541.4
Microsoft Ecosystem

SEC541.4 | Cloud Security Attacker Techniques, Monitoring, and Threat Detection

SANS

Microsoft Ecosystem

Welcome to SEC541.4: Microsoft Ecosystem.

TABLE OF CONTENTS

	PAGE
Malwarebytes Attack	3
Microsoft 365	11
EXERCISE: Microsoft 365 Exchange Investigation	31
SolarWinds Attack	33
Azure Active Directory (AD)	43
EXERCISE: Introduction to KQL	63
Storage Monitoring	65
EXERCISE: Log Analytics Using Azure CLI	85
Detection Services	87
EXERCISE: Microsoft Defender for Cloud and Sentinel	107
Network Traffic Analytics	109
EXERCISE: Azure Network Traffic Analysis	126

This page intentionally left blank.

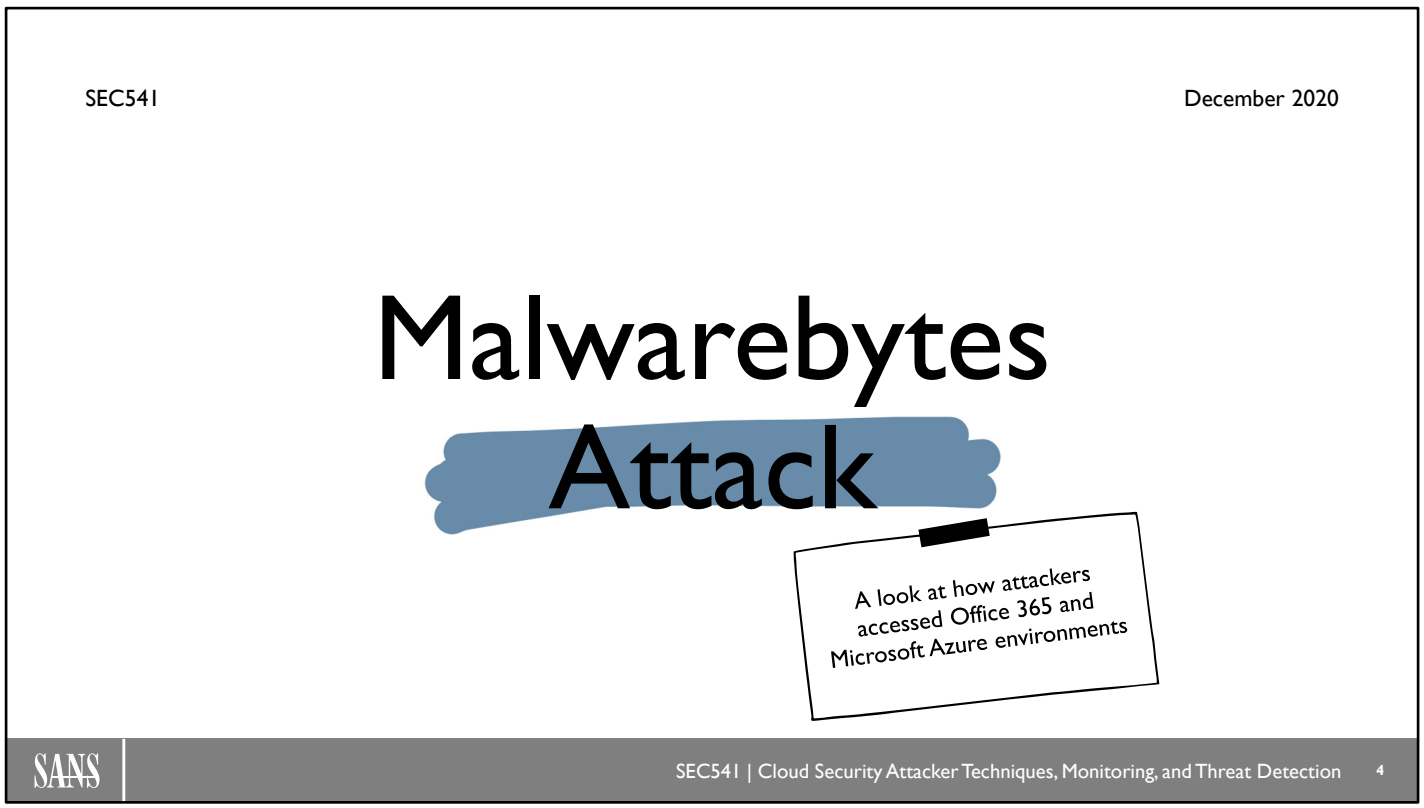
Course Roadmap

- Section 1: Management Plane and Network Logging
- Section 2: Compute and Cloud Services Logging
- Section 3: Cloud Service and Data Discovery
- **Section 4: Microsoft Ecosystem**
- Section 5: Automated Response Actions and CloudWars

Microsoft Ecosystem

1. **Malwarebytes Attack**
2. Microsoft 365
3. EXERCISE: Microsoft 365 Exchange Investigation
4. SolarWinds Attack
5. Azure Active Directory (AD)
6. EXERCISE: Introduction to KQL
7. Storage Monitoring
8. EXERCISE: Log Analytics Using Azure CLI
9. Detection Services
10. EXERCISE: Microsoft Defender for Cloud and Sentinel
11. Network Traffic Analysis
12. EXERCISE: Azure Network Traffic Analysis

This page intentionally left blank.



Unlike the previous three sections, this section focuses not just on one attack, but two. In fact, these two attacks are assumed to be related to the same attack group, APT29. This threat group is believed to be operating from the Russian Foreign Intelligence Service (SVR), is presumably responsible for the first attack we will discuss affecting Malwarebytes, and the much more devastating and far-reaching SolarWinds supply chain attack that will be discussed later in this section.

Both of these attacks involved Microsoft infrastructure—Microsoft Azure and Microsoft 365, to be exact. This will require us to explore the ins and outs of both of these services to more adequately detect attacks that may exist in other cloud environments, but also attacks specific to these infrastructures.

References:
https://attack.mitre.org/groups/G0016/
https://blog.malwarebytes.com/malwarebytes-news/2021/01/malwarebytes-targeted-by-nation-state-actor-implicated-in-solarwinds-breach-evidence-suggests-abuse-of-privileged-access-to-microsoft-office-365-and-azure-environments/

Malwarebytes Attack

Case Study

December 2020

A limited subset of internal company email messages were accessed on December 15, 2020.

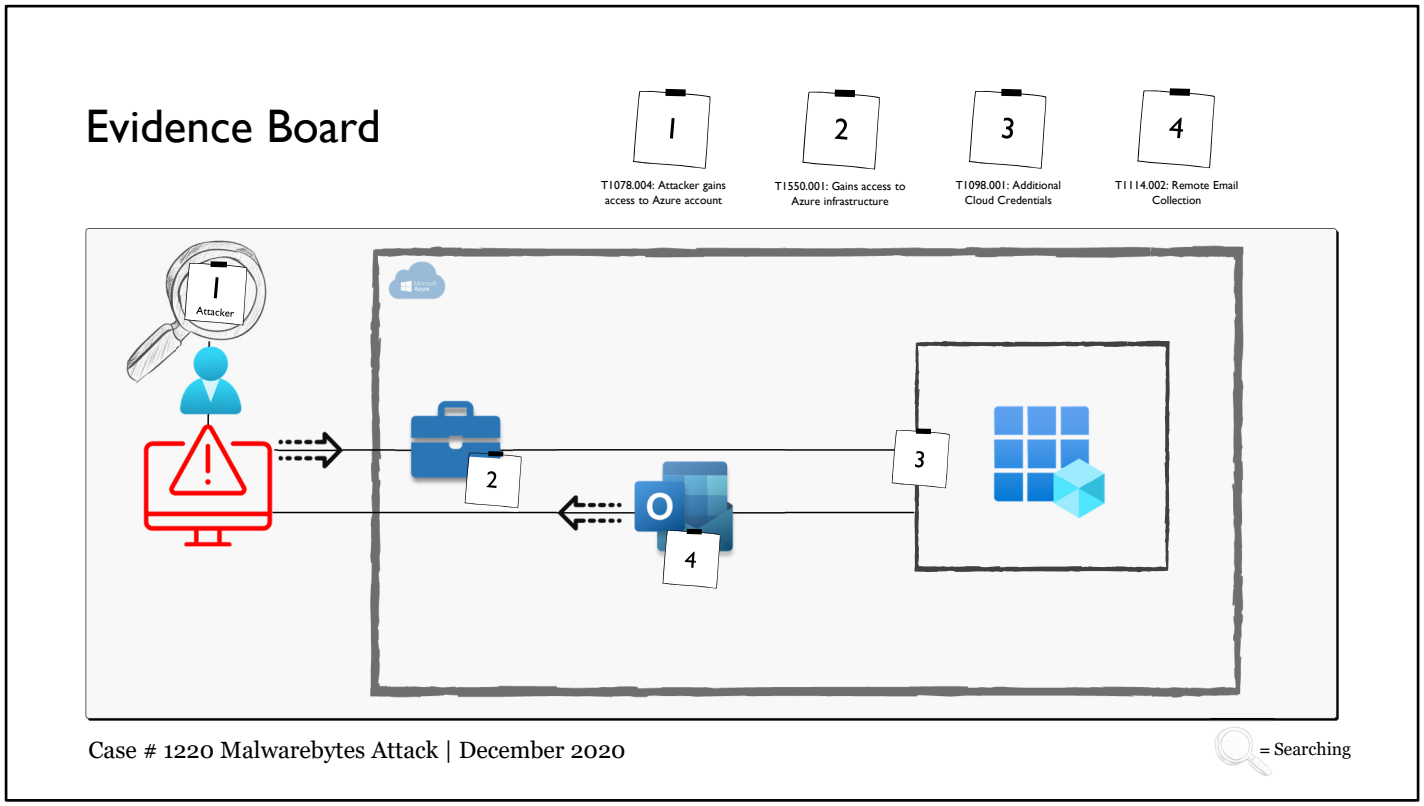
Initial Intrusion: Hackers breached its internal systems by exploiting a dormant email protection product within its Office 365 tenant.

At Risk Infrastructure: Office 365 and Azure infrastructure.

SEC541

To summarize the attack which will be discussed in more detail shortly, Malwarebytes was breached in December of 2020. The initial intrusion is believed to have been a stolen cloud account. From there, the attackers created a backdoor Azure account which gave access to the Microsoft Graph service. This service provided the attacker access to Microsoft 365 components. This leads us to the attackers' main objective: stolen internal email messages.

This means that there are two Microsoft products to attend to in this section: Microsoft Azure and Microsoft 365. The next module will focus on Microsoft 365 and afterward, once we discuss the SolarWinds attack, we will focus on various Microsoft Azure components in much more detail than we had previously.

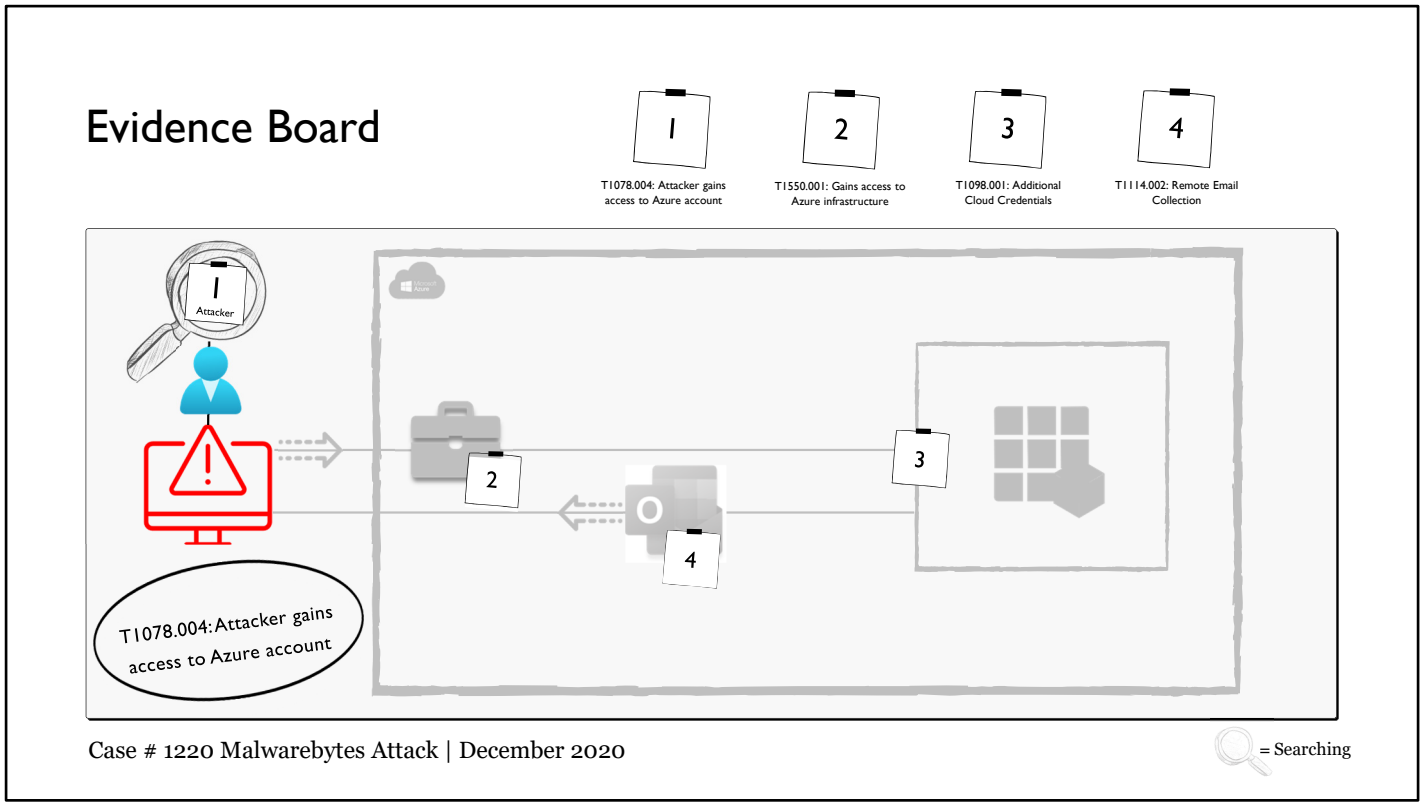


Just like previous breach discussions, the Malwarebytes attack will be explained in a step-by-step manner using our evidence board. This evidence board consists of four main MITRE ATT&CK techniques:

1. T1078.004 (Cloud Accounts)
2. T1550.001 (Application Access Token)
3. T1098.001 (Additional Cloud Credentials)
4. T1114.002 (Remote Email Collection)

There are also some new icons:

- The briefcase (2) represents stolen credentials.
- The blocks (3) represents the Microsoft Graph service.
- The Microsoft Outlook icon (4) represents Microsoft 365's Outlook application.

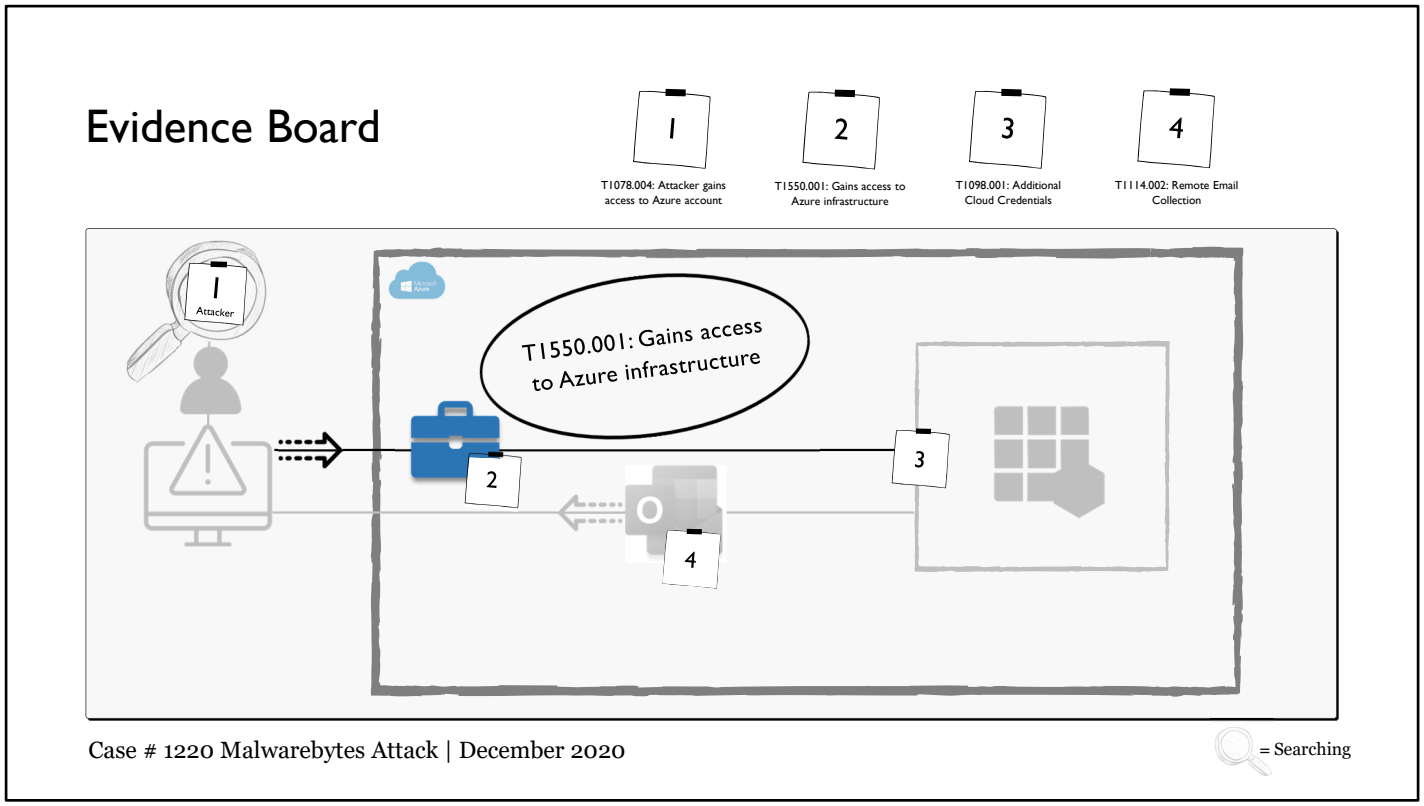


As is usually the case, the initial access to many cloud-based breaches are unclear. Was it a malicious insider? Was it via an easily guessed password? Was Azure managed identity access abused?

As reported by the Australian news outlet iTnews,[1] Malwarebytes cofounder Marcin Kleczynski believes that the initial access abused an application that has privileged access to the Microsoft Azure tenant. What is clear is that the attacker did acquire legitimate credentials to pivot into Malwarebytes' Azure tenant. This same tenant not only provides access to Infrastructure as a Service (IaaS) capabilities in Azure, but also gives the attacker access to the company's Microsoft 365 implementation.

This technique is labeled as T1078.004 (Cloud Accounts). T1078.004 is commonly used to shift the attack campaign to a cloud environment. With this newly acquired access, the attack may have visibility into the internal company infrastructure. Depending on the level of access and how quickly the victim's security team can respond, the cloud-based breach impact can range from simply listing the cloud resources (inflicting very little damage) to a complete takeover. We will see just how bad this breach got in a moment.

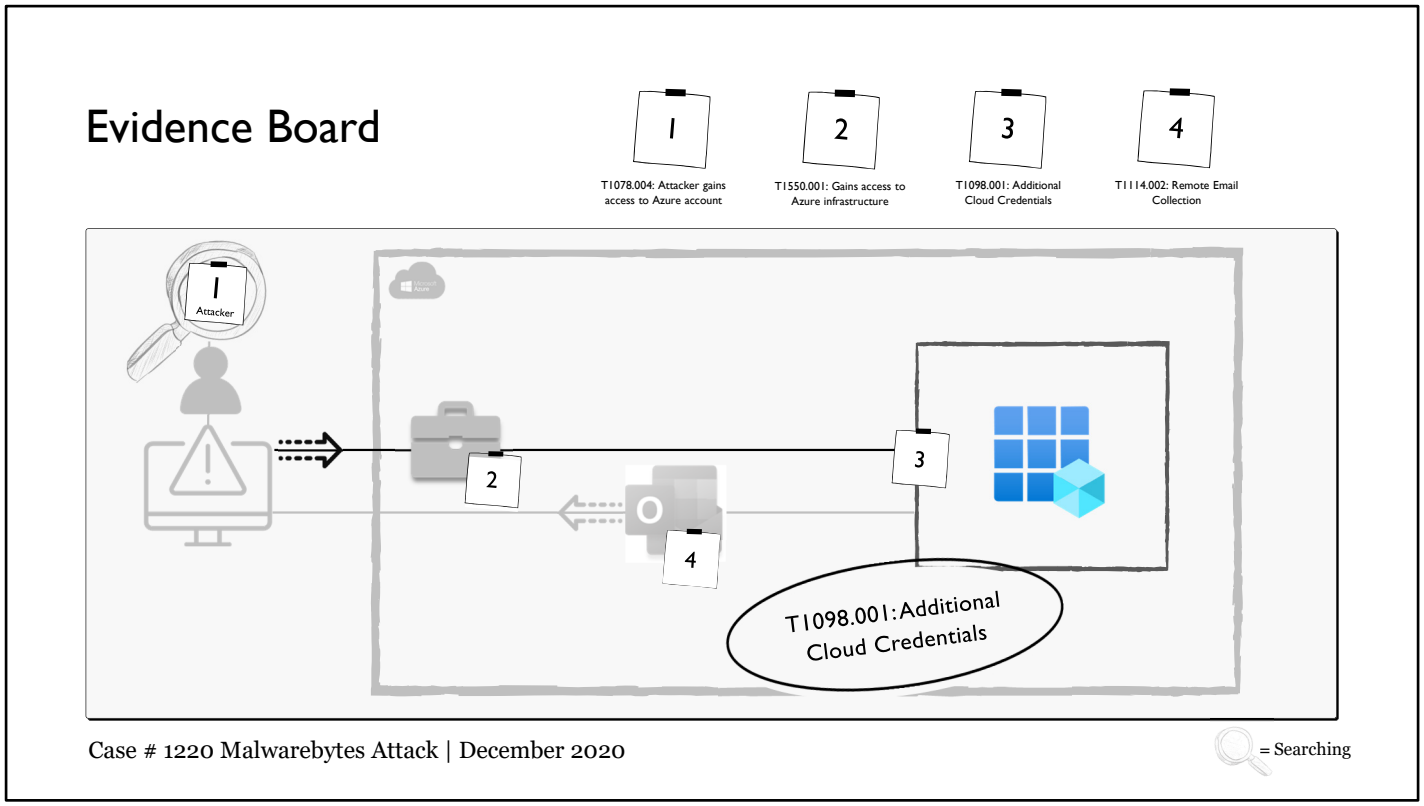
[1] https://www.itnews.com.au/news/security-vendor-malwarebytes-hacked-through-office-365-and-azure-559936



With those stolen credentials came, of course, access to Azure. Since the stolen credentials were suspected to be service principal credentials, it will be worthwhile to discuss another MITRE ATT&CK technique: T1550.001 (Application Access Token). This technique assumes that the attacker acquires some sort of token that, when sent along with an Application Programming Interface (API) request to the cloud provider, allows the action to take place (provided that the account with which the token is associated has the proper rights). [1]

For Azure service principals to be used in Azure, they must acquire a token. To do so, a request must be made to the well-known URL which is in the following format: https://login.microsoftonline.com/{your-tenant-id}/oauth2/token. It is not just a simple HTTP request. The request must be a POST request method and contain the application's client ID, client secret, grant type, and resource that is being accessed. All of this would be very difficult to guess or brute force by an attacker, so this is what leads most to believe an application or user which utilizes this service principal was compromised in some way.

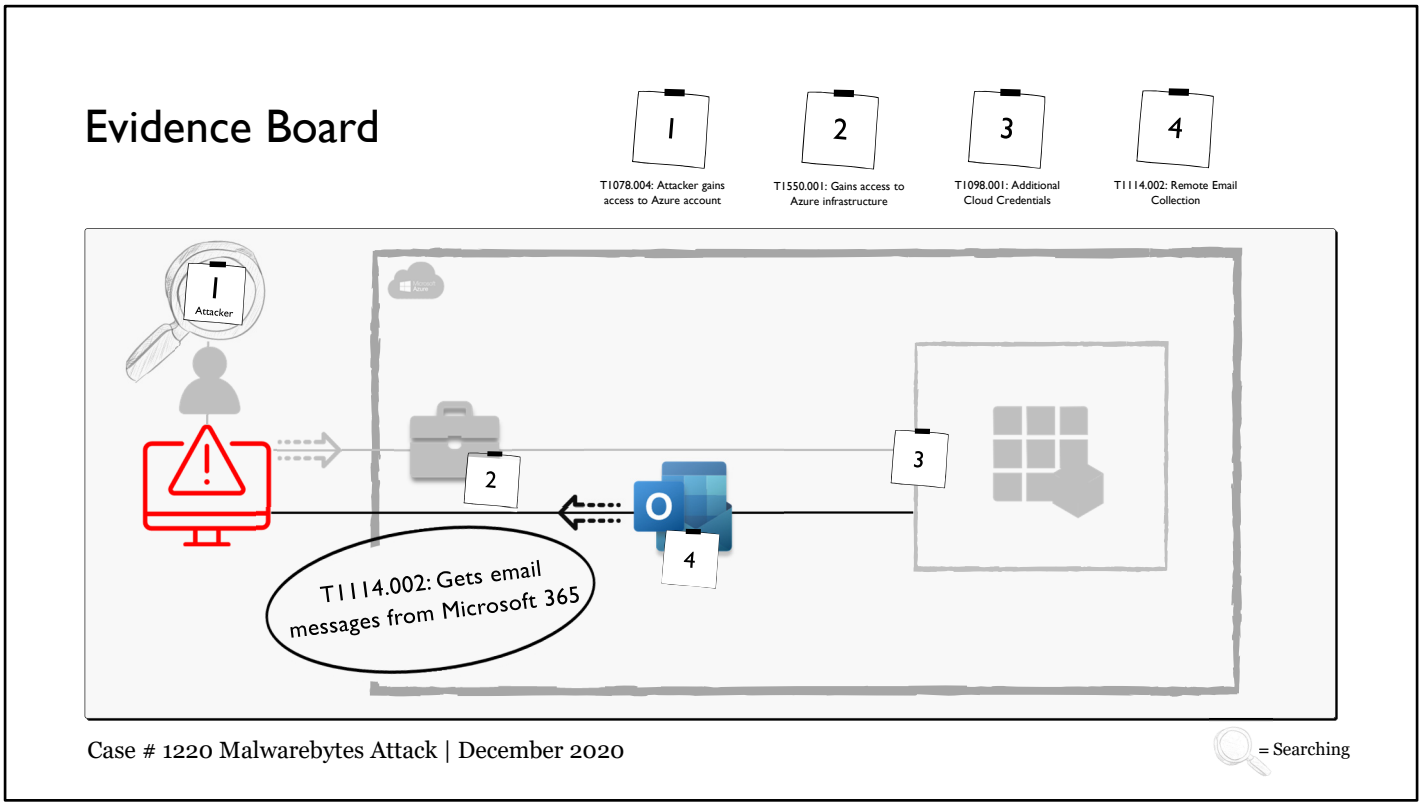
[1] https://docs.microsoft.com/en-us/azure/purview/tutorial-using-rest-apis



Also reported by iTnews,[1] the malicious adversary is believed to have generated their own credentials to maintain persistence in the Microsoft ecosystem. This also implies that the stolen credentials from the first phase of the attack did, in fact, have privileged access because your typical Azure Active Directory (AD) users do not normally have rights to create additional users (unless administrators are very careless with their Azure roles). This leads us to a brief discussion of MITRE ATT&CK technique T1098.001 (Additional Cloud Credentials).

Given these new, elevated credentials, the attacker can continue their campaign through a few different means: using the Azure Portal, using Azure command-line interface (CLI) tool access, or using an approach we will cover in this section, Microsoft Graph. Microsoft Graph allows API calls which use Azure credentials to access Microsoft 365 components.

[1] https://www.itnews.com.au/news/security-vendor-malwarebytes-hacked-through-office-365-and-azure-559936



What could go wrong if the attacker could pivot to Microsoft 365? In the case of Malwarebytes, access to company email. This is due to Microsoft Graph providing access to the many Microsoft 365 applications and their data—to include email in Microsoft Outlook.

This phase of the attack appears to be the MITRE ATT&CK technique T1114.002 (Remote Email Collection). This technique's one and only goal is to exfiltrate data from the victim's environment to an attacker-controlled system residing outside the victim's environment. In the case of Malwarebytes, the stolen email messages were limited only to some internal company email. The data inside of these email messages was undisclosed making the overall impact of the breach unknown as well.

Let's now shift our focus on a product not yet discussed in the course that is necessary to understand to adequately defend: Microsoft 365.

Course Roadmap

- Section 1: Management Plane and Network Logging
- Section 2: Compute and Cloud Services Logging
- Section 3: Cloud Service and Data Discovery
- **Section 4: Microsoft Ecosystem**
- Section 5: Automated Response Actions and CloudWars

Microsoft Ecosystem

1. Malwarebytes Attack
2. **Microsoft 365**
3. EXERCISE: Microsoft 365 Exchange Investigation
4. SolarWinds Attack
5. Azure Active Directory (AD)
6. EXERCISE: Introduction to KQL
7. Storage Monitoring
8. EXERCISE: Log Analytics Using Azure CLI
9. Detection Services
10. EXERCISE: Microsoft Defender for Cloud and Sentinel
11. Network Traffic Analysis
12. EXERCISE: Azure Network Traffic Analysis

This page intentionally left blank.

Cloud Productivity Services

- Why are organizations pushing more and more users to cloud-based productivity solutions?
  - Save organization's a lot of money by managing less hardware and software
  - Increase efficiency of work staff
  - Integrate with other cloud-vendor solutions
  - Vendors are introducing features that are only supported via cloud
- Just like anything else, these services must be managed properly
- **Microsoft 365** is, by far, the largest player in this space[1]

Organizations are moving more and more toward cloud-based productivity services for many of the same reasons they are moving to the cloud in general. First, by simply paying a flat fee per user and minimizing software installs and on-premise hardware requirements, cost can be greatly reduced.

From an efficiency standpoint, many of the vendors offerings tend to play very nice with each other in that they integrate rather easily. Also, efficiency can be increased by allowing the "always-on, connect from anywhere" option to the organization's users that cloud often brings to the table.

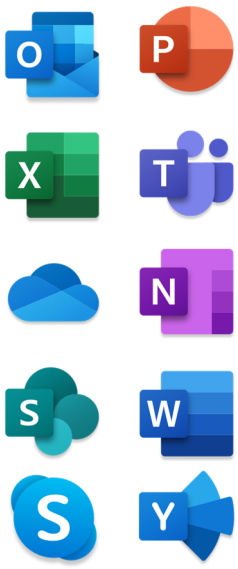
Lastly, vendors, especially Microsoft, seem to almost push users to using their cloud offerings by only offering certain features in their cloud environment and by allowing an easy transition from on-premise solutions to the cloud.

In this module, we will focus on the most popular player in cloud-based productivity services: Microsoft 365.

[1] https://techcommunity.microsoft.com/t5/office-365/office-365-hits-200-million-monthly-active-users/m-p/942813

Microsoft 365

- Provides a large number of **SaaS productivity services**
  - Office Suite
  - Email and Calendar services
  - Meetings and Voice
  - Files and Content
- As of August 2020, three different pricing models (**F3**, **E3**, and **E5**)
- Threat detection and hunting is performed quite a bit differently in this product



Microsoft offers a variety of productivity services in their Microsoft 365 offering. These services include the traditional suite of Microsoft Office products (e.g., Word, PowerPoint, Excel, and much more), email, calendar, meeting and voice options, and file and content storage.

These services are packaged together depending on the pricing model chosen. This also dictates which options within the services the company may be able to utilize and manage. We will cover many services that are only available to E5-licensed users.

Do not fret. At the end of this module, we will cover the Microsoft Graph service which gives one the capability to manually query the Microsoft API services to retrieve many of the same indicators as some of the service components we will discuss, as well as perform other detections and hunts since the APIs can expose much more data to the end user.

Threats Against Productivity Services*

- **Email-based attacks**
  - Spam
  - Phishing
- **Malware staging**
  - Serving malicious content from cloud storage
- **Data threats**
  - Theft: Using stolen credentials to access data stores
  - Inability to expire outdated files
  - Oversharing

* Not a complete list

There is no shortage of threats for these productivity services. Above are just a few examples and we will work through this module to detect evidence of these threats targeting our Microsoft 365 users.

Email threats are ever present dangers when consuming productivity services. Spam is simply annoying, but phishing can become very concerning if our users are tricked into following the malicious sender's actions, clicking dangerous links, or opening malware-laden attachments.

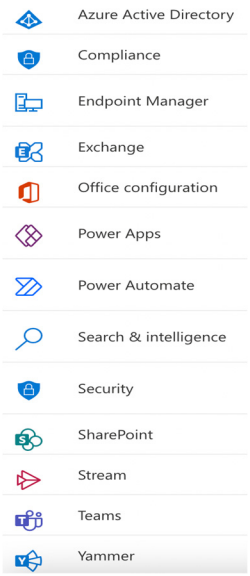
These productivity services could also be leveraged to store malware for later stages of a campaign against either your organization or another organization. This could mean that your cloud services could be used to inflict pain on others!

And last, but not least, as was mentioned many times in previous sections for this course, there could be threats to the data stored in these services as Microsoft 365 comes with a variety of services that could store sensitive information.

Oftentimes, a few of these types of attacks could be used during the same campaign.

Microsoft 365 Admin Centers

- There are a large number of **admin centers** in Microsoft 365
  - Used to manage multiple service components at once or individual components
  - Capabilities within these admin centers are limited
    - Little, if any customization
    - Customizations, when available, are quite rigid
- We will cover three of these:
  - **Exchange**
  - **Compliance**
  - **Security** (now called **Microsoft 365 Defender**)



One should not expect, as soon as their appropriate Microsoft 365 license is procured, that their users and data are secured and that all of the appropriate logging and detections are in place. In fact, very little in the way of security is actually configured, which should not be a foreign concept when using cloud services designed to "just work."

With most Software as a Service (SaaS) solutions, the customer typically has very little influence over the security (i.e., prevent, detect, respond) other than maybe some user management and sharing options. Microsoft 365, however, does give much more power to the organization subscribed to the service. Even still, you will find that the options presented to the end users are not as fully featured as what you may be used to in, say, Microsoft Azure. In fact, some of the detection strategies we will later discuss require the end user to get creative in their approach by querying the Microsoft Graph service directly to gain more insight into the Microsoft 365 resources.

The various components of Microsoft 365 can be configured using a multitude of admin centers. There are many of them, but, over the course of the next few pages, we will dive more deeply into three in particular to describe both the defensive strategies you can employ and, more importantly, how to detect malicious activity in the Microsoft 365 accounts. These admin centers are Exchange, Compliance, and Security (now called Microsoft 365 Defender).

Exchange Admin Center

- Detections available via user- and Microsoft-generated policies
  - **Mail flow**: Can be used to discover mail loops, slow transport rules, and new forwarding rules
    - Security team can be notified via email of suspicious activity
  - **Spam filtering**: Discover likely spam messages
    - Places message into quarantine
    - Can also be used to detect potential phishing from **cousin domains**
  - **Message traces**: Query for email based on sender and receiver addresses
- **Note**: Email content can only be viewed if the email was placed into quarantine! Microsoft Graph will rescue us later.

The first admin center to be discussed is a very crucial one to combat email-based compromises: the Exchange admin center.[1] There are a host of options to control what your users can and cannot do, but we will focus instead on how to detect suspicious activity in users' email accounts (as you would expect in a class such as this).

When you arrive at the Exchange admin center, you will be surprised (or maybe not so surprised based on the discussion on the previous page) that there is very little which can be configured when attempting to identity or hunt for threats. Not all is lost, however. Three very key features that should be investigated are mail flow, spam filtering, and message traces.

In August of 2020, an attack plagued SANS Institute, of all companies, involving an unsuspecting employee that was using the company's productivity suite provider, Microsoft 365. This user fell victim to a phishing attack that, according to SANS's own incident response team, led to a mailbox forwarding rule which was responsible for sending 513 email messages to a suspicious email address. These email messages contained approximately 28,000 PII records.[2]

These types of attacks could happen to any organization but, with the help of the mail flow option, the attack could have been recognized much earlier. Mail flow detections can stop not just new and unexpected forwarding rules from being implemented, but it can also spot issues with the mail services themselves such as mail loops (email being forwarded back and forth between two or more mailboxes—likely due to forwarding rules as well) and slow transports rules.

Spam filtering can be useful to identify suspicious domains sending email to your users and, instead of relying on the users to make the "right call" and disregard the message, you can intercept the message and place it in quarantine. There is one caveat: as of October 2021, the only way to create spam policies is either using the Classic Exchange Admin Center (within the Protection options) or the Security Admin Center (under Threat Management).

[1] https://admin.exchange.microsoft.com
[2] https://www.sans.org/blog/sans-data-incident-2020-indicators-of-compromise/

Cousin Domains

- Also known as “fuzzy domains,” look very similar to legitimate domains
- Often used in phishing campaigns to trick users into following links or taking other actions as it appear legitimate
  - **www.email.com** and **wwwemail.com** look very similar
  - Users may miss the fact that there is a “.” missing in the URL
- Our email systems should block these cousin domains
- Tools like **dnstwist** can help identify possible cousin domains
  - Online: https://dnstwister.report
  - Standalone: https://github.com/elceef/dnstwist

Oftentimes, targeted phishing attacks may try to lure an unsuspecting user to click on a link. This link may look very similar to a legitimate link, but could be controlled by the attacker or, at the very least, contain malicious content that the attacker would like to leverage against this victim. These look-alike domains are known as cousin domains. For example, if a legitimate domain that a user is likely to visit during a normal workday is sec541.com but an attacker owns and operates a domain of secc541.com, the victim, who is not paying close attention, may click the link. You may have missed it as well—the second URL has an extra "c" in it. This is very easy to miss.

A technique that the security team can leverage is to first, look up what those cousin domains may be and second, block access to those domains by using various techniques. The most common would be blocking the domains at a forward web proxy or “sinkholing” the domains. In other words, returning a DNS answer of 0.0.0.0—which is unreachable by the victim. To determine what these cousin domains may be, the security team may utilize a third-party tool like dnstwist.

The security team may also limit any incoming email from these cousin domains. Let's say that the legitimate CEO's email address is ceo@sec541ryanic.onmicrosoft.com. A user may be inclined to do whatever someone sending from ceo@sec541-ryanic.onmicrosoft.com may want them to do (notice the extra dash in the address?)—including sending company secrets or other sensitive data to this spoofed “boss.”

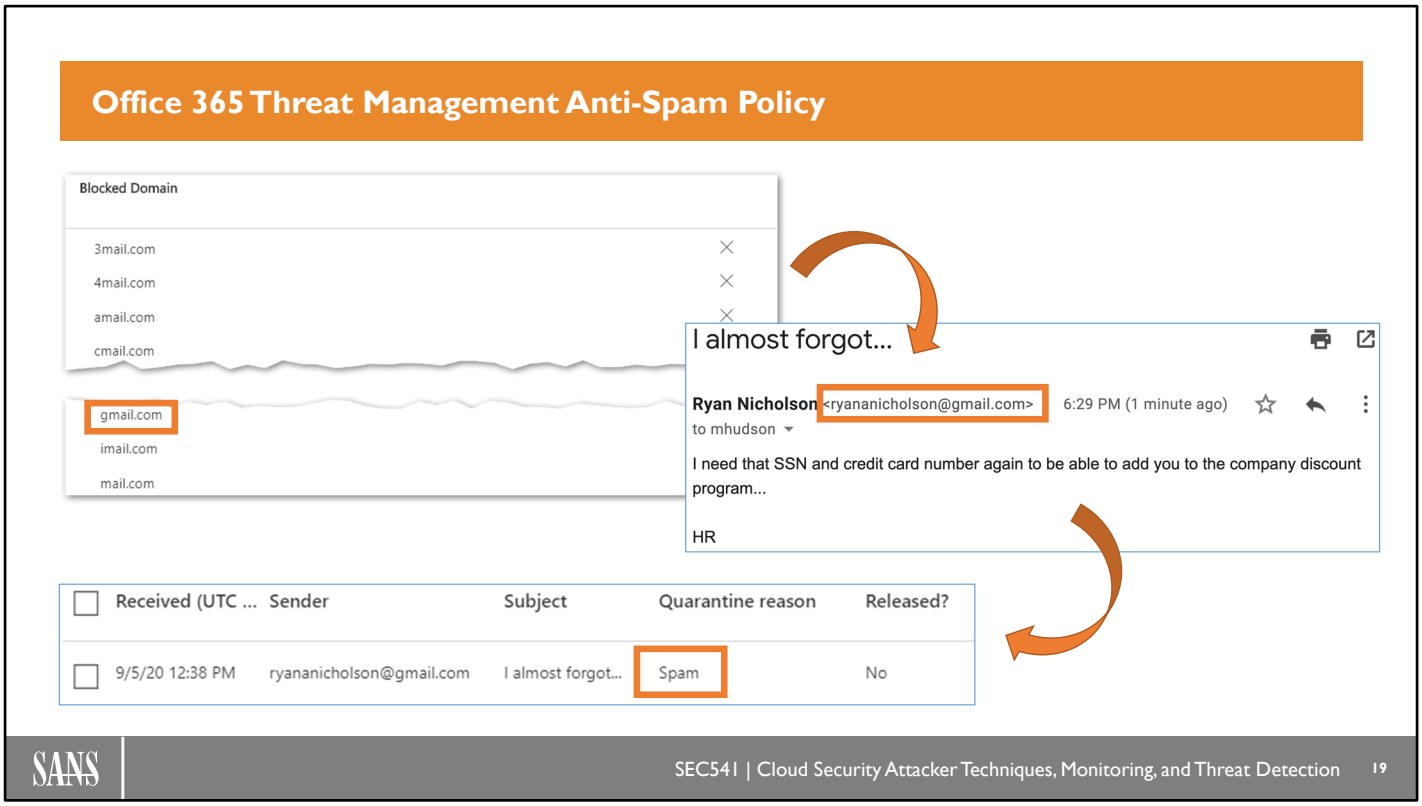
dnstwister.report Results for "email.com"

Found (119) Available (75)

Found Domain	IP Address / A record	MX found?	
émail.com (xn--mail-fpa.com)	54.208.77.124	✓	analyse
ēmail.com (xn--mail-9ua.com)	×	✓	analyse
rmail.com	208.97.170.19	✓	analyse
mail.com	82.165.229.87	✓	analyse
zmail.com	82.165.227.135	×	analyse
eemail.com	216.40.47.17	✓	analyse
meail.com	72.52.179.174	✓	analyse
emnail.com	35.186.238.101	✓	analyse
elmail.com	69.172.201.153	✓	analyse

As mentioned, the first step to block inbound email from cousin domains is to identify those cousin domains. Here we see results from dnstwister.report associated with email.com. Notice how many of them (marked "Found") are already registered and could be used against this organization. These should be the first 118 domains to block (the 119th address is email.com, which is legitimate).

Although not yet registered, the 75 cousin domains marked as "Available" could eventually be used against the organization, so blocking those would not be a bad idea either. In fact, if the organization were to register those domains, an adversary will not have the opportunity. Although potentially costly, it does prevent the use of several cousin domains.

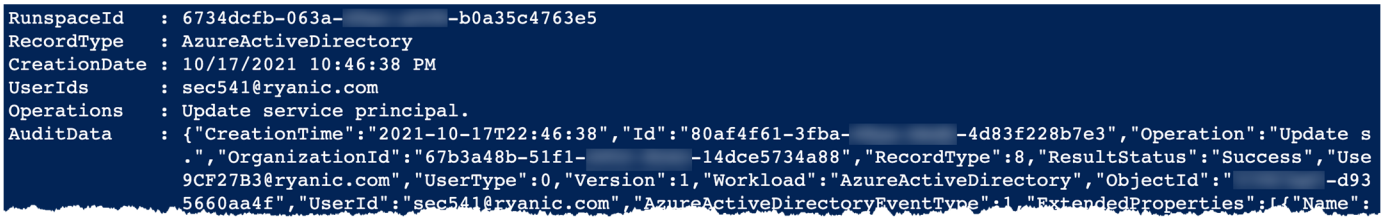


Now that those cousin domains have been captured, an anti-spam policy can be created. Not only will this block spam from those domains, but it will also prevent potential phishing as these email messages are not allowed to enter a user's mailbox without administrative action. In other words, the email is placed in quarantine.

You can see one of the cousin domains for email.com is gmail.com and ryananicholson@gmail.com is trying to send a phishing email to our friend and Microsoft 365 Exchange user, Martha Hudson. Luckily for us, the security team added these cousin domains, and the email did not make it to the not-so-security-minded individual.

Compliance Admin Center

- Used primarily for meeting compliance needs of the organization
- A few options exist for detections:
  - **Audit**: Running list of all Microsoft 365 activity
  - **Content search**: Find data of interest within documents, email, or Skype messages
- Data can be short-lived in the Compliance admin center, so **`Search-UnifiedAuditLog`** can retrieve/search the audit data



The next admin center we will explore is the Compliance admin center. This service has many features to help meet the various compliance needs your organization may be facing, such as retaining data for the required amount of time, identifying data that falls in scope of regulatory requirements, and managing privacy data to name a few. However, we will focus on how this service can be leveraged for detections of malice.

The audit component is used to track user activity within Microsoft 365. This information would prove very valuable when investigating a potential incident or even collecting and analyzing during a hunt. Normally, most data in Microsoft 365 related to user activity or detected threats is short lived. Thirty to ninety days is a pretty typical lifespan for this data. With auditing, you have the power to retain this trail of events for the following intervals: nintey days, six months, nine months, one year, or ten years.

To programmatically access this data outside of the graphical user interface that is the Compliance admin center, a PowerShell cmdlet is available to retrieve and query the audit log called `Search-UnifiedAuditLog`. This cmdlet requires the use of another cmdlet called `Connect-ExchangeOnline` (part of the `ExchangeOnlineManagement` module) in order to query Microsoft 365 services. The output shown in the image above is the result of these two PowerShell commands from within Azure Cloud Shell:

```
Install-Module ExchangeOnlineManagement
Connect-ExchangeOnline
Search-UnifiedAuditLog -StartDate "10/12/2021" -EndDate "10/19/2021"
```

This entry is showcasing a change to an Azure Active Directory service principal which is used to programmatically access the Azure environment. Is this expected? It may be. This type of entry is common in many environments. If the activity is not expected (i.e., there are no known changes that the security team is aware of) it may be cause for concern. This could be privilege escalation or backdoor access to Azure.

Another very interesting component is the content search. Here, one can create a custom search looking for data of interest across a few data sources such as documents, email, and even Skype for Business messages within the Microsoft 365 account.

Data Loss Prevention (DLP)

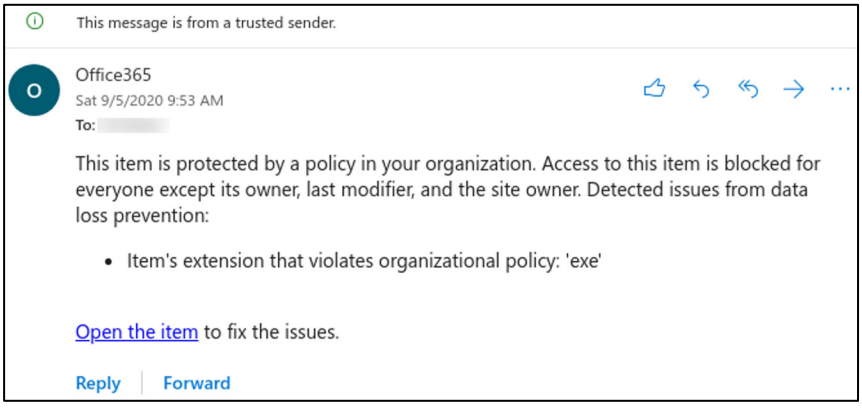
Compliance admin center allows for the creation of DLP policies:

- Step 1: **Identify** data to be protected
  - Many built-in options (e.g., Financial, Medical and Health, or Privacy)
  - Those options contain many regulatory data types (e.g., U.K. Access to Medical)
  - Can create your own
- Step 2: **Name** and **describe** the policy
- Step 3: Choose **storage locations** (i.e., Exchange, SharePoint, OneDrive, and/or Teams)
- Step 4: Create **rule conditions** and **actions**
- Step 5: Decide to enable the policy right away, test the policy (e.g., audit), or keep the policy turned off

Data Loss Prevention (DLP) is a great feature to prevent and detect malicious use of organization data within the Microsoft 365 services. However, this does, again, need to be set up by the end users and you can do so using DLP policies. To create these policies, there are a few steps that the Microsoft 365 Compliance admin center will walk you through:

- Step 1: Identify the types of data to be protected
- Step 2: Name and describe the policy so that viewers can make sense of the policy and why it would have triggered a violation
- Step 3: Choose which of the available storage locations the policy pertains to
- Step 4: Create specific rule conditions (when to trigger) and actions (what to do when triggered)
- Step 5: Enable, test, or disable the policy

Once in place and a violation occurs, both the security team and the user committing the violation can be notified via email as shown in the example below:



This is a result of the user placing an executable file in OneDrive—something that is not permitted by the organization, and enforced by the DLP policy.

Microsoft 365 Defender

Requires a higher or dedicated subscription rolling many capabilities into one service:

- **Incidents and alerts**: Roll-up of all identified suspicious activity
- **Hunting**: Manually search for data or activity of interest
  - Also allows for **custom detection rules**
- **Threat analytics**: Microsoft-provided analytics to identify malice
- **Endpoints:** Perform inventory, vulnerability management, and configuration management of users' devices
- **Email and collaboration**
  - **Explorer** is a new feature that can scan email for content of interest
  - **Attack simulation training** sends benign phishing email to internal users
- **Auditing:** Just as we saw before in the compliance admin center

Microsoft 365 Defender contains a massive amount of defensive capabilities but does not automatically come with all Microsoft 365 subscriptions. As with many licensing models, it can be a bit confusing as to how to acquire Microsoft 365, but Microsoft has a handy guide to help you out at https://docs.microsoft.com/en-us/microsoft-365/security/defender/prerequisites?view=o365-worldwide.

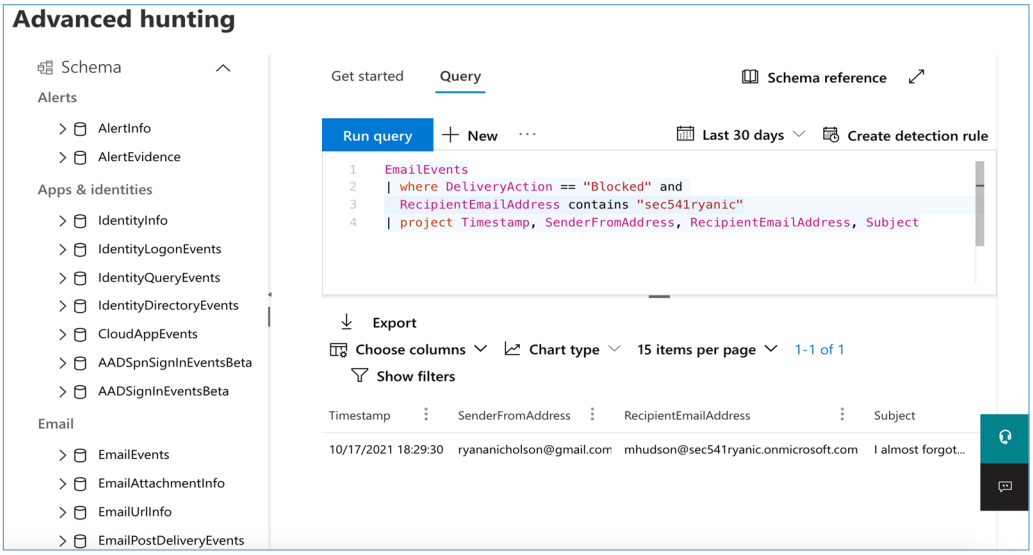
Once enabled, you may find that some of the capabilities offered are redundant to what was discussed earlier, but many more capabilities are now rolled into this one service, saving analysts from clumsily moving from one admin center to the next. For instance, once Microsoft 365 Defender is enabled, you can now centrally view all Microsoft 365-related incidents and alerts in one handy dashboard. Not only this, hunting for data throughout the Microsoft 365 ecosystem becomes much easier within the hunting component. Similarly, if it is email that you are looking to sift through, the Explorer component of email and collaboration can come in quite handy.

Microsoft also helps organizations identify many known threats using the threat analytics feature. These analytics are vendor provided and are frequently updated as new exploits and attack campaigns are discovered "in the wild."

Speaking of threats, what about insider threats related to users clicking an email link or opening attachments? The Attack simulation testing, while not a technical preventative or detective control, allows an organization to help train their users to recognize phishing email. The service offering will send realistic phishing email to your workforce and keep track of the users that reported, ignored, or followed the fictitious attacker's instructions.

Lastly, if devices are enrolled, Microsoft 365 Defender can provide a host of options to protect these endpoints. For example, the organization can have an easy-to-read inventory of systems, manage vulnerabilities, and even conduct configuration management of these endpoints.

Microsoft 365 Defender: Advanced Hunting



Exploring everything which comes from Microsoft 365 Defender can take many pages and hours of discussion, but we will focus on one really nice feature for now: advanced hunting. This feature allows one to sift through data from the following Microsoft Defender offerings:[1]

- Microsoft Defender for Endpoint
- Microsoft Defender for Office 365[2]
- Microsoft Cloud App Security
- Microsoft Defender for Identity

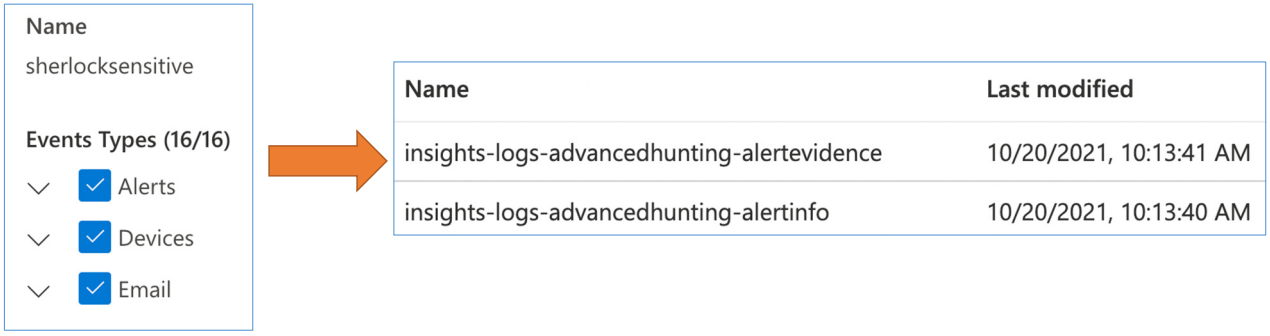
If you notice in the image above, the syntax for the queries should look quite familiar—it is Kusto Query Language (KQL)! The same logic applies here: the data is stored in tables and the tables can be queried using the same KQL syntax learned in previous modules. There is an example search shown above.

To break this query down, we are first looking at the `EmailEvents` table.[3] This table contains information about all email messages that were processed by Microsoft 365 Defender for Office 365. One of the available columns in this table is `DeliveryAction` which signifies whether the email reached its intended destination or was blocked. In this query, we are interested in Blocked messages. Another `where` clause is to check if the email is incoming. This is performed by checking the `RecipientEmailAddress` to contain the organization's domain (`sec541ryanic`). Finally, to limit the column output to the most important pieces of information, the `Timestamp`, `SenderFromAddress`, `RecipientEmailAddress`, and Subject are displayed using `project`.

[1] https://docs.microsoft.com/en-us/microsoft-365/security/defender/advanced-hunting-overview
[2] At the time of this writing, this service was still called Microsoft Defender for Office 365. It is likely to change as Microsoft is using the designation of "Office 365" less frequently.
[3] https://docs.microsoft.com/en-us/microsoft-365/security/defender/advanced-hunting-emailevents-table

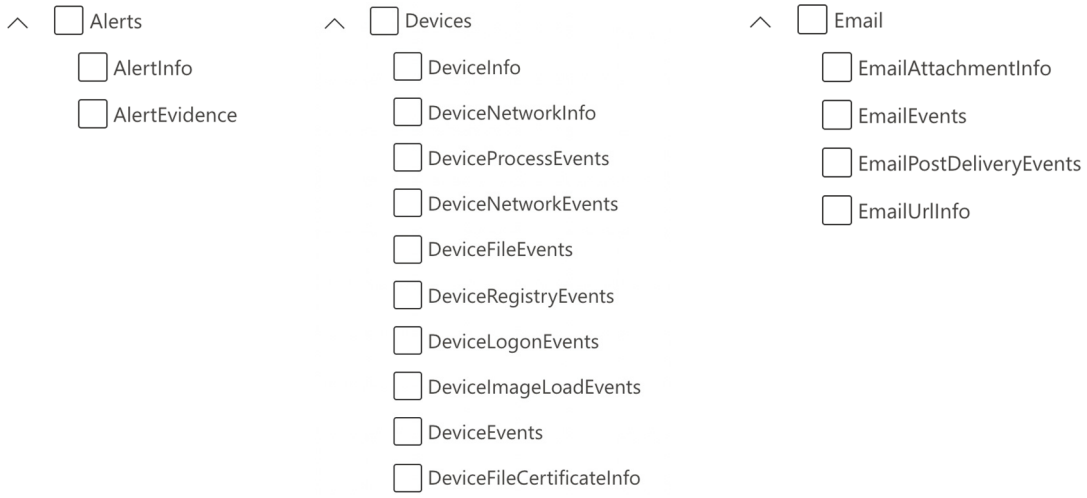
Streaming API

- Microsoft 365 Defender data is limited to **thirty days** in the platform itself, so exporting the data is crucial
- Under **Settings → Microsoft 365 Defender**, the **Streaming API** can be leveraged to ship data to an **Azure service**



This should come as no surprise at this point, but Microsoft 365 Defender data is only available for thirty days in the service itself. This means that, yet again, one must come up with a plan to export this data into somewhere more long-term. The built-in method to do that within the Microsoft 365 Defender service is to leverage what is called the streaming API. This setting is very straightforward in that the user must select which event types to export and where to export it.

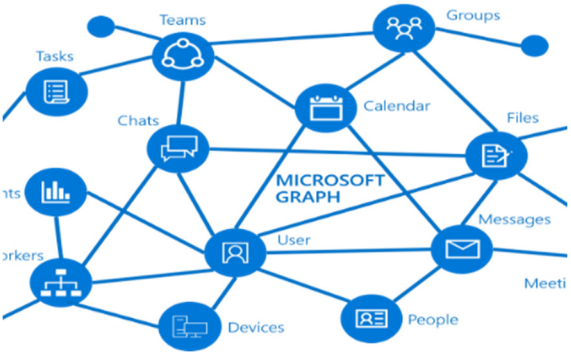
The event types that can be selected fall into three categories:



The export options are limited to either an existing Azure Storage container, an existing Azure Event Hub, or both simultaneously. In the example above, the data is being sent to an Azure Storage account and, when the data arrives, it is placed inside a container in folders representing the type of data stored: in this case, insights-logs-advancedhunting-alertevidence and insights-logs-advancedhunting-alertinfo. There would obviously be more if more data were populated in the Microsoft 365 Defender service to export.

Microsoft Graph

- What if Microsoft 365 Defender is not available or does not provide the capabilities you need?
- Microsoft Graph allows users to access a large amount of data related to Microsoft 365
- Two options to access this dataset:
  - **Microsoft Graph API**: RESTful web queries
  - **Microsoft Graph Data Connect**: Azure Data Factory



You may find that either Microsoft 365 Defender is unavailable given your current licensing model or that, perhaps, this service does not suit all of your needs. Microsoft Graph can be leveraged by anyone (with a valid Microsoft 365 license of course) to access a large amount of Microsoft 365 event data.

At the time of this writing, the following services and features can be searched using Microsoft Graph: *Bookings, Calendar, Delve, Excel, Microsoft 365 compliance eDiscovery, Microsoft Search, OneDrive, OneNote, Outlook/Exchange, People (Outlook contacts), Planner, SharePoint, Teams, To Do,* and *Workplace Analytics.*

There are currently two options to utilize Microsoft Graph to acquire this Microsoft 365 data: using RESTful web queries and interacting with the Microsoft Graph Application Programming Interface (API) or using a service called Microsoft Graph Data Connect.

Before using the Microsoft Graph API, some setup on the user's part will be required that will be explored in an upcoming case study very shortly. This will be worth the extra effort because, as soon as the setup is complete, one can ask for any Microsoft 365 artifact at any time using rather simple, and authenticated, HTTP requests.

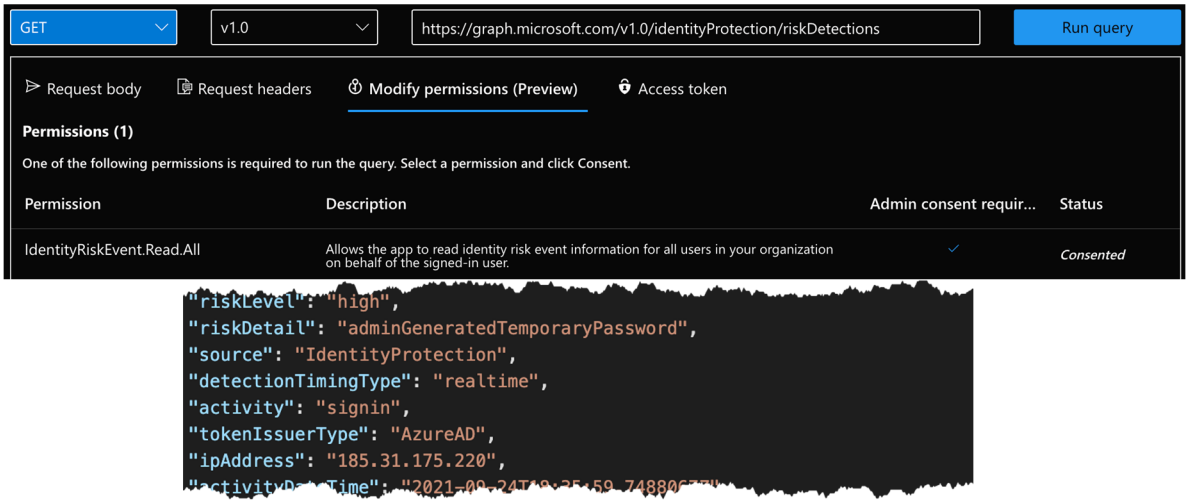
If you would like to, instead, copy the Microsoft 365 data to Azure services, Microsoft Graph Data Connect can use Azure Data Factory to accomplish this at user-configurable intervals.

Image reference:
https://docs.microsoft.com/en-us/graph/images/microsoft-graph.png

References:
https://docs.microsoft.com/en-us/graph/overview
https://docs.microsoft.com/en-us/graph/data-connect-concept-overview

Microsoft Graph Explorer

Online application to test Microsoft Graph API requests



To begin experimenting with Microsoft Graph, Microsoft makes available a very interesting resource called Microsoft Graph Explorer located at https://developer.microsoft.com/en-us/graph/graph-explorer. This service provides many useful sample queries along with a handy interface to modify, if necessary, these queries to access data in Microsoft 365. Before you can start using it, you must authenticate with valid Microsoft Azure Active Directory credentials.

Once connected, you have the ability to select one or many sample queries for many of the Microsoft 365 services. These queries range from simply gathering audit data of past actions, to acquiring documents and email, to even making changes in the environment.

Once a sample query is selected, the HTTP method (shown above as GET), the Microsoft Graph version (shown above as v1.0), and URL (shown above as https://graph.microsoft.com/v1.0/identityProtection/riskDetections) is populated for you. This was all a result of simply selecting the "get risk detections" sample query. It is here that you can make any necessary adjustments to craft a custom request. Not captured in the above image is the ability to see and modify the request body and headers, if necessary.

Before the query can be successful, an admin may be required to consent to the various permissions needed to access the required Microsoft 365 data and resources. This is as simple as clicking a Consent button if the current user is one of the admins. Otherwise, permissions must be added to the user account (as will be shown in the following case study) to allow the account to perform these actions.

Once the permissions are granted and the query executed, results will appear in the bottom pane of the Microsoft Graph Explorer web page. The data from our example query is in JSON format (as is most cloud output returned to users as you have discovered many times in class thus far).

Case Study: Use Microsoft Graph for Threat Hunting

- In this case study, we will need a more flexible approach to sift through email data looking for an attachment of interest:
  - Could have been received by anyone in the company
  - Has MD5 hash of **`3a87a2715eeeda19eba14aa9c5888b1f`**
- This is achieved in four main steps:
  - Create an **App registration** in Azure Active Directory
  - Ensure the App can access Microsoft Graph with proper **permissions**
  - Generate a **bearer token** for authorized access
  - Loop through all email accounts and messages, searching for indicator

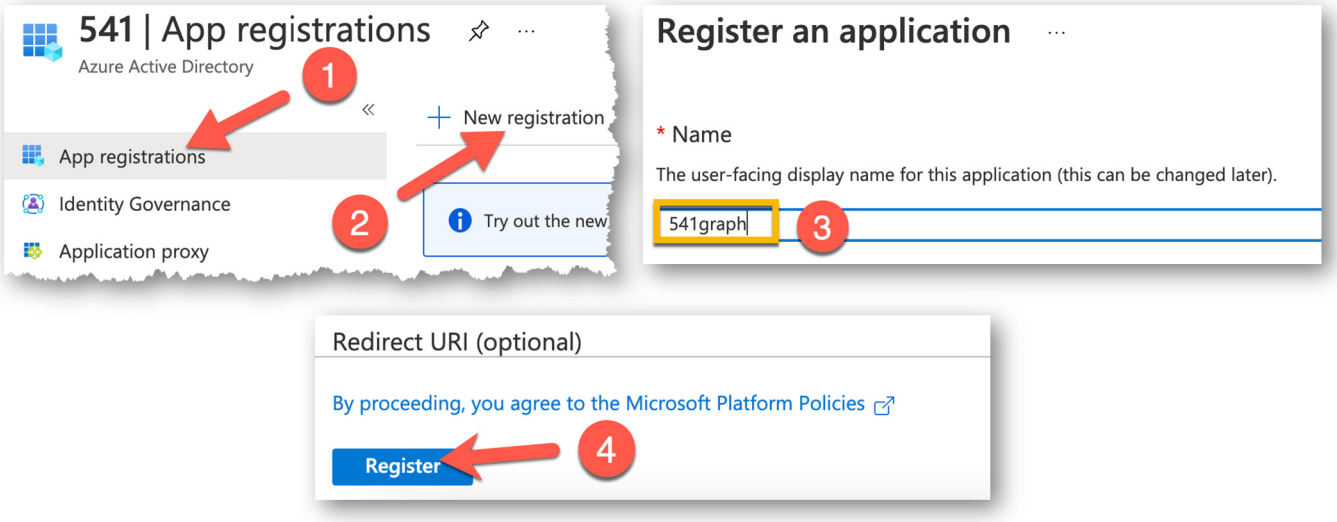
In this case study, we will assume that there is a particular file of interest that we deem malicious that may have been delivered to a Microsoft Exchange user. The data of interest is an email attachment that, when hashed using the MD5 algorithm, outputs the value of `3a87a2715eeeda19eba14aa9c5888b1f`.

As you saw earlier, a fantastic resource to dig through email data would be Microsoft 365 Defender and its advanced hunting options, but unfortunately it lacks a means to search for MD5 hashes of email attachments. We will instead leverage Microsoft Graph to conduct a deep investigation of artifacts in the Microsoft 365 platform.

There are four main steps to meet the needs of this case study, which we will walk through in the following pages:

- We will need to authenticate with Azure Active Directory to perform these Microsoft Graph queries, so an App registration must be created in the service.
- Once the App registration is complete, the application must have proper access to Microsoft Graph (similar to admin consent in Microsoft Graph Explorer).
- Before launching the query, a bearer token must be acquired from Azure Active Directory to send along with the query as proof of authentication and authorization.
- The last step is launch one or more queries to acquire all attachments and hash them—looking for a match to the malicious MD5 sum.

Case Study: Create App Registration

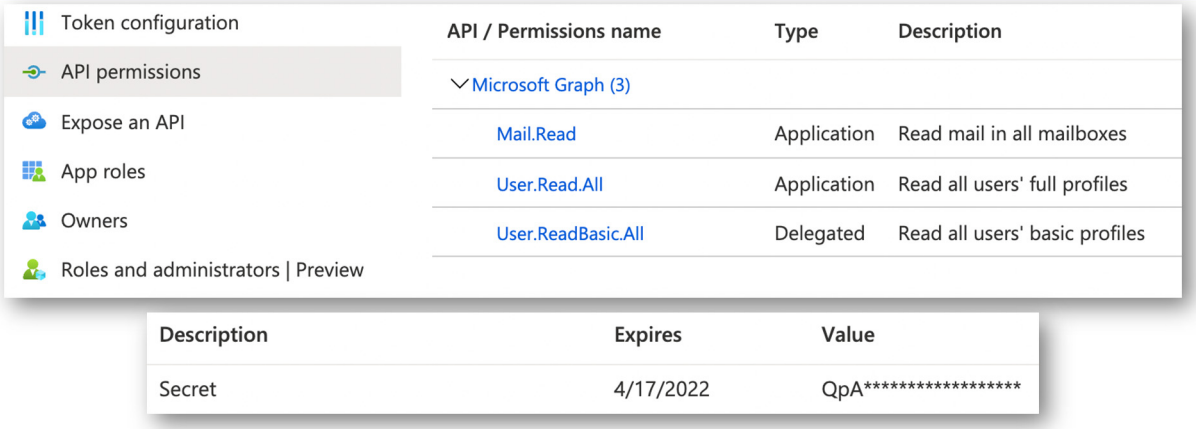


The first step of this process is to create the App registration. This is performed by navigating to Azure Active Directory with an account that has proper permissions to register an application. Once in Azure Active Directory, there is a dedicated App registrations section which you will navigate to by clicking on the proper selection in the left-hand pane (1).

Creating an App registration is very similar to creating most resources in Azure—click the "+ New registration" button (2). When you arrive at the Register an application page, you simply need to give the application a name. In this case, we have given it the appropriate name of 541graph (3).

After the name is entered, you can click on Register at the bottom of the page (4). This first step, which was quite easy, is now complete. On to the next step.

Case Study: Adjust App Permissions and Acquire Token




```
1  TOKEN=$(curl -sXPOST https://login.microsoftonline.com/67b3a48b-51f1-495f-8fd3-
   541541541541/oauth2/token -d 'grant_type=client_credentials&client_id=37f67ed7-
   d93c-49f0-8dab-541541541541&client_secret=QpA7Q~4bRz3QxFiqD6TRY_EYV541541541541
   &resource=https://graph.microsoft.com' | jq -r '.access_token')
```


On this page you will see both the second and third step. The second step requires you to add permissions to the application so that it can access the appropriate resources needed to conduct the investigation. This is performed by navigating to API permissions in the left pane when viewing the application resource.

From here, you will add all Microsoft Graph permissions that will be required. In this case, since you will need to have access to users' email you will need the Mail.Read permission. There are two more permissions that are required for this use case that may seem unrelated but are very necessary: User.Read.All and User.ReadBasic.All. Those permissions are needed because, when querying for email messages, you must know the user ID in Microsoft 365. The final two permissions are used to acquire those user IDs.

The third step, shown in the last screenshot and code block, signifies generating a secret which is used to acquire a bearer token and acquiring that token. The secret is used, along with the application ID (also known as client ID) and application URL to authenticate with Azure Active Directory. In turn, the Azure Active Directory service will respond with a bearer token proving your successful authentication. This token is then used in the next step when issuing queries to Microsoft Graph and provided in the Authorization HTTP header.

Case Study: Use 365emailhasher.ps1 to Hash Attachments

Use PowerShell script along with bearer token to hash all email messages with attachments

```
1  pwsh -c ./365emailhasher.ps1 $TOKEN
```

```
Name                           Value
----                           -----
Recipient                      ryan@sec541ryanic.onmicrosoft.com
Message Subject                Super secret message
Received time                  10/17/2021 10:40:25 PM
Attachment Hash                3a87a2715eeeda19eba14aa9c5888b1f
```

We have a hit!

Now that the bearer token is acquired, you can begin sending queries to Microsoft Graph. Be aware that this token has a finite lifespan (sixty minutes by default) so if you need to make many queries over a long timeframe, you may need to acquire a new bearer token to continue interacting with Microsoft Graph.

To meet this case study, you would be required to send many queries to Microsoft Graph:

Gather user IDs of all Microsoft 365 users
- Query URL: https://graph.microsoft.com/v1.0/users

For each user, gather all email metadata
- Query URL: https://graph.microsoft.com/v1.0/users<userId>/messages

For each email message, gather the attachment information
- Query URL: https://graph.microsoft.com/v1.0/users/<userId>/messages/<messageId>/attachments

After these queries are made, you would still need to extract the attachment's raw bytes, hash them, and then compare the hash to the MD5 sum mentioned previously. Fortunately, one of the course authors created a handy tool to do all of this for you called 365emailhasher.[1] By simply running this PowerShell script and providing it with a valid bearer token, the pertinent email data (i.e., recipient, subject, time, and hash of the attachment) are returned to the screen. The only thing left to do is look for the malicious hash.

And we found it! It appears the on October 17, 2021, ryan@sec541ryanic.onmicrosoft.com received a message with the subject of "Super secret message" containing the malicious attachment.

[1] https://github.com/ryananicholson/365emailhasher

Course Roadmap

- Section 1: Management Plane and Network Logging
- Section 2: Compute and Cloud Services Logging
- Section 3: Cloud Service and Data Discovery
- **Section 4: Microsoft Ecosystem**
- Section 5: Automated Response Actions and CloudWars

Microsoft Ecosystem

1. Malwarebytes Attack
2. Microsoft 365
3. **EXERCISE: Microsoft 365 Exchange Investigation**
4. SolarWinds Attack
5. Azure Active Directory (AD)
6. EXERCISE: Introduction to KQL
7. Storage Monitoring
8. EXERCISE: Log Analytics Using Azure CLI
9. Detection Services
10. EXERCISE: Microsoft Defender for Cloud and Sentinel
11. Network Traffic Analysis
12. EXERCISE: Azure Network Traffic Analysis

This page intentionally left blank.

Lab 4.1 | Microsoft 365 Exchange Investigation

Exercise Duration: 30 Minutes

Objectives

In this lab, we will analyze previously-gathered Microsoft 365 log data and:

- Review Exchange administrator steps to acquire audit data
- Connect to Azure Cloud Shell
- Download lab files from Azure Storage
- Analyze Microsoft 365 Exchange artifacts
- Investigate identified phishing message

The first lab of this section will explore some new tooling provided by Microsoft. It's a major shift from the previous labs, as you will be accessing a provided Azure account to perform all activities in this exercise, and most activities in the rest of the exercises in this section.

You will leverage Azure Cloud Shell to perform your command line-based actions to pull pre-recorded audit data from Azure Storage and analyze those artifacts to uncover who was phished and what the content of the phishing message was.

Course Roadmap

- Section 1: Management Plane and Network Logging
- Section 2: Compute and Cloud Services Logging
- Section 3: Cloud Service and Data Discovery
- **Section 4: Microsoft Ecosystem**
- Section 5: Automate Response Actions and CloudWars

Microsoft Ecosystem

1. Malwarebytes Attack
2. Microsoft 365
3. EXERCISE: Microsoft 365 Exchange Investigation
4. **SolarWinds Attack**
5. Azure Active Directory (AD)
6. EXERCISE: Introduction to KQL
7. Storage Monitoring
8. EXERCISE: Log Analytics Using Azure CLI
9. Detection Services
10. EXERCISE: Microsoft Defender for Cloud and Sentinel
11. Network Traffic Analysis
12. EXERCISE: Azure Network Traffic Analysis

This page intentionally left blank.

SolarWinds Attack

A look at how attackers are attacking the Amazon Web Services (AWS) and Microsoft Azure environments

Our second attack we will investigate is the SolarWinds attack that began in September of 2019. This attack is believed to have been perpetrated by the same, Russian-based threat group as the Malwarebytes breach: APT29. This is also our first breach that is believed to have affected more than just one company and its customers, as this is a supply chain attack.

A supply chain attack, as described by MITRE in their ATT&CK technique T1195.002 (Compromise Software Supply Chain), is an effort by an attacker to implant custom code or alter existing code. The end goal of this code alteration may be to steal proprietary data from downstream customers environments, spread ransomware throughout the compromised network, or, in the case of the APT29 threat group, establish command and control of compromised systems running the infected SolarWinds Orion software.

SolarWinds Attack

Case Study

September 2019

Thousands of enterprises and government agencies worldwide networks and systems compromised.

Initial Intrusion: Malicious code deployed into Orion IT monitoring and management software.

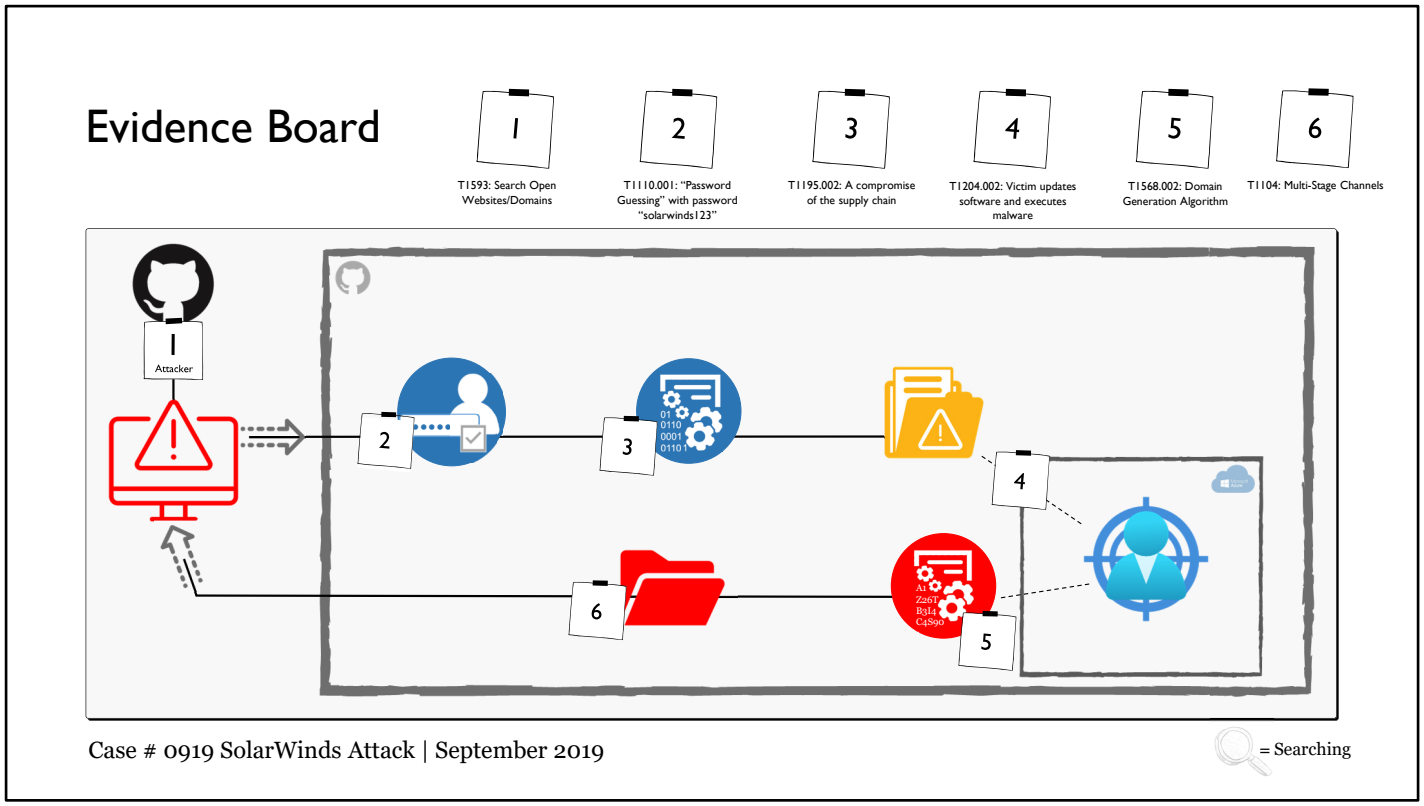
At Risk Infrastructure: Azure infrastructure.

SEC541

The SolarWinds attack is believed to have begun in the third quarter of 2019, but it was not until several months later that the breach was discovered. FireEye was the company that discovered that something was amiss in the SolarWinds Orion IT monitoring and management software, but it was more than one year later on December 8, 2020. In the meantime, the attack had found its way into many enterprises.1

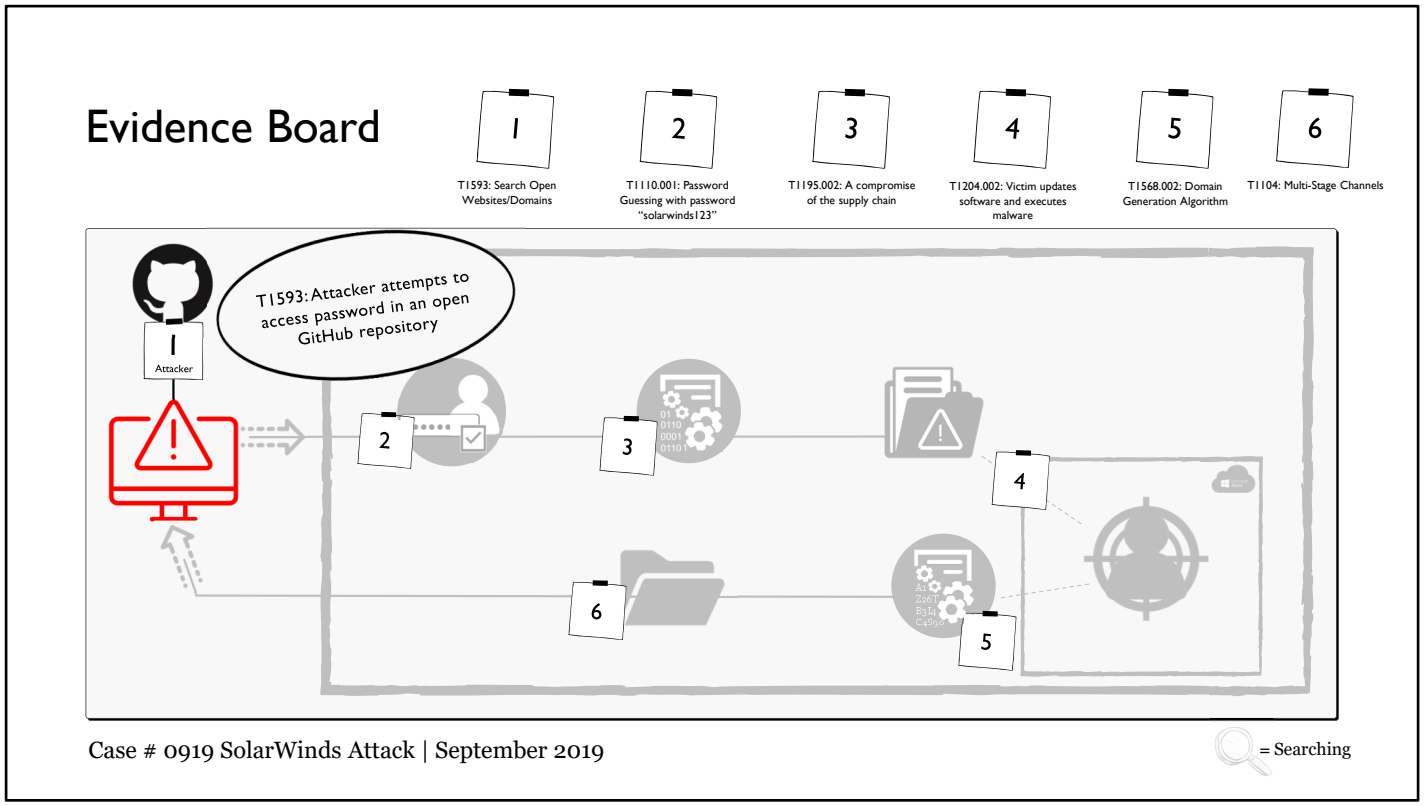
Once the SolarWinds Orion IT monitoring and management software's code was manipulated, this allowed APT29 to gain access to the networks, systems, and data of thousands of SolarWinds customers. It is believed that this supply chain hack is the largest (or amongst the largest) ever to have occurred. In fact, the victims of the breach come from a variety of industries including, but not limited to, financial, technology, and government.2

[1] https://whatis.techtarget.com/feature/SolarWinds-hack-explained-Everything-you-need-to-know
[2] https://www.lexingtonsoft.com/a-timeline-of-the-solarwinds-hack-what-weve-learned/



We will use the above evidence board to step through the SolarWinds breach. Similar to previous attacks, there are uncertainties based on incomplete reporting, but, even if the techniques were not leveraged in the exact way we present, it will be worth it to us to explore these techniques as they are popular amongst highly motivated threat groups. This will allow you to better defend your own environments and more quickly and accurately recognize these attempts to exploit your cloud-based systems.

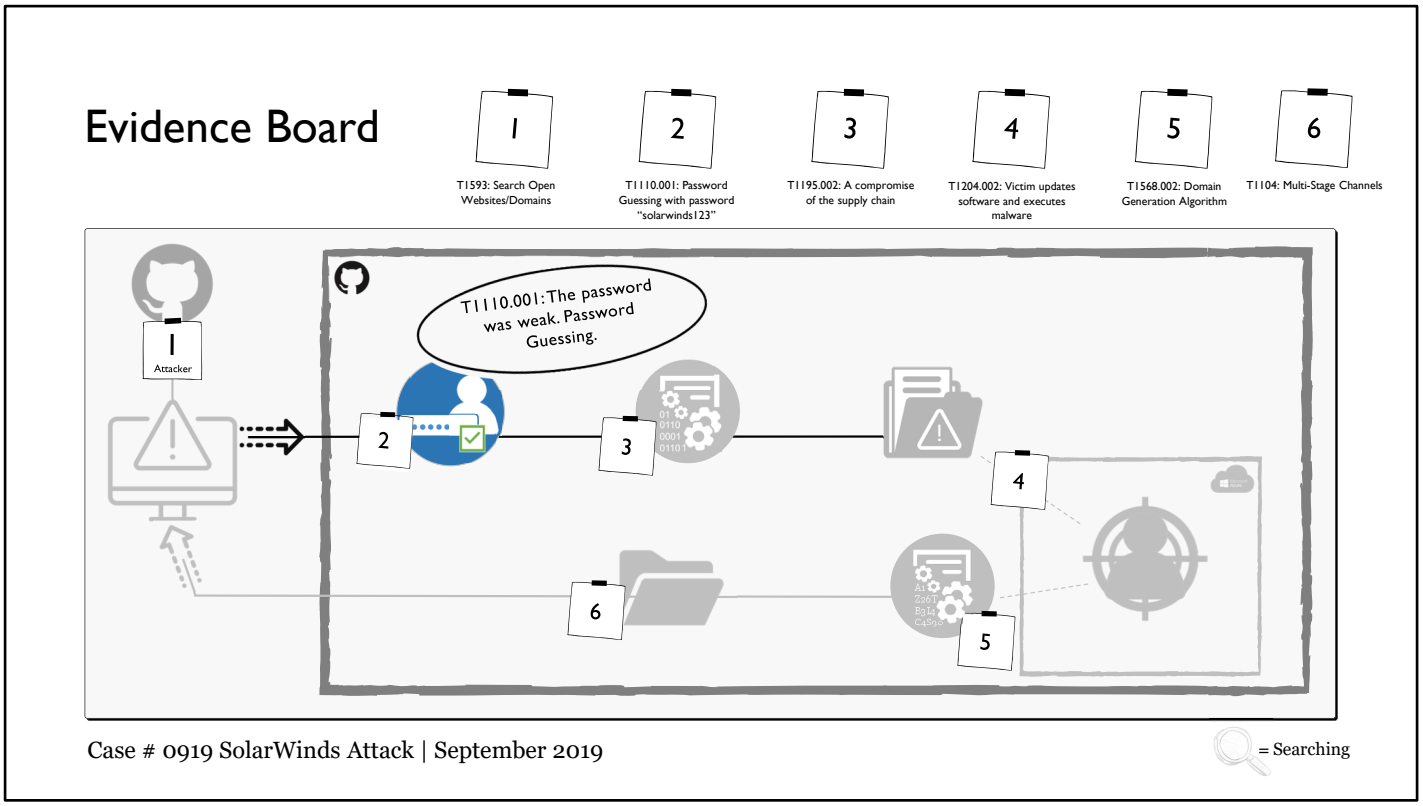
You may notice that there is a new environment we will discuss as well: GitHub. Since many vendors use cloud-based version control systems (also known as source code repositories), GitHub was the one chosen by SolarWinds for their Orion software. Let us get started investigating this breach.



Most proprietary code will be hidden in a private repository. This means that, to access the SolarWinds Orion software, you would need GitHub credentials to access and manipulate this source code. The problem here is that the credentials were visible in another publicly accessible repository totally unrelated to the SolarWinds Orion software.

The MITRE ATT&CK technique leveraged here is T1593 (Search Open Websites/Domains). This technique, as the name implies, involves digging through internet-facing resources such as public-facing websites, source code repositories, and social media, to name a few. This technique can be quite tricky or, in the case of GitHub, impossible to detect. So, if you were to accidently push credentials into a public-facing GitHub repository, you must realize the error of your ways and clean up the vulnerable file before the adversary finds it.

In any case, many security professionals know not to reuse the same password for multiple platforms, projects, or user accounts since, if those credentials are stolen, the attacker can use those same credentials in those other locations. This is known as a credential stuffing attack. Unfortunately, this password was used elsewhere.

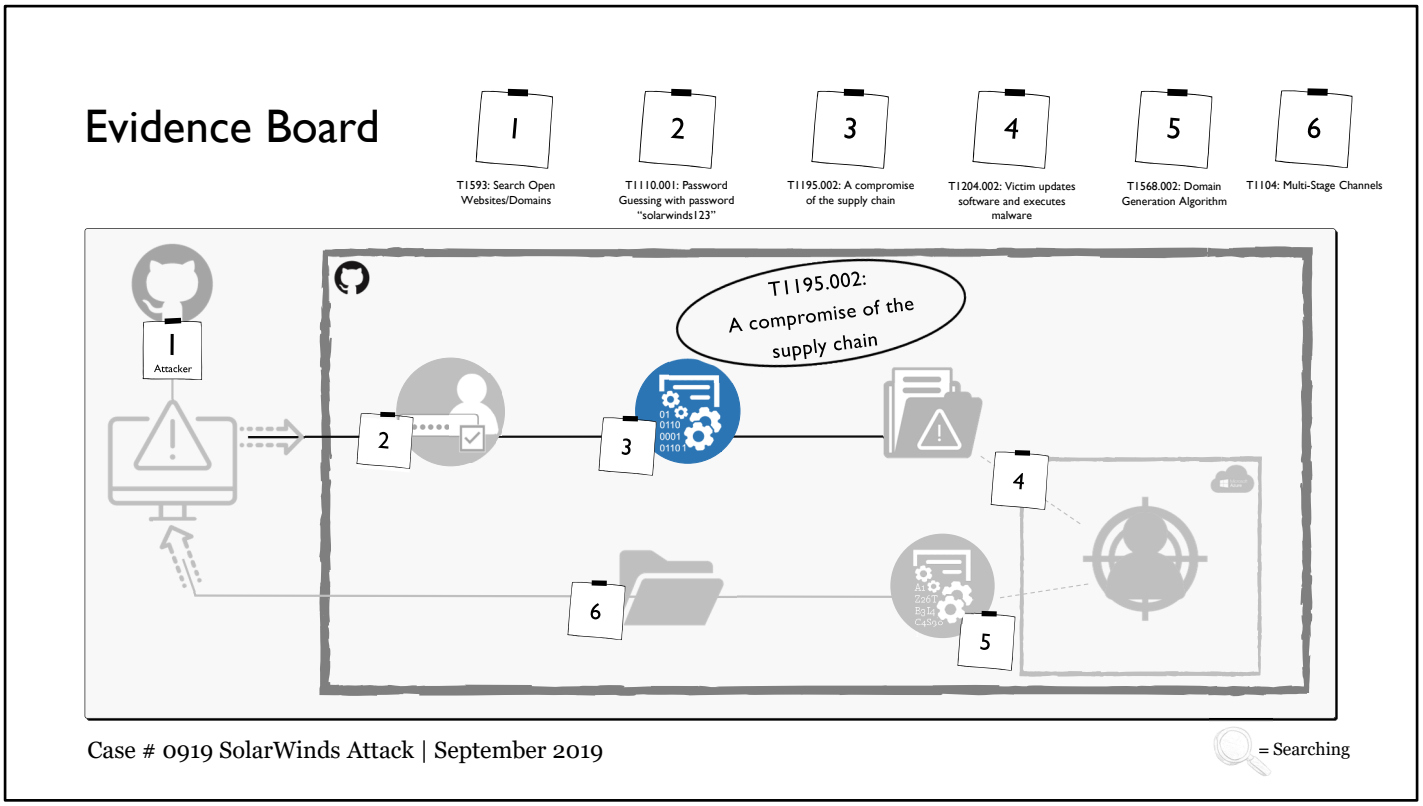


The second MITRE ATT&CK technique used in this breach by APT29 is Password Guessing (T1110.001). Most password guessing attacks, if strong passwords are chosen, are likely to fail. For strong passwords, we rely on the Center for Internet Security (CIS) guidance of fourteen or more characters, two or more special characters, two or more uppercase letters, two or more lowercase letters, and two or more numbers. To go a step further, other industry best practices may add not to include your company's name, your username, or email address as part of the password.

The password that was used to access the SolarWinds Orion source code repository was “solarwinds123”. Not only was this password extremely weak according to those CIS standards (only meeting two of the five requirements) and included the company name, but it was also the same password that was hosted in the public-facing GitHub repository discussed on the previous page. So even if the password was extremely complex, the attacker did not really have to do much "guessing" as they already had direct access to the password.[1]

Now that the credentials were successfully guessed to access the SolarWinds Orion source code, the supply chain compromise can begin.

[1] https://cisomag.eccouncil.org/solarwinds123-did-a-weak-password-result-in-solarwinds-hack/

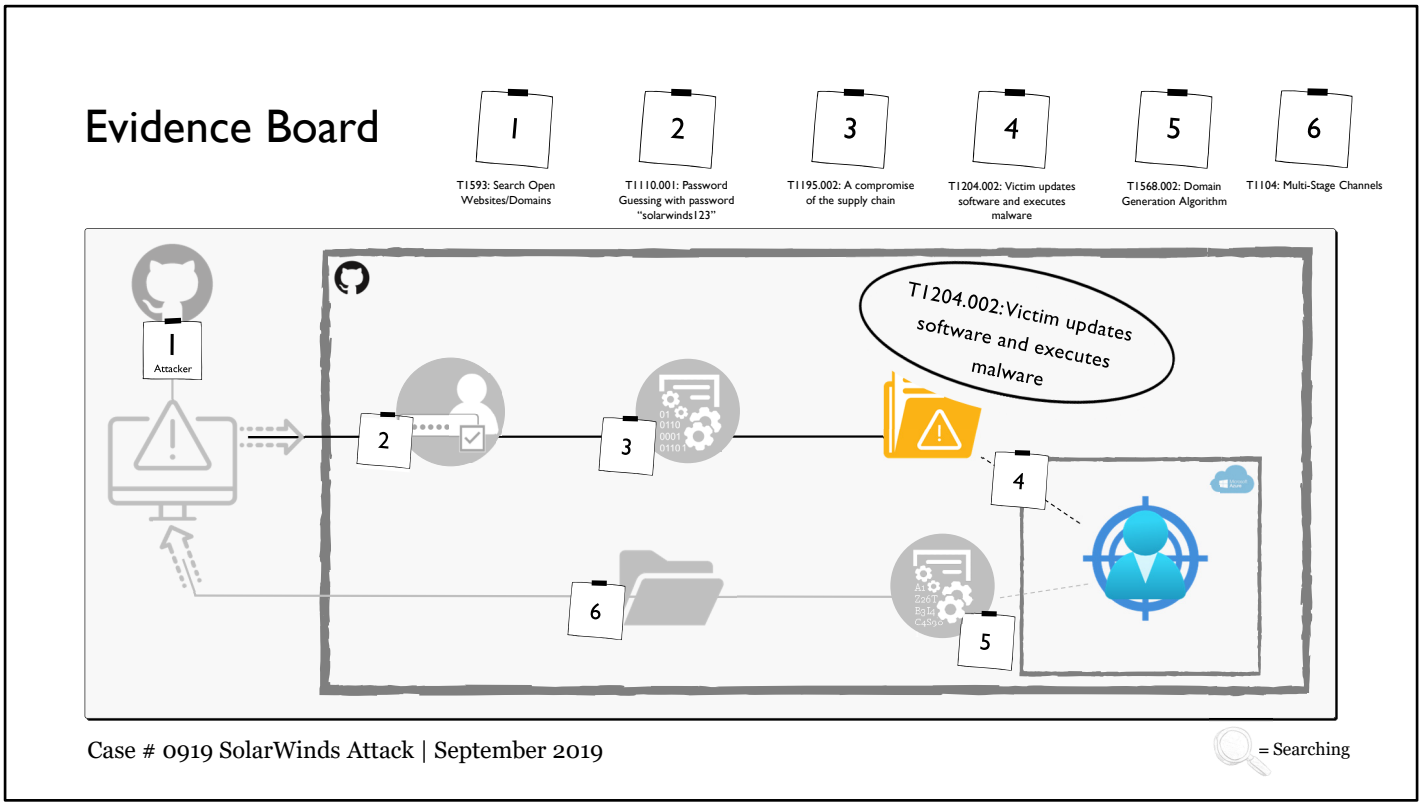


And here is where the real damage was done by the attack group—using MITRE ATT&CK T1195.002 (Compromise Software Supply Chain). It is here that, with just 3,500 lines of code,[1] the attackers added malicious capabilities to the SolarWinds Orion software that, when a SolarWinds customer performed a software update, a new code would be installed and lead to a complete takeover of the infected system running this updated code.

"But how can you defend against an unapproved or out-of-cycle source code update?", you may ask. Many source code repositories, like GitHub, do include functionality to require approvals before changes can be made or, at the very least, maintain a running list of all changes to the code that can be regularly reviewed. Had this review been done, it likely would not have taken over one year before the attackers were discovered. By that time, much damage had already been done.

The updated code included many capabilities—a few of which we will continue to break down when discussing the remaining techniques.

[1] https://www.npr.org/2021/04/16/985439655/a-worst-nightmare-cyberattack-the-untold-story-of-the-solarwinds-hack



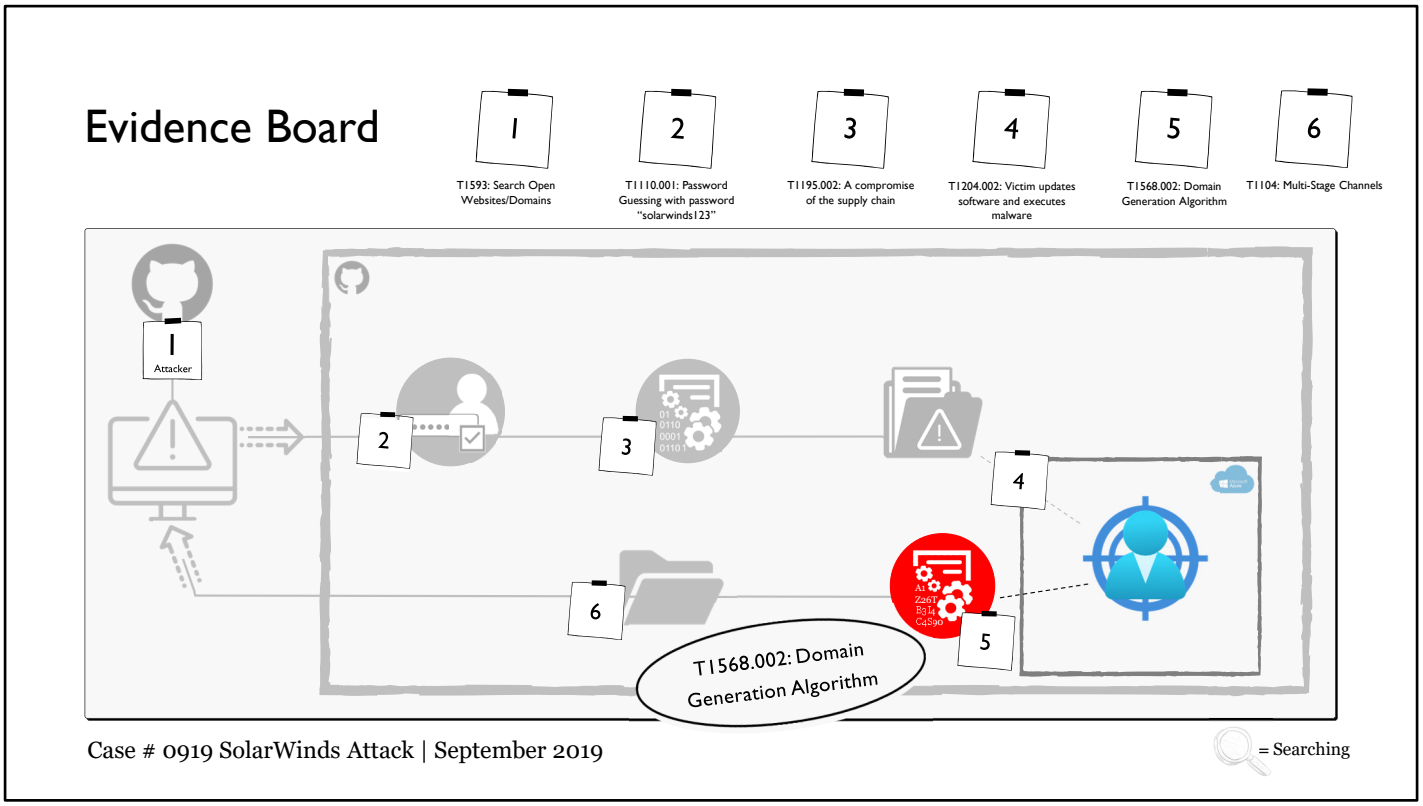
As the malicious variant of the SolarWinds Orion software is just sitting there waiting to be pulled down to an unsuspecting system, it is just a matter of time until a SolarWinds customer performs the update. Once they do and the install is successful, they are now infected and running the attacker's malicious code.

This malicious code is known as SUNBURST and has several moving pieces that we will highlight.[1] The first is communication with an attacker-controlled command and control (C2) server. This server resides on the avsvmcloud[.]com domain. This domain is no longer active as Microsoft and others were involved in seizing this domain to prevent the spread of this malware.[2]

In an effort to evade endpoint protection suites, the SUNBURST malware waits anywhere from twelve to fourteen days to make that first communication.

Let's move on to the next technique that the SUNBURST malware employs.

[1] https://resources.infosecinstitute.com/topic/sunburst-backdoor-malware-what-it-is-how-it-works-and-how-to-prevent-it-malware-spotlight/
[2] https://www.zdnet.com/article/microsoft-and-industry-partners-seize-key-domain-used-in-solarwinds-hack/

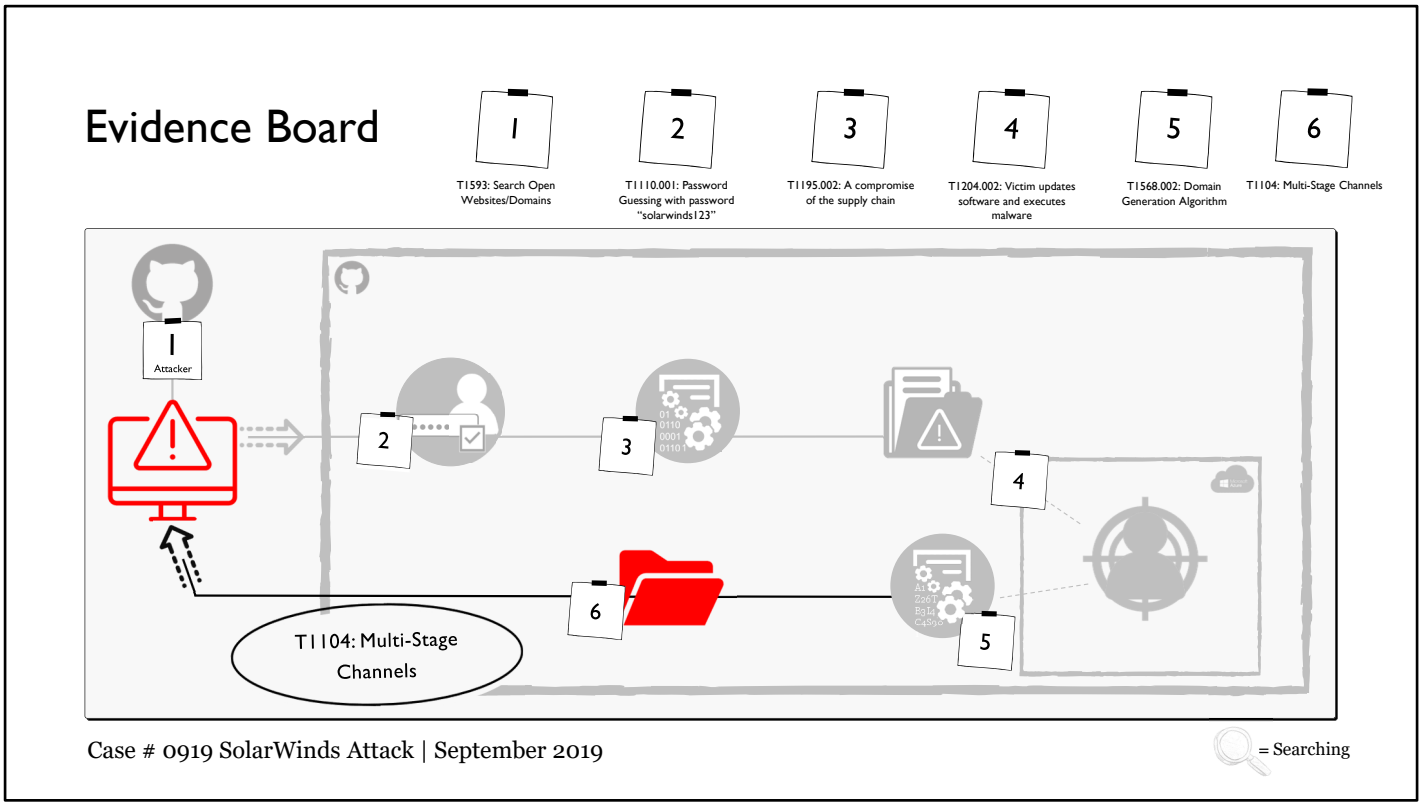


Once the SUNBURST malware checks into the command and control server, a custom hostname on the avsvmcloud[.]com domain is created using a domain generation algorithm (DGA). This DGA helps the threat group uniquely identify the host that is "phoning home" as the hostname will include such indicators as a unique Globally Unique Identifier (GUID), the domain name in which the victim is associated, and the region of the world in which the victim resides.

Asuna Amawaka wrote a very interesting article[1] on how this DGA is formed if you are so inclined to get very deep into the technical details.

A list of 1,722 randomly-generated domain names used by SUNBURST can be found here: https://github.com/bambenek/research/blob/main/sunburst/uniq-hostnames.txt.

[1] https://medium.com/insomniacs/a-look-into-sunbursts-dga-ba4029193947



Just simply having a foothold into the victim environment is not enough to satisfy the attackers. The SUNBURST malware also gives them capability, now that communication is in place between the victim and the command and control servers, to allow the attackers to perform many tasks. These tasks can be performed over a multitude of channels—to include HTTP, ICMP, and DNS protocols.

This leads us into the final MITRE ATT&CK technique used in this breach: T1104 (Multi-Stage Channels). The technique performed here simply uses more than one network channel to perform various actions—making it increasingly difficult to determine the full story or timeline of the incident.

SUNBURST uses many channels for various reasons, such as:

- Exfiltrating secrets from the victim system or network
- Spread the infection to other internal systems
- Modify files or execute PowerShell scripts on the victim system
- Disabling security products on the victim systems

Now that we understand more clearly how the breach started and progressed, let's dig into some more cloud technology and detection techniques to avoid this type of activity (or identify it more quickly) in our own environments.

Course Roadmap

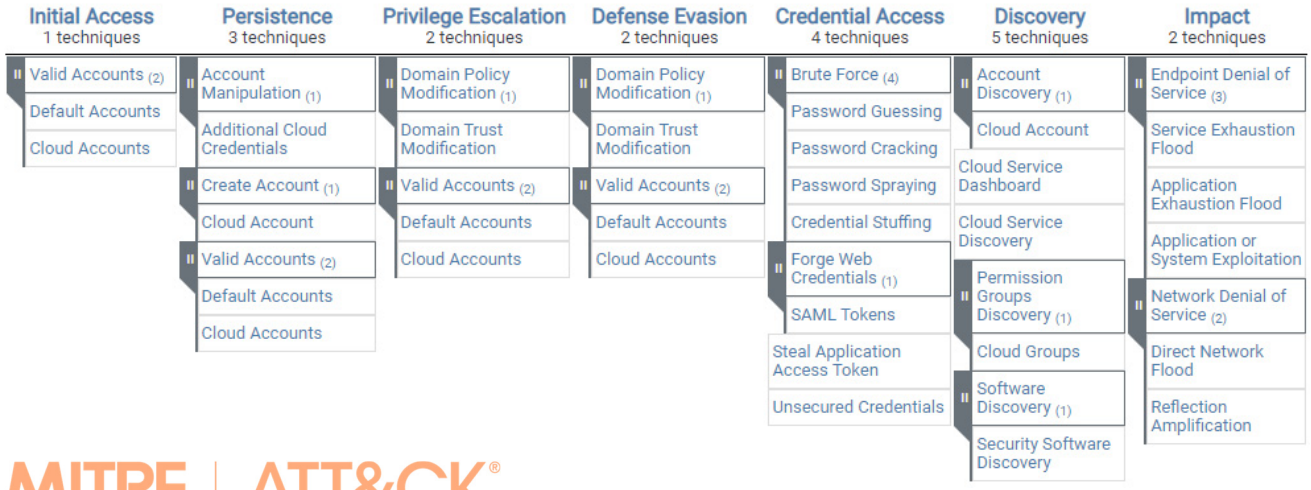
- Section 1: Management Plane and Network Logging
- Section 2: Compute and Cloud Services Logging
- Section 3: Cloud Service and Data Discovery
- **Section 4: Microsoft Ecosystem**
- Section 5: Automated Response Actions and CloudWars

Microsoft Ecosystem

1. Malwarebytes Attack
2. Microsoft 365
3. EXERCISE: Microsoft 365 Exchange Investigation
4. SolarWinds Attack
5. **Azure Active Directory (AD)**
6. EXERCISE: Introduction to KQL
7. Storage Monitoring
8. EXERCISE: Log Analytics Using Azure CLI
9. Detection Services
10. EXERCISE: Microsoft Defender for Cloud and Sentinel
11. Network Traffic Analysis
12. EXERCISE: Azure Network Traffic Analysis

This page intentionally left blank.

MITRE ATT&CK Azure AD Matrix



This module begins by immediately diving into a MITRE ATT&CK matrix. Unlike previous matrices, the Azure AD Matrix, shown above, is focused solely on how Azure's main directory service (Azure Active Directory) is targeted by real world adversaries. The same tactics exist, as is the case with all MITRE ATT&CK matrices, but one can clearly see the vast majority of techniques (entries in the boldened boxes) and sub-techniques (entries in the lighter-shaded boxes) focus mostly on cloud accounts. This makes sense as the primary purpose of Azure AD is to control users' accesses to the greater Azure environment. If an attacker were to hijack these trust relationships, they will have the same level of access as the compromised user account or service principal.

On top of these facts, many of these techniques may seem very familiar. As you have learned repeatedly, many attacks in a non-cloud environment still exist in a cloud environment. To go even further, many of the attacks exist across the spectrum of different cloud environments and service offerings.

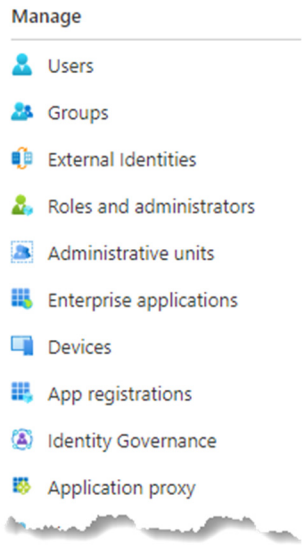
Where things begin to differ is the various techniques outside the scope of a single cloud user account. For instance, if an attacker were to take advantage of a trust relationship between Azure Active Directory and another domain or application, the attack may be able to extend into another on premise or cloud environment.

Reference:
https://attack.mitre.org/matrices/enterprise/cloud/azuread/#

Image reference:
https://attack.mitre.org/theme/images/mitre_attack_logo.png

Knowing Normal in Azure AD

- Just like anything else, one must **know normal**
  - Necessary before identifying potential threats
- Questions to answer:
  - *What users are currently in Azure AD?*
  - *Which groups are in use and who are members of these groups?*
  - *Which roles are assigned and to whom?*
  - *Are guest users or external identities in use and are they approved?*
- Let's explore some possible techniques to begin understanding the Azure AD deployment



Azure Active Directory is a very important service to understand in your environment—not just the ins and outs of how it works and how it can be taken advantage of, but also knowing what is to be expected in the Azure Active Directory deployment in which one is tasked to protect. This should not sound unfamiliar as a key tenet to protect any environment is to understand it well. With that said, there are a few (of many possible) questions to answer before you can effectively attempt to identify potential threats related to Azure Active Directory:

What users are currently in Azure AD?

A record of all approved users in this environment can help here. This can be in the form of simple documentation or a ticketing system that was leveraged by the Azure Active Directory administrators when the accounts were provisioned.

Which groups are in use and who are the members of these groups?

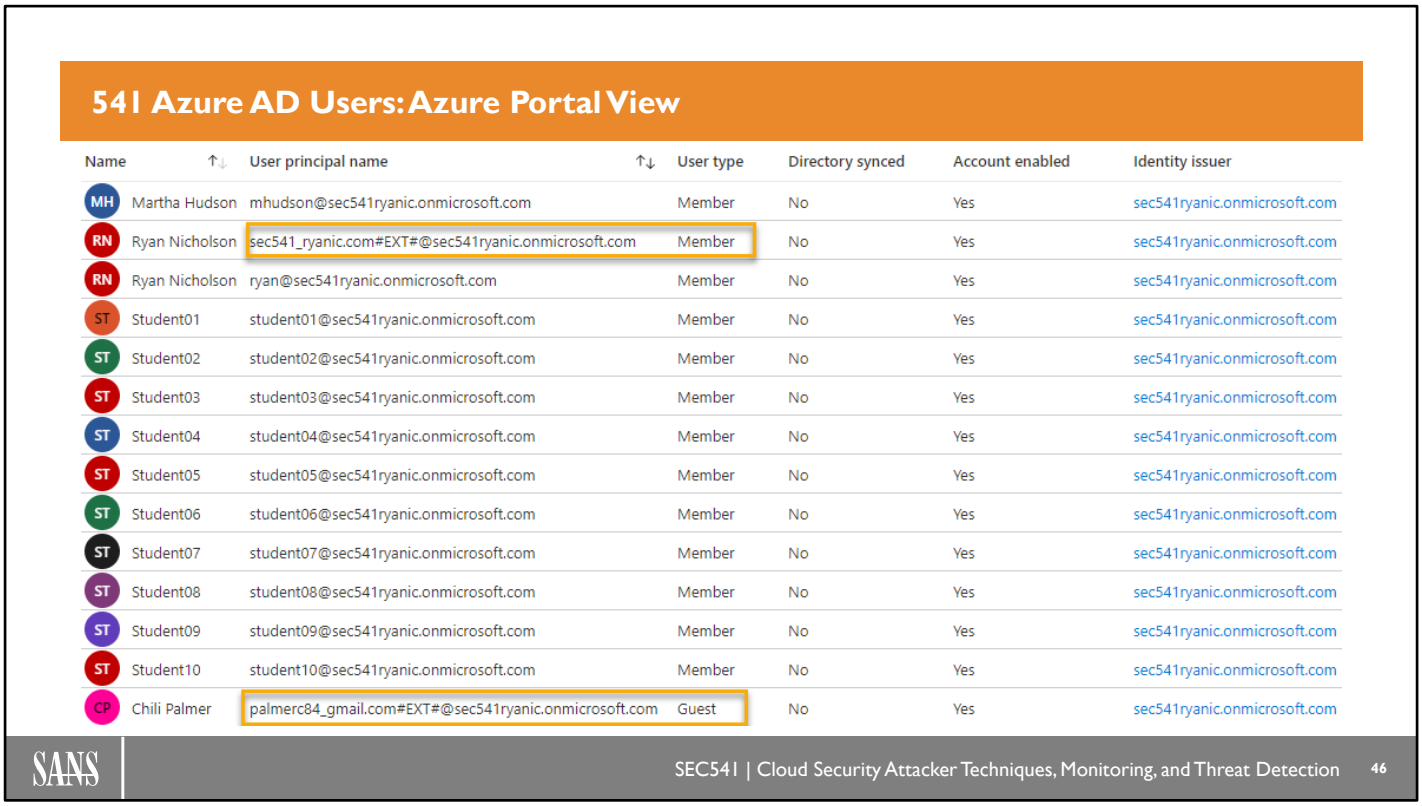
Again, this should be well documented somewhere as one of the industry's best practices is for repeatable permission assignments by utilizing groups. If not, one may be left making educated guesses that the groups are or are not set up properly by reviewing job titles and job descriptions for each user assigned to each group to verify if this data aligns with the permissions granted.

Which roles are assigned and to whom?

Since roles can be assigned directly to users (whether "human" or machine accounts) or groups of users, determine all of the appropriate assignments via manual or automated means.

Are guest users or external identities in use and are they approved?

Guest users should be used sparingly (if they are permitted at all) and tracked as this is an external user account which has been granted access to your Azure tenant in some capacity. Also determine any other means in which external entities are allowed access to the Azure deployment.



Within the Azure Active Directory service inside the Azure Portal, you will find your list of users by simply clicking on the Users option underneath Manage in the left-hand navigation. When you arrive, you will see the list of users. For each user, you can immediately see the following:

- Name: The name given by the Identity issuer to this user account
- User principal name: Request for Comments (RFC) 0822-compliant login name for the user
- User type: Informs the administrator if the user is Member or Guest
- Directory synced: Shows whether the account is synced with an Active Directory server or not
- Account enabled: Displays if the account is enabled (Yes) or disabled (No)
- Identity issuer: The domain issuing the identity

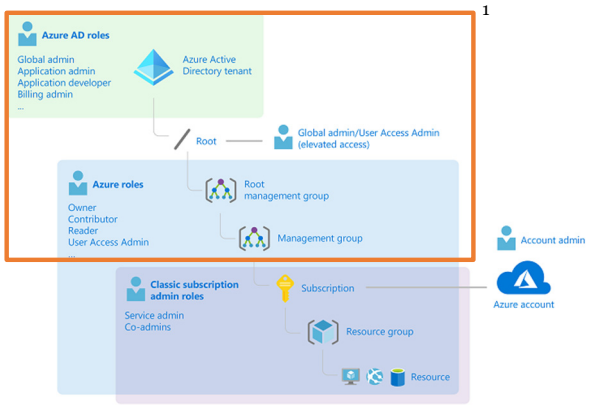
In the above example, you will see the list of users for the class's Azure Active Directory deployment called 541. A few things will stand out. First, most of the User principal names are formatted very similar to that of an email address (e.g., username@domain.ext). [1] However, two stick out: sec541_ryanic.com#EXT#@sec541ryanic.onmicrosoft.com and palmerc84_gmail.com#EXT#@sec541ryanic.onmicrosoft.com. The #EXT# identifies to you that this account is an external account. In this case, the sec541_ryanic.com#EXT#@sec541ryanic.onmicrosoft.com user is the account owner and is expected as the user type is shown as Member.

What about the palmerc84_gmail.com#EXT#@sec541ryanic.onmicrosoft.com user? It you notice, the user type for this user account is Guest. This means that someone invited this external user to the Azure environment. It is a mystery (for now, at least) what kind of rights this user ultimately has. We will get to that later. What is for certain is that the user that invited them was from this Azure account (sec541ryanic.onmicrosoft.com)!

[1] https://www.ietf.org/rfc/rfc0822.txt

Azure Role-Based Access Control (RBAC) (1)

- Provides access to Azure resources
- Roles can be assigned in Azure for various **scopes**:
  - **Azure AD**
    - Assigned via **Azure AD** service and allows/denies access across the entire tenant
  - **Azure**
    - Assigned for management groups by either a Global Administrator or User Access Admin in **Azure AD**



[1]

Once there is a handle on the users and groups provisioned in the Azure Active Directory service, the next crucial pieces of information to acquire would be the permissions utilized in the account. In Azure, roles are the collections of permissions that are attached to users or groups to allow them access to Azure resources.

This gets a little more complex as you cannot simply view all of the roles in one place. This is because when you attach roles to users, they can be scoped to various entities in Azure. First, to allow or deny a user or group access to resources in the entire Azure tenant, Azure Active Directory roles can be leveraged for this deployment. The Azure Active Directory service is where you would view the role to user attachment to get that sense of normal.

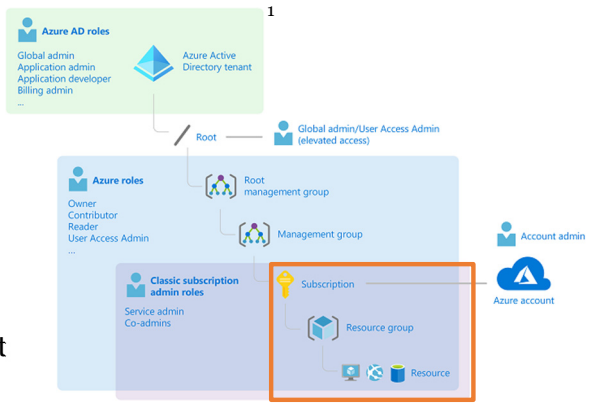
It does not stop there. There are a variety of other places to scope these roles. The next role type is what is called an Azure role which is used when scoping permissions at a management group level. Management groups are used to administer a collection of subscriptions. For instance, if a user needs access to all subscriptions under a certain management group, this service component would be the place to manage the role.[2]

[1] https://docs.microsoft.com/en-us/azure/role-based-access-control/media/rbac-and-directory-admin-roles/rbac-admin-roles.png
[2] https://docs.microsoft.com/en-us/azure/governance/management-groups/overview

Azure Role-Based Access Control (RBAC) (2)

Additional scopes include:

- **Classic Subscription Admin**
  - Assigned via **Subscriptions** service and restricts access to resources within the single subscription
- **Resource Group**
  - Assigned at the resource group and allows access only to resources within the group
- **Resource**
  - Assigned directly at the resource itself, so right single resource



There are three more scopes to mention to get the full picture of where roles can be configured. Next up is the Classic subscription admin role in which an administrator can directly apply permissions that affect all resources deployed within the Azure subscription.

Within the Azure subscription itself, which can be thought of simply as a billing account, resource groups would be the next logical place to apply permission boundaries. It is a resource group in which one or more Azure resources are placed during deployment. The resource group roles, as you could likely guess, control access to all resources within the resource group.

Finally, resource roles can be leveraged to apply permissions directly to a single Azure resource. All of these different scopes can be viewed in the Azure Portal.

That is a lot to keep track of, however. Imagine logging into the Azure Portal and viewing each Azure Active Directory deployment, every management group, every subscription, every resource group, and every resource to generate a list of roles in use. That is quite time-consuming and inefficient! Luckily, there are some examples coming in the next few pages that show how to efficiently collect the deployed roles across the entire Azure deployment.

[1] https://docs.microsoft.com/en-us/azure/role-based-access-control/media/rbac-and-directory-admin-roles/rbac-admin-roles.png

Azure AD Inventory: CLI Approach

- The larger and more complex the Azure AD deployment, the **less efficient** using the Azure Portal for inventory becomes
- Microsoft provides very useful tools to make you more efficient:
  - **Azure Command Line Interface (CLI)**
    - Installation example on Ubuntu:

      ```
      curl -sL https://aka.ms/InstallAzureCLIDeb | sudo bash
      ```
  - **Azure Az PowerShell Module**
    - Installation (PowerShell 7.0.6 LTS, PowerShell 7.1.3, or higher)

      ```
      Install-Module -Name Az -Scope CurrentUser -Repository PSGallery
      -Force
      ```
- Both tools are available in **Azure Cloud Shell**

Clicking through dozens of screens in the Azure Portal (or even hundreds or thousands—depending on the size of the Azure deployment) is likely no one's idea of fun. It certainly is not efficient. There has to be a more scalable and efficient way to handle inventory in an Azure environment.

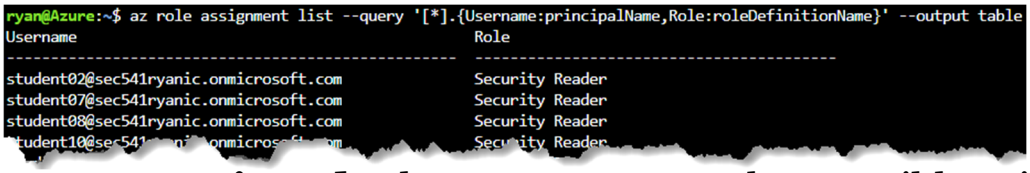
Before we begin discussing such techniques, you may remember that Azure provides multiple tools to interact with your Azure account from a command line environment. The two most common are the Azure Command Line Interface (CLI) and the Azure Az PowerShell module. In fact, you will find that, while there is plenty of overlap in capabilities between the two tools, there are certain cases where you must use one tool or the other. The reason could be that the capability you would like to use is only in one tool or if you are looking to extend the output of the tool, the Bash or PowerShell environment may be more useful—making you lean toward one tool over the other.

That may sound like you will need to have a few dependencies taken care of on your own system: Bash or similar command line environment for the Azure CLI tools and PowerShell for the Azure Az PowerShell module. In fact, many engineers are doing just that—installing the necessary prerequisites on their systems to support both of these options. Azure has a more user-friendly approach in its Cloud Shell product. You may remember this service from exercise 4.1, as it made using these very useful tools (and many more) very simple since the in-browser command line interface comes pre-loaded with these tools.

Azure CLI Role Inventory

- **Subscription-level** roles are relatively easy:

```
az role assignment list --query \
  '[*].{Username:principalName,Role:roleDefinitionName}' --output table
```



- **Azure AD-assigned** roles are not so easy, but possible using PowerShell:

```
Get-AzureADDirectoryRole | % {
  echo $_.DisplayName;
  Get-AzureADDirectoryRoleMember -ObjectId $_.ObjectId | Out-String;
}
```


You will see many handy Azure CLI and Azure PowerShell commands throughout the course just like you did with the AWS commands in the previous sections. Above are two examples to help alleviate the pain of clicking through several screens in your web browser just to collect the roles in use in Azure.

In the first example, the Azure CLI is used to collect all of the subscription-scoped roles. That is, all of the roles configured within the subscription itself that will either allow or deny access to all resource groups and resources within the subscription. The command begins with `az role assignment list`. This is a very noisy command that will list, in great detail, information about each role assignment created. To acquire just a listing of the role name and the user principal name to which it is attached, the `--query '[*].{Username:principalName,Role:roleDefinitionName}'` flag and option is appended. Finally, to make the data a little more human-readable, the Azure CLI allows one to output the data in multiple formats. Here, you see that the output chosen was `table`.

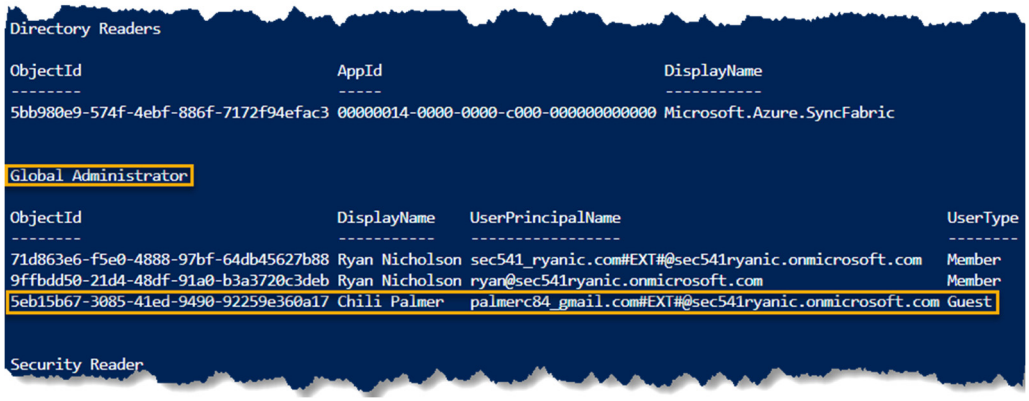
Using the Azure CLI would be great if it afforded the ability to collect all roles scoped in all ways, but, unfortunately, it does not at the time of this writing. However, the Azure Az PowerShell module can help identify the Azure AD-assigned roles. The example command above is a little more complex, so let us break this down into bite-sized chunks.

To begin, the `Get-AzureADDirectoryRole` is used to simply gather all roles scoped at the Azure Active Directory level and their associated information. The data returned from that command is quite busy, but it does not give us the assignments. What it does give us is two key pieces of information: the ObjectId (which is used in the `Get-AzureADDirectoryRoleMember` command) and the name of the role (`DisplayName`).

The percentage sign (`%`) and curly braces (`{` and `}`) are used to shorthand a `for` loop. This `for` loop takes the results of the first command and, for each result, outputs the role name and Azure Active Directory members to which the role is attached (using the `Get-AzureADDirectoryRoleMember` command).

Remember That Guest User?

- Previous PowerShell command output:



- The guest user is a **Global Administrator**!

After running the PowerShell command from an Azure Cloud Shell environment, you can see above some results for the 541 Azure Active Directory deployment. As promised, the multi-line PowerShell command will output the name of the Azure role (e.g., Directory Readers, Global Administrator, and Security Reader) as well as the membership information for each of those roles.

Taking a look at the Global Administrator results gives some interesting information. There are three members with complete control to the Azure Active Directory deployment and every resource tied to this deployment. It appears Ryan Nicholson has two accounts identified by the user principal names sec541_ryanic.com#EXT#@sec541ryanic.onmicrosoft.com and ryan@sec541ryanic.onmicrosoft.com. Both of these are known as fine as those belong to the course author.

The user principal name that stands out is palmerc84_gmail.com#EXT#@sec541.ryanic.onmicrosoft.com. If you remember from just a few pages ago, this is a guest user. Unless this was approved by a member of management or the security team, this guest user may very well be concerning to say the least as they are an external user who has very high privileges in this Azure deployment. We could have also just identified some back door account or persistence a la MITRE ATT&CK technique T1136.003 (Cloud Account).[1]

[1] https://attack.mitre.org/techniques/T1136/003/

Azure AD Log Types

- Fortunately, logs are generated by default in Azure AD for various actions:
  - **Audit Logs**: Similar to Azure Activity Logs, provides information regarding Azure AD changes (e.g., new user created, modification or role assignment, MFA-enforcement removed)
  - **Sign-in Logs**: Records successful and failed log in attempts and context surrounding the attempt
  - **Provisioning Logs**: Captures programmatic user creation activity
- Logs are only stored and are viewable in the Azure AD service for **one month**!
  - You will see how to collect this data for longer-term storage shortly

Now that you have the lay of the land for Azure Active Directory, let's take a look at the types of data that is collected by default.

There are three main types of logs in Azure Active Directory used for specific purposes. First, audit logs can be thought of as the Azure Activity Logs of Azure Active Directory. This means that any create, update, or delete events that occur to an Azure Active Directory component are captured here.[1]

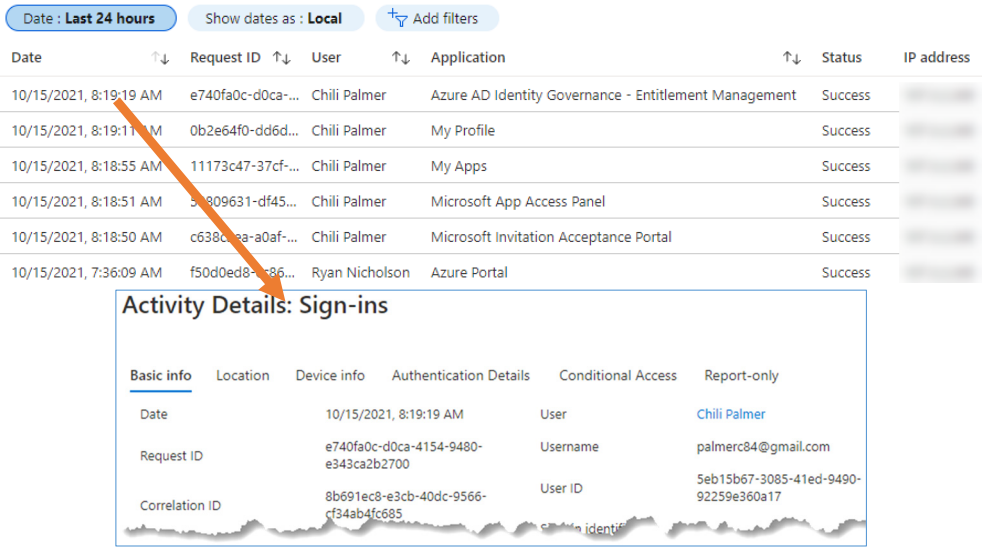
Next, the sign-in logs provide exactly what one would expect: sign-in activity. Not only do these logs provide who logged in (or failed to), what time they logged in (or failed to), and where the attempt came from, but also much more complete data. To name a few interesting bits of information provided, you can see if they attempted to log in with a particular tool or web browser based on their user agent string, the geolocation data for their source IP address, if they used multi-factor authentication (MFA), and much more.[2]

Lastly, provisioning logs will capture any automated user creation or management activity if this is deployed in the Azure Active Directory tenant.[3]

This data is fantastic, but, again, requires the Azure Portal to parse. To add to this, the data is only stored for one month in the Azure Active Directory service![4] What if you are investigating an attack from many months ago?

[1] https://docs.microsoft.com/en-us/azure/active-directory/reports-monitoring/concept-audit-logs
[2] https://docs.microsoft.com/en-us/azure/active-directory/reports-monitoring/concept-sign-ins
[3] https://docs.microsoft.com/en-us/azure/active-directory/reports-monitoring/concept-provisioning-logs
[4] https://docs.microsoft.com/en-us/azure/active-directory/app-provisioning/how-provisioning-works

Azure AD Sign-in Log Example



If one wanted to view the data in the Azure Portal, they would navigate to the Azure Active Directory service and, in the left pane underneath Monitoring, click on Sign-in logs. Here, you would be presented with the last 24 hours of log in attempts and a few select fields for each entry, such as the time, the username attempted, the status, and the IP address of the request. The timespan is easily adjustable (but remember, you can only go back one month) and you can even apply other filters to capture particular data of interest. This can become very valuable if looking for certain key events in a very busy Azure tenant.

To view more complete data other that what is displayed, you can click on any entry to acquire much more detail around the request. The following tabs you will see for the Activity Details pane that will appear and can be considered the most valuable are:

- Basic info: This information is very similar to what was displayed on the main Sign-in logs page. The most valuable data would likely be the date of the event, the status (was the log in successful?), the client application, and, of course, the username.
- Location: The geolocation of the source IP to include the city/state/country, IP address, and autonomous system number (ASN) is found here.
- Device info: This information is based solely upon the user agent string provided by the requestor. Of course this data can be spoofed but could identify things like the application used and even operating system platform.
- Authentication Details: Shows more information related to the authentication, such as if only one factor provided or whether MFA was in use.
- Conditional Access: If set up for this user, identifies how or whether conditional access conditions were met.

Azure Active Directory Identity Protection

- **Identity Protection** can be enabled to help identify suspicious user activity:
  - Leaked credentials
  - Sign-ins from unexpected IP addresses
  - "Impossible traveler"
  - Sign-ins from presumed infected devices
  - Sign-ins from potentially malicious IP addresses
- Policies can be created to react to suspicious activity:
  - Block access
  - Allow access, but require password change
  - Allow access, but require multi-factor authentication (MFA)

A very interesting component in Azure Active Directory is Identity Protection. This feature attempts to provide Azure account owners with insight into suspicious activity in their account. There are many detections that can be discovered on a security team member's behalf by Azure, but they can be broken down into two categories: risky users and risky sign-ins.

To leverage these features, policies must be enabled on the account. There are two policies to be enabled: a user risk policy (to capture risky users) and a sign-in risk policy (to discover risky sign-ins). When enabling these policies, there are a few options to consider.

First, who do you assign the policies to? It is recommended to assign these policies to all users, but you can be selective if you would like.

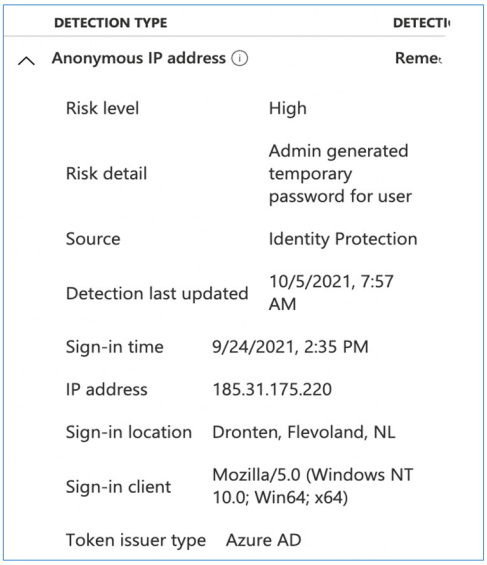
Next, there is an option to determine when the policy is enforced. This is, more or less, you being given the option of how risky must the user or sign-in be before a control is enforced. The three options you have are high, medium and above, and low and above. The higher the selection, the riskier the user or sign-in must be before the control kicks in.

The final option is to determine what happens when the user or sign-in is determined to be risky. This is the control mentioned earlier. The options for a risky user is to block the user (they cannot interact with Azure) or allow the access but add another requirement to their session by forcing a password change. For risky sign-ins, you can also choose to block, but when allowing access, you can force the user to sign in using multi-factor authentication (MFA).

A third policy within Identity Protection that is worth mentioning is the MFA registration policy. This policy can be invoked to force all users (or select users) to use MFA technologies to log fully into their account before interacting with Azure resources. This would be a good thing to enforce before implementing a sign-in risk policy that will require users to log in using MFA if they are behaving, according to Azure, strangely. If they do not have MFA enabled, false positives (users flagged as risky) will not be able to continue working in the environment until an Azure administrator takes action to restore their access.

Azure Active Directory Risky Users, Sign-ins, and Detections

- If Identity Protection is enabled, log data can be generated as suspicious users or sign-ins are discovered
  - **Risky users**: accounts with credentials that could be compromised
  - **Risky sign-ins**: captures suspicious sign-in activity
  - **Risk detections**: allows users to view and manage all risk detections
- Again, data is only kept in Azure Active Directory for a limited time!



Risky users can be determined using the following criteria:

- Leaked credentials: Microsoft acquires credentials from attacker communities and determines if they are used to compromise your own user's credentials.
- Azure AD Threat Intelligence: Microsoft compares your user's activity to known adversarial activity based on their own internal and external intelligence feeds.

Risky sign-ins can be identified by the following criteria[1]:

- Anonymous IP address: Access via a Tor or Virtual Private Network (VPN) connection.
- Atypical travel/Impossible travel: Compares two sign-ins from two locations where it is impossible, using current modes of travel, to travel between the two locations in the delta of time between those logins.
- Malware linked IP address: Sign-in identified from a source IP address known to have been infected with malware or known to be part of a malicious campaign.
- New country: A new or infrequent location was the source of a successful sign-in.
- Password spray: Identifies when a password spray is detected against a collection of users in Azure Active Directory.
- Much more criteria can be found at: https://docs.microsoft.com/en-us/azure/active-directory/identity-protection/concept-identity-protection-risks

Note, again, that the data in these service components within Azure Active Directory are only stored for a very short amount of time. For the risky users and risky sign-ins events, that data is only kept for up to one month. For risk detections, you do have the data available for a bit longer, but only for ninety days. If you need to collect this data for longer (and it is highly recommended that you do), you can export this data to Azure Storage, Azure Log Analytics workspaces, Azure Event Hubs, or sent to a partner solutions using Diagnostics Settings.

[1] https://docs.microsoft.com/en-us/azure/active-directory/identity-protection/howto-identity-protection-investigate-risk

Password Attacks (1)

- **Dictionary Attack**
  - Using a pre-existing list of passwords to attempt against the target
  - If not in the list, attack will be unsuccessful
- **Brute Force Attack**
  - Literally attempting every possible password
  - If given the proper key space, will always succeed
  - Longer passwords or uncommon character sets will slow down this attack significantly
  - Susceptible to account lockouts

A very common tactic adversaries may use to establish a foothold in the cloud and on-premise environments alike would be a password attack. In fact, in a future lab, you will see a password-guessing attack against Sherlock's Azure Active Directory tenant! There are three basic forms of password attacks getting a mention in this module: dictionary, brute force, and password spray attacks.

Dictionary attacks consist of an attacker populating or acquiring a list of common passwords to attempt against a single user account. This password list could be from a previous breach or a list of common words or phrases with common password policy enhancements in mind. For example, if the attacker is targeting Sherlock in particular, a list of passwords may include Sherlock, Baker (since the office is located on this street), Watson (since he works for Sherlock), and have alternate forms of these words to account for complexity requirements. For example, Sherlock could turn into Sherlock123, Sherlock!, $h3rl0ck!, and so on. The drawback for the attacker is that the dictionary must contain a correct password. If it does not, the attack must generate or acquire a new dictionary and try again.

The next password attack is the old-fashioned brute force attack. With this attack, the adversary is attempting every single sequence of characters in a given key space. A key space, in this sense, is a list of possible characters that a user could select from. If the attacker is not careful, they could limit their key space and, although brute force attacks are generally thought to always be successful, may not be successful here as it missed a valid character. For instance, if the attacker is using the English twenty-six-character alphabet as the possible lower- and upper-case characters but is attacking a German-speaking company, passwords with an Ä, Ö, Ü, ß, ä, ö, or ü will not be cracked.

Password Attacks (2)

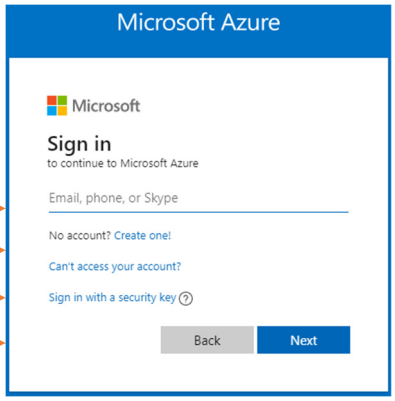
- **Password Spray**
  - Attempting a single password against a list of users
  - Much less likely to lock out accounts if only attempting one password per day/week against each account
  - More difficult to detect if focusing on login failures/account

✘ `sherlock@sherlock.onmicrosoft.com:sherlock!`

✘ `watson@sherlock.onmicrosoft.com:sherlock!`

✘ `mycroft@sherlock.onmicrosoft.com:sherlock!`

✓ `martha@sherlock.onmicrosoft.com:sherlock!`



A newer form of password attack is a password spray. With this attack, the adversary is attempting to use a common password that, even though it is rather complex, may be chosen by an internal user. This may or may not require knowledge about the organization to choose the proper password. For example, no knowledge is required if the attempted password coincides with the current season with some special characters added to it for complexity—something like this: `Summer2020!`.

You may be asking, "why would a season be a good choice?" Great question. Since many password policies require users to change passwords every three months, including the season in the password makes it even easier to remember. When the `Summer2020!` password is due for a change, `Fall2020!` could be the next password, then `Winter2021!`, and so on.

This form of attack is less likely to be detected because, if the attacker is very careful and only attempts one or two passwords within a week or so, accounts are less likely to lock out since users will legitimately log in before the next password spray attempt. Also, many signatures identify a password attack based upon a certain threshold of login failures per account, but not across the whole directory.

Case Study: Azure Active Directory Password Spray (1)

- An indicator of a potential password spray attack could be:
  - Several failed attempts from a single source with multiple usernames attempted
  - Many failed attempts from one or more sources with multiple usernames over a short period of time
- The Azure Active Directory service's **Sign-in logs** in the Azure Portal can help pinpoint a potential authentication attack
  - Sort data by Date field
  - Find several failed attempts to many Azure AD user accounts
  - If no successes, attack failed
  - If a success is mixed in from the same IP as the failures, attack success

And now it is time for another case study. This time, we are concerned with a potential authentication attack against our Azure account. The authentication attack in question is a password spray.

How would one identify a potential password spray in Azure? If you remember, there are a few great sources to acquire sign-in activity: the Azure Portal if the attempts were within the last thirty days or, if the data were exported, Azure Storage, Azure Log Analytics, etc. This log data will capture things such as:

- Time of the sign-in
- The IP address of the sign-in attempt
- The status of the attempt (i.e., Success or Failure)
- User account that was attempted

So, with this, the Azure Portal could be used to spot these attempts if the account is not too incredibly noisy with legitimate logins. If using the Azure Active Directory service within the Azure Portal, the method to identify this attempt could be to first sort the thirty days' worth of data by the Date field.

After that, the analyst would scan through the data manually until they find a large number of failed attempts against several, legitimate Azure Active Directory user accounts from the same or a collection of IP addresses. If no successes are found during this attack or from those IP addresses, the attack can be assumed to not be successful. If, however, a successful login appears from one of the suspicious IP addresses, the attack was successful!

Case Study: Azure Active Directory Password Spray (2)

Successful Password Spray identified in Sign-in logs!

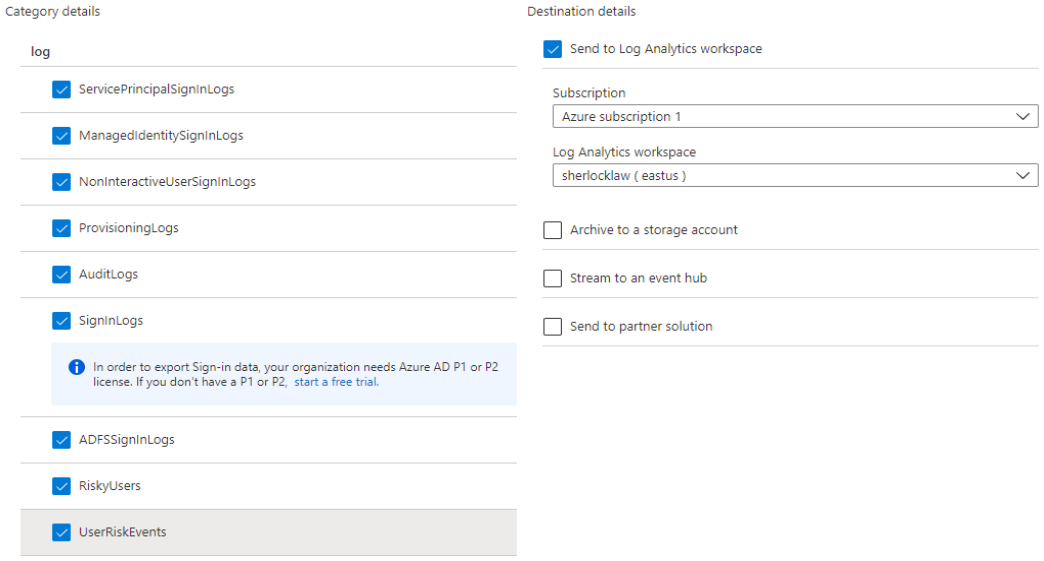
Date	Request ID	User	Application	Status	IP address
10/5/2021, 8:32:30 AM	05de0ec7-4e7f-4873...	Student01	Microsoft Azure CLI	Failure	107.3.2.240
10/5/2021, 8:32:31 AM	eb964e3f-f43a-49eb...	Student02	Microsoft Azure CLI	Failure	107.3.2.240
10/5/2021, 8:32:32 AM	8674053d-c21d-42a...	Student03	Microsoft Azure CLI	Failure	107.3.2.240
10/5/2021, 8:32:32 AM	b130df93-12fc-470b...	Student04	Microsoft Azure CLI	Failure	107.3.2.240
10/5/2021, 8:32:33 AM	51dc4264-4a67-4ca...	Student05	Microsoft Azure CLI	Failure	107.3.2.240
10/5/2021, 8:32:34 AM	89bfcda5-1e92-4d33...	Student06	Microsoft Azure CLI	Failure	107.3.2.240
10/5/2021, 8:32:35 AM	a14ce659-7c76-4a9e...	Student07	Microsoft Azure CLI	Failure	107.3.2.240
10/5/2021, 8:32:36 AM	5b61a0ad-65e9-407...	Student08	Microsoft Azure CLI	Failure	107.3.2.240
10/5/2021, 8:32:37 AM	e6f6c550-46cf-4aad-...	Student09	Microsoft Azure CLI	Failure	107.3.2.240
10/5/2021, 8:32:38 AM	8f4430a5-bb3d-404...	Student10	Microsoft Azure CLI	Success	107.3.2.240

Here is an example of conducting the analysis explained on the previous page. If you look closely, you can see a large number of Failure statuses related to the same source IP address. These attempts shown above are all within the same minute which is synonymous with many authentication attacks. Where a brute force or dictionary attack would likely attack one user account, each user account (Student01 through Student10) are attacked in sequence. This is looking more like a password spray for sure.

Public-facing accounts are likely subject to authentication attacks simply for being exposed to the public internet, so many failures are to be expected. What is not expected, and quite concerning, is the last entry. The same IP address that is attacking Student01 through Student09 (and failed) successfully logged into the Student10 account!

These logs even show you how they were likely making these attempts. It appears that, unless the user agent field was spoofed during this communication, that the suspected attacker is using the Azure CLI tools to attempt to connect. We can conclude that the Student10 account has been logged in during a likely password spray and the attacker was using the Azure CLI to conduct the attack.

Export All the Data!



We were lucky in the case study that the attack was spotted in the Azure Active Directory service in the sign-in log data. What if the attack happened months or even years ago? The data would be lost forever unless the data is exported to another service—either within the Azure ecosystem or to an outside location (e.g., Security Information and Events Management solution). This can be performed by creating a Diagnostics Setting in the Azure Active Directory service.

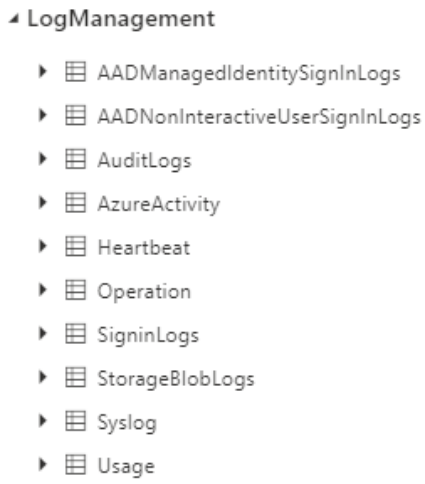
To ensure this data goes somewhere, first set up the destination to accept this data. For instance, if an Azure Log Analytics workspace, Azure Storage account, Azure Event Hub, or partner solution is the intended destination, these services must be up and running before configuring the Diagnostics Setting.

Next, select the log data to be exported. There are plenty of options here, so take a step back and, if the organization is using the solution, enable the log. It may be wise to simply select all options in case the company explores other methods of sign-ins or capabilities that integrate into Azure Active Directory.

Note the warning for SigninLogs: you must have a proper license (Azure AD P1 or P2) to export this log data.

Azure Log Analytics Workspaces Tables

- **AADManagedIdentitySignInLogs**
  - All managed identity (both user- and system-assigned) sign-in attempts
- **AADNonInteractiveUserSignInLogs**
  - Sign-ins performed by a client application or OS component on behalf of a user
- **AuditLogs**
  - Information regarding Azure AD changes
- **SigninLogs**
  - Successful and failed log in attempts and context surrounding the attempts



If the data is exported to Azure Log Analytics workspaces (as it is in the class Azure environment), you will have several tables generated as the log data arrives. Above is not a complete list, but some of the more common tables that are generated.

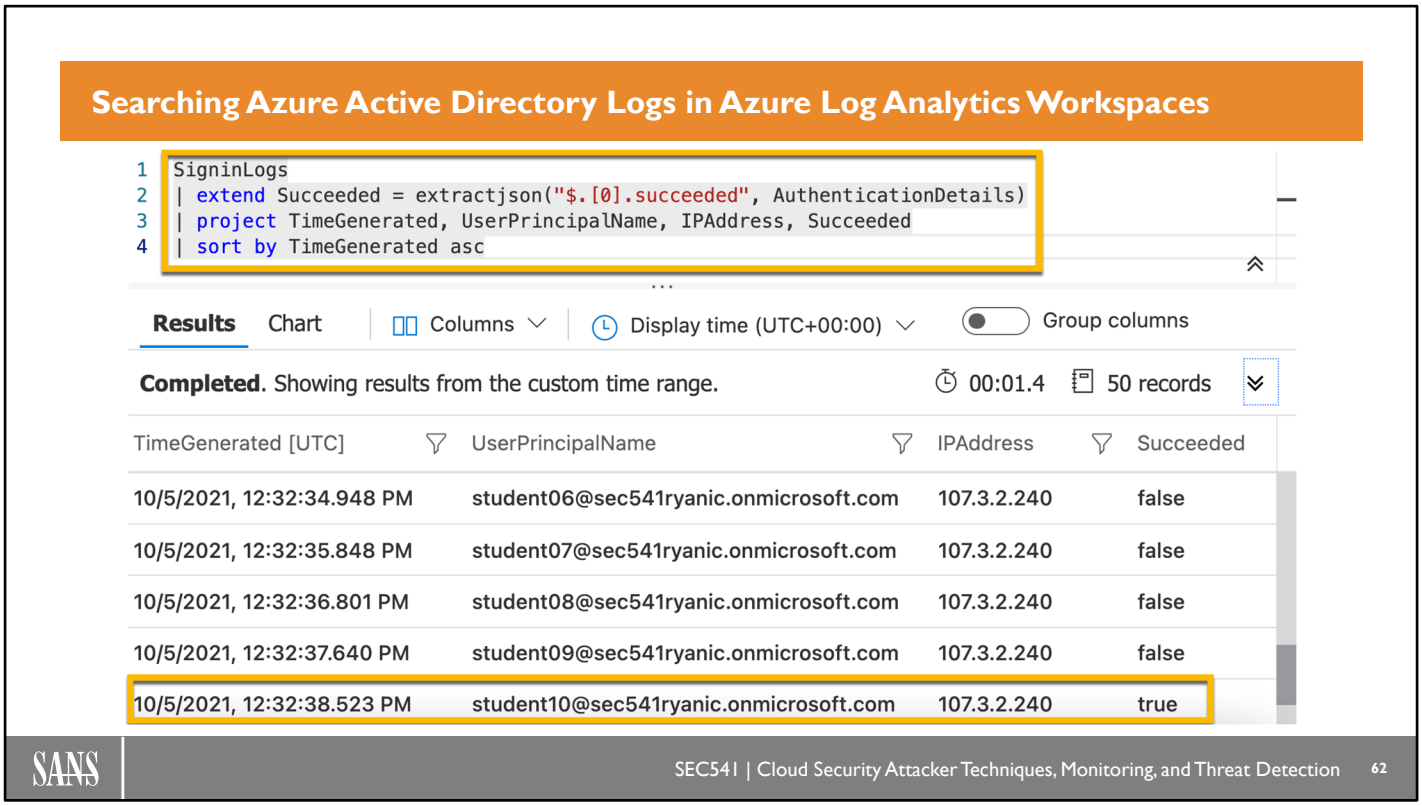
AADManagedIdentitySignInLogs can be very useful if monitoring or investigating when Azure service components (like an Azure Virtual Machine) is leveraging either a user-assigned or system-assigned managed identity.

A different type of interaction is when a user is leveraging some sort of client application or operating system component to interact with the Azure environment. In this case, the AADNonInteractiveUserSignInLogs table would be most helpful to investigate those events.[1]

As mentioned previously, Activity Log-like data related to the Azure Active Directory service is captured in the AuditLogs table. If strange configurations, users, groups, or other resources are noticed in Azure Active Directory, a record of this activity and context around the interaction can be viewed in this table.

Finally, SigninLogs can be very valuable if determining when, how, and from where users are signing into the Azure environment.

[1] https://techcommunity.microsoft.com/t5/azure-sentinel/non-interactive-logins-minimizing-the-blind-spot/ba-p/2287932



If the sign-in log data is stored in an Azure Log Analytics workspace table, it can be analyzed using Kusto Query Language (KQL) queries. In fact, let us return to the case study from a few pages ago for a moment. If you remember, you identified a potential, successful password spray attack against the Azure tenant. This data will (and has at this point in time) age out of the Azure Active Directory service. However, this data was also exported to the Azure Log Analytics workspaces service.

To discover this attack using KQL, let us build a query. First, specify the table to be searched. In this case, it is `SigninLogs`. On the next few lines of the query, we can begin to get a bit more creative as the data output is quite voluminous by simply querying the table. Before restricting the data output, however, we can use the `extractjson` function to pull a nested JSON element from the `AuthenticationDetails` column. Using extend, we are assigning a new column name, Succeeded, to the succeeded value within the `AuthenticationDetails` column.

Next, `project` is used to specify which columns to output. Since we only require the times, usernames, source IP addresses, and whether or not the sign-in was successful, using the filter of `project TimeGenerated, UserPrincipalName, IPAddress, Succeeded` will accomplish this.

Lastly, to put the results in ascending order by the event time, we can leverage `sort by TimeGenerated asc`.

After executing the query by clicking the Run button, we can scroll through the results. Yet again, we are able to locate the potential password spray and the fact that it was successful because a user from a single IP address was rapidly attempting several different user logins and all were unsuccessful but one—student10@sec541ryanic.onmicrosoft.com.

Course Roadmap

- Section 1: Management Plane and Network Logging
- Section 2: Compute and Cloud Services Logging
- Section 3: Cloud Service and Data Discovery
- **Section 4: Microsoft Ecosystem**
- Section 5: Automated Response Actions and CloudWars

Microsoft Ecosystem

1. Malwarebytes Attack
2. Microsoft 365
3. EXERCISE: Microsoft 365 Exchange Investigation
4. SolarWinds Attack
5. Azure Active Directory (AD)
6. **EXERCISE: Introduction to KQL**
7. Storage Monitoring
8. EXERCISE: Log Analytics Using Azure CLI
9. Detection Services
10. EXERCISE: Microsoft Defender for Cloud and Sentinel
11. Network Traffic Analysis
12. EXERCISE: Azure Network Traffic Analysis

This page intentionally left blank.

Lab 4.2 | Introduction to Kusto Query Language (KQL)

Exercise Duration: 30 Minutes

Objectives

In this lab, we will use the Azure Log Analytics workspaces service to discover the beginning of a new attack campaign by:

- Discovering Azure Log Analytics tables
- Exploring Azure SigninLogs fields and conducting a basic search
- Building a KQL query to identify failed logins
- Extending a KQL query to identify a successful authentication attack (T1078.004 and T1110.001)
- Identifying compromised user account activity in ActivityLogs (T1526)

And now, on to the second exercise of this section. During this lab, you will move through various Kusto Query Language (KQL) queries to uncover several artifacts related to a new attack campaign that was targeting the Azure environment you are performing your activities in. These queries will start rather simple but become more complex throughout the lab's exercises. At the end, you will have a decent (but not yet complete) picture of just how this account was attacked.

Course Roadmap

- Section 1: Management Plane and Network Logging
- Section 2: Compute and Cloud Services Logging
- Section 3: Cloud Service and Data Discovery
- **Section 4: Microsoft Ecosystem**
- Section 5: Automated Response Actions and CloudWars

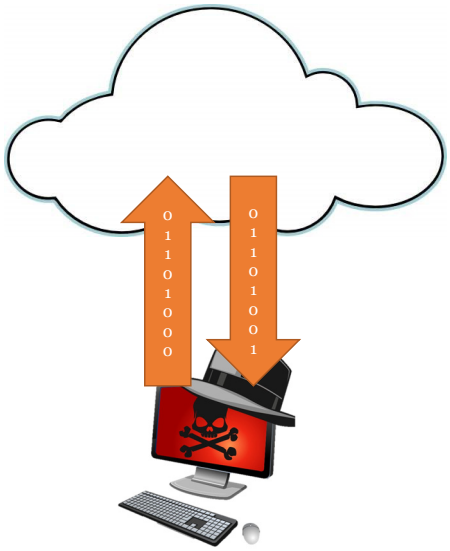
Microsoft Ecosystem

1. Malwarebytes Attack
2. Microsoft 365
3. EXERCISE: Microsoft 365 Exchange Investigation
4. SolarWinds Attack
5. Azure Active Directory (AD)
6. EXERCISE: Introduction to KQL
7. **Storage Monitoring**
8. EXERCISE: Log Analytics Using Azure CLI
9. Detection Services
10. EXERCISE: Microsoft Defender for Cloud and Sentinel
11. Network Traffic Analysis
12. EXERCISE: Azure Network Traffic Analysis

This page intentionally left blank.

Cloud Storage Concerns

- Azure has many services that could house sensitive data
  - Managed database offerings
  - Azure Storage
  - Azure Virtual Disks
- This module focuses on four major concerns
  - Egress of sensitive data
  - Ingress of data to be later exfiltrated
  - Sensitive data in public-facing storage
  - Theft from managed databases



Company and customer data can take many forms., from personal data that, if stolen, could require funds set aside to pay for credit monitoring—something that many companies continue to pay—to proprietary data that, if acquired by a competitor, may remove your company's competitive edge. In any case, protection of this data is crucial for the good of the company and customers.

This sensitive data can be stored in cloud environments in different ways. A typical offering from an Infrastructure as a Service (IaaS) provider is a managed database. With these offerings, the underlying operating system and database application are wholly managed by the provider—making these services more like Platform as a Service offerings (PaaS). The data, however, is read, written, modified, and removed by the customer. Examples of these services from Azure are Azure SQL servers and Azure SQL databases. Both of those services go hand-in-hand.

Azure Storage is a common service in Azure that simply stores the company's files. As you will notice on the following page and as you have learned in a previous section, this service can store data of many different forms. When using virtual machines in an Infrastructure as a Service cloud provider like Azure, those virtual machines require a virtual disk. These virtual disks, just like physical hard drives, can contain not only operating system files, but also this concerning data.

No matter the solution, there are common concerns regarding the data stored within them. Often, the first concern most will think of is this sensitive data leaving their cloud environment. And for good reason. Probase, among many other users of Azure Storage, lacked proper security controls around their blob storage-hosted files and this data was easily exfiltrated. We will discuss these at length over the course of this module.

We are not only concerned about data theft. Attackers may use our own cloud storage against us. For instance, data could be compiled in cloud storage offerings for later sharing or exfiltration. Also, attack tools, instead of being pulled from a domain that may be previously deemed malicious, could be brought into the cloud environment for later use.

Image reference:
http://www.clipartbest.com/cliparts/ace/XA7/aceXA7rri.jpg

Reference:
https://www.techradar.com/news/microsoft-azure-breach-left-thousands-of-customer-records-exposed

Azure Storage

And now, a quick reminder of data types and storage locations for Azure Storage before we dive deeper into the evidence

Offering	Used to store	Log Analytics Table
Blob Containers	Text and binary data	`StorageBlobLogs`
File Shares	SMB- or NFS-accessible files	`StorageFileLogs`
Tables	NoSQL data	`StorageTableLogs`
Queues	Messages between application components	`StorageQueueLogs`

We will begin discussing how to analyze activity within the Azure Storage service. As mistakes are made when configuring this service or attackers gain privileges to crawl these storage locations, we must determine how to capture and analyze the event data made available to us by Azure.

The second book (SEC541.2) discussed the various offerings in this service, but here is a quick recap.[1] First up: Blob containers (Blob stands for Binary Large Object). This service offering is very similar to AWS's Simple Storage Service (S3) offering in that any file can be stored within this service whether it is a simple text file or a binary file.

File shares are very similar to on-premises network-attached storage solutions. Systems inside or outside of the cloud environment can connect using the Server Message Block (SMB) or Network File Share (NFS) protocols to access this data.

Tables and queues can also be leveraged in the Azure Storage service. Tables, as the name may imply, are used to store NoSQL database entries. The target audience for this service are companies who want to send data to a database without the hassle of setting up their own or even a managed database instance. Queues are useful for those wishing to send messages between various components in their cloud-based application.

Note that none of these offerings are logging anything by default. Azure customers must navigate to the Diagnostics Settings of the Azure Storage account and enable the necessary logs and metrics they want to collect and also send that data to a location of their choosing. The options, again, are another Azure Storage account, Azure Log Analytics workspace, Azure Event Hub, or a partner solution. If sending the data to Azure Log Analytics, each of the four offerings from Azure Storage have their own associated table listed in the chart above.

[1] https://docs.microsoft.com/en-us/azure/storage/common/storage-introduction

MITRE ATT&CK Technique T1530: Data from Cloud Storage Object

- *Adversaries may access data objects from improperly secured cloud storage*
- Threat Groups, such as Fox Kitten, are now stealing this data
- Mitigations include:
  - Auditing for non-compliant configurations
  - Encrypting data at rest
  - Use IAM to restrict data access
  - Filter network connections to objects
- MITRE recommends establishing a detection scheme to identify unapproved storage access

MITRE | ATT&CK®

As you may recall, MITRE maintains several matrices outlining the overall tactics and numerous techniques that an adversary may leverage against an organization and its assets. The most popular is the Enterprise Matrix which contains the most of these tactics and techniques.

A smaller, yet pertinent matrix for cloud users is the Cloud Matrix. One technique that falls beneath the Collection tactic is T1530: Data from Cloud Storage Object. This technique states that, if cloud storage is improperly configured, an adversary can access the data housed in this service. In fact, there is an Iranian threat group called Fox Kitten that does exactly this. According to the Cybersecurity & Infrastructure Security Agency (CISA), this group acquired files from one of their victim's cloud storage instances.

There are a few mitigations to be aware of when attempting to restrict or quickly identify unapproved cloud storage behavior in not just Azure, but any cloud environment—most of which relies upon a proper identity and access management (IAM) solution. For Azure, this means utilizing a least privilege approach in Azure Active Directory to limit who can access this crucial data. Security assurance programs must also ensure that previously configured cloud accounts remain secure over time so regular auditing is needed.

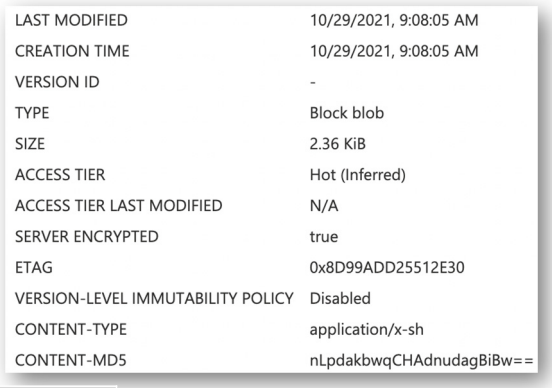
If any of those protections are lacking—and even if they are relatively solid—we must constantly analyze the access attempts in our cloud storage accounts in hopes that, if any malice does exist, it can easily be spotted before the attacker siphons all of the sensitive data from our cloud account. This will be the main focus moving forward in this module.

References:
https://attack.mitre.org/techniques/T1530/
https://us-cert.cisa.gov/ncas/alerts/aa20-259a

Image reference:
https://attack.mitre.org/theme/images/mitre_attack_logo.png

Azure Blob Metadata

- Useful blob metadata includes:
  - Modified/Creation date
  - File type
  - File size
  - MD5 hash of object*
- Retrievable from both Portal (right) and CLI (below)



LAST MODIFIED	10/29/2021, 9:08:05 AM
CREATION TIME	10/29/2021, 9:08:05 AM
VERSION ID	-
TYPE	Block blob
SIZE	2.36 KiB
ACCESS TIER	Hot (Inferred)
ACCESS TIER LAST MODIFIED	N/A
SERVER ENCRYPTED	true
ETAG	0x8D99ADD25512E30
VERSION-LEVEL IMMUTABILITY POLICY	Disabled
CONTENT-TYPE	application/x-sh
CONTENT-MD5	nLpdakbwqCHAdnudagBiBw==

```
az storage blob list --container-name \
  $CONTAINERNAME --account-name $ACCOUNTNAME
```

* If over 64MB, client must specify during upload

You may be tasked to hunt for data stored in an Azure Storage container for many reasons. Of course, you could download every blob housed in the container, but that would be inefficient and cumbersome at best. Lucky for us, Azure generates metadata for blobs uploaded to the container. This data is retrievable using both the Azure Portal (right image above) and the Azure CLI (also shown above). Not all the metadata may be useful in every use case but let us look at a few.

Assume your company has stored historical log data in an Azure Storage container. This data, being log data, is not expected to have changed unless, perhaps, a mistake had been made by an administrator. Even worse, what if a malicious user is trying to cover their tracks by manipulating the records? To catch either one of these, the metadata field of Creation Time would indicate these changes. In fact, being log data, if the Creation Time and Last Modified entries were different, this would show an alteration to this data.

Another interesting field is the file type. Azure does its best to identify what kind of data this blob is. One would assume that this is determined by the "magic bytes"[1] of the file—that is, the first few bytes in the raw data. This is not the case. In fact, the course author generated files with completely random data and supplied a ".pdf" extension. Even though the first few bytes were `7F EB DE 75`, the Content-Type file still specified that this was a PDF file. This must mean that Azure, by default, will use the extension as the Content-Type indicator. This can be overridden, and the content type supplied by the user or client application when the data is uploaded, so not all hope is lost.

The size of the file, indicated by the Size field value, can be very telling if data staging is being performed. Imagine an adversary collecting lots of internal data, compressing it, and uploading it to Azure Storage for exfiltration. The size of the file would likely be quite large and stand out compared to other files stored in that location. Even if that is not their method, the size of any malicious data, once a sense of normal is established, may be indicated if the data is an abnormal size.

[1] https://www.netspi.com/blog/technical/web-application-penetration-testing/magic-bytes-identifying-common-file-formats-at-a-glance/

If the blob is under 64MB in size, Azure will hash and provide the value in the Content-MD5 field. MD5 hashes are hex values (numbers between 0 and 9 and letters between A through F), so why, in this example, does the value contain other letters and the = character? That is because Azure will base64-encode this hash and represent the encoded result as the Content-MD5 field value. If the data is over 64MB, this value is blank unless the customer or application, during upload, provides this base64-encoded hash in the Content-MD5 HTTP request header.

Anatomy of Azure Storage Logs

- **Forty-six** columns in a **`StorageBlobLogs`** table as of October 2021
- Most valuable columns:

Column Name	Description
AccountName	Azure Storage Account targeted by the operation
AuthenticationType	How the user authenticated (e.g., Access Key, Anonymously)
CallerIpAddress	Public IP address and ephemeral TCP port of the client
OperationName	Azure operation that affected the requested resource
StatusText	Shows whether the request failed or was successful
Uri	The full URI of the requested Azure Storage resource
UserAgentHeader	Client platform making the request

Now that the log data is hopefully being collected, we must know what we are dealing with so that we can successfully analyze this data. As an example, the data stored in the `StorageBlobLogs` table in Azure Log Analytics workspaces has many available fields. In certain respects, all forty-six of these columns (as of October 2021) can prove valuable, but we will focus on what will likely be the most valuable in the event of a breach:

- `AccountName`: This column specifies the Azure Storage account that is being interacted with—whether a create, read, update, or delete action.
- `AuthenticationType`: This tells you if and how the user authenticated with the Azure Storage service. You may even see, in the case of public-facing Azure Storage containers, that the data may be accessed anonymously.
- `CallerIpAddress`: The requestor's public IP address can be collected from this column value. Just like any other log collecting a public IP, if the data is proxied, you may only be able to collect the proxy system's IP address as the "real" IP address will be unknown if not expressed in the X-Forwarded-For HTTP request header from the proxying system.
- `OperationName`: This column represents the API call being made to the Azure Storage service. This would help indicate what the user is trying to do (e.g., read a sensitive file, modify a log entry).
- `StatusText`: Was the action successful or not? This column will tell you.
- `Uri`: Notice that "file name" or similar is not mentioned? That is because a field to easily indicate the blob being accessed does not exist. Instead, this column would contain the name of the file but must be extracted from the full URI.
- `UserAgentHeader`: This column will inform you which platform—that is, client software and possibly even operating system—initiated the request. This value would prove valuable in identifying abnormal clients contacting the Azure Storage service.

Discovering Public Azure Storage Containers

- Azure Storage Containers have three **public access levels:**
  - **Private**: No public access to container or blob data
  - **Blob**: No public access to container, but public access to blob data
  - **Container**: Public access to container and blob data
- Can be discovered via the Portal rather easily or via two command-line options:
  - PowerShell:

```
Get-AzStorageContainer -Context $context | `
  Where-Object PublicAccess -ne "Off"
```

  - Bash:

```
az storage container list --account-name $storageAcctName \
  --query [?[].properties.publicAccess=="blob|container"]
```

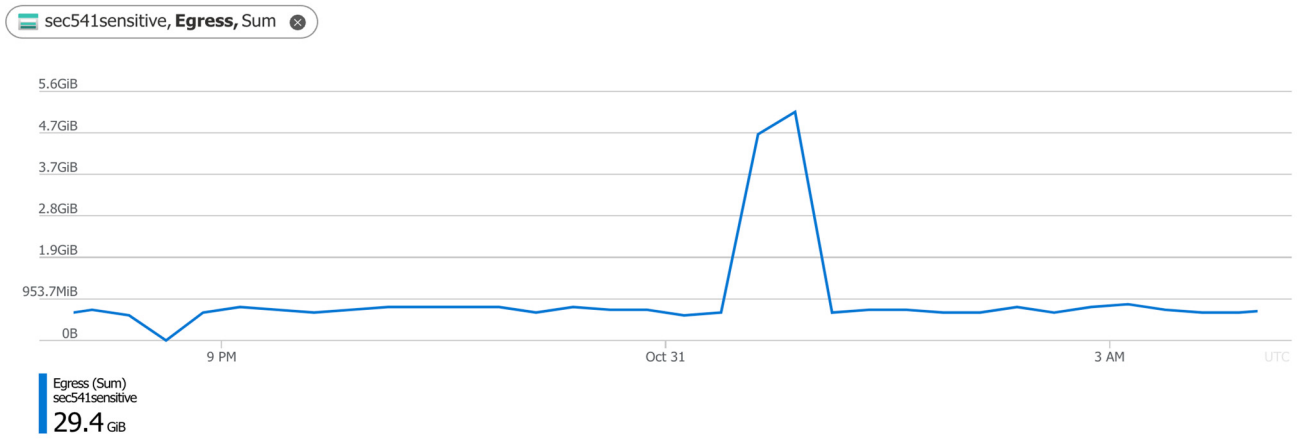
It is quite easy to control the access to Azure blob storage in the Azure Storage service. There are three options to set the container's public access level to. The first option, and the default in Azure, is Private, which means only Azure Active Directory users with proper rights can access the container and its stored data. The second option, Blob, on the other hand, allows users, both inside and outside of Azure, the ability to access the blob data. Lastly, and the one exposing the most to the public, is Container. This option allows for not only reading the blob data, but also listing the container's contents.

Identifying the accidental or malicious exposing of this data can be acquired from the log data, but there are a few easy ways to identify publicly-accessible Azure Storage containers and blobs. First is to view the Azure Portal and the Public access level column, which appears when you navigate to the Azure Storage account. This is easy, but not very efficient, as you would need to navigate to each storage account individually.

To add efficiency, you can leverage the tools provided by Azure: the Azure PowerShell `Get-AzStorageContainer` cmdlet and the Azure CLI tool's `az storage container list` command. These will both output the public access level. The tool outputs, however, are not consistent. Instead of specifying that the public access level is Private, Blob, or Container like in the Portal, the Azure PowerShell `Get-AzStorageContainer` cmdlet will show the `PublicAccess` property as Off, Blob, or Container. Likewise, the Azure CLI's `az storage container list` command will show the `publicAccess` value as null, blob, or container—all of these are lower-case.

Identifying Data Exfiltration with Metrics

Can you venture a guess about when there may have been data exfiltration from this Azure Storage account?



Often overlooked, metrics can provide visualizations at-a-glance if used properly. Unlike several of the logging options we will look at, metrics in Azure Storage are available by default. Above is one such example: a visualization of the amount of data leaving an Azure Storage account named sec541sensitive.

Before analyzing this data, you may be wondering how this visualization was created. To get to the proper screen in the Azure Portal, you must navigate to the Azure Storage account in question. When arriving at the Overview page for the Azure Storage account, on the left pane under Monitoring you will find a Metrics entry to click on. A new page will present you with a default chart where you must make a few selections based up what you would like populated in the chart.

The first option to consider is the time frame. In the above example, which is the default, the last 24 hours was selected.

After the time frame is selected, a namespace selection will determine the scope of the visualization. The options for Azure Storage are Account (as shown above), Blob, File, Queue, and Table.

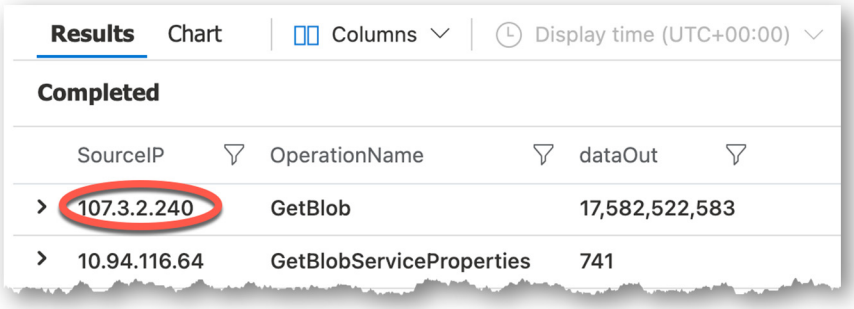
Next is the metric to display. Depending on the namespace selected, these options will change. Since we are trying to visualize a spike in egress from an Azure Storage account, the Egress metric is selected.

Finally, the aggregation. This option determines how the data is represented: the sum of the activity (as selected above) or the average, minimum, or maximum over time.

After selecting the above options, you can clearly see that there was a spike sometime after 12 AM UTC on October 31st. The next logical step will be to review the other events that occurred around this time by pivoting to the log data.

Identifying Data Exfiltration with KQL

```
1  StorageBlobLogs
2  | where TimeGenerated between(datetime("10/31/2021 12:00 AM")..
3    datetime("10/31/2021 3:00 AM"))
4  | extend SourceIP = split(CallerIpAddress, ":")[0]
5  | summarize dataOut = sum(ResponseBodySize) by tostring(SourceIP), OperationName
6  | sort by dataOut
```



Metrics can be the first indicator that something is amiss or abnormal, but more detail is necessary to determine if this is a mere misconfiguration or malice. To gain this level of detail, understanding Kusto Query Language (KQL) queries will come in quite handy. This is assuming, of course, that the storage events are being forwarded to an Azure Log Analytics workspace. Not all hope is lost if the events are shipped elsewhere as you would need to understand the log format in the destination location—whether it is Azure Storage and its JSON-formatted logs, streamed through Azure Event Hub, or sent to a partner solution.

As you may remember, Azure Storage blob logs, when sent to an Azure Log Analytics workspace, land in a table called `StorageBlobLogs`. As most queries do, you can see that the example query starts off with the appropriate table name. The rest of the query will attempt to identify the top source IP addresses by the amount of data that they retrieve from Azure Storage.

Next is to determine the timeframe which, based on the last page, should fall somewhere between midnight and a few hours later as that is when the spike of egress activity occurred. This filtering can be done one of two ways: via the time range selector in the Azure Portal or as part of the query itself—which is what you see on the next two lines of the query as part of the `where` clause.

The `CallerIpAddress` is a very interesting column and will allow you to identify who made the data egress requests. There is one problem, however. The field consists of not just the IP address, but a colon and port number after the IP address indicating one side of a TCP socket. To get around this, a new column (`SourceIP`) is created using the extend option. To create the values for this column, the `CallerIpAddress` column's values are split by the colon character (`:`) and the substring to the left of that character is selected.

The `summarize` option, beginning on the fifth line, is used to add all the bytes coming from Azure Storage and break down the data based on the new `SourceIP` column and the operation that was being performed.

We can now see, based on the results of the query, that 107.3.2.240 is the offending IP pulling all this data from Azure Storage. Next, of course, would be to determine if this is an expected system or not. If not, there may be data exfiltration!

MITRE ATT&CK Technique T1074.002: Remote Data Staging

- *In cloud environments, adversaries may stage data within a particular instance or virtual machine before exfiltration. An adversary may Create Cloud Instance and stage data in that instance.*
- Mitigation is tricky as attacker may already have control of the cloud resources
- Detection is a must
  - Cloud resource creation events such as a new Azure Storage container
  - Data upload events—likely in large batches

MITRE | ATT&CK®

As dangerous as unwanted egress traffic is, we must also consider the dangers of an attacker bringing data into our cloud storage platforms. MITRE again brings us information with its ATT&CK technique T1074.002: Remote Data Staging. This technique involves ingress traffic to cloud storage in an effort to accomplish one of a few things.

First, they may elect to store sensitive company data—be it customer data, proprietary data, or anything else that the company would not want exposed and is worth something to the attacker. Once the data is stored in this single location, the data can be exfiltrated as we previously discussed.

Another popular reason for storing data in the victim's cloud storage solution is to store malicious files that may be used later in the attack campaign. This allows the attacker to retrieve the files without the risk of network- and host-based detection controls flagging the files as they may otherwise be coming from a known malicious domain or IP address. It is not likely that the company would capture their own cloud storage options in their threat intelligence.

To store these files in, say, Azure Storage, an attacker may have access due to lacking security controls such as public-writable permissions. The most likely method, however, is that the attacker already has presence in the environment via credentialed access. This allows the stolen credential or attacker-created cloud account to legitimately access the storage as long as they have the proper identity and access management (IAM) permissions. In the case of Azure Active Directory, the attacker would need `Microsoft.ClassicStorage/storageAccounts/listKeys/action` rights to list the storage account key required to write an Azure Storage blob file.

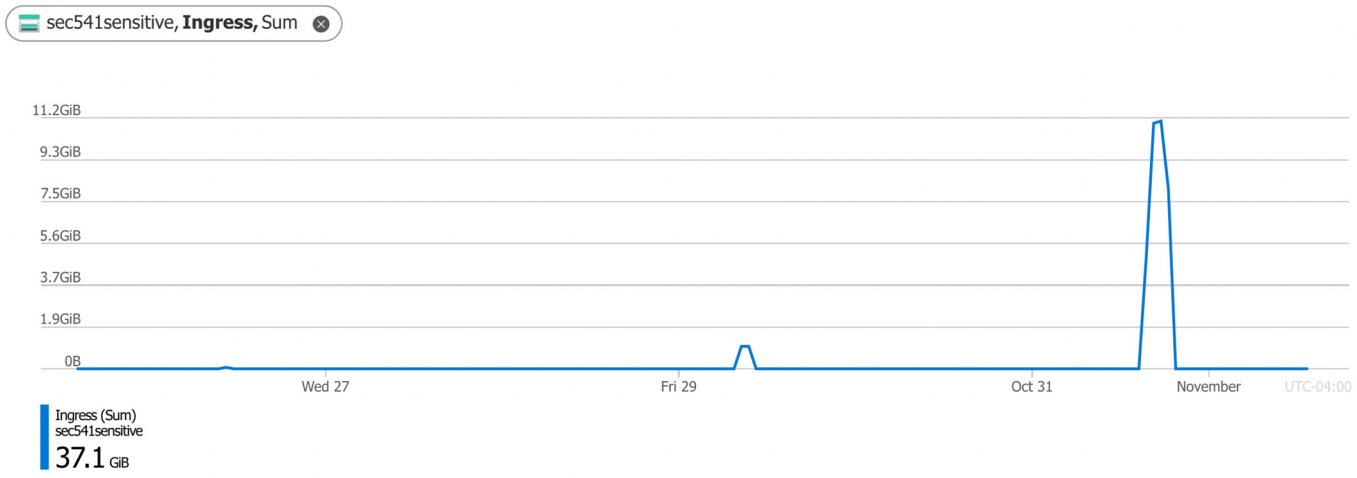
Knowing this, mitigation would involve aggressive, least-privilege-focused rights management. Even then, if a user account is compromised, we also need to focus on detections—which the next few pages will focus on greatly.

Reference:
https://attack.mitre.org/techniques/T1074/002/

Image reference:
https://attack.mitre.org/theme/images/mitre_attack_logo.png

Identifying Data Staging with Metrics

Which day had a large spike in uploaded data?



Just like the last time you saw the Metrics results in Azure Storage, this chart shows another large spike worth investigating. The difference between this chart and the last one is that the last chart was looking at egress data and this one is looking at ingress data. Ingress data is chosen because if an attacker were attempting to stage data for later exfiltration or as a means to store their malicious tools, a spike like this could be a telltale sign of this activity. That is, of course, if the activity does not blend into normal production ingress traffic.

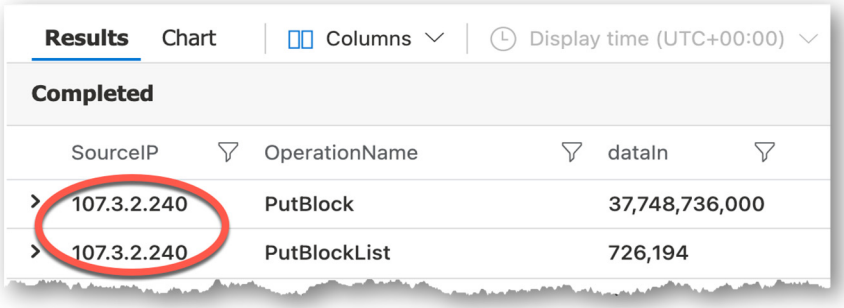
Another difference between this chart and the last is that the last chart's time zone was in Universal Coordinated Time (UTC) and this one is not. This chart, as is the default in Azure, is displayed in the user's local time zone. In this case, the course author was in the Eastern part of the United States. Given the time of year this visualization was captured, it was Eastern Daylight Time (EDT) which is four hours behind UTC—hence the UTC-04:00 which can barely be made out in the bottom right of the chart in light gray.

The time frame you are looking at is not the default of the last 24 hours but of the last week. This is necessary to get a better sense of normalcy in the environment if we did not already know. There are a few small spikes on what appears to be October 26th and 29th, but one spike does seem to stick out on October 31st.

To see the exact time of the spike, the chart does allow to drag and drop over an area (perhaps just to the left and right of the large spike). We will just assume, for now, that the possibly malicious incident happened sometime on October 31st, 2021. Now, let's take a look at what Azure Log Analytics has available for us by crafting a KQL query to investigate this incident.

Identifying Data Staging with KQL

```
1  StorageBlobLogs
2  | where TimeGenerated between(datetime("10/31/2021 12:00 AM")..
3    datetime("11/01/2021 12:00 AM"))
4  | extend SourceIP = split(CallerIpAddress, ":")[0]
5  | summarize dataIn = sum(RequestBodySize) by tostring(SourceIP), OperationName
6  | sort by dataIn
```



The KQL query that appears above is nearly identical to the previous query you saw just a few pages ago. There are a few exceptions that will be pointed out as we break down this query.

As usual, the query begins with the table we are investigating—`StorageBlobLogs`. The `where` statement is where things change from the last query. Since the egress and ingress traffic occurred at different times, we are searching for the activity related to that October 31st peak. To do so, the `TimeGenerated` values must have a value between midnight on October 31st and midnight on November 1st.

The `SourceIP` column must be generated again since we want to know who has been making these requests regardless of the ephemeral port chosen each time. Otherwise, there may be several `CallerIpAddress` entries for the same host.

The next difference which may be hard to spot is the summarize line. Unlike the egress analysis, which is interested in adding the `ResponseBodySize` values for each `SourceIP`, ingress analysis is adding the `RequestBodySize`. Given this, we also elected to change the `dataOut` column name the more appropriate name of `dataIn`.

Once the query is sent to Azure Log Analytics, we see, yet again, that 107.3.2.240 is the culprit. In fact, two `OperationName` values appear: `PutBlock` and `PutBlockList`. The `PutBlock` operation is used to upload chunks of data to a blob and `PutBlockList` identifies the blocks that make up the uploaded blob file.

Azure-Managed Database Services

- Azure provides many database storage options depending on customer needs:
  - Azure SQL
  - Azure Cosmos DB
  - Azure Database for PostgreSQL
  - Azure Database for MySQL
  - Azure Database for MariaDB
  - Azure Cache for Redis

- All remove the need for the customer to manage an underlying OS and database application
  - Just focus on the data!

There is no shortage of options for you to store data in a database in Azure.[1] You could always create your own server by deploying a virtual machine, configuring and hardening the operating system (OS), installing, configuring, and hardening the database application, setting up the schema for the database instance, and then finally get to pushing data into and pulling data out of the database. To ease this burden (and, according to the course authors, it sure is a burden) Azure will address most of those activities for you so you can focus on the more important tasks of creating, reading, updating, and deleting database entries.

The selection of which Azure service to choose really comes down to the database application you are offloading to Azure. This is typically driven by the application that is managing or reading the data—often an artifact of an on-premises deployment moved into cloud. If a Microsoft SQL connection is expected, then Azure SQL would be the proper Azure service. The same can be said for NoSQL- (Azure CosmosDB), PostgreSQL- (Azure Database for PostgreSQL), MySQL- (Azure Database for MySQL), MariaDB- (Azure Database for MariaDB), and Redis-based (Azure Cache for Redis) applications.

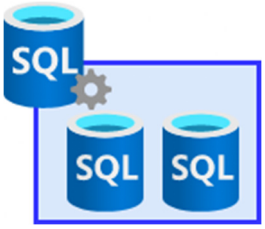
When those offerings are leveraged, it is up to the customer to, as usual, configure the service components properly. "Properly" meaning that operational requirements, security best practices, and availability concerns are fully addressed.

Once the setup of the database environment is taken care of—usually during the deployment of the database instance—the database environment (minus the data) is kept operational by the cloud provider.

[1] https://azure.microsoft.com/en-us/solutions/databases/

Azure SQL Servers and Databases

- **Azure SQL server** resources are required to be in place to host databases
  - Logical container for managing databases
  - Authentication options include:
    - **SQL authentication**
    - **Azure Active Directory**



- Database servers are given a hostname in the well-known **database.windows.net** domain
- **Azure SQL databases** are deployed with these considerations:
  - Compute resources
  - Scalability
  - Networking
  - Logging/auditing

This section would get quite lengthy if we were to conduct a deep-dive into each and every database offering Azure makes available to customers, but let's focus on one in particular: Azure SQL. Why? Azure is most likely going to be the provider that draws the most customers for SQL servers due to Microsoft's claim that "AWS is 5 times more expensive than Azure for Windows Server and SQL Server."[1] That, plus many of the technical challenges and attack techniques related to databases tend to affect each type of database similarly; there is no need to rehash these same facts repeatedly.

To begin with an Azure SQL deployment, there are two components that are necessary to set up as a cloud customer: Azure SQL server and Azure SQL database resources. The order does matter as a database cannot be deployed without a server being first provisioned. When deploying the Azure SQL server, there is little to address as this "server" is merely a logical collection of database instances.

One very important item to address when deploying an Azure SQL server is how to connect to it. There are two options currently supported in Azure. The first is to connect using the tried-and-true SQL authentication native to Microsoft SQL. This works just fine, but the drawback is how to maintain users at scale. If managing multiple users for each and every server, this becomes cumbersome. The other, likely more efficient, option is to use Azure Active Directory for authentication. This allows you to point to existing users and grant them access to the database server and the database instances therein. The downside is that the application must have the ability to connect using Azure Active Directory as well—something that many web applications do not do without some heavy modification to the source code.

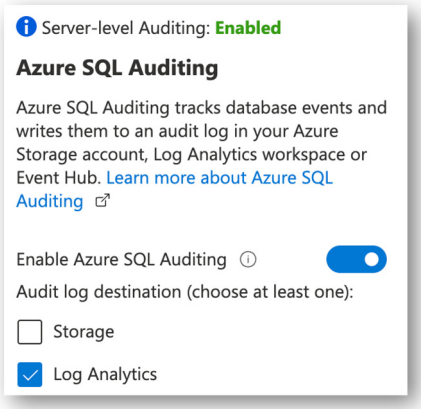
The majority of the configuration for a database deployment in Azure is during the creation of the instance in the Azure SQL database service. There are several factors to consider like how much CPU, RAM, and disk space do you expect to consume? What about how it is connected (over the internet or via a private connection)? Do you require redundancy? What about logging?

Though these questions are important, we will focus our efforts on the evidence that can be at our disposal if this deployed SQL database is under (or was under) attack.

[1] https://azure.microsoft.com/en-us/overview/azure-vs-aws/cost-savings/

Azure SQL Database Auditing

- Two levels of auditing:
  - **Server-level**: All databases will be audited
  - **Database-level**: Only the database configured is audited
  - Server-level will override database-level
- Logs arriving at Azure Log Analytics will be stored in the **AzureDiagnostics** table with a **Category** of **SQLSecurityAuditEvents**
- **Diagnostics Settings** can also be used to capture metrics-related data and ship to the usual locations (e.g., Storage, Log Analytics, Event Hub)



By default, there is little insight into the happenings of the Azure SQL database other than some metric data. We are not discounting that data as you will see shortly how valuable metrics can be in identifying abnormal database behavior, but you do need more visibility. This can be achieved by enabling auditing in either the Azure SQL server or Azure SQL database configuration referred to as server-level and database-level auditing, respectively.[1]

Server-level logging, as the name implies, is configured in the Azure SQL server configuration. When implemented, all databases, regardless of their individual auditing settings, will begin auditing all their interactions—most important of which being the queries issued to the database. If server-level auditing is not possible, there is another option referred to as database-level auditing that allows you to audit the databases of your choosing. This is performed by configuring the auditing settings of each of the appropriate Azure SQL databases.

In either case, when enabling auditing, you must choose the destination of the log data similar to what you have seen with Diagnostics Settings. There are fewer options, however, to send the data to: an Azure Storage container, an Azure Log Analytics workspace, or both simultaneously. As we have in the previous cases in this module, we will view the data in the more capable solution—Azure Log Analytics workspaces.

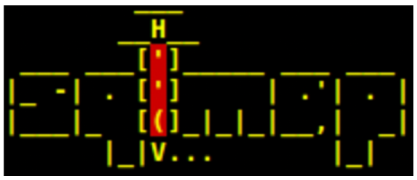
Unlike some other Azure services, when the data arrives in an Azure Log Analytics workspace, it is placed in the generic AzureDiagnostics table, so finding this data requires and extra step: adding `where Category == "SQLSecurityAuditEvents"` to the KQL query.

If more data is required, metrics can also be sent to Azure Storage, Azure Log Analytics workspaces, Azure Event Hub, and partner solutions using Diagnostics Settings configurations as you are now quite familiar with.

[1] https://docs.microsoft.com/en-us/azure/azure-sql/database/auditing-overview

MITRE ATT&CK Software S0225: `sqlmap`

- Attempts to uncover SQL injection flaws in web applications:
  - **Fingerprint** database backend
  - Determine potential **injection avenues**
  - Expose **structure** of databases and tables
  - **Exfiltrate** database table entries
  - Establish a **shell session**
- Default indicators include:
  - Hundreds or thousands of queries sent
  - User-agent starting with **`sqlmap/`**
  - Abnormal database queries



```
SELECT * FROM CCData WHERE First=
'test' AND 5118 IN (SELECT (CHAR(
113)+CHAR(122)+CHAR(120)+CHAR(98)+
CHAR(113)+(SELECT SUBSTRING((
ISNULL(CAST(ExpDate AS NVARCHAR(
4000)),CHAR(32))),1,462) FROM
(SELECT ExpDate, ROW_NUMBER() OVER
(ORDER BY (SELECT 1)) AS LIMIT
FROM sec541.dbo.CCData)x WHERE
LIMIT=2)+CHAR(113)+CHAR(118)
+CHAR(98)+CHAR(113)+CHAR(113)))--
asup' AND Last='test'
```

MITRE | ATT&CK® [1]

A common tool to attack databases by way of vulnerable web application code is sqlmap.[2,3] This tool, as outlined by MITRE ATT&CK software numbered S0225, can be used for many of the attacker's tactics. Here is an incomplete list of some of the more common uses for this tool as they map to MITRE ATT&CK techniques:

- T1595.002 (Vulnerability Scanning): sqlmap can target a web page and identify to the attacker the likely ways that it can be exploited by sending it various stimuli and analyzing the results.
- T1592.002 (Software): The tool can query the web page in many ways and, again, analyze the results to come to a conclusion whether the backend is likely MySQL, Microsoft SQL, PostgreSQL, or other database software.
- T1082 (System Information Discovery): Once an attack vector is chosen by the adversary, sqlmap can be instructed to crawl the database server and identify database instances, tables, and even the database entries themselves—essentially exfiltrating all data.
- T1059.003 (Windows Command Shell) and T1059.004 (Unix Shell): sqlmap may be leveraged, if the system is vulnerable to the technique, to establish a shell session with the database server. This would allow a foothold into the underlying cloud environment which hosts the database server.

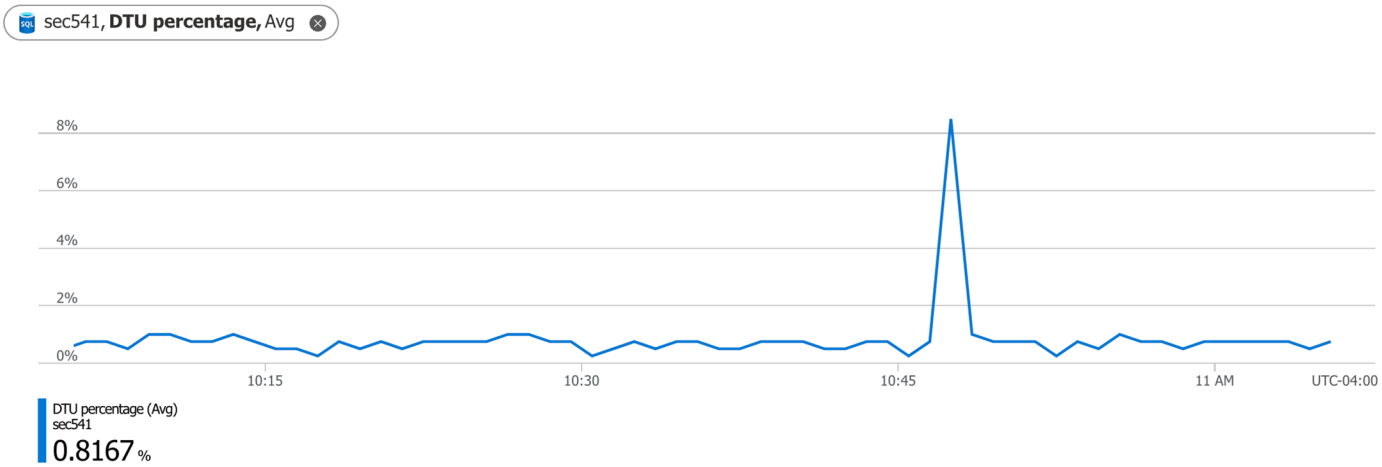
When sqlmap is being used, it is very likely to be noticed if you are paying attention to the log data. This is because the tool is quite noisy—meaning there are several hundreds or thousands of requests being sent to the web server when it is being analyzed or exploited. Not only are the requests numerous, they tend to be quite large in size. Because of this, the web server will be sending very large database queries to the database which are likely much larger than the legitimate requests it normally receives.

Furthermore, unless the attacker is changing it, the user-agent string will start with sqlmap—making the attack quite identifiable in web server access logs.

[1] https://attack.mitre.org/theme/images/mitre_attack_logo.png
[2] https://attack.mitre.org/software/S0225/
[3] https://github.com/sqlmapproject/sqlmap

Identifying sqlmap with Metrics

When was there a large increase in **Data Transaction Units**?



Just like with Azure Storage, the Azure SQL Database service allows you to monitor many metrics related to your database instances such as CPU usage, data metrics like space used and input/output, number of sessions, denied network traffic, and several more including one family of metrics that we will focus on—Data Transaction Units (DTUs).

"What are DTUs?", you may ask. According to Spotlight Cloud, they represent "a mixture of the following performance metrics as a single performance unit for Azure SQL Database."[1] This includes the CPU utilization, memory usage, and data and logging input to and output from the database instance. When creating a SQL database, customers will size their database instance by how many DTUs they expect to consume. The DTU percentage metric can be used to visualize how much of the allocated DTUs are consumed at any given time.

Since SQL injection attack payloads from sqlmap tend to be very aggressive in size and speed, it is highly likely that you will notice a spike in DTU percentage as you can clearly see above. This visualization is showing you the DTU percentage from the last hour, so you get more granularity than what you saw in the previous two examples. Here, you can see a spike of over 8% DTUs in use just after 10:45 Eastern Daylight Time (EDT). Now, this could just be a large, benign transaction occurring or maybe it is an attack against the database.

[1] https://www.spotlightcloud.io/blog/what-is-dtu-in-azure-sql-database-and-how-much-do-we-need

Identifying sqlmap with KQL

```
1  AzureDiagnostics
2  | where TimeGenerated between(datetime("10/29/2021 2:45 PM")..
3    datetime("10/29/2021 3:00 PM")) and Category == "SQLSecurityAuditEvents"
4    and action_id_s == "BCM "
5  | extend statementLength = strlen(statement_s)
6  | sort by statementLength
7  | project client_ip_s, statementLength, statement_s
```

client_ip_s	statementLength	statement_s
107.3.2.240	883	SELECT * FROM CCData WHERE First='test' AND 5705 IN (SELECT (CHAR(113)+CHAR(113)+CH
107.3.2.240	883	SELECT * FROM CCData WHERE First='test' AND 7459 IN (SELECT (CHAR(113)+CHAR(113)+CH
107.3.2.240	883	SELECT * FROM CCData WHERE First='test' AND 7198 IN (SELECT (CHAR(113)+CHAR(113)+CH
107.3.2.240	883	SELECT * FROM CCData WHERE First='test' AND 8636 IN (SELECT (CHAR(113)+CHAR(113)+CH
107.3.2.240	858	SELECT * FROM CCData WHERE First='test' AND 9158 IN (SELECT (CHAR(113)+CHAR(113)+CH

And now to investigate the spike in DTU averages using KQL in Azure Log Analytics. Unlike the Azure Storage blob logs which have their own dedicated table, the data for Azure SQL databases is placed in the `AzureDiagnostics` table.[1]

This table may have data for several unrelated Azure resources. You can narrow the search down to only Azure SQL database data by adding `Category == "SQLSecurityAuditEvents"` to the `where` clause.

There is another condition tacked onto the `where` clause narrowing the results to only `action_id_s` values of `"BCM "` (yes, the space is there on purpose as you will learn shortly). First, `action_id_s` is a four-character (hence the space) indicator of the type of action and `"BCM "` means Bulk Change Map.[2] This records all changes to the database—including the SQL statement. The full SQL statement is what we are most interested in as it may show us odd or malicious database queries passed on by the application or directly by the attacker if they are able to interface with the database.

As noted, one way to detect sqlmap is that, unless your database is accustomed to very lengthy queries, identifying the query length and sorting by the longest may pinpoint this attack tool. That is exactly what is being performed in the fifth line. Here, a new column is being created (`statementLength`) and is used to note the character length of the `statement_s` values.

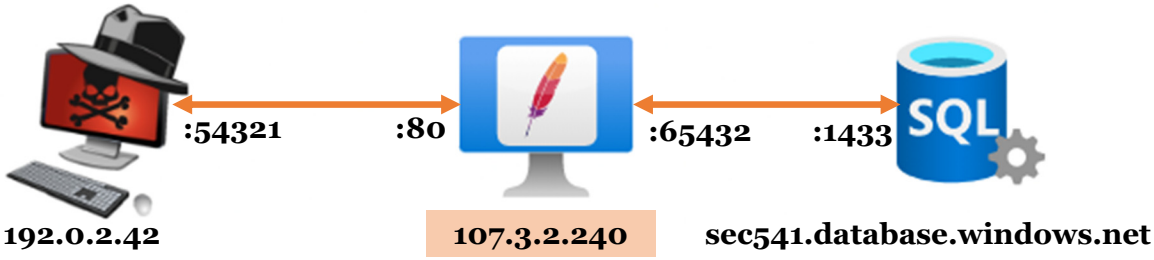
Finally, the last two lines of the KQL query will sort the data by the statement lengths and show you the values of the `client_ip_s` (requestor's public IP address), `statementLength`, and `statement_s` (the full SQL query) columns.

Once more, we have a hit for 107.3.2.240. This attacker is relentless!

[1] https://docs.microsoft.com/en-us/azure/azure-sql/database/audit-log-format
[2] https://sqlity.net/en/2399/bcm/

But Who Was the True Source?

- Azure SQL database logs contain the **`client_ip_s`** column
  - This field is the web server making the request, **not** the attacker!



- Must consult with and correlate the web server access logs!

```
192.0.2.42 - - [29/Oct/2021:06:21:06 -0700] "GET /customers.php?
First=test%27%29%29%3BSELECT%20%28CASE%20WHEN% <snip> -&Last=test
HTTP/1.1" 200 360 "-" "sqlmap/1.5.3#stable (http://sqlmap.org)"
```


To fully understand the Azure SQL database log data being written to the Azure Log Analytics workspace, we must first understand the infrastructure. Failure to do so will cause us to make false assumptions about portions of the narrative we are trying to create related to the SQL injection attack.

In this scenario, we have to determine what is normally communicating with the Azure SQL database instance. This database is used as a backend data storage resource for a web application located on a system with the IP address of—wait for it—107.3.2.240.

That IP looks familiar—it's the attacker's IP address! That is where things get a bit confusing as it appears that there is a malicious user on that web server attacking the database. While this may be true, the likely scenario is that the attack is being proxied by the web server as that is primarily how these types of SQL injection attacks are facilitated—abusing a trusted, public-facing resource to acquire data from a database backend.

To test the hypothesis, a natural log to analyze and correlate to the database events is the web server's access log. To correlate, you will take one of the example `statement_s` column values from the KQL results and conduct a search. This only works with `GET` requests in default Apache access logs since POST data and other HTTP body contents are not captured in this log.

Luckily, we do have a match that you can clearly see in the gray box above. It appears that the true source, which used the web server to aid in the SQL injection attack, is located at 192.0.2.42. This whole time, the attacker was one of our own assets that was compromised by, we believe, 192.0.2.42. The attacker is using our own infrastructure against us!

Course Roadmap

- Section 1: Management Plane and Network Logging
- Section 2: Compute and Cloud Services Logging
- Section 3: Cloud Service and Data Discovery
- **Section 4: Microsoft Ecosystem**
- Section 5: Automated Response Actions and CloudWars

Microsoft Ecosystem

1. Malwarebytes Attack
2. Microsoft 365
3. EXERCISE: Microsoft 365 Exchange Investigation
4. SolarWinds Attack
5. Azure Active Directory (AD)
6. EXERCISE: Introduction to KQL
7. Storage Monitoring
8. **EXERCISE: Log Analytics Using Azure CLI**
9. Detection Services
10. EXERCISE: Microsoft Defender for Cloud and Sentinel
11. Network Traffic Analysis
12. EXERCISE: Azure Network Traffic Analysis

This page intentionally left blank.

Lab 4.3 | Log Analytics Using Azure CLI

Exercise Duration: 30 Minutes

Objectives

In this lab, we will leverage the Azure CLI tools to discover several additional pieces of evidence related to the previously discovered attack by:

- Exploring Azure CLI Log Analytics queries
- Identifying discovery attempts (T1526)
- Finding Managed Identity usage (T1552.005)
- Identifying data exfiltration (T1530)

And now on to the third exercise of the book: Log Analytics Analysis Using Azure CLI. This lab continues analysis of the attack we were analyzing in the second lab. In fact, you will be performing even more analysis using KQL except, in this lab, from the command line.

This lab also includes more MITRE ATT&CK techniques and processes to discover these activities in an Azure environment. You will find that this attack campaign also included cloud resource discovery, "living off the land" by using a legitimate managed identity, and exfiltrating data from cloud storage.

Course Roadmap

- Section 1: Management Plane and Network Logging
- Section 2: Compute and Cloud Services Logging
- Section 3: Cloud Service and Data Discovery
- **Section 4: Microsoft Ecosystem**
- Section 5: Automated Response Actions and CloudWars

Microsoft Ecosystem

1. Malwarebytes Attack
2. Microsoft 365
3. EXERCISE: Microsoft 365 Exchange Investigation
4. SolarWinds Attack
5. Azure Active Directory (AD)
6. EXERCISE: Introduction to KQL
7. Storage Monitoring
8. EXERCISE: Log Analytics Using Azure CLI
9. **Detection Services**
10. EXERCISE: Microsoft Defender for Cloud and Sentinel
11. Network Traffic Analysis
12. EXERCISE: Azure Network Traffic Analysis

This page intentionally left blank.

Microsoft Azure Detection Services

- Azure provides built-in threat detection and alerting services:
  - Microsoft Defender for Cloud
  - Microsoft Sentinel
- Both capabilities require adequate log data stored in an Azure Log Analytics workspace, such as:
  - Azure management activity
  - Azure services' diagnostic logs
  - Virtual Machine logs

- Let's explore each of these as well as many MITRE ATT&CK techniques that these services can discover

There is no shortage of security services in major cloud vendors as evident with our previous discussion on the many, bespoke security services offered by AWS. Azure, as expected and discussed in short earlier in the course, offers security services as well to identify malice in the cloud environment. Where Azure differs is that the services are not as narrow in scope. In fact, there are two that provide a serious amount of detection capabilities regardless of the attacked or compromised resource type—whether that resource is a cloud account, cloud service, or virtual machine operating in, and even outside, of Azure.

The first service we will explore is the newly renamed (as of October 2021[1]) Microsoft Defender for Cloud. This service, formerly known as Azure Security Center, has many elements to spot malicious or suspicious activity in the Azure platform but does require appropriate set up. We will discuss this setup over the following pages and, of course, highlight some sample, common attacks affecting cloud environments and how Microsoft Defender for Cloud can highlight this activity.

Microsoft Sentinel, formerly known as Azure Sentinel, is the second service we will explore. There was much discussion of Security Information and Events Management (SIEM) platforms in the previous book, and this is Microsoft's cloud-native SIEM offering. Just like with Microsoft Defender for Cloud, we will cover several MITRE ATT&CK techniques and highlight how the very powerful Microsoft Sentinel service can pinpoint these attacks before it's too late.

There is one major prerequisite with leveraging either of these services: proper logging must be established for whichever resources you are attempting to protect. This is why so much time was spent discussing various Azure services, what log data is important to collect, and how to collect it.

[1] https://techcommunity.microsoft.com/t5/itops-talk-blog/microsoft-365-and-azure-security-product-name-changes/ba-p/1719167

Microsoft Defender for Cloud

- Multi-purpose security service provided by Microsoft:
  - **Cloud Security Posture Management (CSPM)**: Assess security posture using security benchmarks and various compliance regulations
  - **Cloud Workload Protection (CWP)**: Detect threats in Azure deployments
- Two tiers available:
  - Enhanced security off (free)
  - Enable all (or select) Microsoft Defender for Cloud plans (cost varies)
- For greater visibility, installation of **Log Analytics agents** on VMs can be automated

First up is Microsoft Defender for Cloud. Microsoft claims that this service enhances your security using two different techniques: acting as a Cloud Security Posture Management (CSPM) solution as well as a Cloud Workload Protection (CWP) platform.

CSPMs, as the name may imply, are tasked with identifying deviations from security best practices, benchmark guidance, and compliance regulations. Microsoft Defender for Cloud meets this mark handily by offering an automated and frequent analysis of several common policies. International Organization for Standardization (ISO) 27001, Payment Card Industry (PCI) Data Security Standard (DSS), National Institute for Standards and Technology (NIST) Special Publication (SP) 500-83, and Center for Internet Security (CIS) Azure benchmarks are all offered, just to name some of the more common security best practices and regulations.

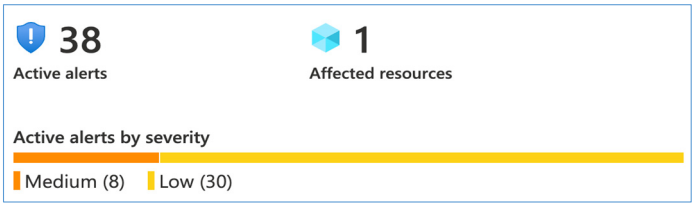
A CWP, on the other hand, is used to identify threats in a cloud-hosted environment. Microsoft Defender for Cloud performs this activity by analyzing the data it has at its disposal related to your Azure management plane, Azure services, and compute environment. Speaking of compute environment, several of the potential alerts that can inform you of suspicious activity rely upon receiving host-level log data. This means that an agent must be installed on the protected systems. At this time, the agent that is expected here is the Azure Log Analytics agent. That is, until Microsoft moves all users over to the Azure Monitor agent.

To really gain the power of Microsoft Defender for Cloud, you must enable certain Enhanced security plans. By default, you will have limited capabilities when Enhanced security is off completely. Those "always-free" capabilities are a continuous assessment of Azure best practices and a secure score based on that analysis. When you enable Enhanced security, you can protect the remaining CSPM and CWP for each plan you enable. So, what are these "plans"? They are categories of cloud services and systems that Microsoft Defender for Cloud can help protect. They are typically priced per protected resource. For example, as of November 2021, you will pay $15 USD per protected virtual machine.

References:
https://docs.microsoft.com/en-us/azure/defender-for-cloud/defender-for-cloud-introduction
https://azure.microsoft.com/en-us/updates/were-retiring-the-log-analytics-agent-in-azure-monitor-on-31-august-2024/

Microsoft Defender for Cloud: Security Alerts

- Powered by **Microsoft Threat Intelligence**[1]
- Requires enabling appropriate plan, such as:
  - *SSH brute force* requires the Servers plan
  - *Unusual deletion in a storage account* requires Storage plan
- Alert data can be streamed to the organization's SIEM
- Since alert data is stored in an Azure Log Analytics workspace, can be retrieved many ways:
  - Az PowerShell cmdlets
  - Azure CLI
  - KQL queries



When you begin to enable Enhanced security plans in Microsoft Defender for Cloud, you not only gain more automated security services, but you also begin to take advantage of Microsoft's own Threat Intelligence. As Microsoft is in many environments and has a large staff, tools, and in-house capabilities, you can utilize their previous work to identify known attacks in your cloud environment.

Threat intelligence is typically a large list of IP addresses, domain names, and file hashes known to have been malicious at one point in time. Microsoft Threat Intelligence provides this and much more. This threat intelligence also includes attacker objectives—very similar to what is seen with MITRE ATT&CK, victimology (i.e., are you a likely target given your industry), and mitigation and remediation information for the identified threats.[2]

If, say, the Servers plan is enabled in Microsoft Defender for Cloud and your VMs are streaming their host data to the platform, you have allowed the service to identify known attack techniques like authentication attacks. What is happening "behind the scenes" is close hold to Microsoft, but one could likely come to the conclusion that the platform is perusing the authentication logs and looking at patterns. Patterns like many failed requests within a given timeframe could be all that is required to determine an authentication attack against the Azure VM.

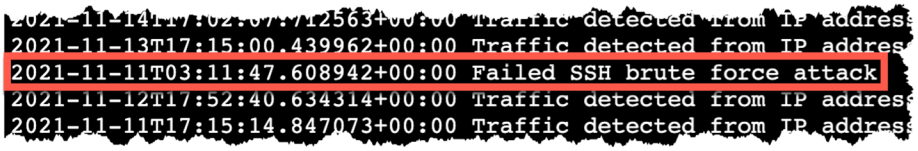
When the threat intelligence features are used and identify an inappropriate actor or behavior, alerts are generated. These alerts can be viewed and drilled into a few different ways: in the Azure Portal (as you see in the Microsoft Defender for Cloud Overview above), Azure PowerShell cmdlets, Azure CLI, and Kusto Query Language (KQL) queries in the Azure Log Analytics workspace housing the Microsoft Defender for Cloud alert data.

If you would like, you also have the option to send this alert data to your corporate SIEM either inside or outside your cloud environment.

[1] https://www.microsoft.com/en-us/insidetrack/microsoft-uses-threat-intelligence-to-protect-detect-and-respond-to-threats
[2] https://docs.microsoft.com/en-us/azure/defender-for-cloud/threat-intelligence-reports

Analyzing Microsoft Defender for Cloud Alerts from the CLI

```
1  az security alert list | \
2    jq -r '.[] | select(.status == "Active") | .timeGeneratedUtc + " " +
3      .alertDisplayName'
4  az security alert list | jq -r '.[] | select(.timeGeneratedUtc ==
5    "2021-11-11T03:11:47.608942+00:00") | .entities[] | select(.type == "ip")'
```



The Azure CLI option you see above uses the `az security alert list` command to extract all alerts from Microsoft Defender for Cloud. There is a catch: the data stored in Microsoft Defender for Cloud is only available for ninety days. If you require this data for a longer time period, you will need to consult the Azure Log Analytics workspace—assuming that the retention period of that workspace is longer than ninety days.

The returned data is quite noisy and includes not only active alerts, but also historical alerts that may have been suppressed. This means that if you only want to view active alerts, some filtering of the returned data is necessary. That is where `jq` will rescue us. By piping (`|`) the results to `jq`, you can select only the entries where the status field value equals `Active`—just like what you see in the example above in lines 1 through 3.

What the above example also includes is an instruction for `jq` to only output two field values: `timeGeneratedUtc` and `alertDisplayName` separated by a space. You can see an example of what is returned in the left screenshot above.

Let's assume that the "Failed SSH brute force attack" caught our eye and we want more information—perhaps *who* (IP address) is attacking our system. This is why the `timeGeneratedUtc` value was captured. We can use this timestamp to generate a second command and `jq` filter (lines 4 and 5) to pull the `entities[]` array from the alert data. This data includes different types of data, so a second select filter is added to only output IP addresses involved.

The second command's output (in the right screenshot above) shows us that the course author's Azure system was likely under attack by a Chinese host at the IP address of 61.177.173.26.

PowerShell Microsoft Defender for Cloud Alert Analysis

```
1  (Get-AzSecurityAlert `
2    | Where-Object {$_.CompromisedEntity -eq "martha" -and $_.AlertDisplayName `
3      -match "Traffic detected from IP addresses recommended for `
4      blocking"}).ExtendedProperties | Format-Table -Wrap
```

```
Key                              Value
---                              -----
investigation Steps              1. Review the IP addresses and determine if they should be communicating with
                                 the virtual machine
                                 2. Enforce the hardening rule recommended by Defender for Cloud which will
                                 allow access only to recommended IP addresses. You can edit the rule's
                                 properties and change the IP addresses to be allowed, or alternatively edit
                                 the Network Security Group's rules directly
destination Port                 22
protocol                         TCP
source IP(s) [Number of attempts] IP: 116.110.3.162 [1]
                                 IP: 106.14.22.73 [1]
                                 IP: 98.126.213.10 [1]
resourceType                     Virtual Machine
killChainIntent                  PreAttack
```


If the Az PowerShell module is imported into your current PowerShell session, you can leverage the `Get-AzSecurityAlert` cmdlet to pull data from Microsoft Defender for Cloud just like in the example above. In this example, we have a very specific hunt to perform in this alert data: find all systems that are communicating with the `martha` VM which are considered risky sources. These "risky sources" are tracked in the Microsoft Threat Intelligence platform and should be identified if spotted in our Azure log data.

To meet the requirements of this hunt, the PowerShell module takes all the returned alerts and pipes (`|`) the results to the `Where-Object` cmdlet to filter the data down using two conditions: the `CompromisedEntity` key must have a string value of `"martha"` and `AlertDisplayName` must have a string value of `"Traffic detected from IP addresses recommended for blocking"`.

If we stopped there, we would see many key/value pairs, but, since `ExtendedProperties` is where the source IP address information is and this field is truncated, we capture just that key value by wrapping our PowerShell command in parentheses, extract the `ExtendedProperties` field, and pipe (`|`) the results to the `Format-Table` command with the `-Wrap` option so that the data is not truncated.

In the results above, we find that, in this single alert, three likely malicious IP addresses (116.110.3.162, 106.14.22.73, and 98.126.231.10) were communicating with `martha`.

MITRE ATT&CK Technique T1110.001: Password Guessing

- Broad technique used by many threat groups to access:
  - Victim cloud environment
  - Exposed operating system management connection
  - Attacker-facing application/service
- Sources in Azure for detection are vast, including:
  - Virtual Machine system logs (e.g., auth.log, Windows Security Events)
  - Cloud Management Plane logs
  - Azure AD Sign-in Logs
- Microsoft Defender for Cloud analyzes these logs for authentication attacks on a regular basis

MITRE | ATT&CK®

The Password Guessing (T1110.001) MITRE ATT&CK technique[1] should seem familiar as it was highlighted a few times throughout the course but let us dive deeper into it. Password guessing is a broad technique in that it could mean that the attacker is performing one or more different approaches to uncover a valid username and password pair. If you remember, the most common approaches are brute force, dictionary, password spray, and credential stuffing attacks.

Once the credential is stolen, the attacker can use this as a foothold into the affected resource—whether it is a cloud account or virtual machine. The seriousness of this breach will greatly vary. For instance, if a read-only Azure account is compromised, while not ideal, is less impactful as a Global Administrator-level account.

No matter which approach the attacker chooses, we must establish logging to capture these authentication attempts. This means capturing virtual machine system and application logs which identify login attempts, cloud management plane logs, and, in the case of Azure, Azure Active Directory (AD) Sign-in Logs.

Once this data is collected and an adversary begins attempting to log into an Azure-based resource over and over again, Microsoft Defender for Cloud may use its logic to detect and alert upon this activity.

If using an Azure Log Analytics workspace to store this alert data, it will arrive in a `SecurityAlert` table. From here, as you will see soon, you can query this data using a KQL query to dig deeper into the alert.

[1] https://attack.mitre.org/techniques/T1110/001/

Identifying Password Guessing with KQL

```
1  SecurityAlert
2  | where Tactics == "CredentialAccess"
3  | extend MaliciousIP = split(split(tostring(Entities),"Address")[1],'"')[2]
4  | project TimeGenerated, DisplayName, MaliciousIP
5  | sort by TimeGenerated asc
```

TimeGenerated [UTC]	DisplayName	MaliciousIP
11/9/2021, 10:30:46.427 PM	SSH - Potential Brute Force	178.62.27.124
11/10/2021, 10:30:33.699 PM	SSH - Potential Brute Force	36.110.228.254
11/11/2021, 10:30:16.182 PM	SSH - Potential Brute Force	212.47.250.50
11/13/2021, 10:31:15.552 PM	SSH - Potential Brute Force	104.248.80.75
11/14/2021, 10:30:27.027 PM	Password spray attack against Azure AD application	107.3.2.240
11/15/2021, 10:30:23.103 PM	Password spray attack against Azure AD application	107.3.2.240

Let us take a look at a sample KQL query used to highlight who is attempting to access credentials in our Azure environment (hopefully) before they are successful and make matters much worse for us. As noted, the alert data in Azure Log Analytics workspaces is stored in the `SecurityAlert` table so that is how the query will begin.

As the alert data is populated, each alert is assigned to a MITRE ATT&CK technique by Microsoft. We can use this to our advantage. Since we are looking for Credential Access tactics, we can add a where clause to filter for `Tactics` column values equal to `"Credential Access"`.

Answering the "Who is attacking us?" question is a bit more difficult. Since the `Entities` column in the `SecurityAlert` table is not a standard value (i.e., it could contain any kind of unpredictable JSON data in any order), we have to get crafty by using a few `split()` methods to extract an IP address and assign the result to a new `MaliciousIP` column using extend.

Now, all that is left is to present the data using project and sort it in ascending order using the sort command. The columns chosen here are:

- `TimeGenerated`, to see when the alert was created;
- `DisplayName`, to see what Microsoft Defender for Cloud is concerned about; and
- `MaliciousIP`, to see who is attacking our credentials.

The results above show a variety of real-world public IP addresses that were a bit too interested in logging into a VM in the course author's Azure account ("SSH - Potential Brute Force") and a user account in the course author's Azure AD deployment ("Password spray attack against Azure AD application").

Microsoft Sentinel

- Microsoft's cloud-based SIEM solution:
  - Query logs in configured **Azure Log Analytics workspace**
  - **Hunt** for threats using pre-defined, complex search queries
  - Generate custom **analytics** to identify threats on your own
  - Create **notebooks** to visualize several query results at once
  - Automate response actions by creating **playbooks**
- Supports ingestion of event data using **Data connectors:**
  - Microsoft platforms
  - Directly supported third-party vendors
  - Syslog or REST-API connections also supported for custom ingestion

Moving onto Microsoft's SIEM offering, Microsoft Sentinel[1] provides an all-in-one platform to identify and analyze threats inside of (and even outside of) Azure. Although this capability is included here, Microsoft Sentinel is not just another place to perform KQL searches against your Azure Log Analytics workspace. You can also hunt using Microsoft-created search queries. And not just one at a time. You can launch hundreds of hunt queries at once and, within just a few minutes, identify many threats that otherwise may have gone unnoticed.

Use can also generate, reuse, and schedule analytics inside Microsoft Sentinel to identify potential malice or suspicious actions by sifting through your Azure data on your behalf. We will discuss this component in more detail on the next page.

Two "books" that are included in Microsoft Sentinel are notebooks and playbooks. Notebooks are used to visualize the results of one more regularly-conducted KQL queries. These are very similar to dashboards in many SIEM products. These notebooks can be authored from scratch, or you can use one of several notebook templates which you can modify to your liking. Playbooks, on the other hand, are created and configured to perform automated response actions if an alert is triggered in a Microsoft security product.

The first step, like any SIEM solution, is to somehow move data into the platform. This is performed in Microsoft Sentinel using Data connectors. These connector configurations vary in complexity: from simply referencing an existing Azure resource or service like an Azure Log Analytics workspace or the Microsoft Defender for Cloud deployment, to setting up a syslog of Representational State Transfer Application Programming Interface (REST-API) connection between Microsoft Sentinel and your third-party data source.

[1] https://docs.microsoft.com/en-us/azure/sentinel/overview

Microsoft Sentinel: Analytics

- Analytics allow you to identify suspicious activity amongst various tables in your Azure Log Analytics workspace
- Three types of analytic rules can be created by users:
  - **Scheduled rules** to generate an alert and/or incident
  - **Microsoft security rules** pull in alerts from other Microsoft products to generate an incident
  - **Near-Real-Time (NRT)** rules running once per minute
- Many "out-of-box" rule templates available to quickly generate your own custom analytics

Analytics in Microsoft Sentinel[1] can be thought of as detection rules to help identify attacker activity by providing the KQL syntax to query your Azure data sources. There are two ways to get started when creating an analytic: building the KQL query from a blank slate or leveraging rule templates provided by Microsoft. If choosing the latter, these rule templates are categorized by MITRE ATT&CK tactics (e.g., Initial Access, Credential Access, Persistence) and which data source is queried. When choosing a rule template, you can use it as is or make tweaks to the very complex KQL code to home in on the data you are interested in identifying.

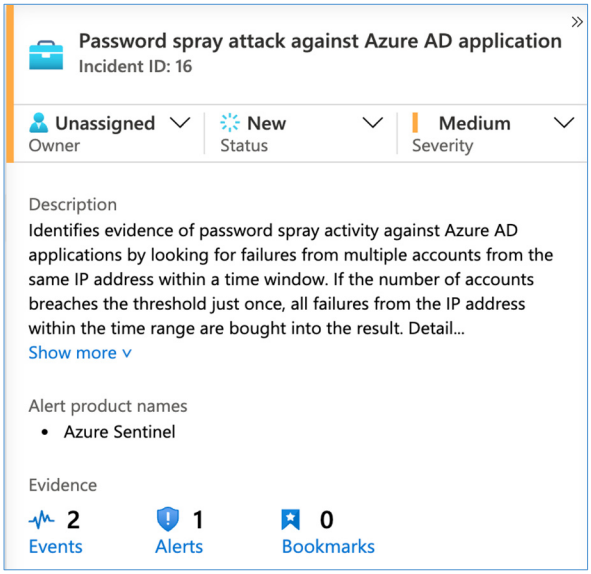
Sure, you could just run these same queries in your Azure Log Analytics workspace on a regular basis, but the real power here is to schedule these analytics to be assessed automatically. If you would like, you can set a scheduled rule to occur every day, month, hour, or however often you feel necessary to analyze your Azure data.

There are two other rule types that you can create as well: Microsoft security rules and Near-Real-Time (NRT) rules. With Microsoft security rules, you can ingest alert data from other Microsoft products—like the just-mentioned Microsoft Defender for Cloud—and use this data as an incident artifact. The other rule type, NRT, like scheduled rules, repeatedly performs the analysis, but does so once per minute (hence the "near-real-time" terminology).

[1] https://docs.microsoft.com/en-us/azure/sentinel/overview#analytics

Microsoft Sentinel: Incidents

- Aggregation of alerts and other relevant data for an investigation
- Generated by Microsoft Sentinel **analytics rules**
- Quickly drill into events, alerts, and entities involved
  - **Events** and **Alerts** links will pivot to Azure Log Analytics query and results
- Use **timeline** feature to view incident's sequence of events



As analytics are being created, they may generate an incident in Microsoft Sentinel. But what determines the criteria for an incident? That is up to you to set up when configuring your analytic. As part of the analytic creation workflow, Incident settings allow you to determine if the alert creates an incident at all and if so, if you should create an incident with each hit of the alert or if you should group the alerts as one major incident if they all occur within a specified timeframe. You can also roll-up multiple alerts into one incident if the alerts export the same entity information.

Once an incident is generated by Microsoft Sentinel, you will be presented with a lot of valuable information. You can drill into the information by clicking the many links presented like the Events and Alerts links you see above. Upon clicking either of those options, you will be forwarded to the appropriate Azure Log Analytics workspace for the Microsoft Sentinel deployment, have a custom query populated and executed, and the results presented—all automatically. This will allow you to quickly grab context needed to analyze the incident.

Keeping track of the order of operations during an attack campaign—especially a very complex campaign—can be troublesome and prone to error. The timeline feature in Microsoft Sentinel's reported incident can help lay out this order of alerts discovered as part of the incident.[1]

[1] https://docs.microsoft.com/en-us/azure/sentinel/investigate-cases

MITRE ATT&CK Technique T1110.003: Password Spraying

- Another authentication attack to access similar resources as T1100.001, but with a twist
  - Brute force/wordlist attacks use a password list against a few accounts, whereas a password spray uses a user list and a single password
- Attacker's approach:
  - Collect list of potential user account names
  - Choose a candidate password to try against all user accounts
  - If successful, they're in!
  - If not successful, try another password against the list of user accounts
- We can build a custom analytic in Microsoft Sentinel to identify a potential password spray

MITRE | ATT&CK®

The next technique we will discuss from MITRE ATT&CK is very similar to the last one in that the goal is to acquire credentials to access a victim system or environment, but Password Spraying (T1110.003) uses a slightly different approach.

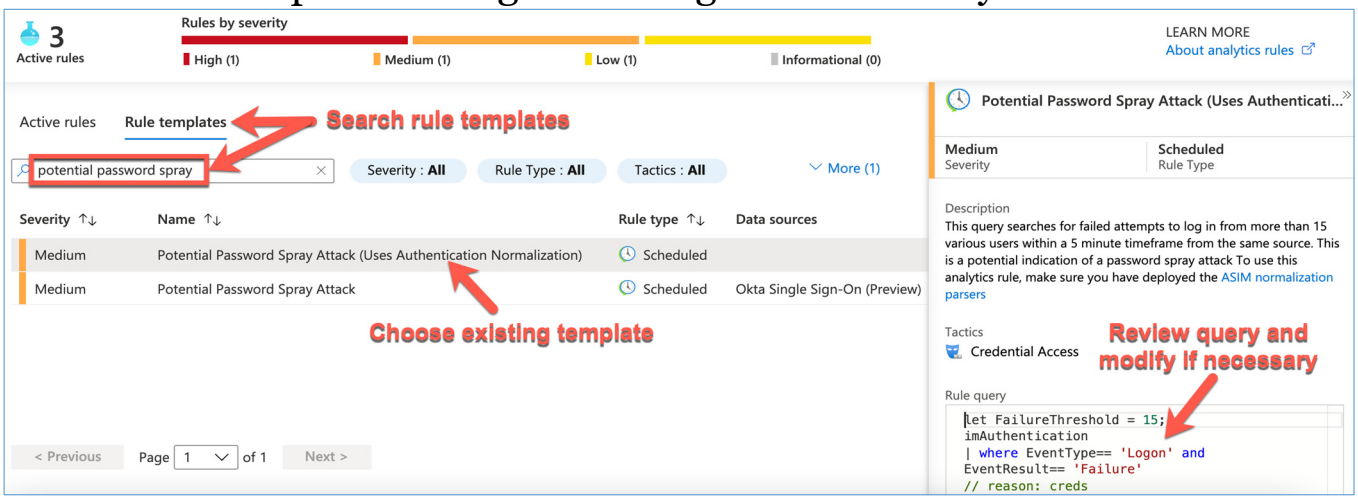
Unlike most authentication attacks where a single or relatively few account(s) are targeted using a large list of password candidates, a password spray attack flips that logic. In this case, the malicious actor will utilize a large list of usernames and a single password. If successful with this first pass, they have successfully entered the victim environment or system. If they are not successful, they may wait a day or so and try the next password.

Why wait to continue the attack? Because, unlike other authentication attacks where the attacker may lock out the targeted account(s), a password spray attempts to avoid this by only failing one or two times between successful connections by the actual person that the account belongs to. Waiting one business day will allow the user to connect as they usually would—resetting the "failed password" counter back to zero.

Identifying the previously-discussed authentication attacks prove trivial in many security products. Password spray attack detections are less prominent in security platforms. Luckily, Microsoft Sentinel contains some analytic templates to help identify T1110.003 in your Azure account.

Building a KQL Query to Detect Password Spraying in an Azure Account

Use rule templates to begin building a custom analytic



To begin identifying password spray attacks in your Azure account, you can use Microsoft Sentinel's built-in rule templates to generate your own custom analytic. This analytic can run at a user-defined interval to automate this analysis.

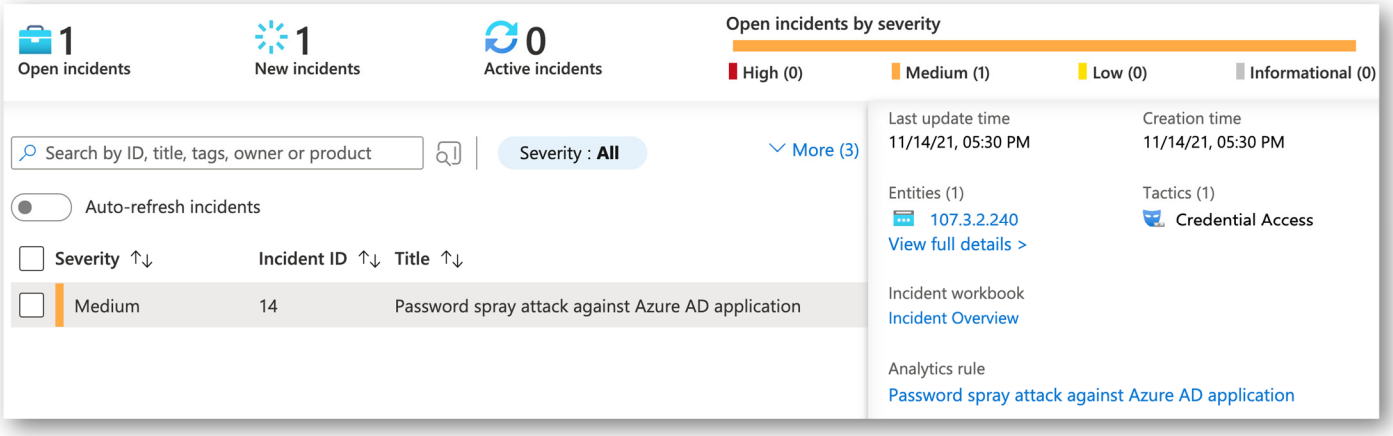
To begin setting this up, you will need to navigate to the Microsoft Sentinel service and click on your already configured resource. In the left-hand navigation pane, you will find an option for Analytics underneath Configuration. Once you click on the Analytics option, you will be shown a list of Active rules. Next to the Active rules tab, you will see a tab for Rule templates. Once you click on that tab, you will see the screenshot above.

Now, to generate the custom analytic using a pre-existing template. There is a search field just under the Active rules and Rule templates tabs where you can input free-form text. Since we are looking for a template for a potential password spray, we enter the text "potential password spray" and the templates listed below will change to only those that match the text we input in the search field. In fact, there are only two. But which one do we want? If you look at the Data sources, one is blank, but one identifies Okta Single Sign-On as the source of the data for that rule. If we do not have Okta login records in our environment, this rule will not provide us with value. That is the case here since Okta is not being leveraged as an identity provider. With this, we may want to try to use the top rule called "Potential Password Spray Attack (Uses Authentication Normalization" to see if we can identify any password spray attacks in our environment.

After clicking on the top result to our search query as it is possibly relevant to our data, we see what the very complex KQL query looks like, and other metadata about the query, in the right pane that appears. If this is the rule you would like to use, click Create at the bottom of the new pane.

It is at this point that you can modify the query and behavior as you see fit. In this sample query, for instance, to be considered a password spray, 15 failed attempts must be present. If your suspected attacker may be targeting a smaller environment or conducting this attack at a slower rate, you may want to modify the first line from `let FailureThreshold = 15;` to `let FailureThreshold = 10;`. Other options that you can change here are the severity, the MITRE ATT&CK Tactic, description, schedule, alert threshold, and incident settings.

Password Spray Incident Caught by Custom Analytic



Once you save the analytic and it runs, you may begin to see alerts and/or incidents. If so, you will see exactly which analytic was responsible for identifying this potential malice as this is the title of the incident.

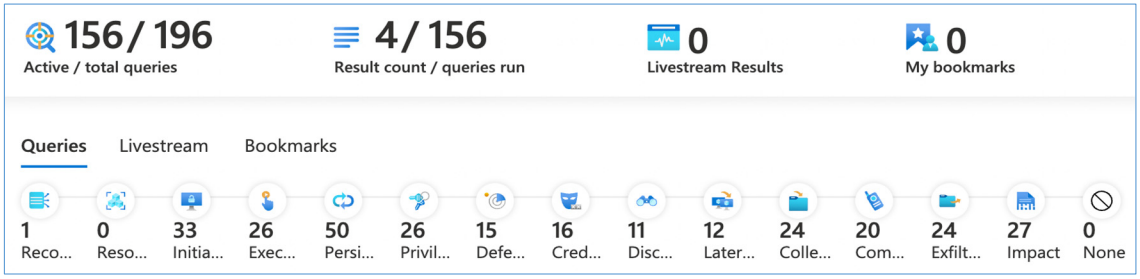
If you were to click on the title, you can see information related to the suspected attack. Information like the creation and last updated times of the incident, the entities involved (which is where you may find the adversary's information), the MITRE ATT&CK tactic set in the analytic configuration, a link to the analytics rule itself, and much more as we discussed previously.

In this example, we find that 107.3.2.240 was the IP address involved in the potential password spray attack that was identified on November 14, 2021, at 5:30 PM.

After reviewing the incident or alert, you may find that this is a false positive. If that is the case, you may need to edit the query used in the analytic configuration to be more granular and reduce the false positive rate.

Microsoft Sentinel: Hunting

- Perform queries across data sets to hunt for suspicious activity:
  - Queries categorized by **MITRE ATT&CK Enterprise Tactic**
  - All queries can be launched at once or individually
  - Searched data can come from Azure as well as outside of Azure
- **Data Connectors** allow ingestion of data from various sources



Up to this point in class, hunting has been a very manual process. While you can still hunt to your heart's content, Microsoft Sentinel provides an automated approach to hunt for common adversarial activities across your various data sets. This is done by leveraging Microsoft-created hunt queries that can be launched individually or all at once. In the example you see above, there are 156 queries that are active in this Microsoft Sentinel resource. By simply clicking Run all queries, the 156 queries will execute, and high-level results will appear as a Result count.

You can see exactly which queries contain results by reviewing and sorting the list of queries in the bottom pane. By clicking on the Results column, you will see which queries had results in order of result count. Here is an example:

Query ↑↓	Provider ↑↓	Data source ↑↓	Results
Signin Logs with expanded Conditional Access Policies	Microsoft	SigninLogs	30
Anomalous Azure Active Directory apps based on authenticati...	Microsoft	SigninLogs	22
Anomalous Sign-in Activity	Microsoft	SigninLogs +1 ⓘ	12
Anomalous sign-in location by user account and authenticatin...	Microsoft	SigninLogs	6

The results shown here identified some anomalous behavior (more on this soon) as well as one related to Signin Logs with expanded Conditional Access Policies.

Hunting, just like the other components in Microsoft Sentinel, requires the data to be made available. That means that data connectors must be ingesting the appropriate data that the hunt queries are sifting through. This data can be from a Microsoft tool or, as discussed earlier, can come from an external source like Amazon Web Services (AWS) or other third-party tools.

Microsoft Sentinel: Threat Intelligence

- Customers can generate their own threat intelligence **indicators** to identify future suspicious activity:
  - **domain-name**
  - **file**
  - **ipv4-addr**
  - **ipv6-addr**
  - **url**
- Configuration includes:
  - Providing the string/IP for the indicator
  - Selecting the **threat type** the indicator poses
  - **Confidence** (0-100) that the indicator is malicious
  - Valid dates of the indicator

We learned earlier that Microsoft provides their own threat intelligence to customers to identify known threat actors and their artifacts, but you, as an Azure customer, have the option to generate and utilize your own threat intelligence. This is part of the Microsoft Sentinel service. It has its own dedicated configuration that can be found by navigating to the Microsoft Sentinel resource and finding the Threat intelligence link under the Threat management heading.

The supported indicators you can create in Microsoft Sentinel are:[1]

- **domain-name**: A known domain name used during attack campaigns. An example could be a stage 2 malware download URL.
- **file**: A hash of a suspicious or malicious file used during an attack. This would be a great indicator to see if systems or cloud resources other than the presumed infected system also contain this file.
- **ipv4-addr**: An IP version 4 address. This indicator can be used to find other communication to an attacker discovered during an investigation. Perhaps the attacker was communicating elsewhere in the Azure ecosystem.
- **ipv6-addr**: This indicator is identical to ipv4-addr, but, of course, IP version 6.
- **url**: A Uniform Resource Locator (URL). This could be a known malware download site, data exfiltration location, or other internet-based system adversaries use to communicate with victim systems.

These indicators, other than the obvious values you must provide, do give the option to add more context in the event that the indicator presents itself in your future log data. For instance, you can identify the threat type. Options for this item include, for example, anomalous-activity, benign, compromised, and malicious-activity, to name a few. Furthermore, you can attribute a confidence level (from 0 to 100), valid dates, name, and description of the indicator.

If there are any results to these intelligence indicators, the Azure Log Analytics table called `ThreatIntelligenceIndicator` will contain data related to these discovered indicators.

[1] https://docs.microsoft.com/en-us/azure/sentinel/work-with-threat-indicators

User and Entity Behavior Analytics (UEBA)

- Not all threats can be identified using atomic indicators:
  - **Behaviors** of users or cloud resources can be monitored to identify suspicious actions
  - This requires comparing each action against a **normal action**
- UEBA technologies automate suspicious activity identification using **machine learning** techniques
- Examples include:
  - Users that access Azure from a new geolocation
  - Compromised virtual machine beaconing to public network
  - Abnormally large data transfer from cloud storage

Static detections like those threat intelligence indicators are great and should be leveraged, but what if you are trying to discover abnormal behaviors in your Azure environment? Those intelligence indicators will likely not pick up on abnormal actions unless the abnormal action also involves a known malicious domain name, IP address, file, or URL.

Luckily, many security vendors (Microsoft included) have another trick up their sleeves: User and Entity Behavior Analytics (UEBA).[1] With UEBA technologies, normal behavior is learned, likely using machine learning, over a period of time. After "normal" is established as a baseline, deviations from this baseline can be scored. Low scores typically mean that the behavior is pretty close to the baseline and higher scores indicate very different behaviors. Typically, there is a score threshold that, when passed, will generate an alert in the platform along with evidence of why it is believed to be a suspicious series of activities.

Some examples of abnormal behavior may include users accessing the protect platform from an unfamiliar location. Unfamiliar could mean that the alert stems from a login activity from a geolocated IP address belonging to a country never seen before for that particular user, or even any users in the environment.

Another example could be keeping tabs on the network traffic in an effort to discover network beacons. Network beacons are regular communication between an infected system and an attacker-controlled device.

A third, and final example (for now) that can prove difficult to analyze manually without these UEBA tools is when there is an abnormally large data transfer from cloud storage. If you remember, it was quite the process to identify this activity, but once the UEBA tools learn what normal ingress and egress cloud storage traffic looks like, it may be easy for the tool to spot anomalous data transfers that could indicate data exfiltration or data staging.

[1] https://cybersecurity.att.com/blogs/security-essentials/user-entity-and-behavior-analytics-explained

Microsoft Sentinel UEBA

- Not on by default when setting up Microsoft Sentinel:
  - UEBA must be configured via **Settings**
    - Select Azure **data sources** to interrogate (e.g., Azure Activity, Signin Logs)
    - Can also choose to analyze security events from Windows VMs
  - Once UEBA is enabled, **Anomalies** must also be turned on in **Settings** to leverage ML-training data to spot abnormal activities
- New **anomaly rules** will be enabled in Analytics

Informational	(Preview) Anomalous Account Creation	Anomaly	Enabled
Informational	(Preview) Anomalous Azure AD sign-in ses...	Anomaly	Enabled
Informational	(Preview) Anomalous Code Execution	Anomaly	Enabled
Informational	(Preview) Anomalous local account creation	Anomaly	Enabled

And here is Microsoft Sentinel, yet again, showing that it has some unique features to help identify suspicious activity. This time, UEBA can be leveraged to indicate anomalous activity based on behaviors identified in the connected data sources.[1]

Just like all other components, this feature is not just "on by default" and requires some configuration before you can take advantage of it. First, you must enable UEBA in your Microsoft Sentinel resources Settings. Enabling requires more than just flipping a switch from off to on. You must also select, from a list of the connected data source, which of the sources Microsoft Sentinel will consider and analyze when performing UEBA analysis.

Continuing in the configuration steps, you must also turn on Anomalies if you want to take full advantage of new, anomaly-based analytics in Microsoft Sentinel. Once you do this, you will find new, enabled rules appear when navigating back to the Configuration → Analytics section of the Microsoft Sentinel resource.

And now, it is just a matter of giving Microsoft Sentinel some time to learn the environment. It is mentioned by Microsoft that it takes, on average, 30 days to learn normal in an Azure environment, but once this normal is learned, those anomaly rules will begin to trigger any monitored actions that deviate too far from normal.

[1] https://docs.microsoft.com/en-us/azure/sentinel/identify-threats-with-entity-behavior-analytics

MITRE ATT&CK Technique T1078.004: Cloud Accounts

- Stolen Azure AD credentials can prove detrimental:
  - Whatever the user has privileges to, so does the attacker
  - If Global Administrator account is lost, so is the entire account!
- Stolen credentials can be tough to identify, but here are some useful indicators:
  - Logins from new locations/countries
  - Logins during strange times
  - Unusual/abnormal activity by logged in user
- Microsoft Sentinel can track this behavior and populate the **`BehaviorAnalytics`** Azure Log Analytics table with its findings

MITRE | ATT&CK®

We covered two specific MITRE ATT&CK techniques related to authentication attacks: T1110.001 (Password Guessing) and T1110.003 (Password Spraying), but what if the compromise is not captured by our detections? Perhaps Microsoft Sentinel's UEBA can come in handy.

Before we test our theory, let us dig into MITRE ATT&CK technique T1078.004 (Cloud Accounts).[1] This technique implies that the attacker is using a stolen cloud credential to access the management plane of our cloud environment. This access can be from any of the same tools we legitimate cloud users would use and more. Tools like the web user interfaces, command-line utilities from the cloud providers, software development kits, and even attack tools that specialize in ripping apart a cloud environment.

The activity that the malicious actor is performing, unless blocked by identity and access management permissions, will appear benign to the untrained eye. UEBA technologies, however, may pick up that the attacker's behavior is a bit strange if accessing from a never-before-seen IP address, during unusual timeframes, or if the action they are performing is abnormal.

If using Microsoft Sentinel and its UEBA capabilities happens to catch these attacker actions, an entry will appear in your Microsoft Sentinel-backed Azure Log Analytics workspace table, `BehaviorAnalytics`. It is here that you can learn more about why Microsoft Sentinel believed something was a bit odd about these actions.

[1] https://attack.mitre.org/techniques/T1078/004/

Identifying T1078.004 With Microsoft Sentinel UEBA

```
1  BehaviorAnalytics
2  | where ActivityInsights.CountryUncommonlyConnectedFromAmongPeers == True
3  | project TimeGenerated, ActivityType, UserName, SourceIPAddress,
4    ActivityInsights
```

TimeGenerated [UTC]	ActivityType	UserName	SourceIPAddress	ActivityInsights
ActivityInsights	{"FirstTimeUserConnectedFromCountry":"False","FirstTimeUserConnectedViaBrowser":"False","Fir			

AppUncommonlyUsedAmongPeers	True
BrowserUncommonlyUsedAmongPeers	True
CountryUncommonlyConnectedFromAmongPeers	True
FirstTimeUserAccessedResource	False
FirstTimeUserConnectedFromCountry	False
FirstTimeUserConnectedViaBrowser	False

Once the data arrives in your Azure Log Analytics workspace, you can begin to query the `BehaviorAnalytics` table. In the above example, we are searching this table for specific behaviors—users accessing Azure from an uncommon country. Uncommon in this case means that this user, or any user for that matter, has never logged in from the identified country before, until now.

To look for this particular behavior, the **`where`** clause is used to look at the `ActivityInsights` column's subfield of `CountryUncommonConnectedFromAmongPeers` to check if the value is set to `True`. If it is, tell us more. This is where the project is leveraged to display the following fields:

- `TimeGenerated`: You should know this by now as you have seen this column many times, but in case you forgot, this column will display the time that the behavior was identified.
- `ActivityType`: The high-level activity that occurred in the environment. It is not present above, but the type identified in that column is `LogOn`.
- `UserName`: This column represents the Azure Active Directory (AD) user who performed the action. This is potentially the compromised user.
- `ActivityInsights`: This data is populated by Microsoft Sentinel UEBA to give more context to the behavior and why it may have been identified as anomalous.

Course Roadmap

- Section 1: Management Plane and Network Logging
- Section 2: Compute and Cloud Services Logging
- Section 3: Cloud Service and Data Discovery
- **Section 4: Microsoft Ecosystem**
- Section 5: Automated Response Actions and CloudWars

Microsoft Ecosystem

1. Malwarebytes Attack
2. Microsoft 365
3. EXERCISE: Microsoft 365 Exchange Investigation
4. SolarWinds Attack
5. Azure Active Directory (AD)
6. EXERCISE: Introduction to KQL
7. Storage Monitoring
8. EXERCISE: Log Analytics Using Azure CLI
9. Detection Services
10. **EXERCISE: Microsoft Defender for Cloud and Sentinel**
11. Network Traffic Analysis
12. EXERCISE: Azure Network Traffic Analysis

This page intentionally left blank.

Lab 4.4 | Microsoft Defender for Cloud and Sentinel

Exercise Duration: 30 Minutes

Objectives

In this lab, we will investigate two detection services: Microsoft Defender for Cloud and Microsoft Sentinel by:

- Viewing alerts in Microsoft Defender for Cloud
- Using Azure CLI to discover alerts related to previously discovered attack
- Utilizing Microsoft Sentinel to hunt for authentication attacks against this Azure account
- Manually running the password spray against Azure AD application analytic rule

Now it is time to shift our focus onto the two major Azure security services we have just covered: Microsoft Defender for Cloud and Microsoft Sentinel. You will use a combination of Azure Portal navigation and CLI tools to view existing alerts, hunt for malicious activity, and craft your own analytic rules to uncover suspicious activity in the hosted Azure environment.

On top of analyzing artifacts from the author-created attack like you explored in exercises 4.2 and 4.3, this lab will also contain live data from real-world attackers targeting this account and its resources.

Course Roadmap

- Section 1: Management Plane and Network Logging
- Section 2: Compute and Cloud Services Logging
- Section 3: Cloud Service and Data Discovery
- **Section 4: Microsoft Ecosystem**
- Section 5: Automated Response Actions and CloudWars

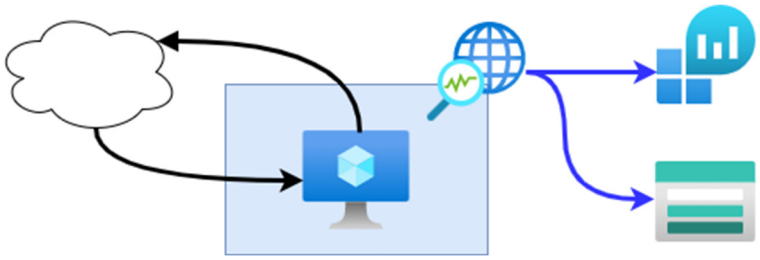
Microsoft Ecosystem

1. Malwarebytes Attack
2. Microsoft 365
3. EXERCISE: Microsoft 365 Exchange Investigation
4. SolarWinds Attack
5. Azure Active Directory (AD)
6. EXERCISE: Introduction to KQL
7. Storage Monitoring
8. EXERCISE: Log Analytics Using Azure CLI
9. Detection Services
10. EXERCISE: Microsoft Defender for Cloud and Sentinel
11. **Network Traffic Analysis**
12. EXERCISE: Azure Network Traffic Analysis

This page intentionally left blank.

Where We Left Off

- Azure provides methods to capture network data for analysis:
  - Network Security Group (NSG) Flow Logs
  - Variable Packet Capture
- Without network data, you may be missing part of the story:
  - Are we being scanned?
  - Are there any abnormal connections in our virtual network (VNet)?
  - Did the malware call out to any other internal or external hosts?



The first book of this class covered, albeit at a high level, two options for capturing network data in an Azure environment: Network Security Group (NSG) flow logs and NSG variable packet capture. The former is the offering which provides you with network metadata (data about the network traffic) and the latter captures the raw network data. As we discuss in this module, you will find both to be necessary in certain scenarios and situations. No matter which is chosen, it is important to note the significance of these options so that you can make the determination of which is the best tool for the job.

Without some visibility of network traffic, it will be extremely difficult to recognize several attacks in your cloud environment. For example, let us imagine it is very early in an attack campaign and the attacker performs what most attackers would—an active network scan. If you were to capture this network data at the host itself through endpoint applications, you may stop the scan in its tracks.

But what if the all of the network scan traffic does not reach the intended target due to network security controls? The endpoint solution will not see the traffic and have a false sense that no scan is occurring—revealing to the attacker the approved and available ports that do reach the target. If you were, however, to utilize the aforementioned tools, you would regain this visibility and be able to recognize and react to an adversary's scan attempts.

And it is not just inbound traffic you must be concerned about. Yes, early in the attack campaign the attacker may be attempting to find a way in, but later in the attack—when the pivoting, malicious tool downloads, and data exfiltration is occurring—you will be also be interested in outbound traffic.

It also must be said (and you probably know what is about to be said) that knowing normal will be absolutely crucial to be able to determine the abnormal, and possibly malicious, traffic. Flow data and packet captures prove invaluable to learn the normal happenings in your cloud network.

As you may remember from the first book, Azure's Network Watcher service provides the option to capture Network Security Group (NSG) flow data. This data can be output to two different locations: an Azure Storage container and/or an Azure Log Analytics workspace.

In the above image, you will see the flow data stored in an Azure Storage container called insights-logs-networksecuritygroupflowevent. The name of that container was not created by the course author to make it apparent just what kind of data is inside the container, but automatically created and named as soon as the NSG flow configuration is set. In fact, this name is not customizable.

The directory structure of the insights-logs-networksecuritygroupflowevent Azure Storage container is quite unusual. The structure begins with the folder resourceid= and has many subfolders until you arrive at the actual flow data: The SUBSCRIPTIONS, subscription ID of the Azure account, RESOURCEGROUPS, resource group name which contains the NSG, PROVIDERS, MICROSOFT.NETWORK, NETWORKSECURITYGROUPS, the name of the NSG, a breakdown of the time of the flow record (year, month, day, hour, and minute), and then the MAC address of the cloud resource creating the flow data.

Finally, you will arrive at a JSON-formatted file containing one or more flow records for the given time. You *can* analyze the data by clicking on the file and the Edit option that will appear in the new pane but reading raw JSON data in a web browser is neither efficient nor effective. See below what that data looks like in the Azure Portal:


```
{"records":[{"time":"2021-11-12T07:00:13.4695298Z",
"systemId":"9acd1e19-1781-4c23-be09-9101793c9c4e","macAddress":"000D3A8EB182",
"category":"NetworkSecurityGroupFlowEvent","resourceId":"/SUBSCRIPTIONS/
138D1821-EFB5-4CDC-80FA-7E1221ABCF2C/RESOURCEGROUPS/SHERLOCK/PROVIDERS/MICROSOFT.
NETWORK/NETWORKSECURITYGROUPS/VM-NSG",
"operationName":"NetworkSecurityGroupFlowEvents","properties":{"Version":2,"flows":
[{"rule":"DefaultRule_AllowInternetOutBound","flows":[{"mac":"000D3A8EB182",
"flowTuples":["1636700358,10.0.0.4,20.42.73.133,51676,443,T,O,A,B,,,,","1636700364,
10.0.0.4,20.42.73.133,51676,443,T,O,A.E,12.5055,11,5079","1636700378,10.0.0.4,20.
```


With that, the data should be moved to a system or application which can be used to parse the data more adequately.

Analyzing JSON-Formatted NSG Flow Logs

- Step 1: Download all JSON files

```
az storage blob download-batch -d . -s insights-logs-
networksecuritygroupflowevent --pattern '*.json' --
account-name $storageAccount
```

- Step 2: Use **find** command to identify all JSON file locations

```
find . -name *.json
```

- Step 3: Extend **find** command with **jq**

```
find . -name '*.json' -exec jq -r
'.records[].properties.flows[].flows[].flowTuples[]'
{} \;
```


In each of those directories in the insights-logs-networksecuritygroupflowevent Azure Storage container, created every 10 minutes or every hour—whichever is configured in the NSG flow settings—you could download each of those P1TH.json files. This is, however, slow and cumbersome. Using the Azure CLI tools will certainly be more effective as you can download all of the flow logs to a remote system with just one command:

```
az storage blob download-batch -d . -s \
  insights-logs-networksecuritygroupflowevent --pattern '*.json' \
  --account-name $storageAccount
```

The command above uses the Azure CLI tool to download a batch of files which end in .json from the insights-logs-networksecuritygroupflowevent Azure storage container and save it to the current working directory.

From here, you can locate all of the flow event files by running the first `find` command that you see above (`find . -name *.json`). This second `find` command is very useful:

```
find . -name '*.json' -exec \
  jq -r '.records[].properties.flows[].flows[].flowTuples[]' {} \;
```

This command does two things: finds all of the .json files and, for each of the results, extracts the flowTuples (the flow records)—presenting them to the screen.

As you now have access to all of the flow data, you can further process and analyze this data as you see fit to track down a current incident or perform threat hunting by comparing the results to the normal traffic observed in the Azure virtual network.

Flow Event Format

- Results can be cryptic

```
1633593505,10.0.0.4,40.79.154.87,33340,443,T,O,A,E,12,5592,10,5025
1633593520,10.0.0.4,40.79.154.87,33374,443,T,O,A,B,,,,
1633593525,10.0.0.4,40.79.154.87,33374,443,T,O,A,E,12,4985,11,5079
1633593540,10.0.0.4,40.79.154.87,33394,443,T,O,A,B,,,,
1633593545,10.0.0.4,40.79.154.87,33394,443,T,O,A,E,12,4996,10,5025
```

- Still need to make sense of the data

  `epochTime,sourceIP,destinationIP,sourcePort,destinationPort,protocol,direction,decision,state,inPackets,inBytes,outPackets,outBytes`

- **`cut`** can come in handy

  Example: **`... | cut -d ',' -f2`** (Source IP)

When reviewing the JSON data's flowTuples entries, the results can be difficult at first to understand. That is, until you have an understanding of the comma-delimited results and what the data in each position represents. The structure is as follows:[1]

Position	Field	Description
1	epochTime	UNIX-formatted time of the flow record
2	sourceIP	Initiator's IP address
3	destinationIP	Recipient's IP address
4	sourcePort	Initiator's source port
5	destinationPort	Recipient's listening port
6	protocol	Layer 4 protocol used (T for TCP and U for UDP)
7	direction	Inbound (I) or Outbound (O) direction
8	decision	Whether the traffic was allowed (A) or denied (D)
9	state	State of the flow if NSG flow version 2 (B = Beginning of flow, C = Continuation of previous flow, E = End of flow)
10	inPackets	How many packets were sent by the initiator
11	inBytes	Number of bytes sent by initiator
12	outPackets	How many packets were returned by the recipient
13	outBytes	Number of bytes returned by the recipient

[1] https://docs.microsoft.com/en-us/azure/network-watcher/network-watcher-nsg-flow-logging-overview#log-format

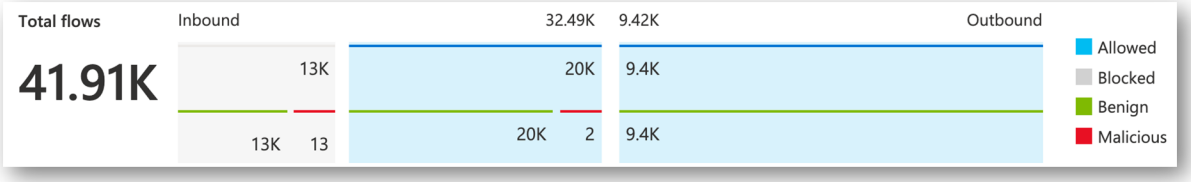
Since each of the values are in a predictable location, some Linux or Windows PowerShell commands can be used to parse or extract the needed data. For example, if you are interested in all the destination IP addresses communicated with, you can extend the second find command from the previous discussion with the pipe character (`|`) and the `cut` command like so:

```
find . -name '*.json' -exec \
  jq -r '.records[].properties.flows[].flows[].flowTuples[]' {} \;
  | cut -d ',' -f3
```

The `cut` command in this example is using the comma character (`,`) as a delimiter (`-d ','`) and then, using the `-f` flag, selects the third entry (`-f3`).

Network Metadata in Azure Log Analytics

- Flow data can also be processed with **Traffic Analytics:**
  - Select Azure Log Analytics workspace to send to
  - Select interval (i.e., hourly or every ten minutes)
- Enriches existing flow data



- Processed data arrives in two different tables:
  - **`AzureNetworkAnalytics_CL`**
  - **`AzureNetworkAnalyticsIPDetails_CL`**

When configuring NSG flow logs, one very useful option is to enable Traffic Analytics. This instructs Azure to not only generate flow log data, but also provide metrics and enrich the flow data by providing information about the IP addresses that it sees (e.g., geolocation of the IP address). There is some configuration of this capability to consider such as where to send the data (i.e., Log Analytics workspace—which is required) and the interval in which the analysis will be performed. The acceptable intervals are rigid—either ten minutes or one hour.

Metric data can be visualized when navigating to the Network Watcher service's Traffic Analysis section. As you see above, the number of allowed, blocked, benign (non-malicious), and malicious connections are available in a simple visualization. If reviewing this data on a regular basis, an analyst can understand the normal traffic patterns of the Azure environment.

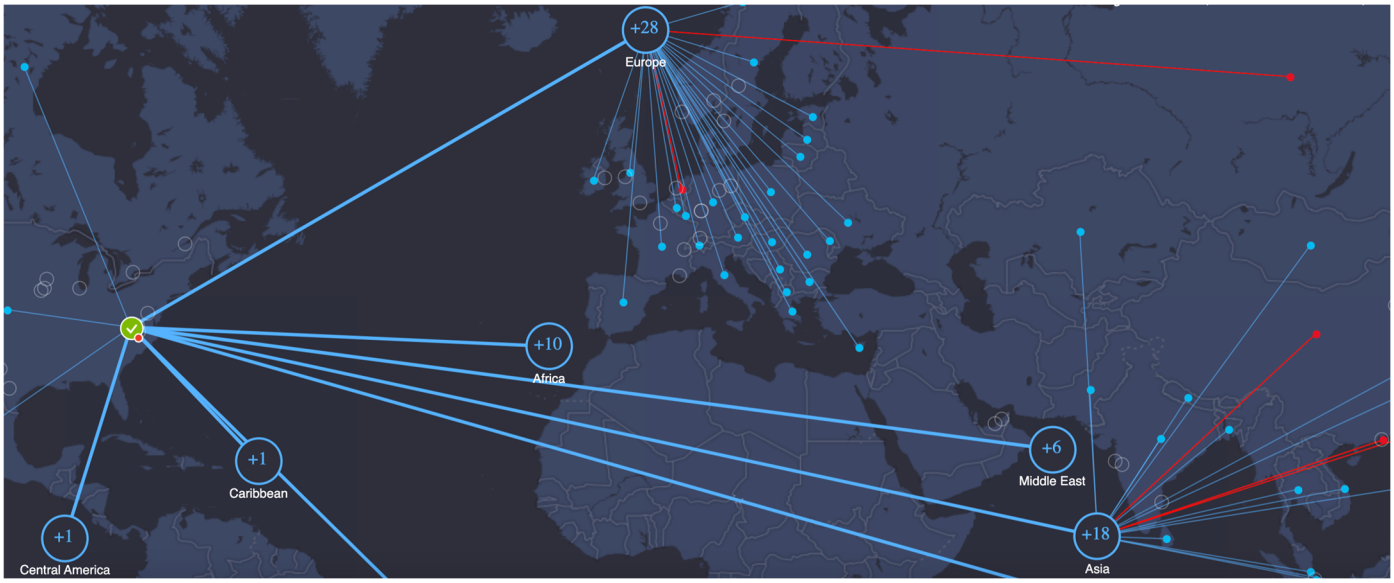
But how does Azure Network Watcher determine a malicious flow? According to Microsoft: "Traffic Analytics relies on Microsoft internal threat intelligence systems to deem an IP as malicious. These systems leverage diverse telemetry sources like Microsoft products and services, the Microsoft Digital Crimes Unit (DCU), the Microsoft Security Response Center (MSRC), and external feeds and build a lot of intelligence on top of it. Some of this data is Microsoft Internal."[1, 2]

Since the Traffic Analysis option requires export to an Azure Log Analytics workspace, two tables are created as the data is processed:

- `AzureNetworkAnalytics_CL`: This table includes all flow-related data and can be searched more effectively using an Azure Log Analytics workspace search using column names instead of understanding the flowTuple structure.
- `AzureNetworkAnalyticsIPDetails_CL`: This table contains WHOIS lookup information about any public IP addresses that were discovered. If the public IP address is deemed malicious, threat types and descriptions of the threat are included.

[1] https://docs.microsoft.com/en-us/azure/network-watcher/traffic-analytics
[2] https://docs.microsoft.com/en-us/azure/network-watcher/traffic-analytics-faq

Traffic Analysis Geo Map View



Before moving on to Azure Log Analytics workspace searches, there is another valuable visualization provided by Azure in the Traffic Analysis section of the Azure Network Watcher service: the Traffic Analysis Geo Map View. When viewing this map, you can shift the view around by clicking and dragging and zooming in and out to display the map how you would like.

All the small, white circles you see are the Azure regions. The white circle filled with green and a check mark is the Azure region which is being monitored by Azure Network Watcher. If you click on the green circle, you will begin so see which parts of the world are either communicating with a system in this Azure region in an inbound direction or the cloud systems in that Azure region are communicating with in an outbound direction to those locations.

You can also expand the geolocations in communication with the Azure region's asset(s) by clicking on the location name. You see this above with Europe and Asia. Both of these geolocations show both benign traffic (narrow blue lines) and traffic deemed by Microsoft as malicious (red lines).

Again, this type of information can be useful to learn the traffic sent to and from your cloud-based systems but may be more useful for those that learn best or recognize deviations to normal behavior using this type of visualization.

Valuable AzureNetworkAnalytics_CL columns

Column Name	Description
DestPort_d	Requested TCP or UDP port
DestPublicIP_s / SrcPublicIP_s	Public IP address either initiating (SrcPublicIP_s) or receiving (DestPublicIP_s) the first network connection
SrcIP_s / DestIP_s	Azure VM address that is the source/destination of the flow
InboundPackets_d / OutboundPackets_d	Number of packets received coming from or going to the network security group
InboundBytes_d / OutboundBytes_d	Number of bytes received coming from or going to the network security group
FlowDirection_s	Initial direction of the connection
FlowStatus_s	Determines if the session was allowed or denied
L4Protocol_s	Transport layer protocol (i.e., TCP, UDP)
VM_s	Resource group and name of the participating VM

As the Azure Network Watcher-created data is arriving in an Azure Log Analytics workspace, we must know what we are dealing with so that we can successfully analyze this data. As an example, the data stored in the `AzureNetworkAnalytics_CL` table in Azure Log Analytics workspaces has many available fields.[1]

In certain respects, all columns can prove valuable, but we will focus on what will likely be the most valuable in the event of a breach or if you are performing additional analysis of a suspicious flow:

- `DestPort_d`: This column contains the recipient's listening TCP or UDP port.
- `DestPublicIP_s/SrcPublicIP_s`: Depending on if it is the requestor (`SrcPublicIP_s`) or a responder (`DestPublicIP_s`) of a flow contains a public IP address, this column will show that internet-routable address.
- `SrcIP_s/DestIP_s`: This column contains the source IP address (`SrcIP_s`) or destination IP address (`DestIP_s`) of the Azure virtual machine (VM) involved in the flow—depending on which side of the communication the VM is a part of.
- `InboundPackets_d/OutboundPackets_d`: These columns show the number of packets in the specified direction of the flow.
- InboundBytes_d/OutboundBytes_d: These columns shows the number of bytes in the specified direction of the flow.
- `FlowDirection_s`: If looking for an inbound flow (**`I`**) or outbound flow (**`O`**), this column will prove useful.
- `FlowStatus_s`: This column displays if the flow was allowed by the Azure NSG (**`A`**) or denied (**`D`**).
- `L4Protocol_s`: The transport layer protocol used in the flow is specified in this column. **`T`** specifies TCP and **`U`** stands for UDP.
- `VM_s`: This column includes both the resource group and VM name involved in the flow.

[1] https://docs.microsoft.com/en-us/azure/network-watcher/traffic-analytics-schema

MITRE ATT&CK Technique T1595.002: Vulnerability Scanning

- Either early in an attack campaign or during an attempt to pivot, attackers use network access to:
  - Find **listening ports and applications** to attack
  - Identify **misconfigurations** in network devices
  - Discover **flaws** in underlying applications
- Mitigations include locking down the network traffic to **approved hosts**, but some applications need internet exposure
- Detection techniques include:
  - Reviewing **flow data**
  - Inspecting the **traffic content**

Before we begin looking into ways to identify suspicious or malicious activity in our Azure Log Analytics workspace data, we must understand how these attacks are performed. Yet again, MITRE ATT&CK comes to the rescue as it includes many network-based techniques performed by known adversaries. You have already seen this technique, Vulnerability Scanning (T1595.002),[1] in the Capitol One attack from the third book. This was when the attacker was believed to have scanned the public-facing web application and found that it was vulnerable to a Server-Side Request Forgery (SSRF) which allowed the attacker to steal and use AWS credentials. In fact, we were performing this activity ourselves in an effort to discover vulnerabilities before the attacker did.

But what if the attacker is using this technique against us? How can we identify this activity? We will get to that in a moment. But first, what if vulnerabilities are present?

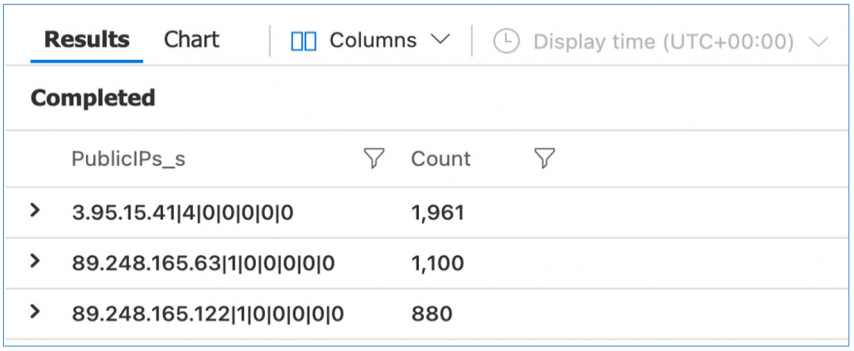
MITRE provides the recommendation to lock down the traffic further, which is a great idea. If the vulnerable service is not presented to or reachable by the attacker, they cannot exploit it. There is, however, one problem with this approach in some cases: what if the application requires public access? An example of this would be an Azure VM serving a company's web page. It may be unacceptable to restrict the network traffic, so we must focus on detection.

Now back to the two questions from earlier: what if the attacker is conducting a vulnerability scan and how could we spot this scan? Network-based detection methods can help. Azure Network Watcher is the go-to service in Azure to view the flow data and also to inspect the attacker's payload itself if the traffic is not encrypted.

[1] https://attack.mitre.org/techniques/T1595/002/

Identifying Network Scanning with KQL

```
1  AzureNetworkAnalytics_CL
2  | where FlowStartTime_t between(datetime("9/27/2021 00:00:00")..
3    datetime("9/27/2021 23:59:59")) and FlowDirection_s == "I"
4  | summarize Count = count() by PublicIPs_s
5  | sort by Count desc
6  | limit 3
```



If the Azure Network Watcher flow data is arriving at an Azure Log Analytics workspace, the `AzureNetworkAnalytics_CL` table will likely provide us with very telling information that could identify some of the more common network-based attacks against our Azure VMs. We will dissect this KQL query to show how, with just a few lines, how we can see the top three public IP addresses that are communicating with our VM in an inbound direction—indicating systems which may be scanning us:

```
AzureNetworkAnalytics_CL
| where FlowStartTime_t between(datetime("9/27/2021 00:00:00")..
  datetime("9/27/2021 23:59:59")) and FlowDirection_s == "I"
| summarize Count = count() by PublicIPs_s
| sort by Count desc
| limit 3
```

The first line, as you would expect, indicates the Azure Log Analytics table that houses the data we are interested in searching and retrieving. In this case, we are interested in the data inside the `AzureNetworkAnalytics_CL` table.

The second line (which wraps around to the third line) is the `where` clause. This clause contains two parts: the timeframe in which we are searching (September 27th, 2021 in this example) and the direction of the flow. With that, `FlowStartTime_t` and the `FlowDirection_s` columns are leveraged.

The next line, beginning with `| summarize` is taking the narrowed down data from the previous lines and counting the occurrences of each unique entry in the `PublicIP_s` field. This will be the basis of determining which public systems may be scanning our Azure virtual network's systems.

The last two lines (`| sort by Count desc` and `| limit 3`) sort the data by the new **`Count`** column created by the summarize instruction by the highest to lowest number and then display the first three entries.

In the above example, it appears that 3.95.15.41, 89.248.165.63, and 89.248.165.122 may be scanning this environment.

MITRE ATT&CK Technique T1571: Non-Standard Port

- Attackers may call out to **command and control (C2)** systems using **uncommon ports:**
  - HTTP over port 8888/tcp
  - SSH over port 2222/tcp
- Primary mitigation is to **restrict internet-bound traffic** to approved ports and protocols
- Detection can be performed by:
  - Reviewing **flow data**
  - Inspecting the **traffic content**
  - Reviewing logs with network connection establishment information (e.g., Zeek conn.log, Sysmon network connection events)

Attackers will not just conduct vulnerability scans over the network, so we will move on to another very popular technique reported by MITRE in their ATT&CK framework: Non-Standard Port (T1571).[1] Once an attacker exploits a system in a network, they may perform command and control (C2). This allows the attack to remotely control a system within the Azure virtual network. This means that the attacker can invoke system-level actions on the system and continue their campaign in many ways, such as:

- Pivot to other Azure VMs over the network
- Steal cloud credentials and perform read, write, or delete actions against cloud services
- Exfiltrate data from the affected system
- Destroy or disrupt the local VM files and services
- And pretty much any other MITRE ATT&CK technique you can imagine—all from within the victim's cloud environment!

This C2 traffic may utilize abnormal port numbers (to the affected network, that is) that can be observed by an analyst utilizing the same flow logging option we covered earlier. "But how?", you may ask. One method is to identify the normal ports in use by legitimate cloud systems and compare that traffic to what is observed in the flow data. If there is a new port and/or protocol, inspect the traffic metadata more deeply and question the system owner to determine if this traffic is expected.

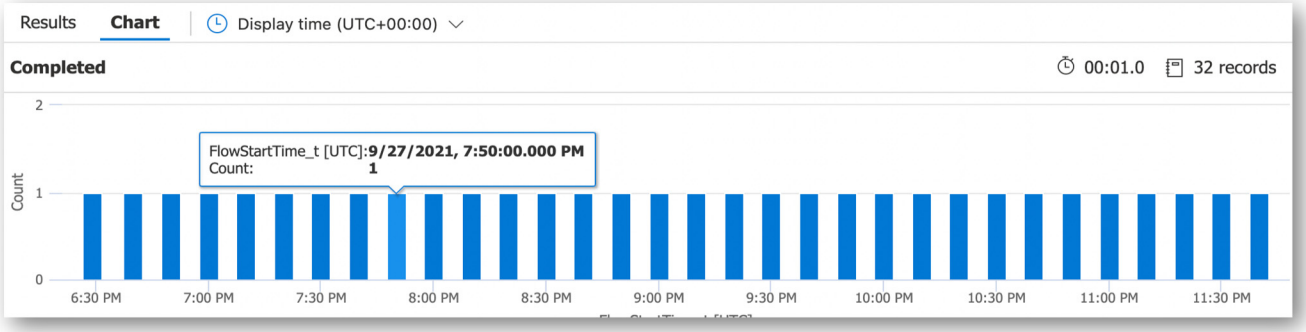
Looking more deeply may also include capturing network packets to inspect the actual payload of the suspicious traffic. As mentioned previously, this means that, to gain an understanding into this payload, the traffic must be in plaintext format (i.e., not encrypted). This capture, while not currently supported at scale in Azure, can be invoked by utilizing Azure Network Watcher to instruct the affected Azure VM to start a capture and store this capture either on the system or in Azure Storage for review.

Regarding mitigation of the risk that C2 could call outbound from your cloud environment, you can, as a cloud customer, restrict the Azure NSG to only allow approved outbound ports and protocols to known hosts. This is necessary as Azure's default NSG configuration allows unrestricted outbound traffic.

[1] https://attack.mitre.org/techniques/T1571/

Identifying Command and Control with KQL

```
1  AzureNetworkAnalytics_CL
2  | where TimeGenerated between(datetime("9/27/2021 00:00:00")
3    ..datetime("9/27/2021 23:59:59")) and PublicIPs_s contains "3.95.15.41" and
4    FlowDirection_s == "O" and DestPort_d == 443 and L4Protocol_s == "T"
5  | summarize Count = count() by bin(FlowStartTime_t, 1m)
6  | project Count, FlowStartTime_t
7  | render columnchart
```



To identify this C2 traffic, we can again consult the `AzureNetworkAnalytics_CL` Azure Log Analytics table. Since this table houses flow data, we can analyze the ports and protocols in use to spot potential C2. Spotting C2 can take many forms. One of which was discussed: identifying abnormal (or not-yet-seen) ports, protocols, or hosts. What if, in the case of the example query above, we already suspect a malicious IP address communicating with a victim in our Azure environment? We already saw this suspect IP address in a previous query, so let us hunt for more evidence. In fact, let's visualize the traffic while we are at it.

To break down the query, we will start with the first line which, as you expect, is the table name of `AzureNetworkAnalytics_CL`. From here, the second through fourth lines contain a lengthy `where` clause which applies a filter for the timeframe of this traffic occurring on September 27, 2021, but also includes a few more filters. The second filter will narrow down the search to all `PublicIP_s` column entries that contain the 3.95.15.41 address. The last three parts of the `where` clause look for outbound traffic to TCP port 443. Why that port and protocol? Just call it a hunch as that port and protocol is very common when an attacker is trying to blend in with normal, benign traffic.

The `summarize` command looks a bit different than the last time we saw an example of this. This line will take the narrowed-down results and count the number of occurrence of TCP port 443 to attacker IP traffic and "group them" into one-minute blocks of time. This way, if there were a pattern, we could visually see it in some form of chart—something we cannot easily do with lines of log data.

This visualization comes into play with the final two lines of the query. The `project` line does what you would expect: inform Azure Log Analytics that you want to only see the `Count` and `FlowStartTime_t` data. The `render` line changes how the data is presented. In this case, a `columnchart` is utilized. When creating a `columnchart`, it is important to know how to send the data to this command so that the x- and y-axes are correct. Here, you can see that the time is on the x-axis and `Count` is on the y-axis. This was no accident or good luck. That is why `project` and its order is so important. The y-axis for the `columnchart` will be the first field sent using `project` and the second field is the x-axis.

With all that, what are we looking at here? We are looking at what appears to be beaconing. Beaconing is identified by a regular "heartbeat" calling back to the attacking system—likely awaiting instructions.

MITRE ATT&CK Technique T1071: Application Layer Protocol

- As traffic over uncommon ports may stand out or be blocked, malware may use a **common port** and **common application**
  - This helps blend in with normal traffic
  - Example: Meterpreter can communicate using HTTP over 80/tcp
- Prevention options are limited
  - **Proxy** outbound web services with security devices
  - Leverage **host-based prevention** suites
- Detection requires payload inspection
  - Discovery in Azure will require packet capture of some sort (i.e., on the host itself or using **variable packet capture**)
  - If encrypted, detection is extremely difficult

The final technique we will discuss in this book is MITRE ATT&CK technique T1071 (Application Layer Protocol).[1] Unlike the T1571 (Non-Standard Port), attackers may use common ports, protocols, and even the matching application for that port/protocol pair to perform C2 of victim systems. A very popular attacker framework, Metasploit, allows an attacker to do just that.

Meterpreter is a component of Metasploit which is used for C2 of a victim system and has many options of how to perform this C2. For instance, and related to this technique, the chosen Meterpreter flavor could leverage a reverse HTTP listener over TCP port 80. This means that all outbound C2 traffic initiated from the victim system will use legitimate HTTP traffic over its common port to communicate with the attacker's system. This communication can include shell command instructions, retrieval of system information, exfiltration of data, and pretty much anything else the malware can leverage on that impacted system.

Prevention and mitigation efforts are much trickier here as these ports, protocols, and applications are likely used in the victim environment and filtering of this outbound traffic is extremely difficult without some help from some network-based security appliances. One such appliance could be an outbound web proxy which can inspect the web traffic and block any attempts to communicate with known malicious systems or block traffic that, itself, appears malicious.

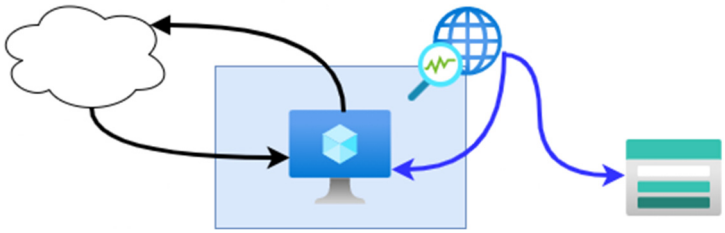
We may want to start paying closer attention to the endpoints themselves and enable them to block these malicious outbound attempts through the use of a host-based prevention solution like Antivirus, Antimalware, or other security application.

And now, detection. To detect this malicious traffic, we can no longer utilize flow data for more than just identifying already identified malicious actors. Instead, we must use a capability to review or inspect the traffic for malice. This is, as of the time of this writing, not possible to perform at scale in Azure using a vendor-supplied service like you saw from AWS in a previous book.

[1] https://attack.mitre.org/techniques/T1071/

Azure Network Watcher Variable Packet Capture

- Sole packet capture option in Azure at this time
- Triggered many ways
  - Portal
  - PowerShell cmdlet
  - Azure CLI
  - REST API



- Once triggered, will store full packet capture for the machine in:
  - Azure Storage
  - Locally on the system
- Analysts can then use tools to analyze captured data

The only Azure-provided solution to view network payload is the variable packet capture option in the Azure Network Watcher service.[1] This service is invoked by the customer manually using one of many Portal or command line options or can be triggered by customer-developed response automation and will perform a packet capture of the selected system which the packet capture file will be stored in either Azure Storage or on the Azure VM itself. Hardly convenient, but better than no option at all.

There is a caveat and prerequisite to consider before initiating the packet capture. The caveat is that the packet capture can only run for a limited time—up to five hours. Once the five hours have elapsed, another packet capture will need to be requested. As for the prerequisite, the Azure Network Watcher service must be able to effectively communicate with the Azure VM in which the packet capture will begin recording. This means that the `AzureNetworkWatcher` extension is already installed on the system.

Not only must the packet capture be requested to start, but it must also be requested to stop using one of the techniques mentioned above. So, if you are building an automation or even invoking the packet capture at will, remember to stop it. Otherwise, you will have quite the large packet capture containing five hours worth of data to sift through.

Once the packet capture is started, stopped, and saved in the appropriate location, it is time to put you network analysis tools to work. There are many, but some of the course authors' favorites are: Wireshark, tshark, Zeek, Suricata, Snort, and tcpdump. All of these tools are relatively easy to install on Linux systems (and some will work on Windows as well) and provide a powerful set of features to identify malicious activity in your newly-recorded network captures.

[1] https://docs.microsoft.com/en-us/azure/network-watcher/network-watcher-packet-capture-overview

tshark

- Command-line version of Wireshark
  - Network protocol analyzer
  - Reads existing PCAPs or captures real-time network data
  - Presents network data back to console in human-readable form
- As **`tshark`** understands several protocols, may be used to identify malicious, unencrypted traffic
- "Follow TCP Stream" is difficult to track compared to Wireshark's GUI-based variant
  - **colorize-tshark-stream.sh** was created to distinguish between client and server traffic

When installing Wireshark on your analysis system, you also get the luxury of its command-line variant, `tshark`.[1] This tool can read your packet capture or even capture data in real-time on the system and present the data in a more human-readable format. This will allow you to more quickly perform your analysis instead of parsing raw network bytes yourself and only then understanding what is happening.

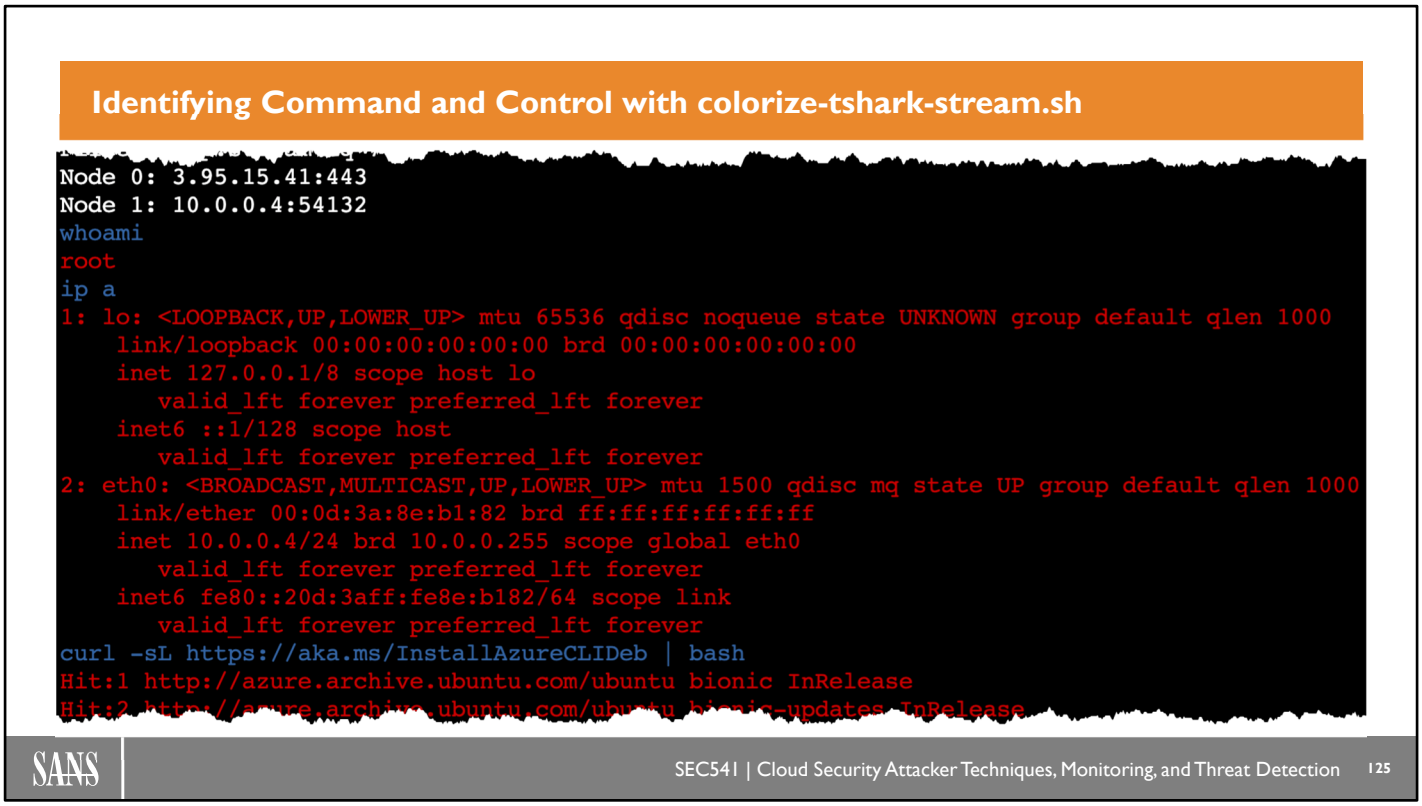
The `tshark` tool also understands many common protocols—showcasing more data to you about the underlying application communication. For instance, if there is C2 traffic using HTTP as its application to send and receive traffic, `tshark` will be able to quickly show you the HTTP methods, headers, Uniform Resource Identifiers (URIs), User-Agents, and request and response bodies. With this knowledge, you may quickly identify what the attacker is instructing the system to perform or what data is being returned to the attacker's system.

One very handy capability that exists in both Wireshark and `tshark` is the ability to follow a TCP stream. Since TCP communication can be tracked by these tools, they can analyze this traffic for you and show you the human-readable layer 7 data sent between the client and server. Wireshark makes the distinction between client and server easy to see as they will apply the red color to the client and a blue color to the server involved in the communication.

The `tshark` application, however, does not do this. All the text will be the same as what the terminal is using. For example, if the terminal is a black background with white lettering, both the client and server traffic will be this same coloring—making it difficult to distinguish between the client and server traffic.

A course author set out to solve this problem by creating a Bash script which can take a `tshark` "Follow TCP Stream" command as input and restore the coloring to the communication—just like Wireshark does. The tool, `colorize-tshark-stream.sh`, will color the client traffic in red and the server traffic in blue. This tool is included in your Inspector-Workstation AWS instance in your lab environment so feel free to use it back at the office!

[1] https://www.wireshark.org/docs/man-pages/tshark.html



The `colorize-tshark-stream.sh` tool was created while creating the next lab in this book. During the creation of the lab, "Follow TCP Stream" was used to showcase the C2 traffic used during the attack which you analyzed in several of this book's labs. The tshark command first leveraged in the above example was:

```
tshark -r ~/labs/sec541-labs/logs/capture.pcap -q -z \
  follow,tcp,ascii,744
```

This command will follow the 744th TCP stream (which was determined to be suspicious) and show you the data in ASCII format. As great as tshark is at showing you this data, it is quite hard to read as all the text is the same color—whether client to server or server to client traffic.

The `colorize-tshark-stream.sh script` steps in to solve this problem. By simply feeding the command with a `tshark` command used to follow a stream, the script will attempt to color the client to server traffic in red and the server to client traffic in blue. We can see the result of this command above:

```
~/labs/sec541-labs/scripts/colorize-tshark-stream.sh \
  'tshark -r ~/labs/sec541-labs/logs/capture.pcap -q -z \
  follow,tcp,ascii,744'
```

Since the client ran malware to connect back to the attacker's server, the blue traffic is the Linux shell command that the attacking server instructs the victim to perform. The red traffic, on the other hand, is the standard output of the executed command performed by the victim system. The attacker's commands look like run-of-the-mill interactions to learn more about the system (`whoami` and `ip a`) and a tool install (`curl -sL https://aka.ms/InstallAzureCLIDeb | bash`). The tool, in this case, is the Azure CLI tool. Why would they want to leverage this tool? Find out in the next lab.

Course Roadmap

- Section 1: Management Plane and Network Logging
- Section 2: Compute and Cloud Services Logging
- Section 3: Cloud Service and Data Discovery
- **Section 4: Microsoft Ecosystem**
- Section 5: Automated Response Actions and CloudWars

Microsoft Ecosystem

1. Malwarebytes Attack
2. Microsoft 365
3. EXERCISE: Microsoft 365 Exchange Investigation
4. SolarWinds Attack
5. Azure Active Directory (AD)
6. EXERCISE: Introduction to KQL
7. Storage Monitoring
8. EXERCISE: Log Analytics Using Azure CLI
9. Detection Services
10. EXERCISE: Microsoft Defender for Cloud and Sentinel
11. Network Traffic Analysis
12. **EXERCISE: Azure Network Traffic Analysis**

This page intentionally left blank.

Lab 4.5 | Azure Network Traffic Analysis

Exercise Duration: 30 Minutes

Objectives

In this lab, we will analyze Azure Network Security Group (NSG) flow data using Azure Log Analytics queries by:

- Identifying network port scans using Azure Network Security Group (NSG) flow data (T1046)
- Investigating inbound traffic from known attacker IP address
- Identifying allowed traffic from attacker to victim
- Discovering traffic sourced from victim back to attacker
- Finding fallback channels (T1008)
- Bonus: Use tshark to finally uncover the attacker's command and control (C2) traffic (T1219)

This final exercise of the book will continue the attack campaign from the last three labs. This lab, however, will focus on the network artifacts generated by the Azure Network Security Group (NSG) service. The flow data that is available will be queried using the appropriate Azure Log Analytics table to identify the MITRE ATT&CK techniques T1046 (Network Service Scanning) and T1008 (Fallback Channels).

On top of this, there is a bonus exercise where you move back to your Inspector Workstation and use tshark to see exactly what was being sent over the attacker's command and control (C2) channel.

Section 4 Wrap Up

- We have reached the end of the Microsoft Ecosystem discussions:
  - Discussed the Malwarebytes and SolarWinds breaches
  - Analyzed Microsoft 365 log data
  - Leveraged KQL to identify several MITRE ATT&CK techniques
  - Used cloud native security products to gain a better understanding of the attack campaign plaguing our Azure environment
  - Reviewed network data to understand the command and control communication between the victim and attacker
- The next book will focus on Automated Response Actions and CloudWars!

This page intentionally left blank.

COURSE RESOURCES AND CONTACT INFORMATION

AUTHOR CONTACTS

Shaun McCullough
smccullough@sans.org

Ryan Nicholson
ryananicholson@gmail.com

SANS INSTITUTE
11200 Rockville Pike, Suite 200
N. Bethesda, MD 20852
301.654.SANS(7267)

CLOUD RESOURCES
sans.org/cloud-security
Twitter: @SANSCloudSec

SANS EMAIL
GENERAL INQUIRIES: info@sans.org
REGISTRATION: registration@sans.org
TUITION: tuition@sans.org
PRESS/PR: press@sans.org

This page intentionally left blank.