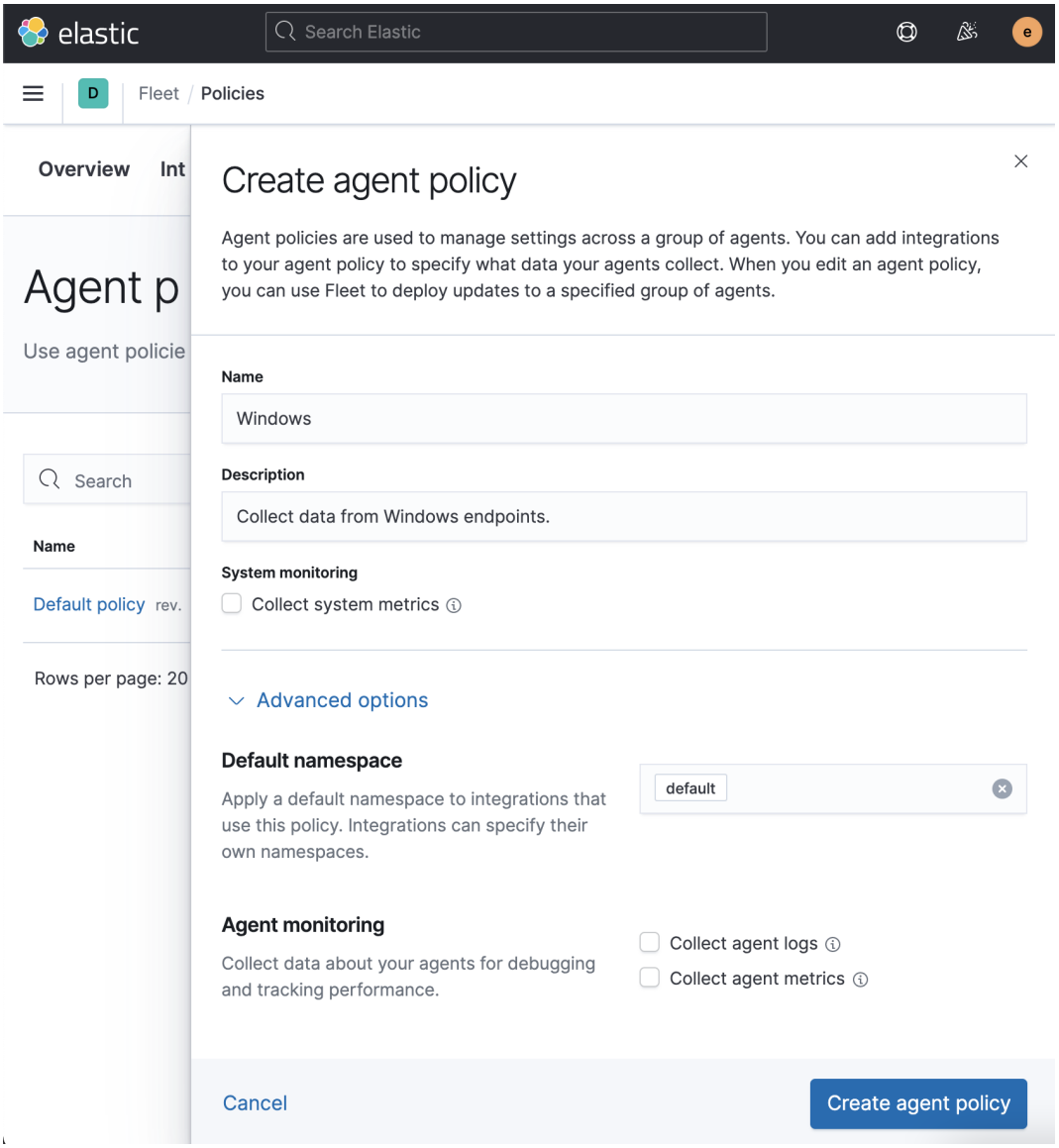

Figure 6.8 – Creating a Windows agent policy

Click on the Windows policy you just created and then click on the blue **Add integration** button in the middle of the screen. Integrations are prebuilt packages that are configured to collect and parse certain types of logs. As an example, we're going to add the Windows integration to collect and parse Windows logs.

When the **Add integration** window comes up, simply type `Windows`, select it, then scroll down and uncheck **Collect Windows perfmon and service metrics**. Click on the dropdown for **Collect events from the following Windows event log channels** to see what kinds of logs are collected. Click the blue **Save integration** button:

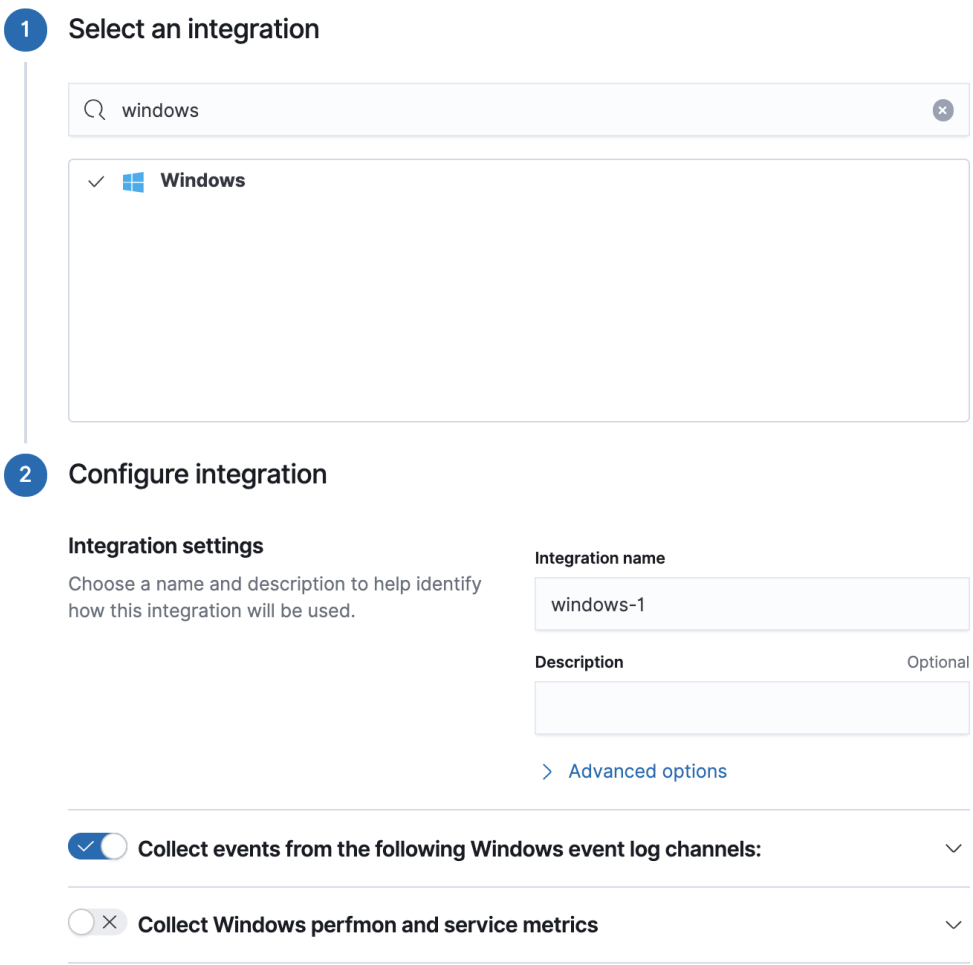

Figure 6.9 – Adding Windows integration

Repeat these steps to add the **Endpoint Security** integration to the Windows policy. At the end, you should have two integrations for the Windows policy, **Windows** and **Security**:

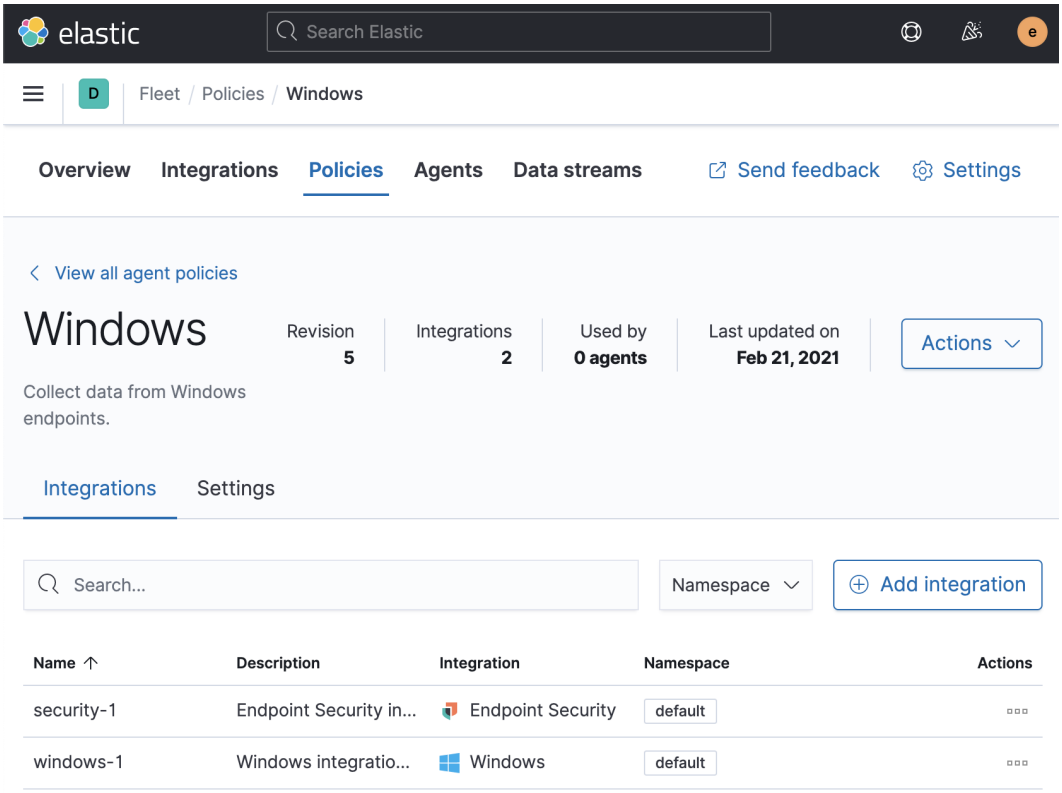

Figure 6.10 – The Windows policy with integrations

Before moving on, let's perform some customization on the security integration.

Click on the **security-1** policy, then click on the three dots next to the integration and select **Edit integration**:

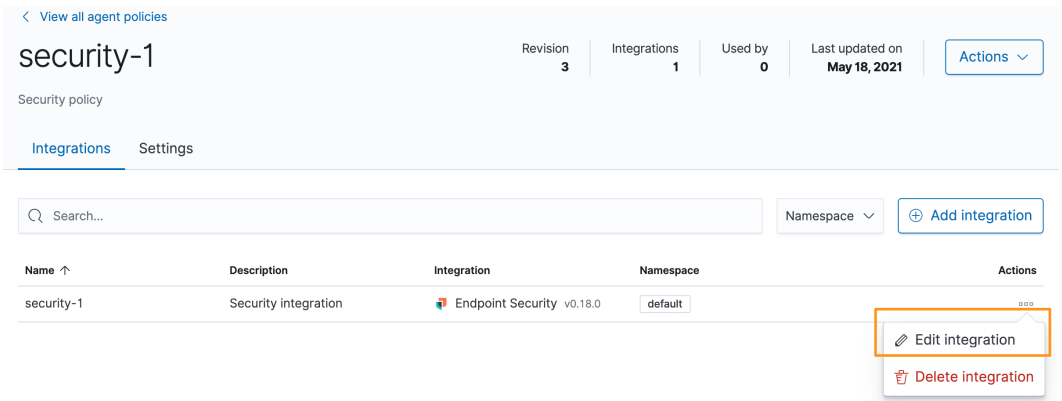

Figure 6.11 – Edit the security integration

We're going to set the **Malware** and **Ransomware Protection Level** settings to **Detect**. We're not trying to test how good the prevention features are for Elastic Agent; rather, we want malware to detonate and for us to collect data. If you use this in production environments, you'll want to choose **Prevent** for the **Protection Level** under **Malware** and **Ransomware**:

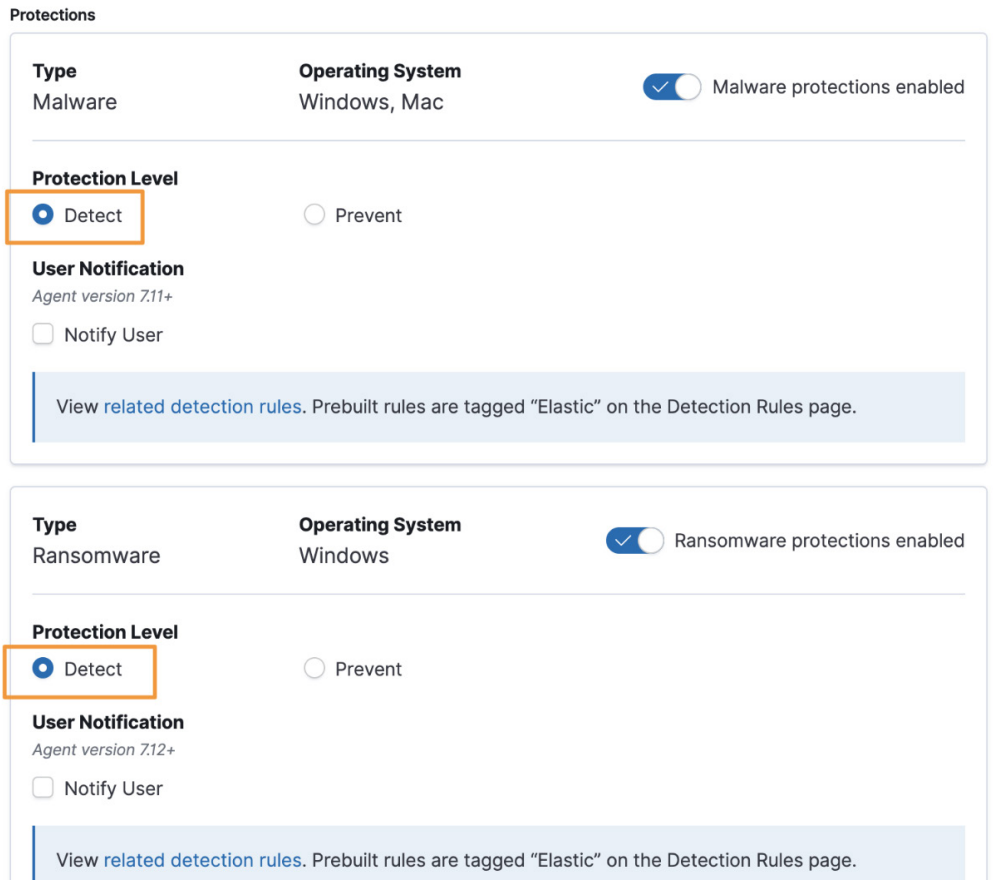

Figure 6.12 – Set Malware and Ransomware Protection Level settings to Detect

Finally, at the bottom of the security integration page, enable **Register as antivirus** so that we don't clash with Windows Defender. Click **Save integration**:

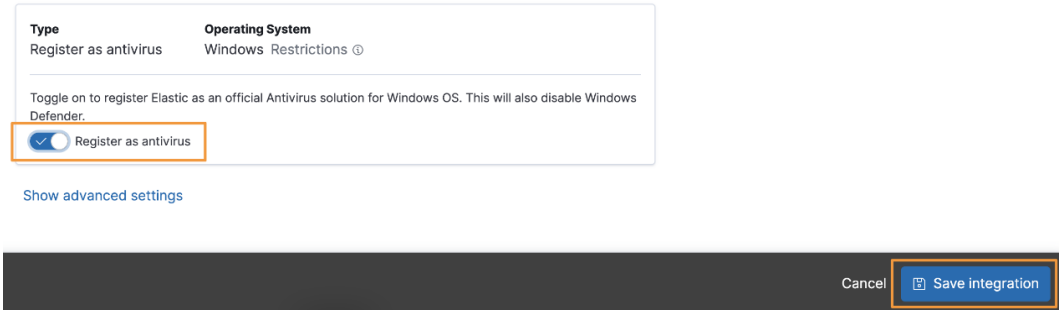

Figure 6.13 – Register Elastic Agent as antivirus

In this section, we configured the Elastic Agent policies to collect Windows and security-relevant data using Fleet. Next, we'll deploy Elastic Agent and begin to collect data.

Deploying Elastic Agent

Let's download Elastic Agent, apply a Fleet policy, and run it as a service. First, we need to collect the Elastic Agent binary, so from the Windows VM, do the following:

- Download Elastic Agent: `https://www.elastic.co/downloads/elastic-agent`.

Just like with the Beats, find the `elastic-agent-{version}-windows-x86_64.zip` file that was downloaded, right-click it, and extract it to the `c:\Program Files` directory. However, unlike Beats, we'll use an API to authenticate to the Elastic Stack instead of a KeyStore.

Before we install the agent, we need to collect the enrollment token from Kibana. So, browse to Fleet, open your **Windows** policy, click on **Actions**, and select **Add agent**:

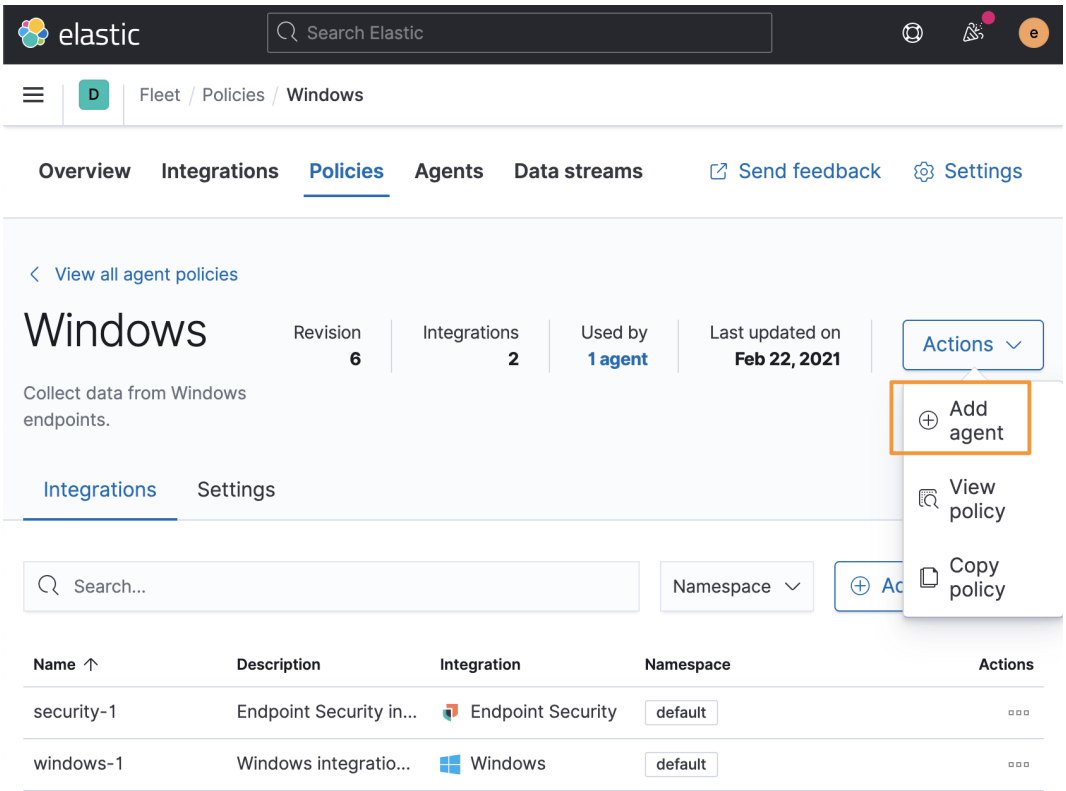

Figure 6.14 – The Add agent option in Fleet

Scroll to the bottom on the slide-out window and there will be a **copy** button for Windows. Select that to copy the installation/enrollment command to your clipboard. Ensure that the `--url` switch has the proper IP address of your intnet IP. If it does, go back to the *Enabling Detection Engine and Fleet* section in *Chapter 5*, *Building your Hunting Lab – Part 2*, and ensure you have properly configured your Fleet settings:

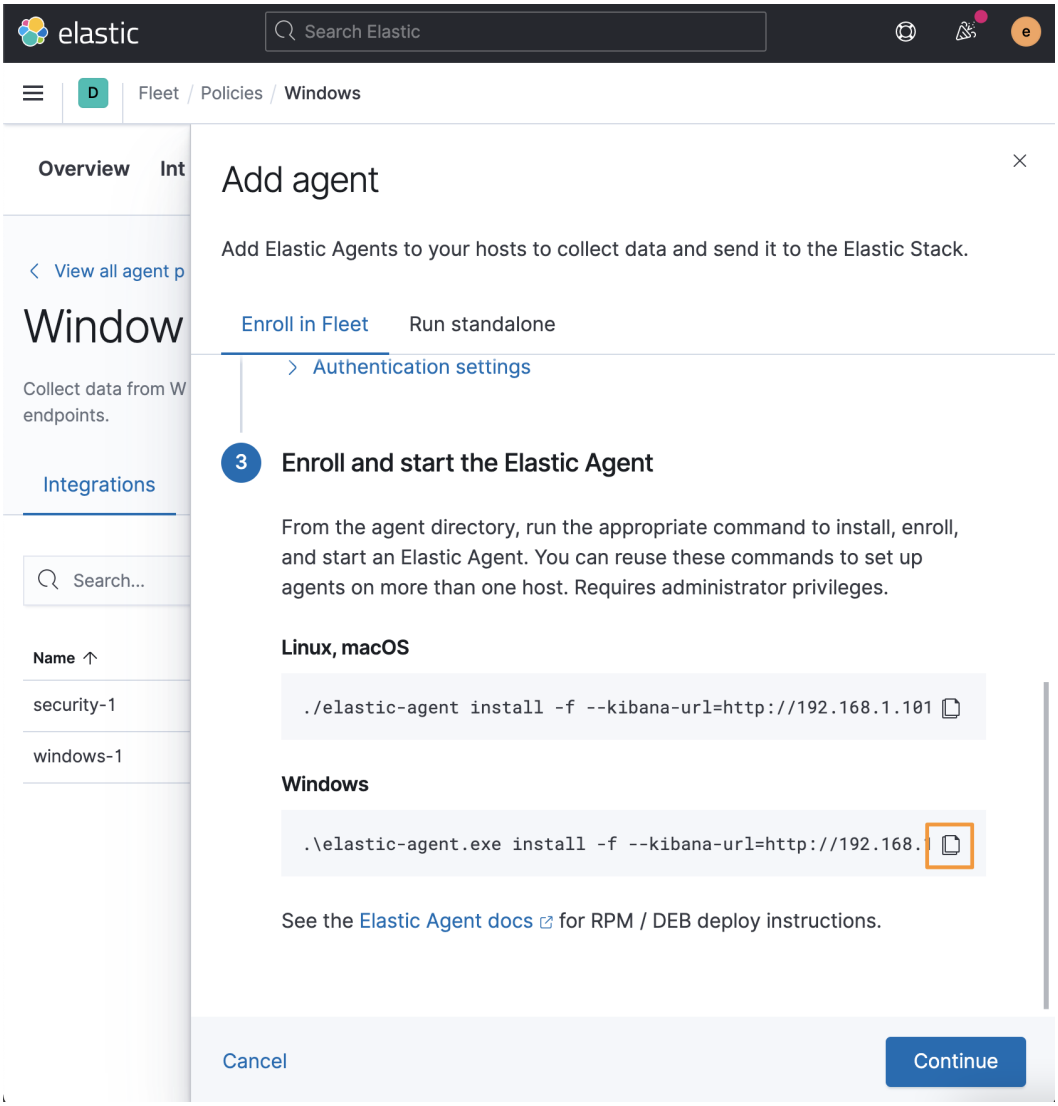

Figure 6.15 – Agent installation and enrollment command

Back on the Windows VM, once you're extracted the ZIP archive to `c:\Program Files\elastic-agent-{version}-windows-x86_64`, browse into that directory and run the installation and enrollment script you copied from Fleet (your enrollment token will be different). Remember to add `--insecure` because we're not using TLS. If you forget, it will give you a reminder (your IP address should be your intnet IP and your token is auto-generated, so it may be different than the following example, but the **copy** button will populate it with all of the proper information for your instance):

```
.\elastic-agent.exe install -f --url=https://172.16.0.3:8220
--insecure –enrollment-token=Vk80cXlIY0IyTG9wSmVOU0NnOU86ckloUj
dUVlpTOFdPVm1WNjJTbU54UQ==

The Elastic Agent is currently in BETA and should not be used
in production

Successfully enrolled the Elastic Agent.
Installation was successful and Elastic Agent is running.
```

Now, back in Kibana in the **Agents** tab, you should see your host reporting into Elasticsearch:

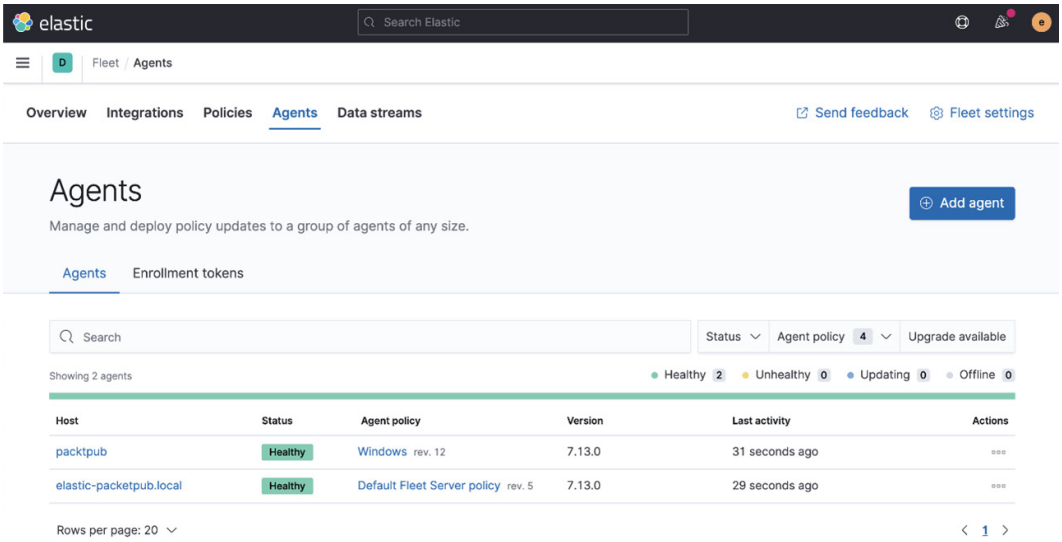

Figure 6.16 – Successful reporting of Elastic Agent into Fleet

Finally, head over to the **Discover** tab and ensure that you're seeing data. In the `logs-*` Index Pattern, you should see data:

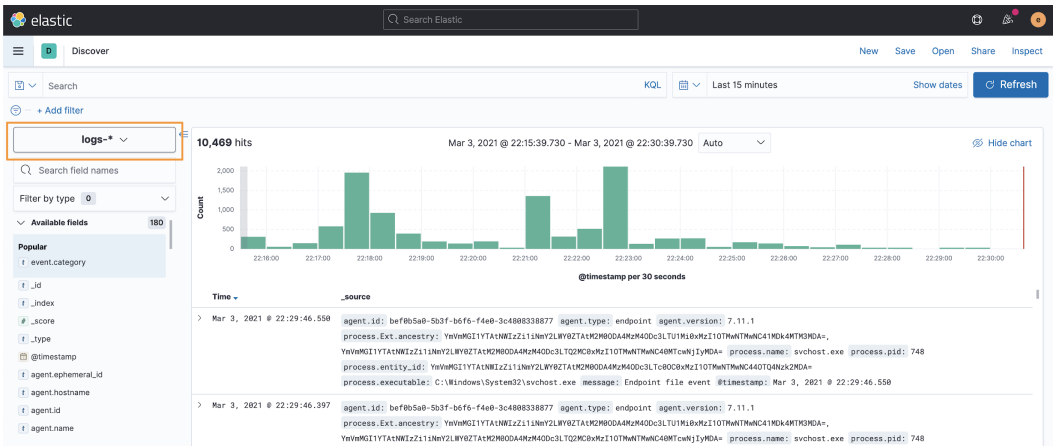

Figure 6.17 – Successful data from Elastic Agent

Now that we have Elastic Agent configured, installed, and enrolled in Fleet, we're ready to start collecting data for threat hunting!

Summary

We built on the hunting lab that you created in the previous chapter in preparation for wading into learning how to perform searches and create visualizations and dashboards in Kibana.

In this chapter, you learned how to configure the collection agents and tools you installed in *Chapter 4*, *Building Your Hunting Lab – Part 1 and Chapter 5, Building Your Hunting Lab – Part 2*. Additionally, we covered the configuration of Fleet used to manage Elastic Agent. This knowledge will help you not only maintain and adapt your collection policies going forward in your lab, but also in production environments.

In the next chapter, we'll be learning how to use the **Kibana Query Language** (**KQL**) and the **Event Query Language** (**EQL**) to perform focused searches on our data.

Questions

As we conclude, here is a list of questions for you to test your knowledge regarding this chapter's material. You will find the answers in the *Assessments* section of the *Appendix*:

1. Winlogbeat is used to collect what kind of data?
   a. Windows event data
   b. Windows performance metrics
   c. Metrics about Beats installed on Windows systems
   d. Windows network information
2. Packetbeat is used to collect what kind of data?
   a. Packet captures
   b. Network traffic between Kibana and Elasticsearch
   c. Application-type network events
   d. Network performance metrics
3. What is the central management app for Elastic Agent?
   a. Fleet
   b. Beats Central Manager
   c. Group Policy
   d. System Center Configuration Manager
4. What are additions to Fleet policies called?
   a. Modules
   b. Plugins
   c. Inputs
   d. Integrations
5. Which of the following Beats reports Sysmon events as a module?
   a. Elastic Agent
   b. Winlogbeat
   c. Packetbeat
   d. Auditbeat

Further reading

To learn more about the subjects we covered in this chapter, check out the following links:

- Winlogbeat: `https://www.elastic.co/guide/en/beats/winlogbeat/current/index.html`
- Sysmon: `https://docs.microsoft.com/en-us/sysinternals/downloads/sysmon`
- Packetbeat: `https://www.elastic.co/guide/en/beats/packetbeat/current/index.html`
- Elastic Agent: `https://www.elastic.co/guide/en/fleet/current/elastic-agent-installation-configuration.html`

7

Using Kibana to Explore and Visualize Data

So far, we've spent a great deal of time introducing you to the various parts of the Elastic Stack and building infrastructure that will be used to create and collect data for analysis. In this chapter, we'll learn how to navigate inside the Discover app, spend time exploring data using different types of query languages, create visualizations that facilitate the presentation of our data in a hunting context, and finally, arrange those visualizations onto dashboards to help organize our hunting methodologies.

In this chapter, you'll learn how to create queries, saved searches, visualizations, and dashboards throughout the Kibana dashboard. These skills will be built upon as we continue to dig deeper into the Elastic Security solution.

In this chapter, we'll take a look at the following topics:

- The Discover app
- Query languages
- The Visualization app
- The Dashboard app

Technical requirements

In this chapter, you will need to have access to the following:

- The Elastic and Windows **virtual machines** (**VMs**) that were built in *Chapter 4*, *Building Your Hunting Lab – Part 1*
- A modern web browser with a UI

The code for the examples in this chapter can be found at the following GitHub link: `https://github.com/PacktPublishing/Threat-Hunting-with-Elastic-Stack/tree/main/chapter_7_using_kibana_to_explore_and_visualize_data`.

Check out the following video to see the Code in Action:
`https://bit.ly/3ikQ827`

The Discover app

The **Discover app** is the main view of your data stored in Elasticsearch. If you had no other solutions, visualizations, or dashboards, you could still explore all of your data with Discover.

Using Discover, we'll learn how to leverage the true strength of Kibana – filters. Raw searching capability is, of course, very powerful. However, in threat hunting, frequently, you don't exactly know what you're looking for, so simply blindly searching through data will result in suboptimal results (if any). Filters, as we'll discuss in more detail, allow you to surgically examine data to discover what's hidden inside.

> **Important note**
>
> The Discover app only shows the first 500 events for a search. This is for performance. We will use time selections and filters to uncover and zero in on the data of interest. You can't efficiently search through more than 500 events, so while it can be confusing (or frustrating) at first, this truly makes sense and forces you to use the full power of Kibana.

So, let's get right into it. Fire up your Elastic and victim VMs. When logging into Kibana, you'll land on the home screen with tiles for the three solutions we discussed in *Chapter 3*, *Introduction to the Elastic Stack*. Instead of getting into those yet, let's go straight to the heart of it all, Discover.

Click on the **hamburger menu** in the upper-left corner of your screen and then select **Discover** underneath the **Analyze** header. There is a lot in **Discover**, but most of it is fairly self-explanatory. We'll walk through it all and point out how to utilize it to search through your data:

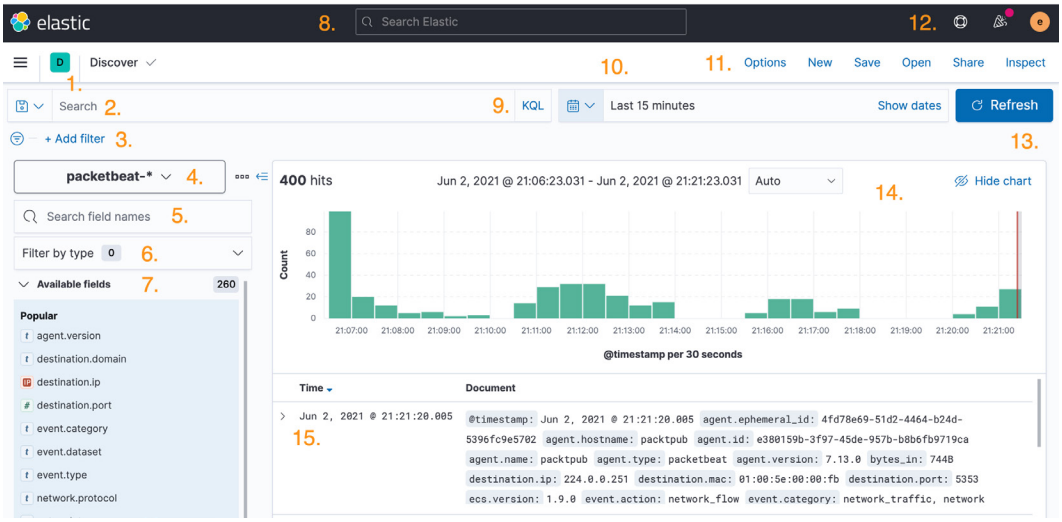

Figure 7.1 – The Discover app

In the **Discover** app, we'll explore these functions:

1. The spaces selector
2. The search bar
3. The filter controller
4. The Index Pattern selector
5. The field name search bar
6. The field type search
7. Available fields
8. The Kibana search bar
9. The current query language selector
10. The date picker
11. The Action menu
12. Support information
13. The search/refresh button
14. The timebox
15. The Event view

In the following sections, we'll discuss each of these in detail.

The spaces selector

Spaces organize saved objects, such as saved searches, visualizations, and dashboards, for specific use cases and teams. What makes spaces valuable beyond their simple organization is that you can apply entitlements, or in-app permissions, to specific spaces.

For example, if you wanted to create a space for an engineering team, a security operations team, a helpdesk, or a hunt team, you could do so in a way that the helpdesk would only have access to objects needed to do their job rather than everything.

While the spaces selector isn't an access tool (that is, it doesn't control who can log in to Kibana), it leverages entitlements so that you can create a unique experience for different users/teams.

The search bar

The **search bar** is exactly what it sounds like; it's where you will perform searches against your data. We will spend a lot of time with this as we explore query languages later in the chapter.

The filter controller

Normally, when you create a filter, you do so within your data after performing a few searches and deciding what you want to filter in or out. Using the **Filter Creator**, you can apply a filter without having to search through your data first.

For example, if we knew we only wanted to see TLS network data, we could simply create a filter for that network protocol by clicking on **Add filter** and defining the field and value.

Once you have applied a filter, you can interact with it by clicking on the filter:

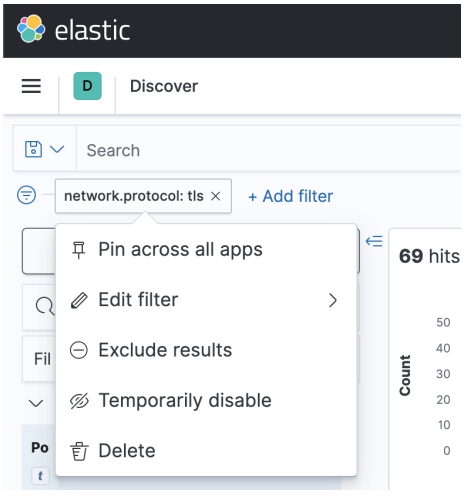

Figure 7.2 – Interacting with an applied filter

You can *pin* filters across apps. This is helpful when you've got specific filters applied to a search and want to jump to another app. Pinning them will take them with you as you move between apps.

You can manually edit the filter to change the **field name**, if it is or isn't present, or change what data is currently inside the field.

Additionally, you can include or exclude results. This action allows you to invert the filter.

You can also disable the filter.

Finally, you can simply delete the filter.

Additionally, by clicking on the inverted triangle, we can make changes to all of the filters at once. Interacting with applied filters is a powerful technique when it comes to sorting and sifting through data.

The Index Pattern selector

Clicking on the **Index Pattern** selector will allow you to change between your available index patterns. If you remember from *Chapter 3*, *Introduction to the Elastic Stack*, index patterns allow you to select what data you're going to search (or define properties for).

If you click on the Index Pattern selector, you will see several index patterns:

- **Logs-***: This is network and event data from the Elastic endpoint.
- **Metrics-***: This is metric data.
- **Packetbeat-***: This is network data from Packetbeat.
- **Winlogbeat-***: This is Windows event data from Winlogbeat.

We'll be using these index patterns in the future to define what buckets of data we'll be searching through.

The field name search bar

The **field name** search bar allows you to search for specific field names without having to dig through your data to find them. From here, you can also click on the field to view the top 5 events, what percentage these events make up, and how many of the top 500 events have these top 5 fields:

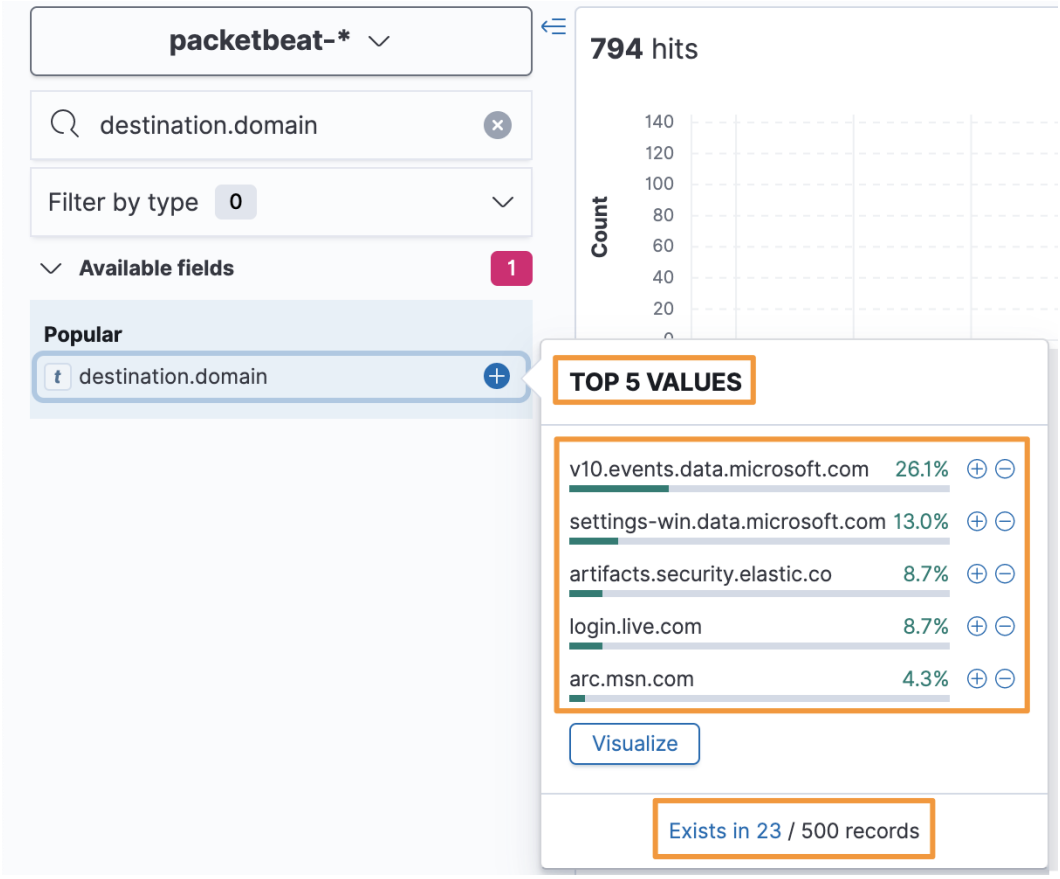

Figure 7.3 – The field name search bar

In the preceding screenshot, in the `packetbeat-*` index pattern, I searched for the `destination.domain` field and clicked on it. Now I can view how the top 5 values are represented and that the `destination.domain` field is available in 23 of the first 500 records.

Additionally, you can click on the `+` or `-` buttons next to the fields to filter in or filter out that field.

The field type search

This function continues along the path that allows you to apply filters without having to search through your data first. This allows you to search for specific field types (for instance, IP addresses, dates, and whether they're aggregatable), but I honestly don't know whether I've ever used it for anything beyond curiosity.

Available fields

This section shows you the available fields. This will update as you apply field names to the **Field Name** search bar.

Clicking on the + button next to a field allows you to add it as a column in the Event view. This field will be displayed even if there is no data. So, as we saw in the preceding example with the `destination.domain` field, that field is only populated in 23 of the first 500 events; the `destination.domain` field will simply show a -:

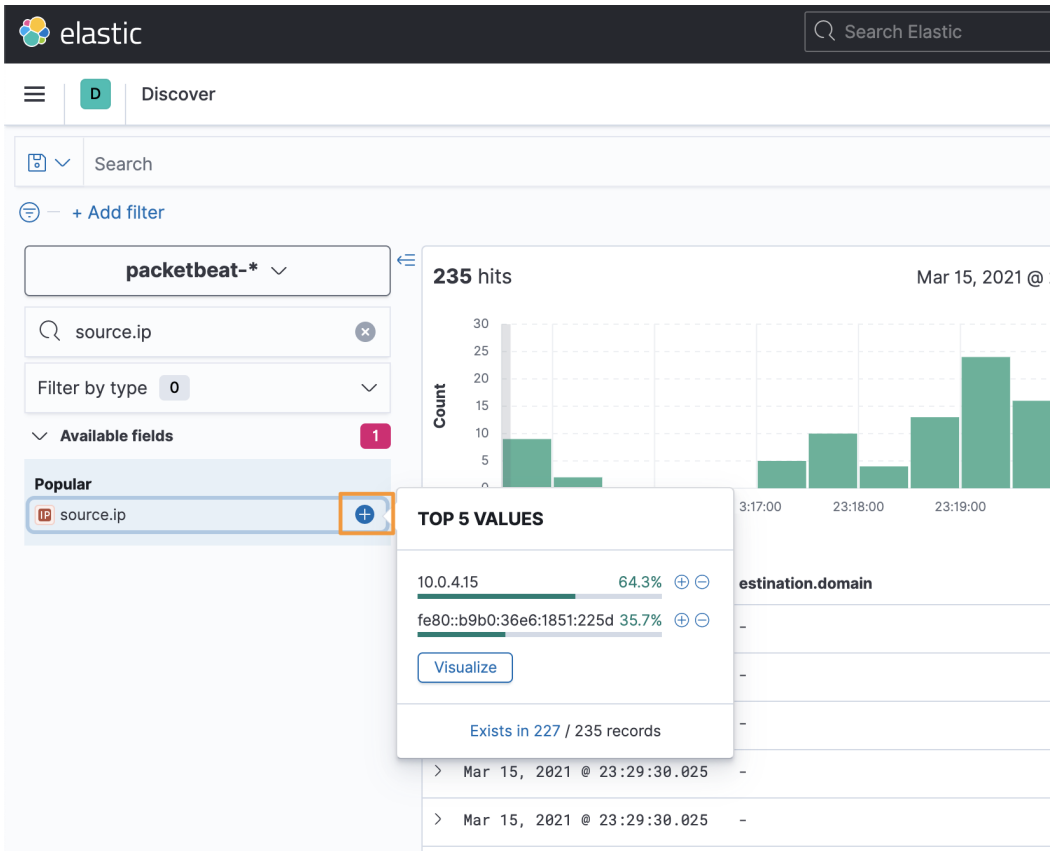

Figure 7.4 – Adding columns to the Event view

If we compare the `destination.domain` field to the `source.ip` field, we can see that the data is displayed:

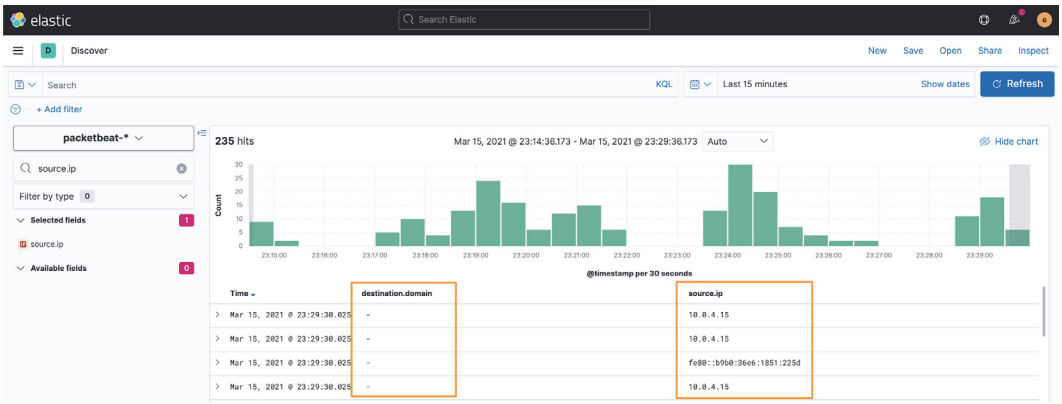

Figure 7.5 – Fields with and without data

As we discussed at the beginning of the section, filters are what make Kibana powerful, and we'll be using these fields to their full advantage as we continue.

The Kibana search bar

Using the **Kibana** search bar, you can search for solutions, apps, and configurations from anywhere in **Kibana**. For example, if you want to go directly into **Fleet**, you can type `Fleet` into the **Kibana** search bar and go directly there:

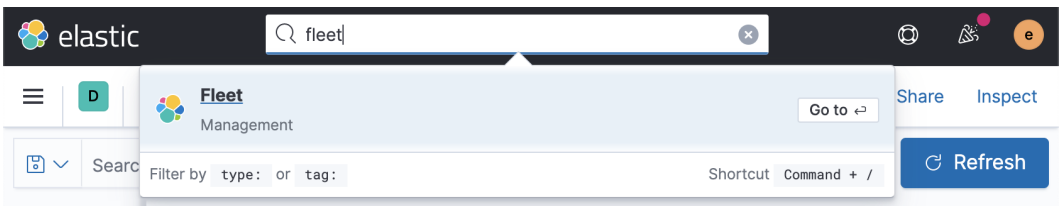

Figure 7.6 – The Kibana search bar

This is a great time-saver when hopping between apps and solutions.

The query language selector

Using the **Query Language Selector**, you can change between the **Kibana Query Language** (**KQL**) and Lucene. The current query language is displayed at the end of the search bar. We'll discuss Lucene and KQL in the *Query languages* section.

The date picker

Using the **Date Picker**, you can select an absolute time (for example, March 15, 2021, 12:00:00:001), a relative time (17 minutes ago), or a series of commonly used times such as *Last 15 minutes*, *Today*, or *Last 24 hours*. By default, Kibana converts the event timestamp into the local time.

One of the most common issues with searches not displaying any results when there should be data is that the **date picker** is set to the wrong date/time. When I'm not seeing data that I'm expecting, I usually go big by setting the date to *Last 90 days* (or something similar) just to make sure there isn't a date issue that's preventing me from getting results.

The Action menu

From the **Action** menu, we can create a new search, save a search, open a saved search, share a search, and inspect a search.

Creating a new search

Clicking on the **New** button resets you to your default index pattern, removes any queries, removes all of the added columns, and removes all of the unpinned filters. It will not reset your **Date Picker** or any pinned filters.

Saving a search

Clicking on the **Save** button allows you to save your current search to include filters. This does not include the **Date Picker**. We'll spend more time with saved searches in the *Visualize app* section of this chapter.

Opening a saved search

Clicking on the **Open** button allows you to open any saved search. You'll notice you have several saved searches already. These were loaded when we ran the setup for the different Beats in *Chapter 5*, *Building your Hunting Lab – Part 2*.

Share

Share is a very powerful action item that allows you to share your current search with others. An extremely helpful feature is that you can use the **Short URL** toggle to make the URL more manageable. Shortened URLs allow you to take large and complex URLs and shorten them into manageable links. Here are some examples of unshortened and shortened URLs.

An example of a shared URL (unshortened):

```
http://172.16.0.3:5601/app/discover#/?_g=(filters:!(),refre
shInterval:(pause:!t,value:0),time:(from:now-15m,to:now))&_
a=(columns:!(destination.domain,source.ip),filters:!(),index:'
packetbeat-*',interval:auto,query:(language:kuery,query:''),so
rt:!())
```

An example of a shared URL (shortened):

```
http://172.16.0.3:5601/goto/3b92487c8016b06d1f61b4e1fb442fc0
```

As you can see, in the shortened URL, it is much easier to share tickets, cases, and communications.

The **Share** URL includes the index patterns, filters, columns, and time.

Inspect

The **Inspect** menu is extremely useful when you're trying to understand more about how queries are being sent to and from **Elasticsearch**.

While this has its place and can be used for some advanced functionality, it is beyond the scope of this section.

Support information

This will let you know what version of Kibana you're currently using. This is helpful when tracking the features and capabilities of the Kibana project and if you need support.

The search/refresh button

This button is used to run a query or rerun a query. One example where you'd rerun a query is if your **Date Picker** is set to *Last 15 minutes* and you want to rerun for the current 15-minute time window or if you wanted to check for new data.

Applying filters does not require a refresh.

The timebox

The **Timebox** shows a chart of your data and how much data is in each time interval:

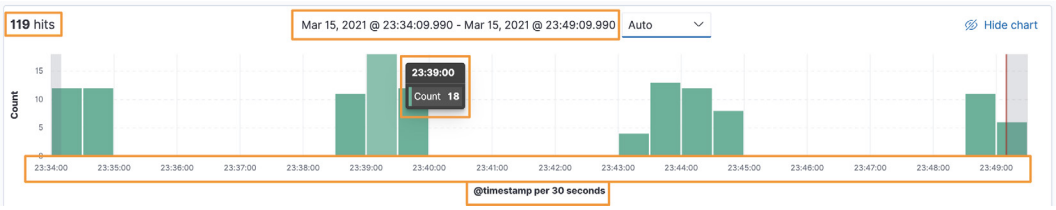

Figure 7.7 – The timebox

In the preceding screenshot, we can view exactly how many events happened in our time window (*119*), the time range of *Mar 15, 2021 @ 23:34:09.990 - Mar 15, 2021 @ 23:49:09.990*, how many events happened in each time window (*18*), the time scale, and how much time is in each column (that is, *30 seconds*).

You can click and drag inside the **Timebox** to zoom in on certain time dates or even just click on a column to look at the data in that column. This changes the **Date Picker** to exactly the time range you're looking at. You can also change it back by using the **Date Picker**.

The Event view

Finally, the events!

When you look at the **Event view**, you're seeing a collapsed view of the fields and data that are in your event. To expand this and view all of your data, click on the > button:

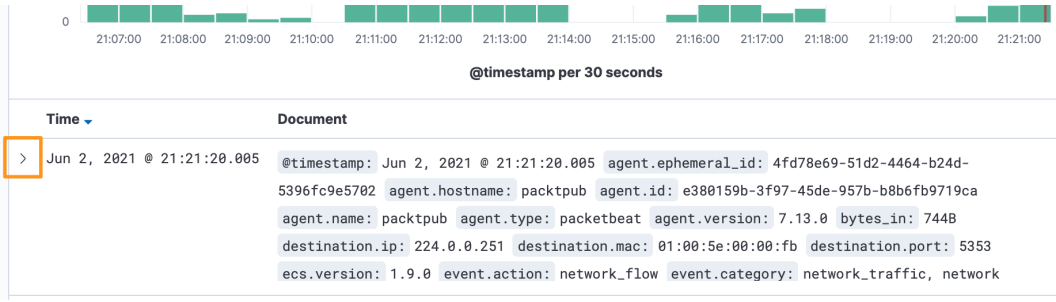

Figure 7.8 – The Event view

Once you've expanded your view, you can see all of the fields and data that are in each event. Imagine digging through 100, 500, or 1,000 events like this – no, thank you! As we discussed at the beginning of the section, Kibana only displays the first 500 records, and we use filters to zero in on the data that we're interested in.

Applying filters before searching through our data is great when you know what you're looking for; however, sometimes, that's not the case. Once we've expanded an event, we can add fields to the columns for analysis or even apply filters to the data by clicking on the + or – buttons:

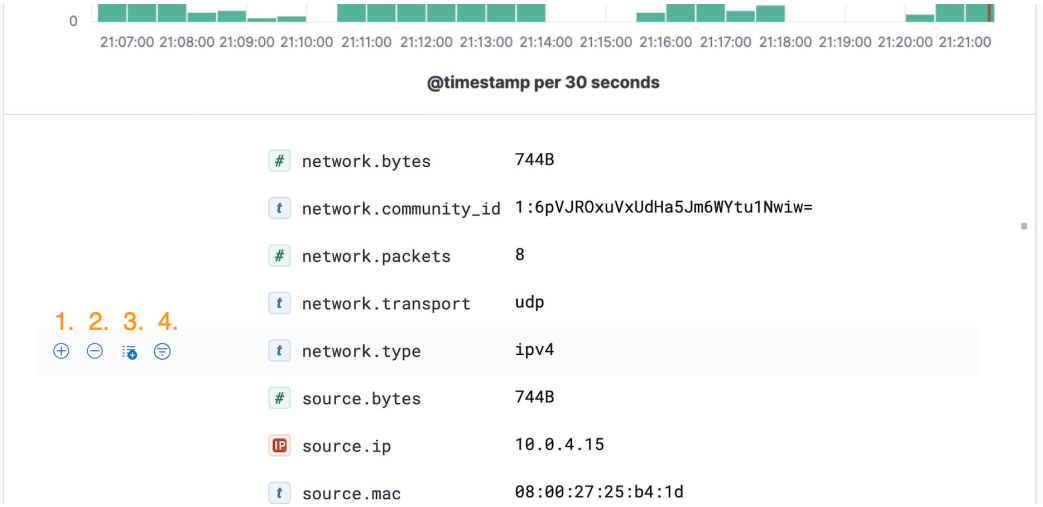

Figure 7.9 – The Event view – the filter application

To apply filters and columns to data from within the Event view, we can simply use the buttons next to the data:

1. Clicking on the **+** button will add this field and data as a filter, meaning that it will show all of the events that have `network.type` as `ipv4`.
2. Clicking on the **–** button will remove this field and data as a filter, meaning that it will show no events that have `network.type` as `ipv4`.
3. Clicking on the **Column** icon will add this field and data as columns to the **Event view**.
4. Clicking on the inverted triangle will create a filter to only display results that have data in the `network.type` field – irrespective of the data in that field.

We've spent some time exploring Discover. The best way to get good at searching in Discover is to practice. You don't need special data, you don't need specific labs, and you don't need to have an actual goal – simply interact with a rich dataset (like the one we generated with our victim machine) and see where your curiosity takes you.

Exercise

As I mentioned earlier, you don't need a specific dataset or a goal to get good at Discover, but let's set up some objectives to help get you going. This is all going to be done without a single search – just using filters.

Use any method you want to complete this, but try to use the **Filter Controller**, **Field** search bar, and the **Event view** at a minimum. Remember to adjust your **Date Picker** to a large window to make sure your victim machine has sent the type of data we're looking for.

Use the following elements to create a view in the Discover app:

- Select the `packetbeat-*` index pattern.
- Create a filter to display events where the `network.protocol` field has a value of `tls`.
- Create a filter to display events where the `destination.domain` field exists.
- Create a filter to display events where the `tls.server.x509.issuer.organization` field has a value of `Microsoft Corporation`.
- Add the following columns to the Event view:

  `source.ip`

  `destination.domain`

  `destination.port`
- Create a shortened URL of this search.
- Save this search as *Chapter 6 - Discover App Intro*.

Using the preceding elements, we can view how the Discover app will display our data:

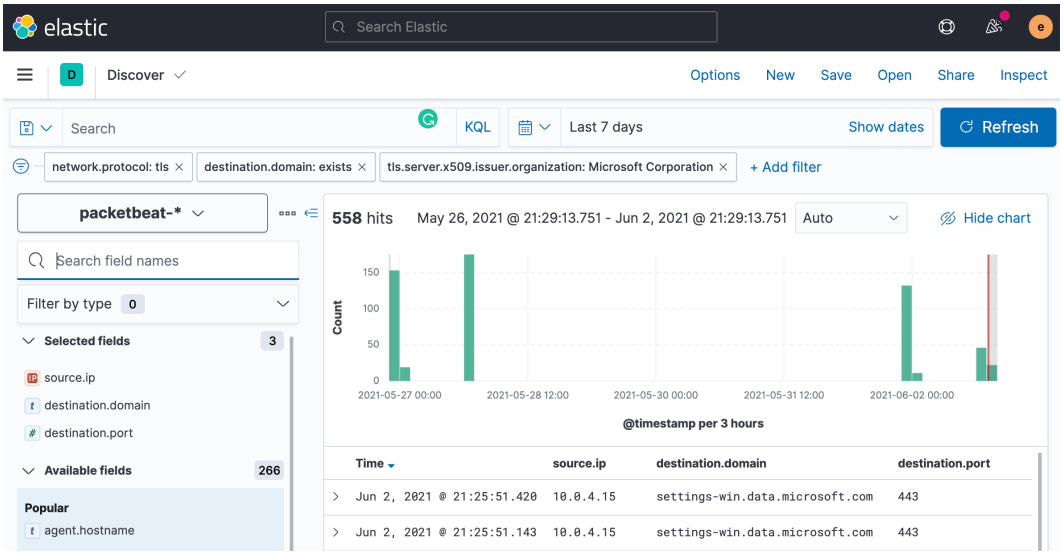

Figure 7.10 – Exercise results

Next, create three additional saved searches in the **Packetbeat** index pattern with the following information:

- Save this as *Chapter 6 - HTTP Traffic* with the following filters:

  `event.category`: network

  `event.type`: connection

  `event.kind`: event

  `network.protocol`: `http`
- Save this as *Chapter 6 - TLS Traffic* with the following filters:

  `event.category`: network

  `event.type`: connection

  `event.kind`: event

  `network.protocol`: `tls`
- Save this as *Chapter 6 - DNS Traffic* with the following filters:

  `event.category`: network

  `event.type`: connection

`event.kind`: event

`network.protocol`: `dns`

We will be using the three protocol saved searches later in the chapter in order to create visualizations and dashboards.

In this section, we explored the various parts of the Discover app. While there is a tremendous amount of information that is presented in Discover, once we get into it, the interface is fairly intuitive and easy to approach.

Additionally, we learned the power of filters and how to explore a large amount of our data without ever writing a query. Just imagine how much your skills will accelerate once we actually search for data instead of simply carving up the data we're presented with.

Next, we'll move on from applying filters to writing queries with Lucene, KQL, and the **Event Query Language** (**EQL**).

Query languages

Within Kibana, we can use one of three languages to query our data – with those being Lucene, KQL, and the EQL.

As mentioned in *Chapter 3*, *Introduction to the Elastic Stack*, Elasticsearch is built upon Lucene, which is a search engine library written in Java. However, before we dive too deeply into Lucene, it should be noted that this language is generally unused in newer versions of Kibana barring a few exceptions, notably, when searching using a **regular expression** (**regex**). A regex is written to identify specific characters in a string. They can be simple searches, such as finding a specific word or phrase, or more complex searches, such as finding the sixth word of a sentence but only if the sentence starts with the word "The" and ends with "?".

Because of this, we'll discuss Lucene in a bit more detail and explore a useful threat hunting example using regex. However, please note that we'll be using KQL for almost all of our threat hunting.

Let's start up both our Elastic and victim VMs and ensure that the event data is being reported into Elastic. As mentioned earlier, you can check to make sure that your ecosystem is properly functioning by simply going into the Discover app in Kibana and validating that you can see the current data from the victim machine in the following index patterns:

```
logs-*, packetbeat-*, winlogbeat-*.
```

Let's generate some network traffic that we can make queries against in Kibana. Go into your victim VM, open up Command Prompt (`cmd.exe`), and use cURL to generate some traffic. If you prefer to use a web browser (for example, Edge) on the victim machine, you can do that as well:

```
$ curl -L https://packtpub.com
$ curl -L https://elastic.co
$ curl -L http://neverssl.com
$ curl -Lk https://neverssl.com
```

Once we have the stack up and running and have generated some network data, we can begin by writing some basic queries in Lucene and then moving on to KQL and EQL to implement more advanced searches.

Lucene

The Lucene search syntax is not vastly different from the majority of other search languages in that you define a field, a value, and then chain these searches together.

For example, if we wanted to search for the source IP address of `10.0.4.15`, in Lucene, we'd search the following:

```
source.ip : 10.0.4.15
```

The results should only show us network traffic that has the source IP address of `10.0.4.15` in the event. That's pretty self-explanatory, right?

However, what if we wanted to only see network traffic that has a source IP of `10.0.4.15` and a destination port of `80` or ports `80` and `443`? In this scenario, we can simply chain the searches together.

Using your host machine, open up Kibana by browsing to `http://localhost:5601` and logging on with the `elastic` username and passphrase. Let's go to **Discover** and select the `packetbeat-*` index pattern so that we can search the network data we just created.

Kibana selects **KQL** as the default syntax for queries. This can be temporarily set to **Lucene** in **Discover**, or we can set it permanently in the **Advanced Settings** of Kibana. For now, we're just going to set it temporarily. So, click on the **KQL** hyperlink and set the **Kibana Query Language** toggle to **Off**:

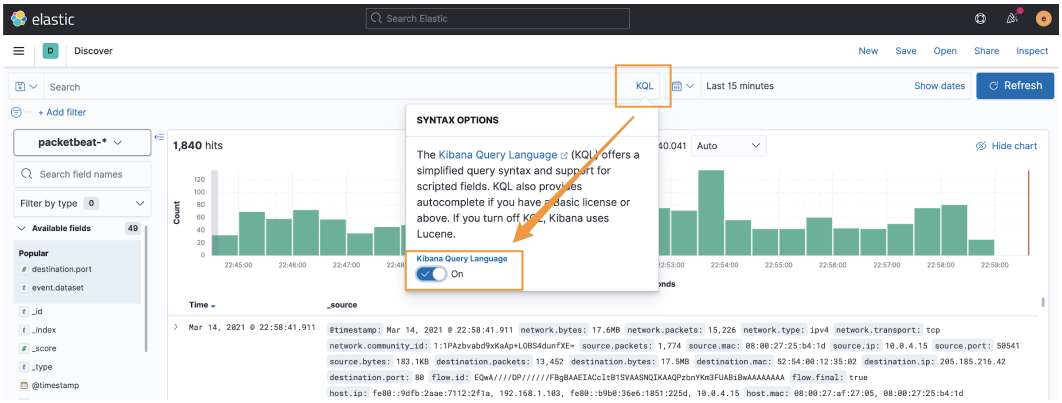

Figure 7.11 – Setting the search syntax to Lucene

Next, let's explore what kinds of source IP addresses we have in our dataset. In the **Search for field names** bar, type in `source.ip`. The `source.ip` field will be displayed underneath **Available fields**. If you click on `source.ip`, you should see a handful of IP addresses. You might have different IP addresses than I do, but the concept is the same; you should have multiple source IP addresses in your dataset:

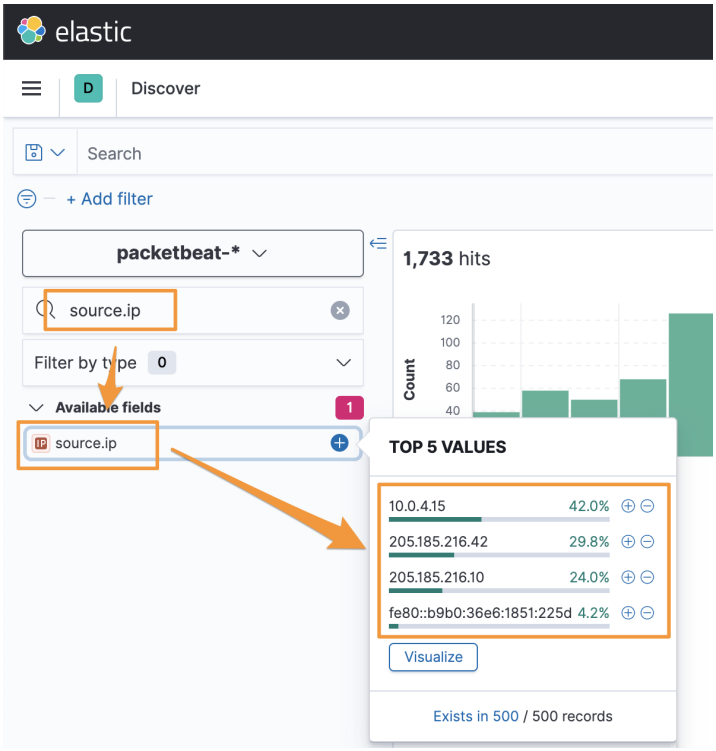

Figure 7.12 – Viewing source IP addresses in the dataset

Let's narrow it down to only the source IP addresses of our victim machine. In the search bar, type in `source.ip : 10.0.4.15` and your results will change to only show network traffic that originated from your victim machine. If you repeat the previous steps, you should see `10.0.4.15` as the only source IP in your dataset:

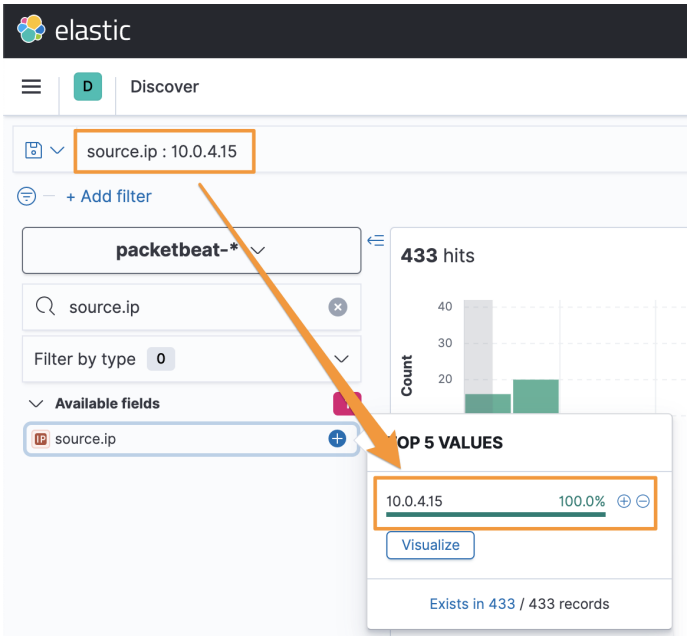

Figure 7.13 – Searching for the victim machine source IP address

Again, this is pretty self-explanatory. Next, let's chain a search together. Let's search for your victim machine and network traffic that is only going to port `443`. To connect Lucene searches together, we need to use a capitalized `AND` operator. So, in our example, this should appear as follows:

```
source.ip : 10.0.4.15 AND destination.port : 443
```

Now you'll see only traffic from your victim machine to port `443`.

> **Important note**
>
> As mentioned earlier, to connect Lucene queries, you need to use capitalized Boolean operators (that is, AND, OR, and NOT). If you don't capitalize them, Lucene will interpret them as being unrelated and simply show you all of the results from each search grouping. For example, typing in `source.ip : 10.0.4.15 AND destination.port : 443` will only show results that include both the source IP and destination port that you specified. If you search for `source.ip 10.0.4.15` and `destination.port : 443`, you will see all of the events that have the source IP you specified and all of the events that have the destination port you specified, even if they aren't related to each other.

I mentioned at the beginning of the section that one advantage Lucene has over KQL is that it can perform regex queries. To highlight this, let's write a basic regex to identify specific network traffic. This isn't a book on regex, so we'll keep it simple.

When using Lucene, you tell Kibana you're passing a regex by wrapping `/` characters around the regex.

So, let's run a query for the network traffic we generated at the beginning of the section. In Kibana, run the following search to return results for any events that have a destination domain of `packtpub`, `elastic`, or `neverssl` and a top-level domain of `.com` or `.co`:

```
destination.domain:/(elastic|packtpub|neverssl).(com|co)/
```

In the following screenshot, we can view how the results of the query are displayed in **Discover**:

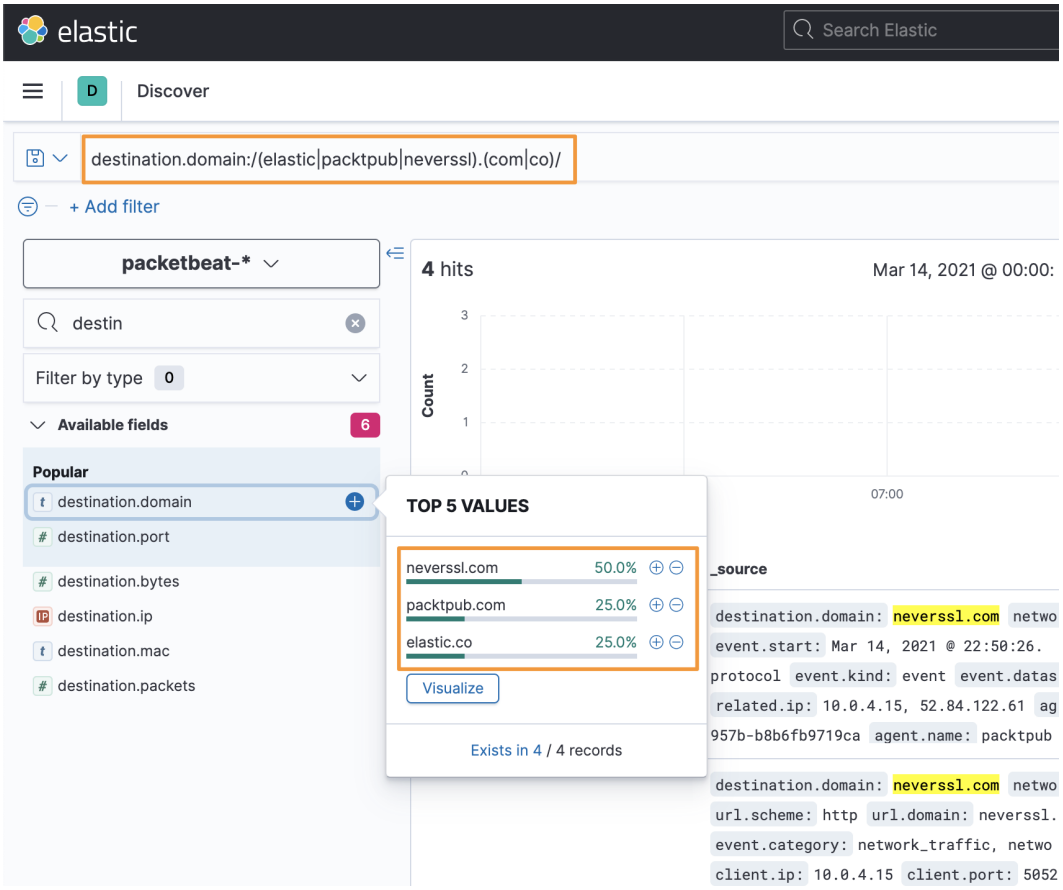

Figure 7.14 – Using Lucene to perform a regex query

While this simple example could be accomplished in other ways, which we'll cover in more detail later, we quickly wanted to demonstrate how to use a regex in the event that you have a specific query that requires it. As a more real-world example, here is a regex rule (from the official Elastic documentation at `https://www.elastic.co/guide/en/security/current/cobalt-strike-command-and-control-beacon.html`) that identifies specific traffic that uses a destination domain; it has a subdomain with exactly three characters, another subdomain called `stage`, and another subdomain with exactly eight digits. This would be difficult to find without a regex:

```
event.category:(network OR network_traffic) AND type:(tls OR
http) AND network.transport:tcp AND destination.domain:/[a-z]
{3}.stage.[0-9]{8}\..*/
```

Using Lucene, we performed a few basic searches and described how to search for data using regular expressions. Lucene is a powerful search library, but it does have some nuance and subtleties that can cause confusion or mistakes. To make searching the data in Kibana a bit more approachable, Elastic created KQL.

KQL

KQL was created by Elastic as a way to search through Elasticsearch data in Kibana. While searching with Lucene is available, KQL has a lower barrier for entry and can even suggest fields, operators, or values based on what is available in the dataset.

> **Important note**
>
> Learning your data is paramount to being able to hunt through it. As I mentioned in the previous section, spend more time on Discover so that you can learn what data you have and what fields that data is stored in. If you're using a Beat to collect data, you can check out the exported fields from the Beats documentation to get an idea of the available fields, but the values are something that you should take time to learn.

Start your VMs and validate that data is present in **Elasticsearch**. For these examples, we'll be entering queries into the **Search Bar** in Discover and using the **Packetbeat** index pattern. Remember to check your **Date Picker** and ensure you're set to use the KQL syntax. Feel free to review *The Discover app* section if you need a refresher on these settings.

Since we're not using the **victim VM** to generate a tremendous amount of traffic, I recommend that you set your **Date Picker** to **Today** and spend some time on your **victim VM** by browsing the internet for several minutes.

Terms queries

Terms queries are exact searches, meaning you're looking for specific terms. You can use spaces to separate terms, and only one term is required to get a match. You can use quotes to match phrases.

KQL doesn't require a field; we can just ask for any existence of a term. Let's just search for some basic terms. In the search bar, type in the following:

```
network
```

This will return all results for documents that have the word `network` anywhere.

Let's search for all of the HTTP traffic. This time, let's get a bit more specific and provide a field name:

```
network.protocol : http
```

Here, you will only view HTTP network data.

Next, let's search for the HTTP response status phrase:

```
http.response.status_phrase : ok
```

Here, you will only view HTTP response codes of `200` (OK).

Let's try another phrase in the same field; for example, let's try the following:

```
http.response.status_phrase : "partial content"
```

Here, you will only see results that have HTTP response codes of `206` (partial content).

In this section, we discussed using KQL to perform basic terms queries. As you get more data and explore it, you'll come up with additional queries to search for.

Boolean queries

KQL supports using Boolean operators (for instance, `or`, `and`, and `not`). By default, `and` has higher precedence than `or`, but that can be overridden with the use of parentheses. Boolean queries can be very simple or complex.

Building on the example from the *Terms queries* section, let's stay focused on the HTTP protocol. Let's search for HTTP traffic that has an HTTP response status code of `200` using the `and` operator:

```
network.protocol : http and http.response.status_code : 200
```

This will return HTTP traffic that returns an HTTP status code of `200`. This kind of basic association can help to track redirects of network activity when researching possible command and control infrastructures. Additionally, this can be used to track when an infected system might have gotten successful connections versus redirects or `404` errors, which could mean that there is some sort of command and control logic incorporated inside the implant.

Next, we can expand our search by using an `or` operator. We can use parentheses to include multiple values for a single field with `field : (value or value)`:

```
network.protocol : http and http.response.status_code : (200 or
301)
```

This will show us HTTP traffic that has a response code of `200` or `301` (moved permanently).

Now, maybe we don't want to see HTTP traffic that is providing images. We can do that too, using the `not` operator:

```
network.protocol : http and not http.response.headers.content-
type : image/png
```

Whoops. Looking at my data, I still see some images that have the `.jpeg` extension. That's no problem; let's combine the `or` and `not` operators together:

```
network.protocol : http and not http.response.headers.content-
type : (image/png or image/jpeg)
```

In this section, we discussed using KQL to perform Boolean queries. As you get more data and explore it, you'll come up with additional queries to search.

Range queries

For numerical data, you can use range queries to search for top and bottom values using the normal >, >=, <, and <= mathematical operators.

Network data has some great numerical data in network port numbers. Let's search for source ports greater than `5000`:

```
event.category : network and source.port > 5000
```

Next, let's search for network data that has a source port between `5000` and `7000`. Note that it is not required to wrap the `source.port` fields in parentheses, but I do it as a way to organize my queries. When you get complex queries, being able to control how fields are evaluated is very important – both for accuracy and sanity:

```
event.category : network and (source.port >= 5000 and source.
port <= 7000)
```

Lucene has an advantage here in that it can query ranges slightly differently. With Lucene, you can make the same query, but you can use brackets to define the range. I still wouldn't use Lucene for this use case, but I wanted to call it out:

```
event.category : network AND source.port : [5000 TO 7000]
```

In this section, we discussed using KQL to perform Boolean queries. As you get more data and explore it, you'll come up with additional queries to search.

Date queries

We can also use KQL to incorporate date queries into our search bar. Generally, we would use the **Date Picker** for this, but there are some situations in which you might want to perform date queries in the search bar.

Like range queries, date queries use mathematical operators. For example, to look at all data in March of 2021, you could use the following:

```
@timestamp > "2021-03"
```

It is very important to note that this search works just like any other query in that it relies on the **Date Picker** to define what window of time you're actually looking at. So, if you ran the preceding search but your **Date Picker** was set to *Last 7 days*, you'd only see the last seven days. I strongly recommend not using this unless you identify a specific use case that requires this.

Exists queries

Now that we've poked at our data a bit, you might have noticed that, sometimes, it'd be easier to only display events that have specific fields. We can create filters to do this for us, but we can also do this on the search bar using an `exists` query.

What we're saying with an exists query is that we want to see every event that includes this field. So, as an example, we want to see every event that includes a destination domain. We could assume that would be the HTTP and TLS network protocols, and simply filter or search on those. But we can also use an exists query, which can be faster:

```
destination.domain : *
```

This will only display results that have the `destination.domain` field.

In this section, we discussed how to use KQL to perform exists queries. As you get more data and explore it, you'll come up with additional queries to search.

Wildcard queries

Wildcard queries allow you to use a wildcard in a field or a term to search for data with multiple values (such as `Windows 10` and `Windows 7`) or fields that might have multiple subfield sets (such as `host.os.family` and `host.os.platform`).

To search for all results where the host OS is Windows, use the following:

```
host.os.name : Win*
```

To search for all results where the `host.os.family` and `host.os.platform` subfield sets are either Windows 10 or 7 (or XP, 2000, ME, and 8), use the following:

```
host.os*:win*
```

In this section, we discussed how to use KQL to perform wildcard queries. As you get more data and explore it, you'll come up with additional queries to search.

In this section, we explored KQL and the different types of queries that you can perform. As I mentioned at the end of each query type, as you explore your data, you'll experiment with many ways to use KQL to uncover more information about your data. The more you dig, the more you'll learn about what types of queries and practices work best for you.

Next, we'll discuss EQL and some of its additional capabilities over Lucene and KQL.

EQL

KQL is a powerful language that allows you to directly explore your data and make basic queries for one or multiple fields. To perform more advanced queries, Elastic has released EQL. EQL is based on a query language of the same name from a company that Elastic joined forces with, that is, Endgame.

Currently, EQL is available almost exclusively in the Security app (that is, in the detection engine and timeline). But it is also in the DevTools app in Kibana, which allows you to make queries against the Elasticsearch API.

At a high level, EQL allows you to express relationships (such as sequences, times, and categories) between events. This is different from Lucene or KQL in that you can create searches that return results only when certain events happen in relation to other events.

Here is one example of some macOS detection logic (taken from the official Elastic documentation at `https://www.elastic.co/guide/en/security/current/macos-installer-spawns-network-event.html`):

```
sequence by process.entity_id with maxspan=1m
  [ process where event.type == "start" and host.os.family ==
"macos" and
      process.parent.executable in ("/usr/sbin/installer", "/
System/Library/CoreServices/Installer.app/Contents/MacOS/
Installer") ]
  [ network where not cidrmatch(destination.ip,
      "192.168.0.0/16",
      "10.0.0.0/8",
      "172.16.0.0/12",
      "224.0.0.0/8",
      "127.0.0.0/8",
      "169.254.0.0/16",
      "::1",
      "FE80::/10",
      "FF00::/8") ]
```

In the preceding snippet, the `process.entity_id` field is used to link the `process.parent.executable` field to outbound network events. This rule is triggered when the native macOS installer program attempts to make a network connection within 1 minute. This kind of relationship cannot be queried in Lucene or KQL. Instead, this could be used in the detection engine or in a timeline, which are both in the Security app.

EQL is almost a programming language on its own. So, we'll focus on two of the most common use cases for security and refer you to Elastic's official documentation for additional learning.

Basic syntax

The basic syntax for EQL relies on an **Elastic Common Schema** (**ECS**) event category along with a condition and a keyword that connects them.

> **Important note**
>
> ECS is a large, complex, and well-structured schema used to uniformly describe data in Elasticsearch. ECS could be its own book and is beyond the scope of this book. It is cataloged in the official Elastic documentation. We will also discuss the uses of ECS in *Chapter 12*, *Sharing Information and Analysis*.

The available ECS event categories are as follows:

- `authentication`
- `configuration`
- `database`
- `driver`
- `file`
- `host`
- `iam`
- `intrusion_detection`
- `malware`
- `network`
- `package`
- `process`
- `registry`
- `session`
- `web`

These conditions are options for the specific dataset. So, for the `process` fieldset, there are args, names, executables, and more. Generally, it is best to follow ECS guidelines, but any data in the `event.category` field will work.

For example, the event category could be `process` and the condition could be the process name of `svchost.exe`:

```
process where process.name == "svchost.exe"
```

We have discussed the basic syntax required for an EQL query. Next, we'll discuss sequences.

Sequences

By using sequences in EQL, you can describe a series of events based on their order. For sequences, each event needs to follow the basic syntax we discussed earlier with an event category and event condition. These events are surrounded by `[ ]`. The events are in reverse chronological order with the most recent event listed last.

For example, if you had a sequence to identify when an event had an executable file followed by a `network` event, it could appear as follows.

```
sequence   [ file where file.extension == "exe" ]   [ network
where true ]
```

Sequences are a differentiator between KQL and EQL in that you can describe the order of events that can be powerful in threat hunting, as you can combine process and network events and how they occur.

We briefly discussed EQL as a powerful query language and provided a few brief examples of how it can be used. Next, we'll use the skills we learned in Discover and use KQL to make rich visualizations and dashboards.

The Visualize app

Now that we've experimented with a few searches and received the resulting text, let's try to visualize it in a way that we can display the data so it can be consumed at a glance to understand the operational security picture as well as to facilitate actual threat hunting.

As before, start up both your Elastic and victim VMs and ensure that the event data is being reported into Elastic. Additionally, we'll be using the three saved searches you created during *The Discover app* section at the beginning of the chapter. If you don't have these searches saved, please refer to those sections.

All of the visualizations are extremely interactive, and they allow you to hover over them to get introspection into a specific data point or to click and apply filters directly to the visualization.

Click on the **hamburger** menu and then select the **Visualize App**.

> **Important note**
>
> As we move through the next two sections on visualizations and dashboards, it is important to remember they are meant to facilitate analysis, highlight data of note, and uncover data patterns. I commonly see visualizations that appear to be created because they look cool or *because they can*. I don't want to stifle creativity, but your screen real estate is limited, so it's best to make use of visualizations that usefully describe data.

There are several different types of visualizations that you can experiment with, each with different use cases. We'll cover the three types that I find most helpful for threat hunting, but you should feel free to find what works best for you.

Considerations

It can be easy to load up visualizations with *all the things*, but generally, visualizations will be part of a dashboard. So, it is best to make specific and direct visualizations together with the others on a dashboard, which tell the whole story.

Visualizations run a search *behind the scenes*, so when we're creating a visualization, it is best from a performance perspective to use a saved search. This will perform the search and then the visualizations will simply apply filters to a single search instead of each visualization running an independent search. If you're experimenting with *what works*, you don't need to use a saved search, but when you move the visualization to a dashboard, this is recommended.

Visualizations are extremely powerful, and with that, they bring a lot of complexity and options. We're going to cover the steps that are required to make good hunt dashboards, but there is a tremendous amount regarding visualizations that we just will not have time to cover. I strongly encourage you to experiment with your data.

The data table

In my opinion, the data table is the most useful of all visualizations for threat hunting because you can get specific information, both aggregated and readable.

To make a data table, from within the **Visualize App**, click on the blue **Create visualization** button, select **Aggregation-based**, and then **Data table**. Next, we'll select the **index pattern** that we're going to use; keeping with our network-based theme, click on `packetbeat-*`.

You will be presented with a blank data table that simply shows you the number of Packetbeat records (in my dataset, this is 895) you have within your allotted time window. The default is **Last 15 minutes**:

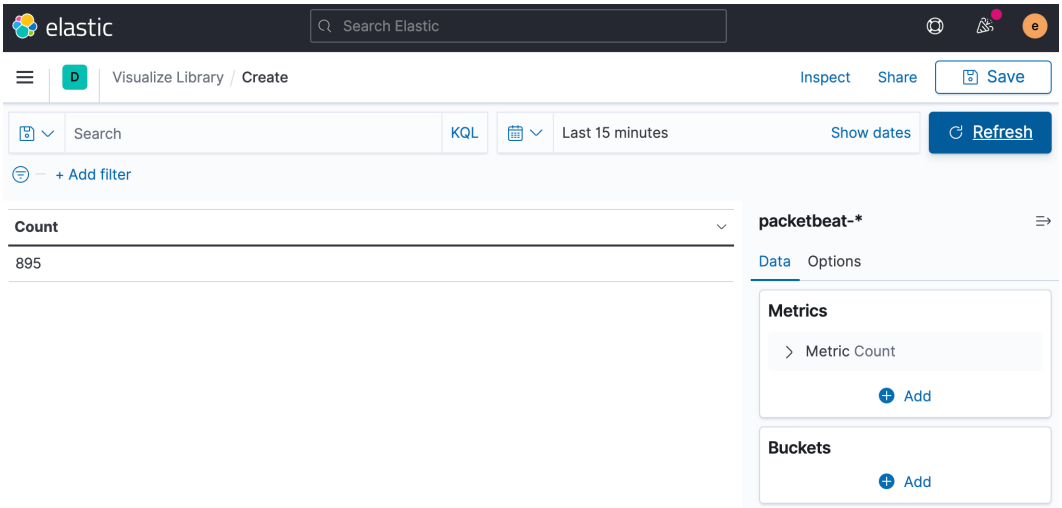

Figure 7.15 – Blank data table

Next, click on **Add** under **Buckets**, select **Split Rows**, and then **Terms** as the **Aggregation**. Finally, select `network.protocol` as the **Field**. You can leave everything else as its default setting, but feel free to experiment with other configurations.

When you click on **Update**, you'll have a count of each record aggregated by their network protocol:

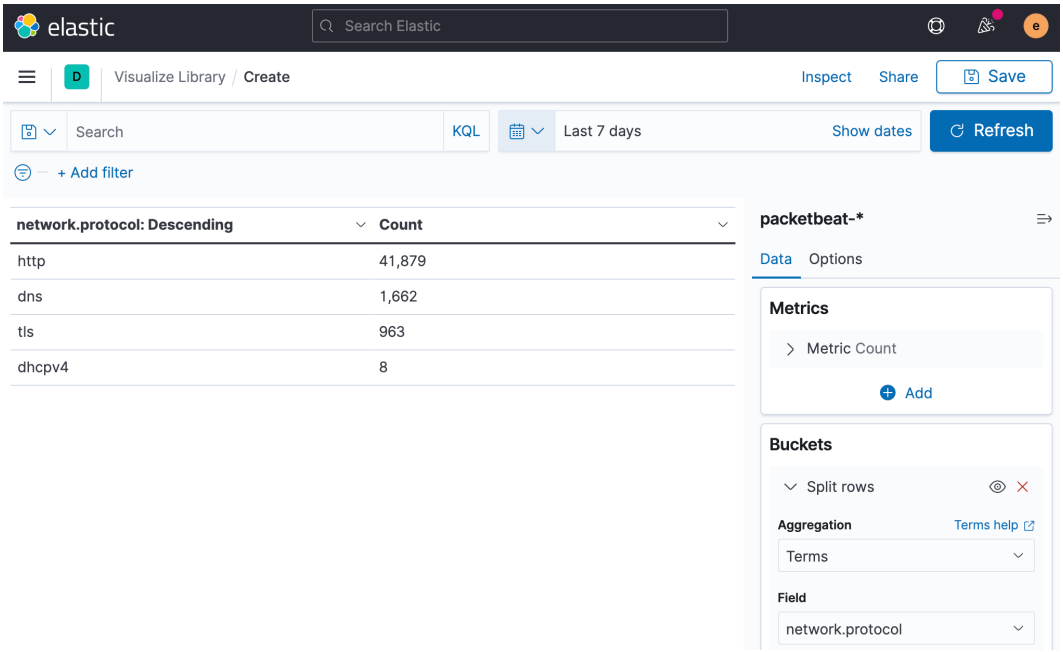

Figure 7.16 – The network protocol data table

That's useful information, but let's add some more data to it. Using the same method, under **Buckets**, click on the **Add** button, select **Terms**, and select `destination.domain` as the field. You should notice two important things:

1. The aggregation now includes the protocols and domains for the top five domains.
2. `dhcpv4` and `dns` have disappeared.

The network protocol and domain show the top five domains and what protocol was used to visit them. If you'd like to see the bottom five, in the appropriate **Bucket** menu, you can change the **Order** from **Descending** to **Ascending**. If you'd like to see more than five, you can change the **Size**.

`dhcpv4` and `dns` have disappeared because there are no destination domains associated with that network protocol. So, unlike the view in **Discover**, which uses a - if the fields are null, the data table does not display them at all. If you would like to view these null fields, you must toggle the **Show missing** switch in the **Buckets** menu:

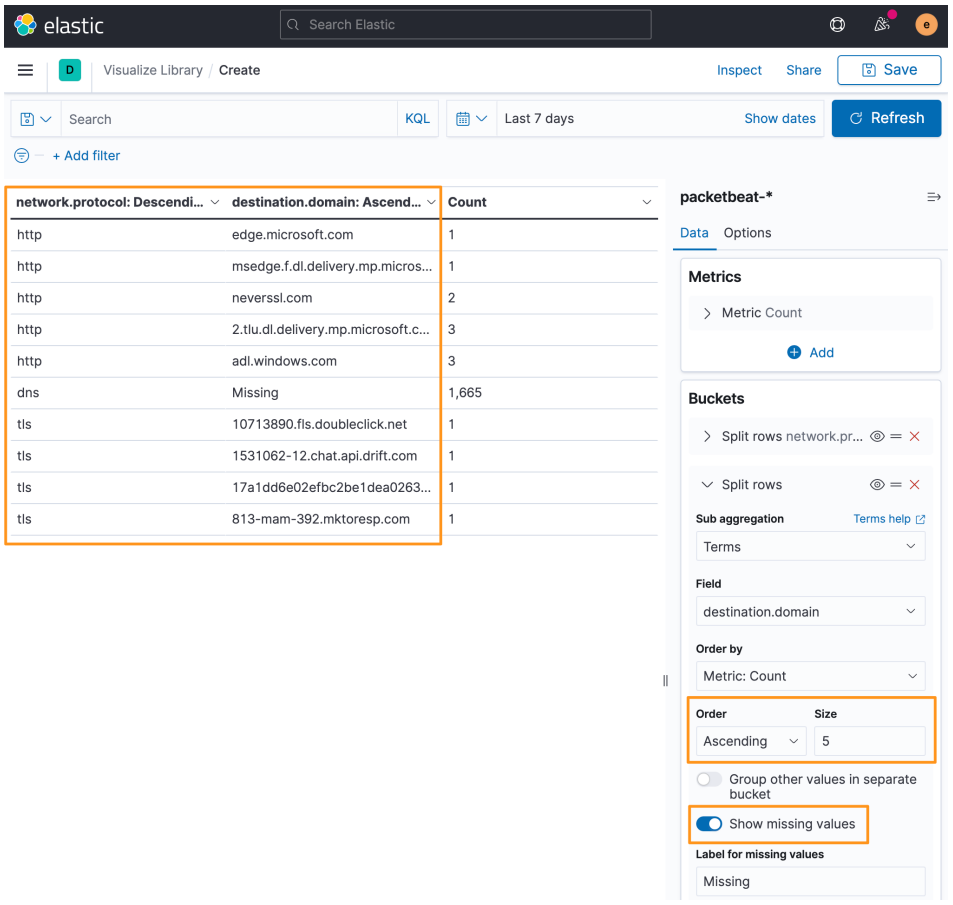

Figure 7.17 – The network protocol and domain data tables

Visualizations work just like anything else in Kibana; when you click on the text, you can then click the + and - buttons to filter the data based on what you're seeing.

Experiment with the `destination.domain` and `destination.ip` fields and the `url.full` and `destination.domain` HTTP fields.

Using network data collected by Packetbeat, we have discussed how to use a data table to aggregate different data types together. Data tables are a powerful tool in aggregating data to find previously unknown data. Next, we'll discuss bar charts.

Bar charts

A bar chart can help identify trending information and is useful for dashboarding to display information at a distance. A bar chart can be created in the same way as a data table, but ensure you select either a horizontal or vertical bar chart instead of a data table.

To keep our comparisons the same, let's create a bar chart that displays `network.protocol`.

Under **Buckets**, click **Add**, and select **X-axis**. Then, select **Terms** as your aggregation and `network.protocol` as your field. Click on the blue **Update** button:

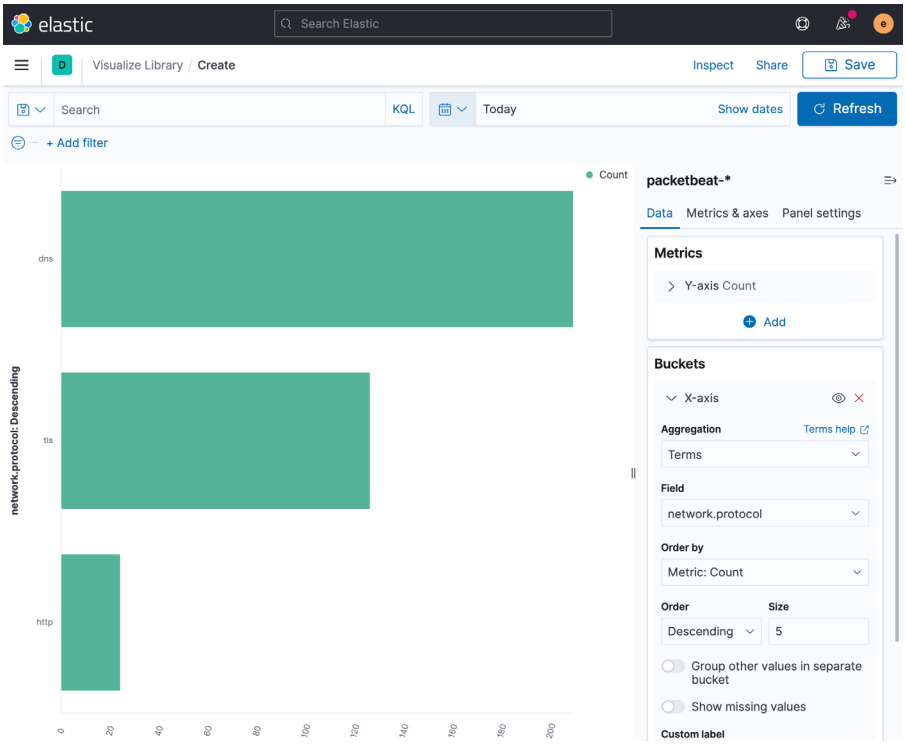

Figure 7.18 – The network protocol bar chart

Bar charts are great on dashboards in which to show protocols as they relate to data table visualizations.

Pie charts

Pie charts are great visualizations that can be used to show percentages of things. I commonly see pie charts that appear to be just layers upon layers upon layers upon layers of information, which makes the chart, while great looking, completely useless. As I mentioned earlier, keep it simple. The power of dashboards will make direct visualizations much more powerful than a single bloated visualization.

You can create a pie chart in the exact same way you created a data table and bar chart. In the **Bucket** menu, split the slices and add `network.protocol`. Again, we'll see the same network protocol data arranged in a donut:

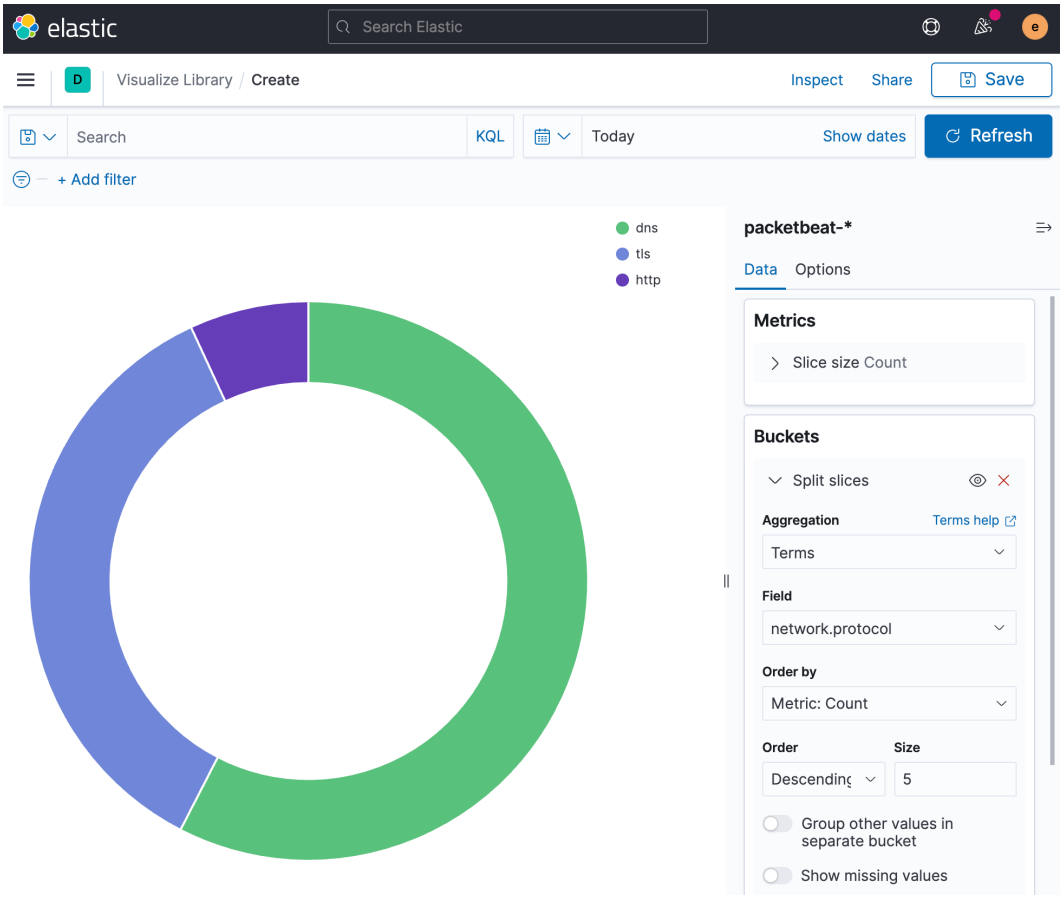

Figure 7.19 – The network protocol pie chart