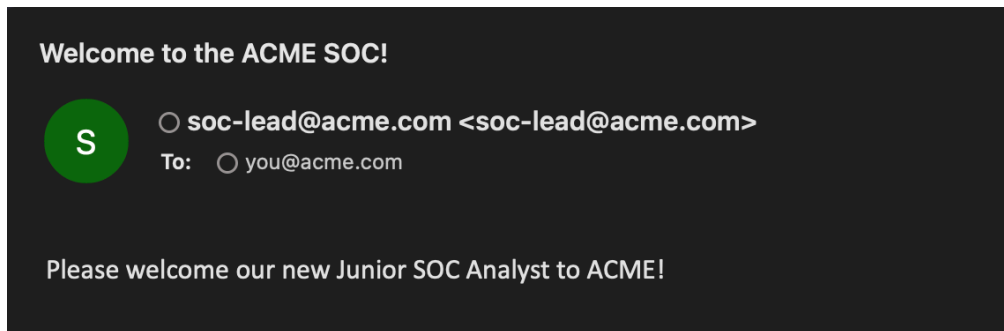


## A Cybersecurity Game

# Introduction: Welcome to ACME

Welcome to ACME Corporation! 🤖 Today is your first day as a Junior Security Operations Center (SOC) Analyst with our company. Your primary job responsibility is to defend ACME and its employees from malicious cyber actors.



Like all good companies, ACME 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 analysing these logs, you may be able to find malicious activity happening within the organization's network and identify the hackers that are performing these actions.



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



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

- ✓ Use Kusto Query Language (KQL) to manipulate data in Azure Data Explorer (ADX)
- ✓ Pivot across multiple data sets to answer targeted questions
- ✓ Identify malicious cyber activity in audit logs including: email, web traffic, and endpoint logs
- ✓ Use multiple “pivoting” techniques to track the activity of one or more Advanced Persistent Threat (APT) actors
- ✓ Leverage third party data sets such as PassiveDNS to discover unknown actor infrastructure based on known actor indicators (e.g. domains and IPs)
- ✓ Build a threat intelligence report describing a threat actor’s Techniques, Tactics, and Procedures (TTPs)
- ✓ Make recommendations on what actions a company can take to harden their environment and mitigate against a specific threat actor

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

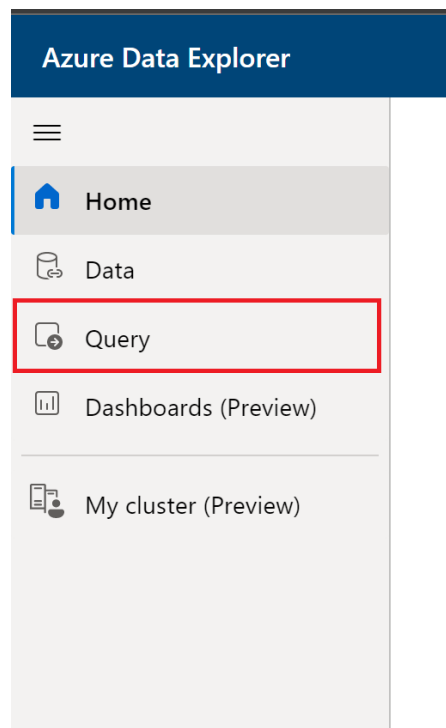
# The walkthrough

## Getting Set Up in Azure Data Explorer (ADX)

ADX is the primary tool used in the ACME 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.

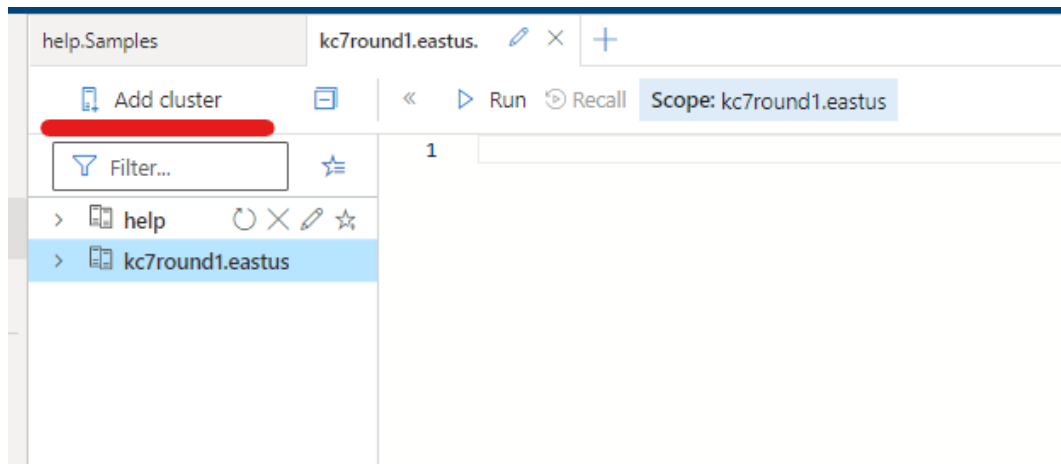
Let's get you logged in and started with ADX:

1. Go to <https://dataexplorer.azure.com/> and 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 ACME'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.

2. Add a new cluster with the following URL: **kc7round1.eastus**
  - Click **add cluster**
  - Enter **Connection URI**: *kc7round1.eastus*



### Add cluster

Connection URI

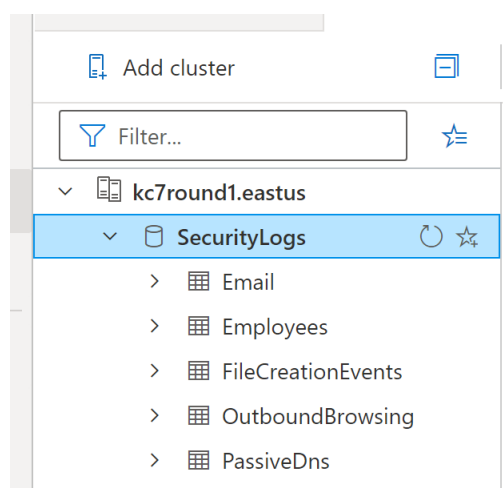
kc7round1.eastus

Display name

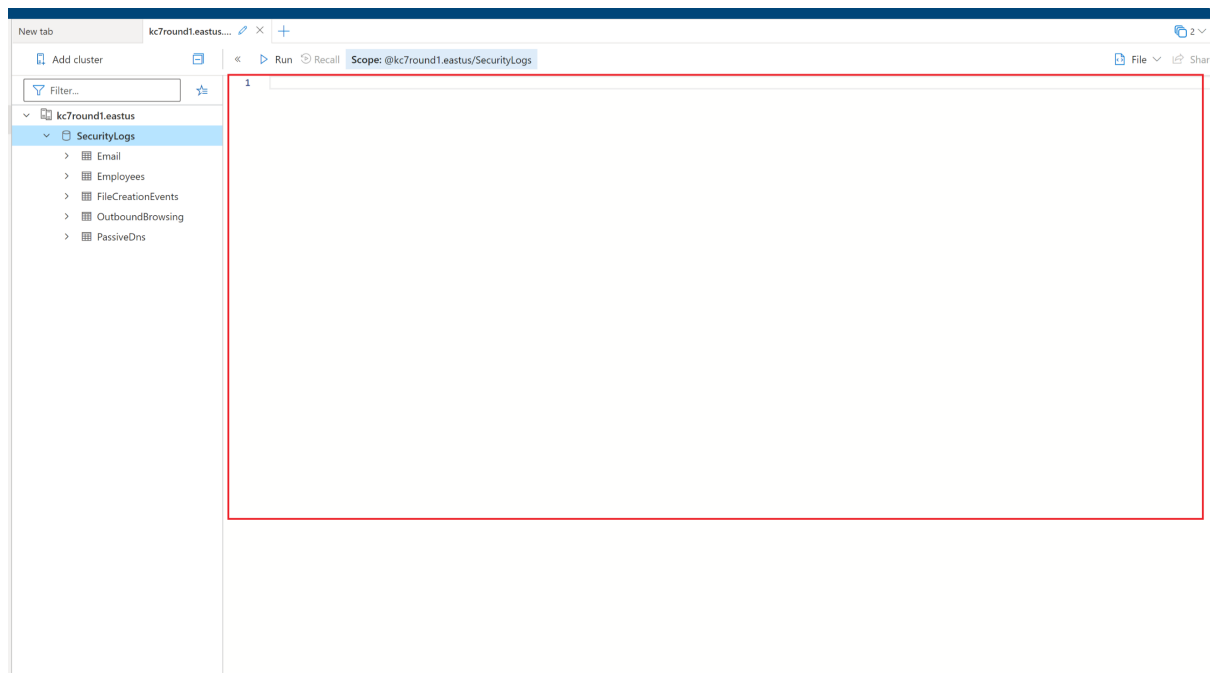
kc7round1.eastus

Add
Cancel

- Expand the dropdown arrow next to the **kc7round1.eastus** cluster. You should then see one database, called **SecurityLogs** inside it. Next, expand the dropdown arrow next to the **SecurityLogs** database. Finally, click on the **SecurityLogs** 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 **SecurityLogs** database contains five 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 four types of internal log data. As you'll see in ADX, each log type corresponds to a table that exists in the **SecurityLogs** database:

- Email
- Employees
- FileCreationEvents
- OutboundBrowsing

 **Key Point – Over the Horizon (OTH) data:** You'll notice there's a fifth table in the database that we didn't discuss above: **PassiveDns**. This dataset, rather than being an internal security log, is one that we've purchased from a 3<sup>rd</sup> 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.

- PassiveDns


You'll learn more about what each of these datasets actually means 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 "ACME" and the domain is "acme.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 "Joshua" by filtering on the *name* column in the **Employees** table.

```
Employees  
| where name has "Timothy"
```

3. 🤔 Each employee at Acme is assigned an IP address. Which employee has the IP address: "10.70.249.73"

While performing their day-to-day tasks, ACME 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 that 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 "William Steele":

```
Email
| where sender == "william_steele@acme.com"
```

4. 🤔 How many emails did Erin Lester 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 "acme"
| distinct sender
| count
```

\*Note here that the distinct operator returns all of the unique senders with the term "acme" 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 "kickoff" in the subject?

When employees at ACME 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.

For example, if we want to figure out what websites Donna Willis visited, we can look in the Employees table to find Donna's IP address:

```
Employees
| where name has "Donna Willis"
```

This tells us that her IP address is "172.19.58.249." We can now look for that IP address in the **OutboundBrowsing** table to determine what websites she visited.

```
OutboundBrowsing
| where src_ip == "172.19.58.249"
```

6. 🤔 How many unique websites did "Michelle Salazar" 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 has 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 "tackle"? (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 “zonehike.info” resolve to?

🧠 Let statements – making your life a bit easier:

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

For example, what if we want to look at all the web browsing activity from employees named “William”?

First, you would need to go into the **Employees** table and find the IP addresses used by these employees.

```
Employees
| where name has "William"
```

name	ip_addr	email_addr
William Parker	10.97.241.125	william_parker@acme.com
William Steele	192.168.80.149	william_steele@acme.com

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 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 ("10.97.241.125", "192.168.80.149")
```

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 “William.”

We can accomplish this in a more elegant way by using a [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 them repeatedly.

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

On the left of the *let* statement is the variable name (“William\_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 william_ips = Employees
| where name has "William"
| distinct ip_addr;
OutboundBrowsing
| where src_ip in (william_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 things.

```

let william_ips = Employees
| where name has "William"
| distinct ip_addr;
OutboundBrowsing
| where src_ip in (william_ips)

```

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

```

let william_ips = Employees
| where name has "William"
| distinct ip_addr;
OutboundBrowsing
| where src_ip in (william_ips)

```

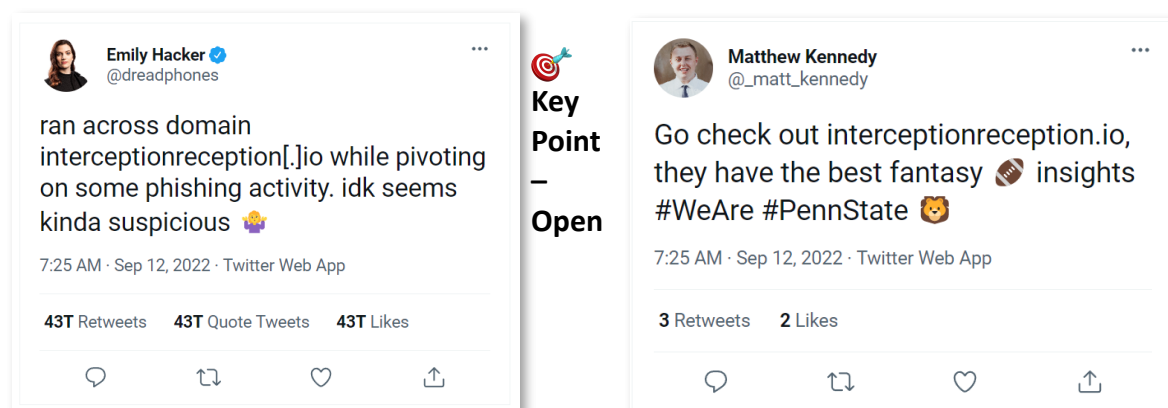
After we store the value of a query into a variable using the `let` statement, we can refer to it as many times as we like in the rest of the query. The stored query does not show any output. Remember, however, that your KQL query must have a tabular statement – which means that you must have another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

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

## 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!

A security researcher twitted that the domain "*interceptionreception.io*" was being used by hackers. Apparently the hackers are sending this domain inside credential phishing emails.



**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. Which users in our organizations were sent emails containing the domain *interceptionreception.io*?
2. Did we block any of the emails containing that domain? Who actually received one of these emails? (hint: the "accepted" field in the **Email** table tells you whether or not the email was blocked. Blocked emails will show as false).
3. What other domains shared the same IPs as this *interceptionreception.io*? Can you find the full list of domains associated with this actor based on PassiveDNS data? (hint: you can use the **has\_any** operator to check for multiple values in a field. E.g. where field has\_any ("x", "y", "z"))
4. What email addresses did the hackers use to send these domains?
5. Did users click on any of the links in the phishing emails?

# Final Challenge and Presentation

The US-CERT (United States Computer Emergency Readiness Team) issued an advisory that the domain “**vikingsandvilestthanking.io**” was being used in a recent phishing campaign by an advanced hacking group. Your supervisor has asked you take the lead on this investigation.



Here is your chance to shine! ✨ As the lead analyst for this investigation you are tasked with the following:

1. Identify indicators related to the initial tipped domain – which maybe belong to the hackers
2. Find all emails sent by the hackers
  - a. What email addresses did they use?
  - b. Which ACME employees received these emails?
  - c. Were any of the emails blocked?
3. Did any of the email recipients click on the links?
4. Is there anything we can learn about the hackers that will help us block their activity in the future (TTPs)?
5. What mitigations/measures can we take to protect ACME?

You should create a short threat intelligence briefing for the executives of ACME co. Your presentations should:

- provide a brief overview of the incident against ACME
- discuss impact to the organization
- present any findings on the nature of the adversaries
- make recommendations on mitigations that AMCE should implement against the malicious actors

Be sure to give the hackers a name when you are discussing them.

## Appendix: Answers to first challenge

1. Which users in our organizations were sent emails containing the domain *interceptionreception.io*?

Execute in [[Web](#)] [[Desktop](#)]

[[cluster\('kc7round1.eastus.kusto.windows.net'\).database\('SecurityLogs'\)](#)]

Email

```
| where link has "interceptionreception.io"
| distinct recipient
```

recipient
diane_brooks@acme.com
alan_patel@acme.com
william_steele@acme.com
seth_parker@acme.com
emily_evans@acme.com
michelle_salazar@acme.com

2. Did we block any of the emails containing that domain? Who actually received one of these emails?

Execute in [[Web](#)] [[Desktop](#)]

[[cluster\('kc7round1.eastus.kusto.windows.net'\).database\('SecurityLogs'\)](#)]

Email

```
| where link has "interceptionreception.io"
| where accepted != "true"
| count
```

Count
0

3. What other domains shared the same IPs as this *interceptionreception.io*? Can you find the full list of domains associated with this actor based on PassiveDNS data?

Execute in [[Web](#)] [[Desktop](#)]

[[cluster\('kc7round1.eastus.kusto.windows.net'\).database\('SecurityLogs'\)](#)]

```
let ips = PassiveDns
```

```

| where domain == "interceptionreception.io"
| distinct ip;
PassiveDns
| where ip in (ips)

```

ip	domain
149.68.141.192	falsifiesacyclic.info
149.68.141.192	terracingacyclic.io
149.68.141.192	interceptionreception.io
149.68.141.192	ballzone.info
149.68.141.192	beatifytackle.info
149.68.141.192	hikeonforgerfalsifies.info
149.68.141.192	beatifyforger.info
149.68.141.192	reception-receptionball.info
149.68.141.192	falsifiespass.io
149.68.141.192	touchdownbeatify.info
149.68.141.192	hikeandbeatifyhike.info
149.68.141.192	touchdownorinterceptionbeatify.info
149.68.141.192	jeopardisedphotochemicallybeatify.info
149.68.141.192	beatifywithpassback.io
149.68.141.192	touchdownreception.info
149.68.141.192	jeopardisedorbackjeopardised.io
149.68.141.192	backortacklefalsifies.info
149.68.141.192	towersontouchdownterracing.io
149.68.141.192	touchdowninbeatifyfalsifies.io

149.68.141.192	beatifyreception.info
149.68.141.192	towerszonereception.info

4. What email addresses did the hackers use to send these domains?

Execute in [\[Web\]](#) [\[Desktop\]](#)

[\[cluster\('kc7round1.eastus.kusto.windows.net'\).database\('SecurityLogs'\)\]](#)

```
let ips = PassiveDns
| where domain == "interceptionreception.io"
| distinct ip;
let malicious_domains = PassiveDns
| where ip in (ips)
| distinct domain;
Email
| where link has_any (malicious_domains)
| distinct sender
| take 3
```

sender
rgiillikely@yahoo.com
beastmodedifferencedifference@yahoo.com
diversitieschastelyfrightedstupendouslyrgiii@yandex.com

5. Did users click on any of the links in the phishing emails?

Yes (59 of them)

Execute in [\[Web\]](#) [\[Desktop\]](#)

[\[cluster\('kc7round1.eastus.kusto.windows.net'\).database\('SecurityLogs'\)\]](#)

```
let ips = PassiveDns
| where domain == "interceptionreception.io"
| distinct ip;
let malicious_domains = PassiveDns
| where ip in (ips)
| distinct domain;
let malicious_links = Email
| where link has_any (malicious_domains)
| distinct link;
OutboundBrowsing
| where url in (malicious_links)
| join kind=leftouter Employees on $left.src_ip == $right.ip_addr
```



```
| summarize firstSeen=min(['time']) by url, src_ip, email_addr
| take 3
```

url	src_ip	email_addr	firstSeen
http://touchdownbeatify.info/public/files/public/proposal.pptx	10.97.24 1.125	william_parker@acme.com	2022-01-07 05:15:17.421429
http://terracingacyclic.io/modules/online/online/free_money.xls	10.97.24 1.125	william_parker@acme.com	2022-01-06 00:35:13.820320
http://falsifiesacyclic.info/online/share/Resume.pptx	10.97.24 1.125	william_parker@acme.com	2022-01-02 01:14:34.331509