A Cybersecurity Game

Introduction: Welcome to The Krusty Krab

Welcome to the Krusty Krab! 🥳 Today is your first day as a Junior Security Operations Center (SOC) Analyst with our company.

The Krusty Krab is a mid-size fast food chain serving the greater Bikini Bottom metropolitan area. The Krusty Krab is best known for its delectable Krabby Patties™, kelp shakes, and sea dogs. Because of its wild success, some of the Krusty Krab's competitors would benefit from stealing its secret formula and reproducing it themselves. Of note, the Chum bucket - a minor competitor - has been less than ethical in its attempts to infiltrate the Bikini Bottom patty market.

Your job is to defend the Krusty Krab and its employees from malicious cyber actors looking to steal the secret formula.

Like all good companies, The Krusty Krab collects log data about the activity its employees perform on the corporate network. These security audit logs are stored in Azure Data Explorer (ADX) - a data storage service in Azure (Microsoft's cloud). You will use the Kusto Query Language (KQL) to parse through various types of security logs. By analyzing these logs, you can help us determine whether we're being targeted by malicious actors.

You can find full documentation on ADX here: https://docs.microsoft.com/en-us/azure/data-explorer/kusto/query/tutorial?pivots=azuredataexplorer

🧠 **By the end of your first day on the job, you should be able to:**

- ✓ Use the Azure Data Explorer
- ✓ Use multiple data sets to answer targeted questions
- ✓ Find cyber activity in logs including: email, web traffic, and server logs
- ✓ Use multiple techniques to track the activity of APTs (Advanced Persistent Threats)
- ✓ Use third party data sets to discover things about your attackers
- ✓ Build a threat intelligence report
- ✓ Make recommendations on what actions a company can take to protect themselves

The attackers have gotten a head start, so let's not waste any more time… time to get to work!

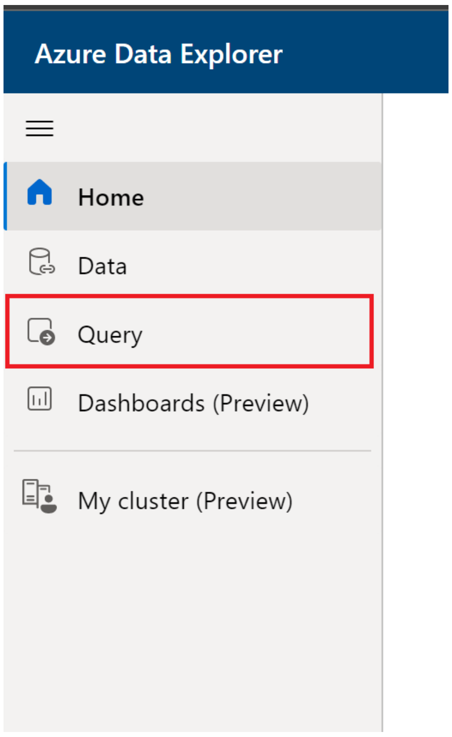
The walkthrough

Getting Set Up in Azure Data Explorer (ADX)

ADX is the primary tool used in the The Krusty Krab SOC for data exploration and analysis. The great thing about ADX is that it is used by cyber analysts at many of the smallest and largest organizations in the world.

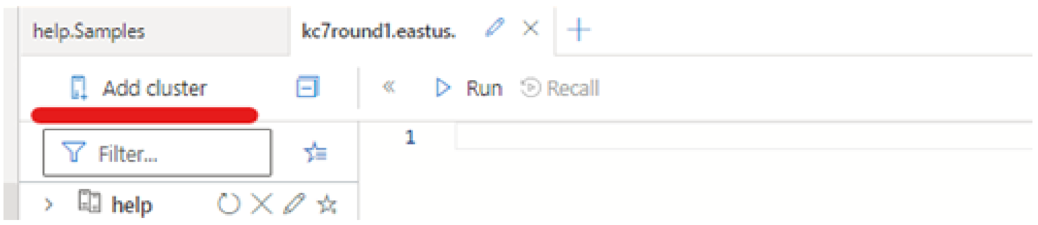
Let’s get you logged in and started with ADX:

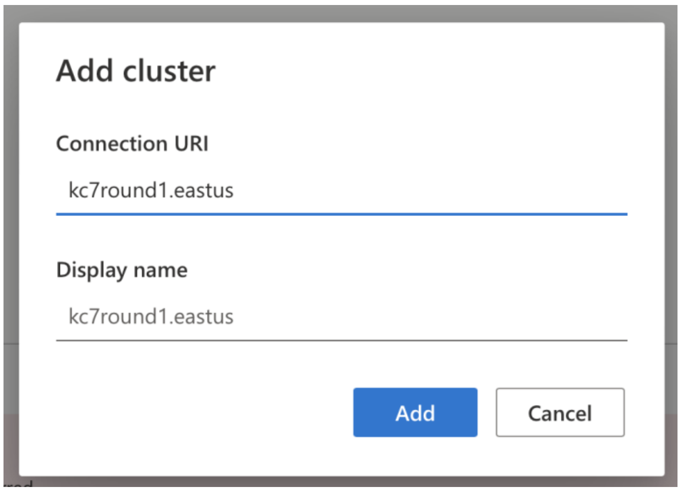
1. Go to https://dataexplorer.azure.com/ and login using your Microsoft account.
2. Click the Query tab on the left side of the screen.



Data in ADX is organized in a hierarchical structure which consists of **clusters**, **databases**, and **tables**. All of The Krusty Krab’s security logs are stored in a single cluster. You’ll need to add this cluster to your ADX interface so you can start looking at the log data. Don’t worry, our HR department is really fast. I’m sure they’ve already processed your access requests.

3. Add a new cluster using the cluster URI provided by your instructor
   - **Click add cluster**
   - **Enter Connection URI:** kc7round1.eastus




Add cluster

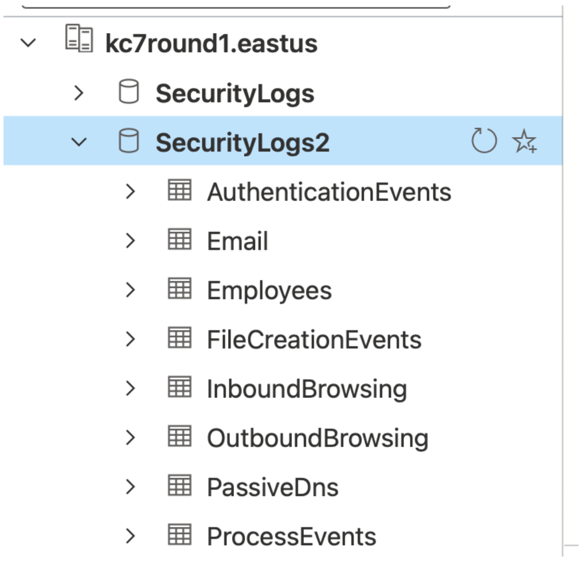
Connection URI

kc7round1.eastus

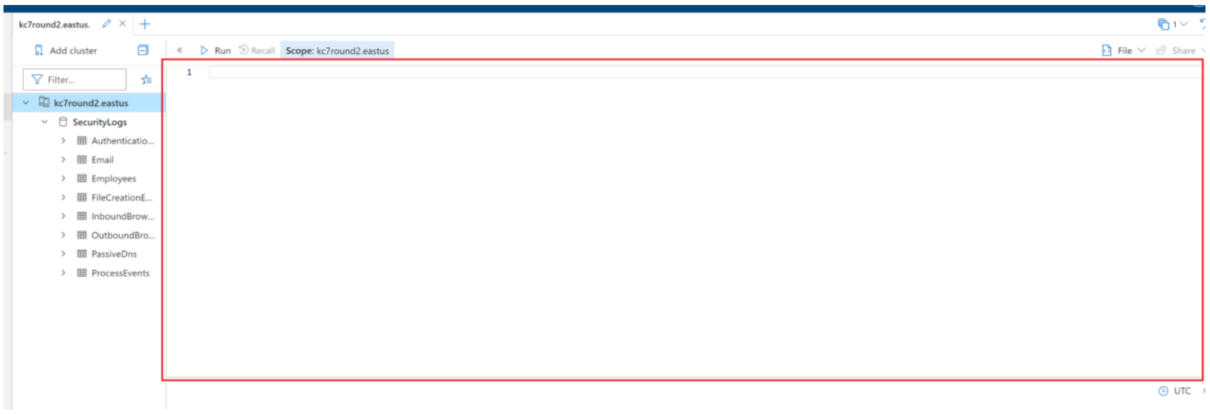
Display name

kc7round1.eastus

4. Expand the dropdown arrow next to your cluster. You should then see one database, called **SecurityLogs2** inside it. Next, expand the dropdown arrow next to the **SecurityLogs2** database. Finally, click on the **SecurityLogs2** database. Once you've done this, you should see the database highlighted- this means you've selected the database and are ready to query the tables inside.



The big blank space to the right of your cluster list is the *query workspace.* That's where you'll actually write the queries used to interact with our log data.



Okay, enough introductions… let's get your hands on the data.

First look at the data…

The **SecurityLogs2** database contains eight tables. Tables contain many rows of similar data. For security logs, a single row typically represents a single thing done by an employee or a device on the network at a particular time.

We currently have eight types of log data. As you'll see in ADX, each log type corresponds to a table that exists in the **SecurityLogs2** database:

Table Name	**Description**
Employees	Contains information about the company's employees
Email	Records emails sent and received by employees
InboundBrowsing	Records browsing activity from the Internet to devices within the company network
OutboundBrowsing	Records browsing activity from within the company network out to the Internet
AuthenticationEvents	Records successful and failed logins to devices on the company network. This includes logins to the company's mail server.
FileCreationEvents	Records files stored on employee's devices
ProcessEvents	Records processes created on employee's devices
PassiveDns (External)	Records IP-domain resolutions

🎯 **Key Point – Over the Horizon (OTH) data**: One of the tables listed above is not like the others – **PassiveDns.** Rather than being an internal security log, PassiveDns is a data source that we've purchased from a 3rd party vendor. Not all malicious cyber activity happens within our company network, so sometimes we depend on data from other sources to complete our investigations.

You'll learn more about how to use each of these datasets in just a minute. First, let's just run some queries so you can practice using KQL and ADX.

KQL 101

Type the following query in the workspace to view the first rows in the **Employees** table. Press “run” or “shift + enter” to execute the query.

```
Employees
| take 10
```

The `take` operator is a powerful tool you can use to explore rows in a table, and therefore better understand what kinds of data are stored there.

🎯 **Key Point – What to do when you don’t know what to do:** Whenever you are faced with an unfamiliar database table, the first thing you should do is sample its rows using the `take` operator. That way, you know what fields are available for you to query and you can guess what type of information you might extract from the data source.

The **Employees** table contains information about all the employees in our organization. In this case, we can see that the organization is named “The Krusty Krab” and the domain is “krustykrab.com”.

1. 🤔 *Try it for yourself! Do a* `take 10` *on all the other tables to see what kind of data they contain.*

We can use `count` to see how many rows are in a table. This tells us how much data is stored there.

```
Employees
| count
```

2. 🤔 *How many employees are in the company?*

We can use the `where` operator in KQL to apply filters to a particular field. For example, we can find all the employees with the name “Linda” by filtering on the *name* column in the **Employees** table.

`where` statements are written using a particular structure. Use this helpful chart below to understand how to structure a `where` statement.

where	field	operator	"value"
where	name	has	"Linda"

```
Employees
| where name has "Linda"
```

The has operator is useful here because we're looking for only a *partial* match. If we wanted to look for an employee with a specific first and last name (an *exact* match), we'd use the == operator:

```
Employees
| where name == "Linda Dusel"
```

3. 🤔 *Each employee at The Krusty Krab is assigned an IP address. Which employee has the IP address: "192.168.2.189"*

While performing their day-to-day tasks, The Krusty Krab employees send and receive emails. A record of each of these emails is stored in the **Email** table.

🎯 **Key Point – User Privacy and Metadata:** As you can imagine, some emails are highly sensitive. Instead of storing the entire contents of every email sent and received within the company in a database that can be easily accessed by security analysts, we only capture email *metadata.* Email metadata includes information like: the time the email was sent, the sender, the recipient, the subject line, and any links the email may contain. Storing only email metadata, rather than entire contents, helps protect the privacy of our employees, while also ensuring that our security analysts can keep us safe. Sometimes even metadata can reveal sensitive information, so it's important that you don't talk about log data with other employees outside the SOC.

We can find information about the emails sent or received by a user by looking for their email address in the *sender* and *recipient* fields of the **Email** table. For example, we can use the following query to see all the emails sent by "Francisca Vasquez":

```
Email
| where sender == "francisca_vasquez@krustykrab.com"
```

4. 🤔 *How many emails did Danny Monson receive?*

We can use the distinct operator to find unique values in a particular column. We can use the following query to determine how many of the organization's users sent emails.

```
Email
| where sender has "krustykrab"
| distinct sender
| count
```

*Note here that the distinct operator returns all of the unique senders with the term "envolvelabs" in their domain. However, we can further use the count operator from above to figure out exactly "how many" of those senders there are.

5. 🤔 *How many users received emails with the term "custard" in the subject?*

When employees at The Krusty Krab browse to a website from within the corporate network, that browsing activity is logged. This is stored in the **OutboundBrowsing** table, which contains records of the websites browsed by each user in the company. Whenever someone visits a website, a record of it stored in the table. However, the user's name is not stored in the table, only their IP address is recorded. There is a 1:1 relationship between users and their assigned IP addresses, so we can reference the **Employees** table to figure out who browsed a particular website.

If we want to figure out what websites William Lancaster visited, we can find his IP address from the Employees table.

```
Employees
| where name == "William Lancaster"
```

The query above tells us his IP address is "192.168.0.149". We can take his IP address and look in the **OutboundBrowsing** table to determine what websites he visited.

```
OutboundBrowsing
| where src_ip == "192.168.0.149"
```

6. 🤔 How many unique websites did "Katrina Watson" visit?

Although domain names like "google.com" are easy for humans to remember, computers don't know how to handle them. So, they convert them to machine readable IP addresses. Just like your home address tells your friends how to find your house or apartment, an IP address tells your computer where to find a page or service hosted on the internet.

🎯 **Key Point – Practice Good OPSEC:** If we want to find out which IP address a particular domain *resolves to*, we could just browse to it. But, if the domain is a malicious one, you could download malicious files to your corporate analysis system or tip off the attackers that you know about their infrastructure. As cybersecurity analysts, we must follow procedures and safeguards that protect our ability to track threats. These practices are generally called operational security, or OPSEC.

To eliminate the need to *actively resolve* (that is- directly browse to or interact with a domain to find it's related IP address) every domain we're interested in, we can rely on *passive DNS* data. Passive DNS data allows us to safely explore domain-to-IP relationships, so we can answer questions like:

- *Which IP address does this domain resolve to?*
- *Which domains are hosted on this IP address?*
- *How many other IPs have this domain resolved to?*

These domain-to-IP relationships are stored in our **PassiveDNS** table.

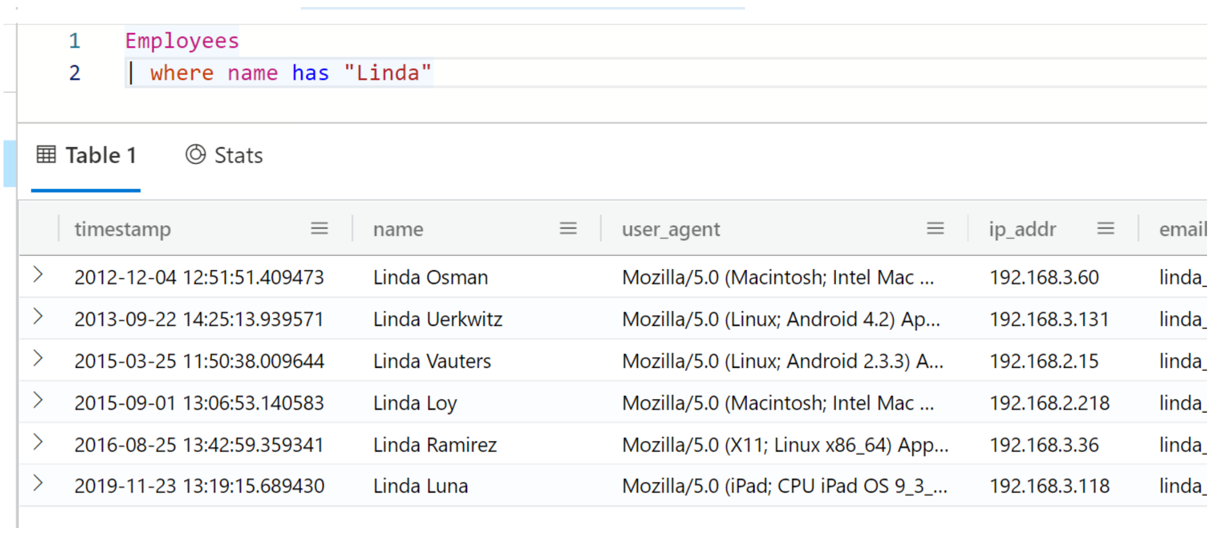
7. 🤔 *How many domains in the **PassiveDNS** records contain the word "jellyfish"? (hint*: use the `contains` operator instead of `has`. *If you get stuck, do a* `take 10` *on the table to see what fields are available.)*

8. 🤔 *What IPs did the domain "jellyfish.net" resolve to?*

🤯 **Let statements – making your life a bit easier:**

Sometimes we need to use the output of one query as the input for a second query. The first way we can do this is by manually typing the results into next query.

For example, what if we want to look at all the web browsing activity from employees named "Linda"?
First, you would need to go into the **Employees** table and find the IP addresses used by these employees.

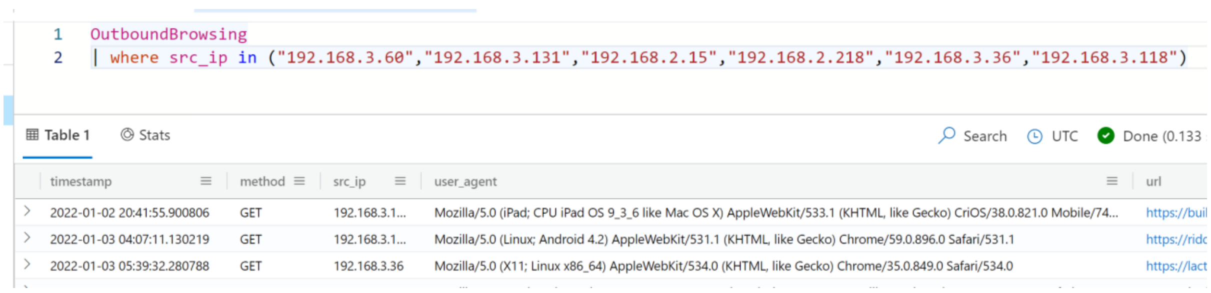


```
Employees
| where name has "Linda"
```

Table 1 | Stats

timestamp	name	user_agent	ip_addr	email
2012-12-04 12:51:51.409473	Linda Osman	Mozilla/5.0 (Macintosh; Intel Mac ...	192.168.3.60	linda_
2013-09-22 14:25:13.939571	Linda Uerkwitz	Mozilla/5.0 (Linux; Android 4.2) Ap...	192.168.3.131	linda_
2015-03-25 11:50:38.009644	Linda Vauters	Mozilla/5.0 (Linux; Android 2.3.3) A...	192.168.2.15	linda_
2015-09-01 13:06:53.140583	Linda Loy	Mozilla/5.0 (Macintosh; Intel Mac ...	192.168.2.218	linda_
2016-08-25 13:42:59.359341	Linda Ramirez	Mozilla/5.0 (X11; Linux x86_64) App...	192.168.3.36	linda_
2019-11-23 13:19:15.689430	Linda Luna	Mozilla/5.0 (iPad; CPU iPad OS 9_3_...	192.168.3.118	linda_

Then, you could manually copy and paste these IPs into a query against the **OutboundBrowsing** table. Note that we can use the *in* operator to choose all rows that have a value matching any value from a list of possible values. In other words, the == (comparison) operator looks for an exact match, while the *in* operator checks for any values from the list.



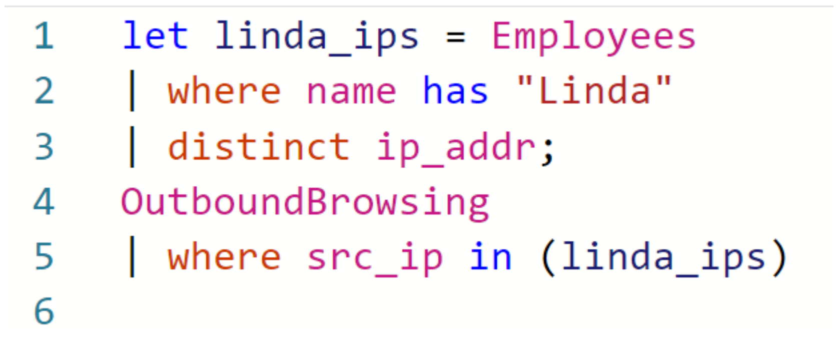
```
OutboundBrowsing
| where src_ip in ("192.168.3.60","192.168.3.131","192.168.2.15","192.168.2.218","192.168.3.36","192.168.3.118")
```

Table 1 | Stats | Search | UTC | Done (0.133

timestamp	method	src_ip	user_agent	url
2022-01-02 20:41:55.900806	GET	192.168.3.1...	Mozilla/5.0 (iPad; CPU iPad OS 9_3_6 like Mac OS X) AppleWebKit/533.1 (KHTML, like Gecko) CriOS/38.0.821.0 Mobile/74...	https://buil
2022-01-03 04:07:11.130219	GET	192.168.3.1...	Mozilla/5.0 (Linux; Android 4.2) AppleWebKit/531.1 (KHTML, like Gecko) Chrome/59.0.896.0 Safari/531.1	https://ridc
2022-01-03 05:39:32.280788	GET	192.168.3.36	Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/534.0 (KHTML, like Gecko) Chrome/35.0.849.0 Safari/534.0	https://lact

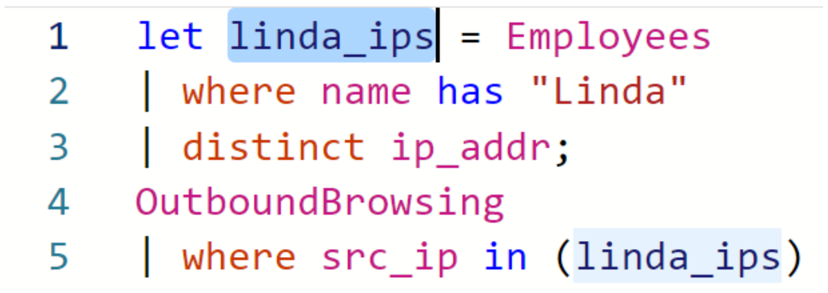
Although this is a valid way to get the information you need, it may not be as elegant (or timely) if you had 100 or even 1000 employees named "Linda."

We can accomplish this in a more elegant way by using a [let statement,](let statement) which allows us to assign a name to an expression or a function. We can use a `let` statement here to save and give a name to the results of the first query so that the values can be re-used later. That means we don't have to manually type or copy and paste the results repeatedly.



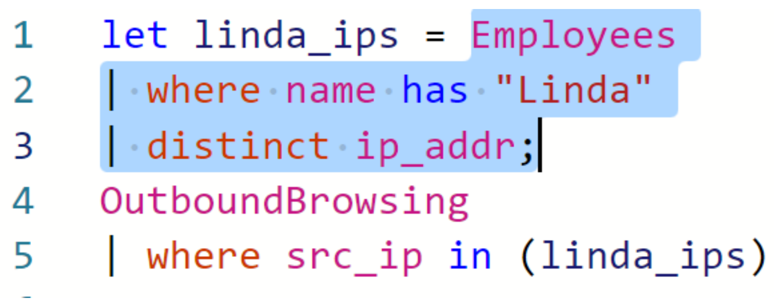
```
let linda_ips = Employees
| where name has "Linda"
| distinct ip_addr;
OutboundBrowsing
| where src_ip in (linda_ips)
```

On the left of the `let` statement is the variable name ("linda_ips" in this case). The variable name can be whatever we want, but it is helpful to make it something meaningful that can help us remember what values it is storing.



```
let linda_ips = Employees
| where name has "Linda"
| distinct ip_addr;
OutboundBrowsing
| where src_ip in (linda_ips)
```

On the right side of the `let` statement in the expression you are storing. In this case, we use the `distinct` operator to select values from only one column – so they are stored in an array – or list of values.



```
let linda_ips = Employees
| where name has "Linda"
| distinct ip_addr;
OutboundBrowsing
| where src_ip in (linda_ips)
```

The `let` statement is concluded by a semi-colon.

```
let linda_ips = Employees
| where name has "Linda"
| distinct ip_addr;
OutboundBrowsing
| where src_ip in (linda_ips)
```

After we store the value of a query into a variable using the `let` statement, we can refer to it as many times as we like in the rest of the query. The stored query does not show any output. Remember, however, that your KQL query must have a tabular statement – which means that you must have another query following your `let` statement.

9. 🤔 How many unique URLs were browsed by employees named “Karen”?

🎯 **Key Point – Pivoting:** Part of being a great cyber analyst is learning how to use multiple data sources to tell a more complete story of what an attacker has done. We call this “pivoting.” We pivot by taking one known piece of data in one dataset and looking in a different dataset to learn something we didn’t already know. You practiced this here when we started in one dataset – the Employees table – and used knowledge from there to find related data in another source – OutboundBrowsing.

Let’s find some hackers

Now that you’ve completed your initial round of training, you’re ready to work your first case in the SOC!

🎯 **Key Point – Open Source Intelligence (OSINT):** Security researchers and analysts often use free, publicly available data, like Twitter! We call this public data OSINT, and it can be a great way to get investigative leads. Like all public data sources on the internet, you should follow up any OSINT tip with rigorous analysis, rather than blindly trusting the source.

Answer the following questions related to this tip:

1. How many users in our organization were sent emails containing the domain chum.net?
2. One of the employees who was targeted has a 'y' in their name. Who is this employee?
3. How many of the emails containing chum.net were blocked? (hint: the “accepted” field in the Email table tells you whether or not the email was blocked. Blocked emails will show as false).
4. How many unique IP addresses does chum.org resolve to?
5. How many unique IP addresses does fun.com resolve to?
6. How many unique IP addresses does genius.org resolve to?
7. How many unique domains does IP 43.243.201.76 host?
8. How many unique domains does IP 101.143.139.17 host?
9. How many “.net” domains resolve to IP 192.224.255.208? (hint: you can use the in operator to check for multiple values in a field. E.g. | where field in (“x”, “y”, “z”)
10. How many ".com" domains resolve to IP 43.243.201.76?
11. Which ".pizza" domain resolved to the same IP as fun.com
12. Which ".net" domain resolved to the same IP as chum-fun.food
13. How many email addresses did the hackers use to send these domains (the .net domains you found in question #23)?
14. How many of these emails had mismatched sender and reply-to addresses? (You can use the != operators to check if two things are not the same)
15. How many users clicked on a link containing the domain "fun.com"?

16. What was the timestamp of the first employee that clicked on a phishing link containing the domain "fun.com"?
17. What was the first domain sent by "krabbs@yandex.com"? (Hint: You can click on the event_time column header to sort the rows by time. Then, find the link for the email that was sent first.)
18. What was the first email address used to send an email containing a link to "picket.pizza"?
19. Which email address was used to send a link to "bucketdomination.com"?
20. Based on the data you've seen (email addresses, domains, and subject lines), which SpongeBob character do you believe is responsible for the malicious activity targeting the Krusty Krab?