Microsoft at your BEC and (API) Call

Richard Smith, Senior Consultant (Security Risk Advisors)
October 2024

WHOIS

I’m Richard Smith, a Senior Consultant with Security Risk Advisors.

- 4 years cybersecurity experience
    - SOC defense & leadership
    - CySA+
    - DevOps engineering
- 10 years infrastructure IT experience
    - Desktop IT
    - Systems and Network Administration
    - Virtualization (VMware, Nutanix, Citrix)

Email: richard.smith@sra.io

1.
Storytime

Are you sitting comfortably?

Meet Jim

- Jim is the CISO at St Quentin's Hospital, a medium-sized health care organization.
- They use Microsoft 365 for email.
- They're cost-conscious, but in a highly-regulated industry.
- Securing PHI/PII is of vital importance.

Jim is very concerned…

- …about the number, scope, and cost of Business Email Compromise attacks.

- The cost of a breach can be cripplingly high.

If there is a cyber incident…

- Like, say, a compromised user account…
- …that has access to a mailbox…
- …that contains sensitive information…

How do you know what the hacker saw?

- Previously, due to audit gaps caused by licensing issues…
- …you had to assume the intruder saw EVERYTHING…
- …and you had to report that they saw EVERYTHING…
- …and you would be fined as if they saw EVERYTHING…

Most data breaches are affected by this auditing gap.

- Perry Johnson & Associates, May 2023: 8,952,212 impacted
- MIE (Medical Informatics Engineering), July 2015: 3.9 million impacted
- It’s likely that in a lot of cases, the actual number of records accessed is much lower than reported.

Jim has an idea!

- What if we could show exactly which emails were accessed in a breach?
- What if Microsoft made these logs available for export to any platform?
- What if there were indicators of compromise that could be leveraged for SIEM alerts?

2.
Accessing Email Audit Logs

It’s not quite as simple as it sounds

First, make sure the logs are enabled

- Go to https://compliance.microsoft.com/auditlogsearch
- Or use PowerShell:
  - Connect to Exchange Online PowerShell
  - `Set-AdminAuditLogConfig -UnifiedAuditLogIngestionEnabled $true`
- Source: https://learn.microsoft.com/en-us/purview/audit-log-enable-disable

The audit logs are there...

- But how do we see them? How do we interpret them? Store them?

We can get logs a couple of ways....

- Exchange Online PowerShell has the `Search-AdminAuditLog` cmdlet.
- Or we can get logs via the Office 365 Management API.

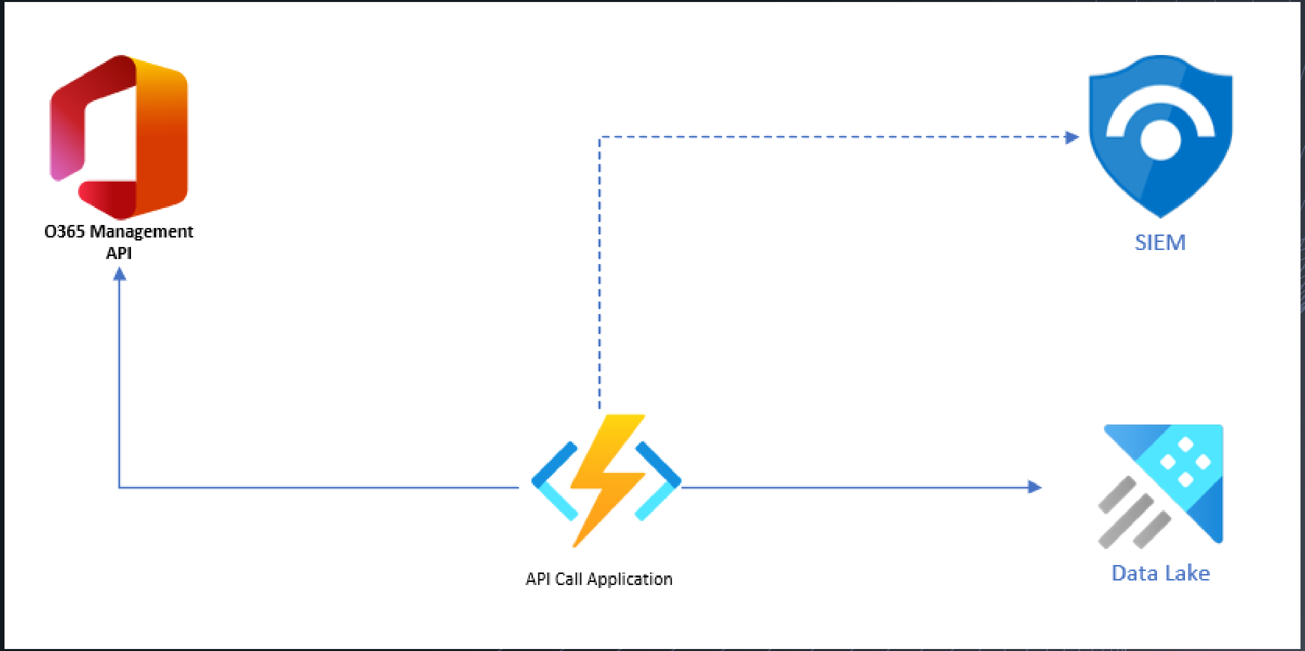
They're not natively exportable...

- The logs cannot be exported natively from Purview.
    - To export the logs to a storage blob, data lake or SIEM, we will have to get creative.
- Fortunately...

They're exportable via API!

- Office 365 Management API comes to the rescue!
- We'll need to process, filter and shape the data to parse it correctly in our datalake
    - We use Cribl for this. Other tools are available.
- So our overall data flow will look like this...

This is the Overall Data Flow

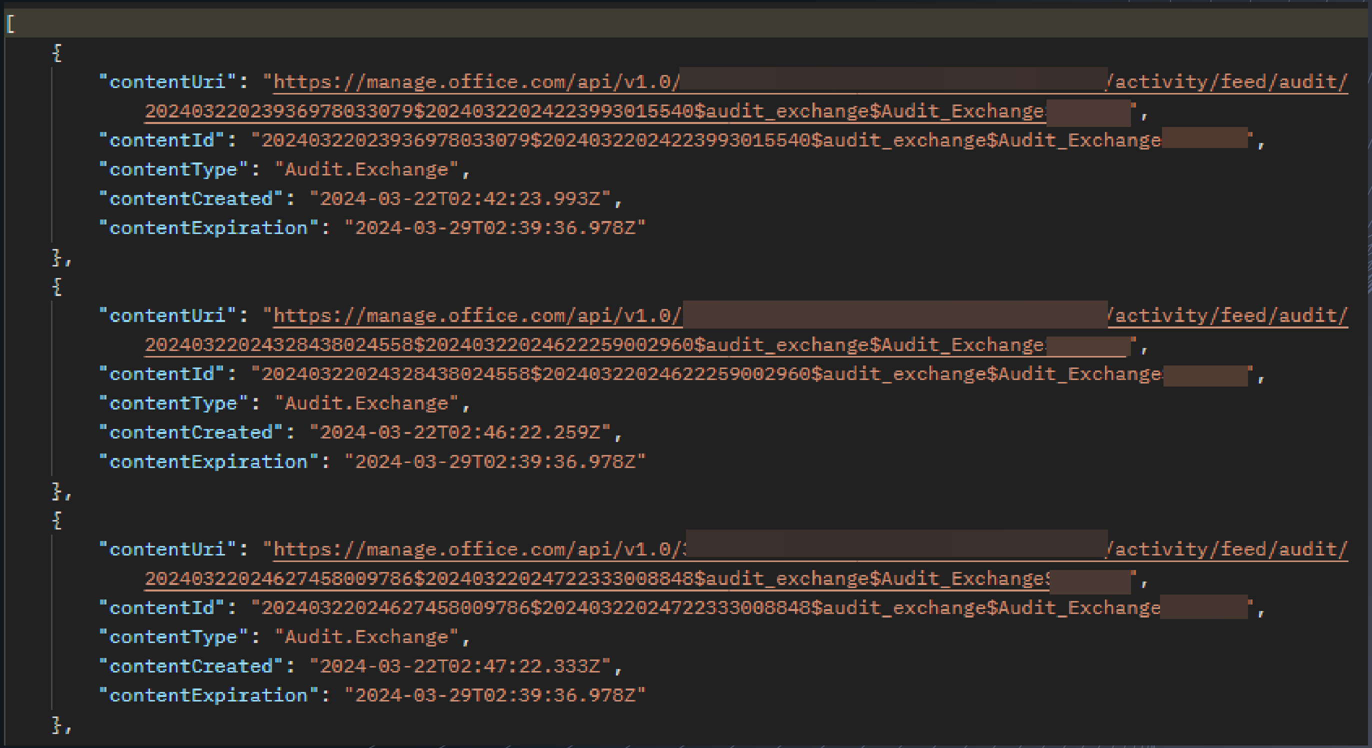


Let’s take a look at the API’s response

- Jim’s security engineers are ready to start implementing the solution.
- Can’t wait to see the results of a POST command to the API.
- They hit the API endpoint with a bearer token they got using their API creds, and ask for Exchange Audit logs.

Are we ready???

Let’s see those logs...



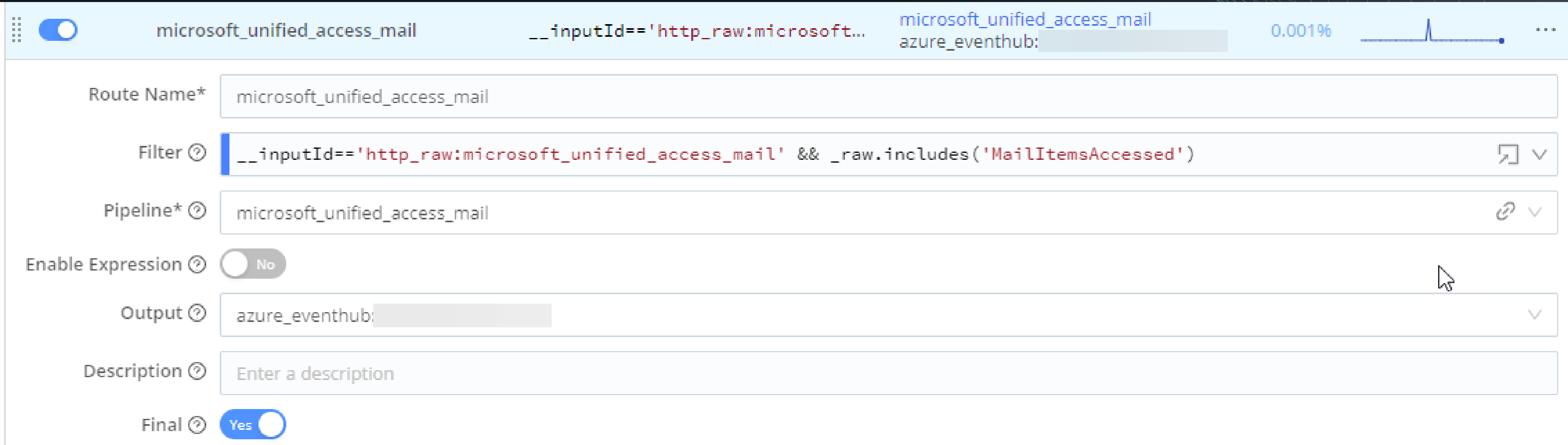
```
[
    {
        "contentUri": "https://manage.office.com/api/v1.0/[redacted]/activity/feed/audit/
            20240322023936978033079$20240322024223993015540$audit_exchange$Audit_Exchange[redacted]",
        "contentId": "20240322023936978033079$20240322024223993015540$audit_exchange$Audit_Exchange[redacted]",
        "contentType": "Audit.Exchange",
        "contentCreated": "2024-03-22T02:42:23.993Z",
        "contentExpiration": "2024-03-29T02:39:36.978Z"
    },
    {
        "contentUri": "https://manage.office.com/api/v1.0/[redacted]/activity/feed/audit/
            20240322024328438024558$20240322024622259002960$audit_exchange$Audit_Exchange[redacted]",
        "contentId": "20240322024328438024558$20240322024622259002960$audit_exchange$Audit_Exchange[redacted]",
        "contentType": "Audit.Exchange",
        "contentCreated": "2024-03-22T02:46:22.259Z",
        "contentExpiration": "2024-03-29T02:39:36.978Z"
    },
    {
        "contentUri": "https://manage.office.com/api/v1.0/[redacted]/activity/feed/audit/
            20240322024627458009786$20240322024722333008848$audit_exchange$Audit_Exchange[redacted]",
        "contentId": "20240322024627458009786$20240322024722333008848$audit_exchange$Audit_Exchange[redacted]",
        "contentType": "Audit.Exchange",
        "contentCreated": "2024-03-22T02:47:22.333Z",
        "contentExpiration": "2024-03-29T02:39:36.978Z"
    },
```

They're just Storage Blob URIs!

- They're not logs at all.
- Each line item in the audit log is a link to an Azure Storage Blob. That's where the actual logs are stored.
- So we have to POST to the API to get the list of Azure Storage Blob URIs.
- Then we GET the logs stored at each URI and forward them.

We want to filter in access logs

- The API endpoint returns all Office 365 mail audit logs.
    - Create, delete, send etc.
- We only want MailItemsAccessed.
- We use filtered routes in data processing tools to only send MailItemsAccessed to the Data Lake.



microsoft_unified_access_mail __inputId=='http_raw:microsoft... microsoft_unified_access_mail azure_eventhub:[redacted] 0.001%

Route Name*: microsoft_unified_access_mail

Filter:

```
__inputId=='http_raw:microsoft_unified_access_mail' && _raw.includes('MailItemsAccessed')
```

Pipeline*: microsoft_unified_access_mail

Enable Expression: No

Output: azure_eventhub:[redacted]

Description: Enter a description

Final: Yes

Filtered route in Cribl

We want to Detect Throttling

- “If more than 1,000 MailItemsAccessed audit records are generated in less than 24 hours [on a single mailbox], Exchange Online stops generating auditing records for MailItemsAccessed activity.”
  - https://learn.microsoft.com/en-us/purview/audit-log-investigate-accounts

We can send throttled accounts to the SIEM

- When an account is throttled, the audit log sets the `IsThrottled` flag to `true`.
- We can filter and route logs containing `IsThrottled:true` to our SIEM and build a high-fidelity detection rule.

3.
Building an App in Node.JS

Taking this from concept to reality

Now we know how to get the logs

- We authenticate to the API.
- It returns the ‘logs’.
- Since the ‘logs’ are just storage blob URIs, send another request to access the logs at each blob.

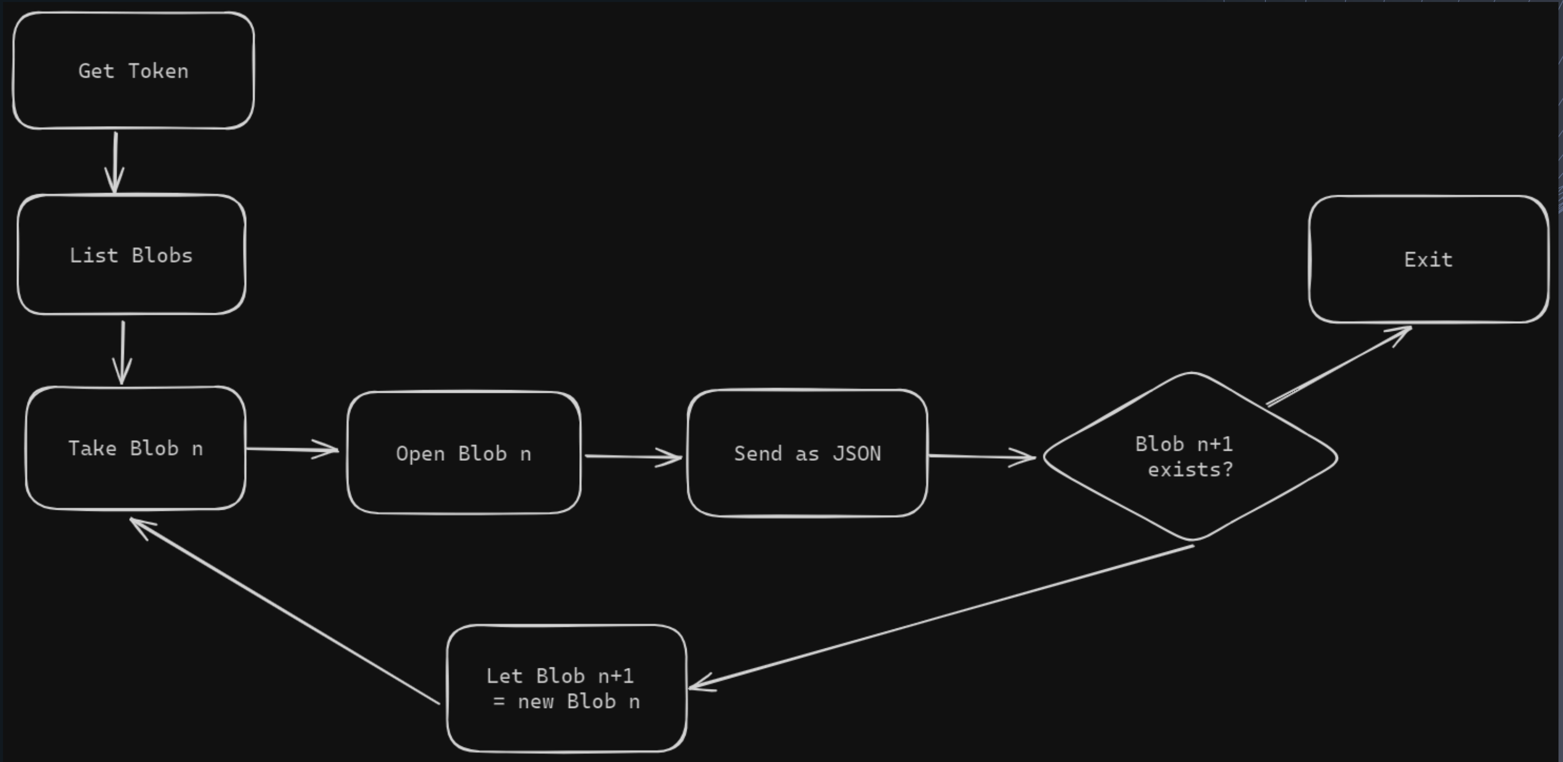
Let’s send them to our Data Lake and SIEM

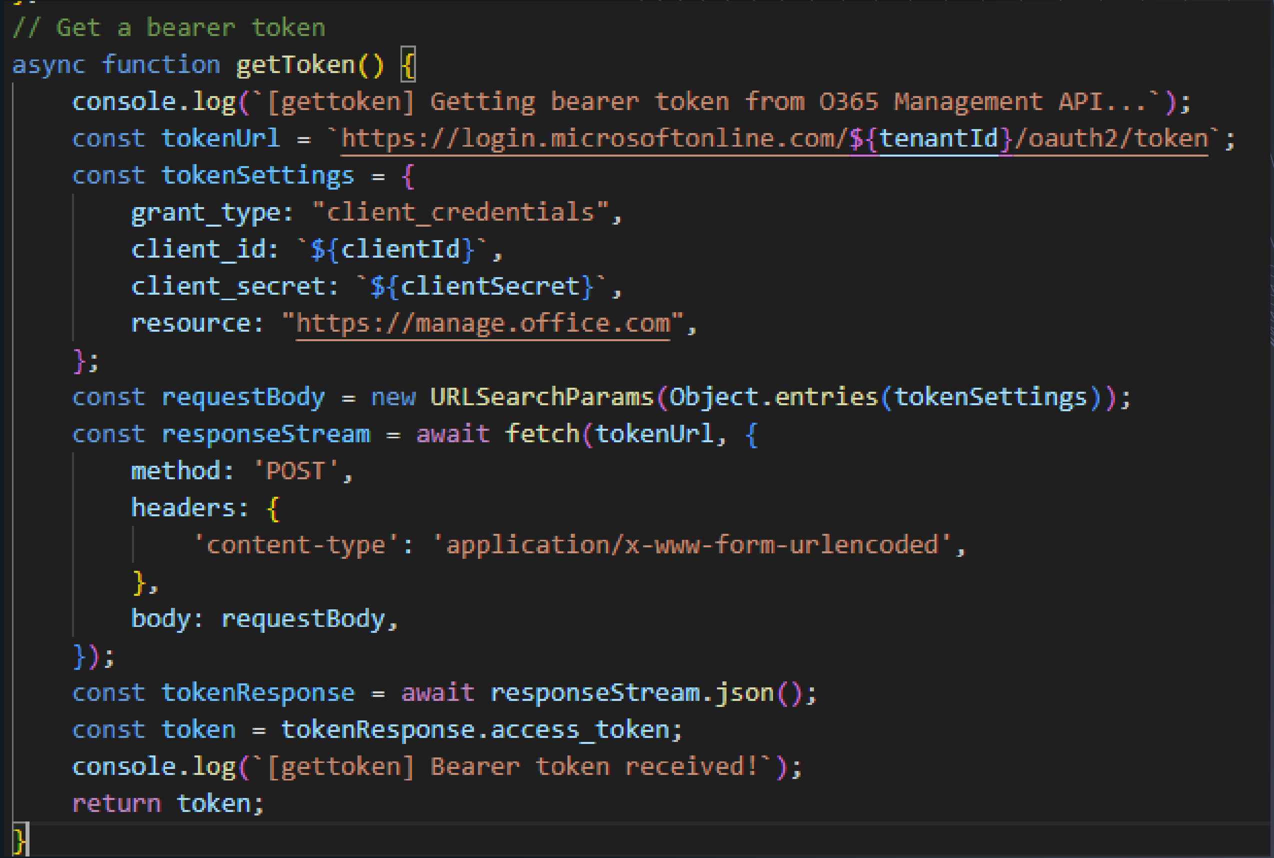
- This way we have actionable data on throttled accounts going to SIEM (high-cost, short-term log retention – reduce volume as much as possible).
- We also have all MailItemAccessed records in long-term, cheap storage for investigation and enrichment.

We can use a Serverless Function

- No need to deploy and maintain a VM to run the app.
- Cheaper than a Cloud VM.
- We used Node.js running in an Azure Function App, but most cloud solutions will support any runtime (e.g. Python, Powershell).

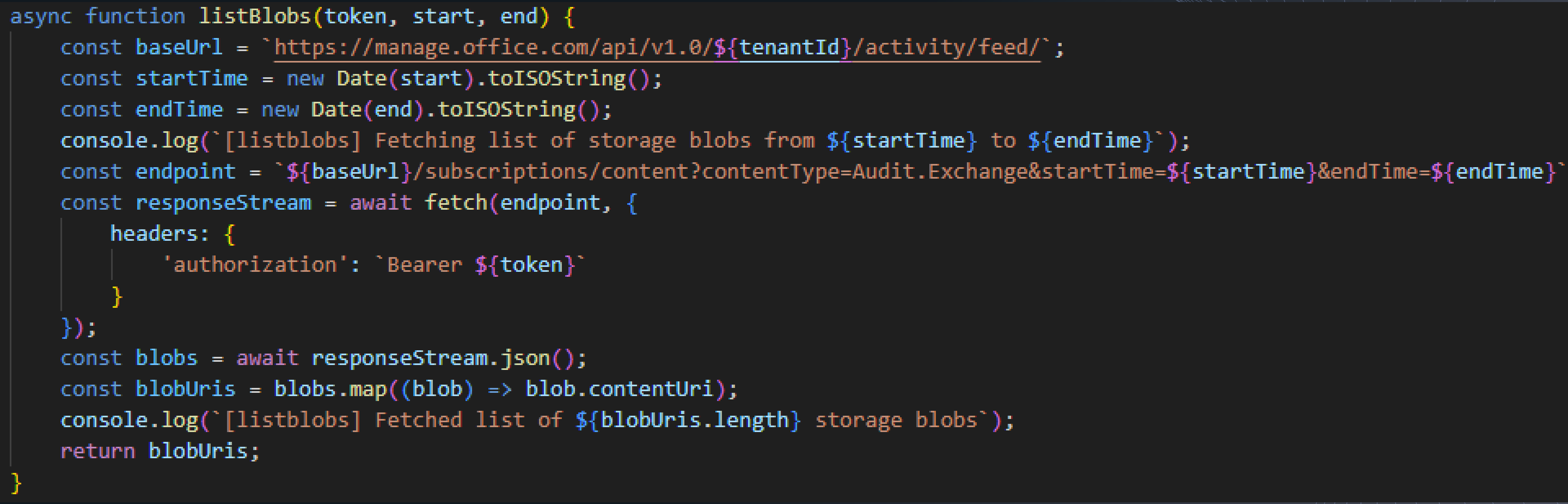
Mapping out the Application Logic





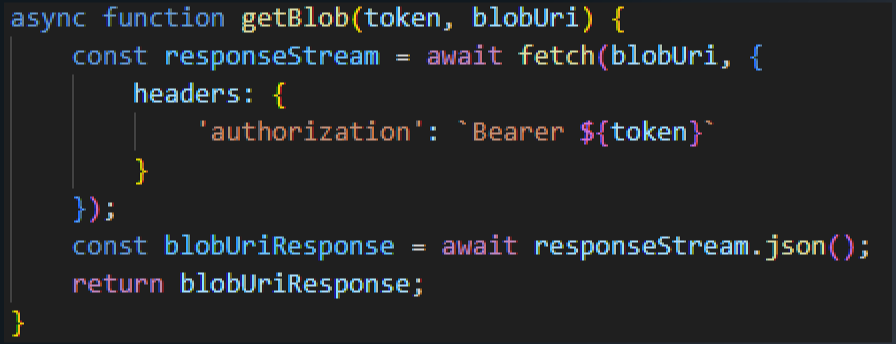
```
// Get a bearer token
async function getToken() {
    console.log(`[gettoken] Getting bearer token from O365 Management API...`);
    const tokenUrl = `https://login.microsoftonline.com/${tenantId}/oauth2/token`;
    const tokenSettings = {
        grant_type: "client_credentials",
        client_id: `${clientId}`,
        client_secret: `${clientSecret}`,
        resource: "https://manage.office.com",
    };
    const requestBody = new URLSearchParams(Object.entries(tokenSettings));
    const responseStream = await fetch(tokenUrl, {
        method: 'POST',
        headers: {
            'content-type': 'application/x-www-form-urlencoded',
        },
        body: requestBody,
    });
    const tokenResponse = await responseStream.json();
    const token = tokenResponse.access_token;
    console.log(`[gettoken] Bearer token received!`);
    return token;
}
```

Get Token



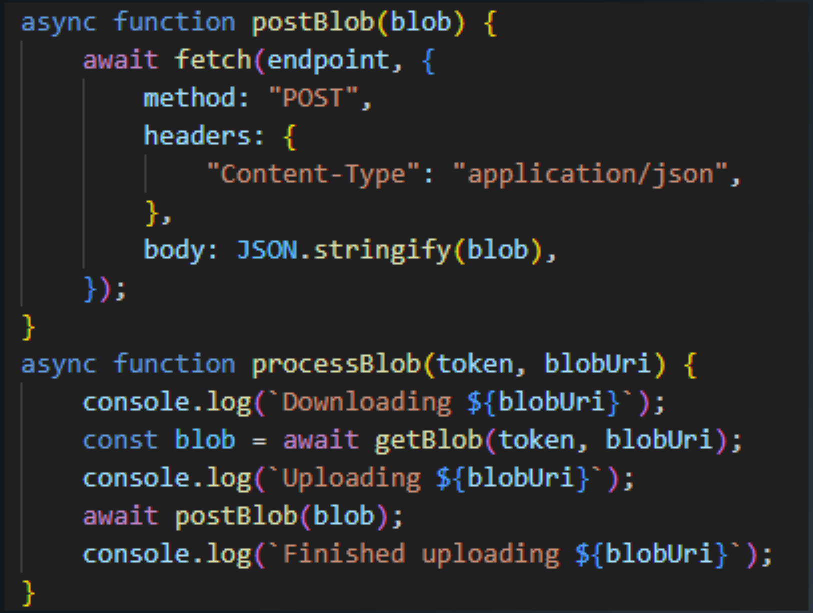
```
async function listBlobs(token, start, end) {
    const baseUrl = `https://manage.office.com/api/v1.0/${tenantId}/activity/feed/`;
    const startTime = new Date(start).toISOString();
    const endTime = new Date(end).toISOString();
    console.log(`[listblobs] Fetching list of storage blobs from ${startTime} to ${endTime}`);
    const endpoint = `${baseUrl}/subscriptions/content?contentType=Audit.Exchange&startTime=${startTime}&endTime=${endTime}`
    const responseStream = await fetch(endpoint, {
        headers: {
            'authorization': `Bearer ${token}`
        }
    });
    const blobs = await responseStream.json();
    const blobUris = blobs.map((blob) => blob.contentUri);
    console.log(`[listblobs] Fetched list of ${blobUris.length} storage blobs`);
    return blobUris;
}
```

List Blobs



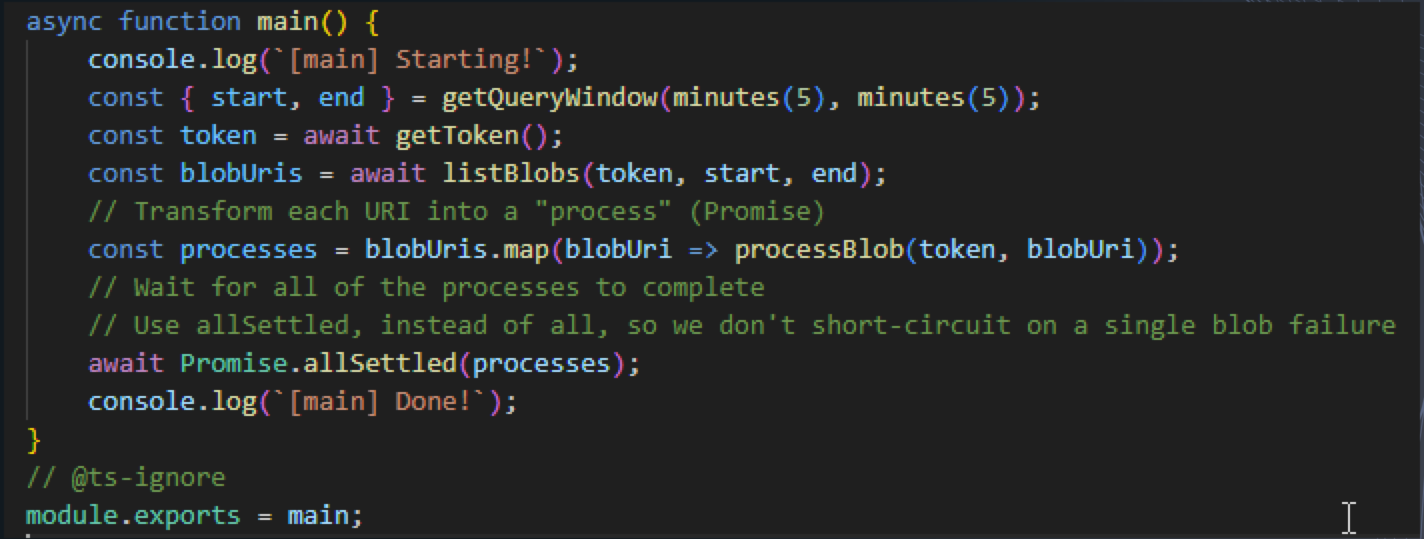
```
async function getBlob(token, blobUri) {
    const responseStream = await fetch(blobUri, {
        headers: {
            'authorization': `Bearer ${token}`
        }
    });
    const blobUriResponse = await responseStream.json();
    return blobUriResponse;
}
```

Get Blob



```javascript
async function postBlob(blob) {
    await fetch(endpoint, {
        method: "POST",
        headers: {
            "Content-Type": "application/json",
        },
        body: JSON.stringify(blob),
    });
}
async function processBlob(token, blobUri) {
    console.log(`Downloading ${blobUri}`);
    const blob = await getBlob(token, blobUri);
    console.log(`Uploading ${blobUri}`);
    await postBlob(blob);
    console.log(`Finished uploading ${blobUri}`);
}
```

Process and Post Blob Contents



```
async function main() {
    console.log(`[main] Starting!`);
    const { start, end } = getQueryWindow(minutes(5), minutes(5));
    const token = await getToken();
    const blobUris = await listBlobs(token, start, end);
    // Transform each URI into a "process" (Promise)
    const processes = blobUris.map(blobUri => processBlob(token, blobUri));
    // Wait for all of the processes to complete
    // Use allSettled, instead of all, so we don't short-circuit on a single blob failure
    await Promise.allSettled(processes);
    console.log(`[main] Done!`);
}
// @ts-ignore
module.exports = main;
```

OK, let’s put this all together!

To **sum up:**

- We get a bearer token by presenting our API creds.
- We use that token to get a list of storage blob URIs.
- We download the contents of each storage blob and send for processing and filtering.
- Then the data goes to our data lake & SIEM.

4.

Putting it to Use

What if we had an actual incident?

Oh no! St Quentin's got compromised!

- Jim's SOC team discovered that an attacker gained access to a user account and was able to log in to their email inbox.
- From our org's MFA logs we can see that a request was sent from a suspicious IP address (logged) and accepted.
- The attacker had access from 7am UTC on 3/14/2024 to 1:30pm UTC on 3/18/2024.

Time to investigate!

- We have the IP address of the hacker.
- We have the timeframe of their access.
- What emails did the hacker access?

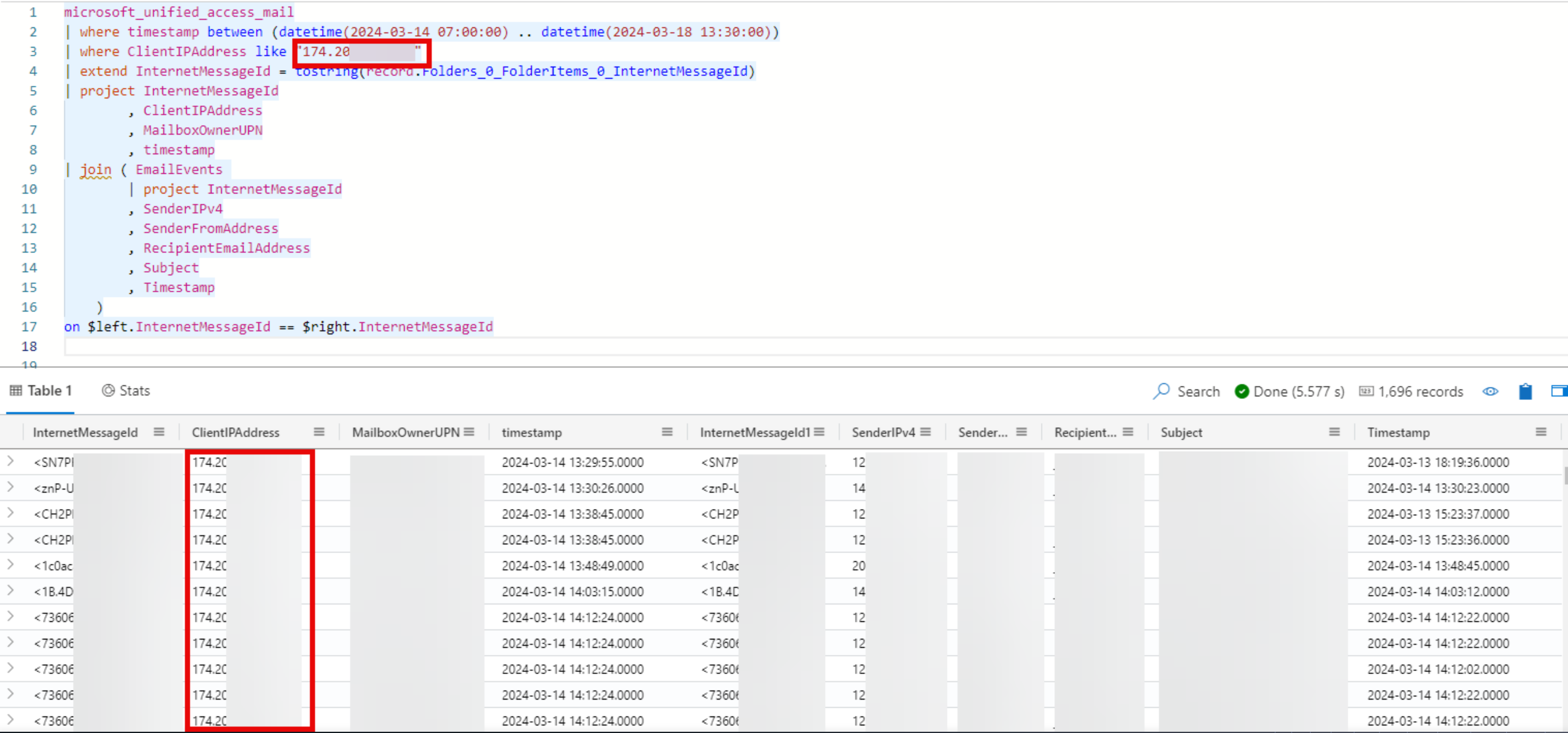
Time to investigate!

- Has the IsThrottled flag been set to True?

- Is there an alert in the SIEM?

Diving in to the Data Lake!

- The audit logs only identify the email by the Internet Message ID.
- But we can cross-reference the Office 365 EmailEvents with our audit logs (if you’re sending these to your Data Lake)
- We join on the InternetMessageId field as the unique identifier and we can see sender and recipient info!



```
microsoft_unified_access_mail
| where timestamp between (datetime(2024-03-14 07:00:00) .. datetime(2024-03-18 13:30:00))
| where ClientIPAddress like '174.20[redacted]'
| extend InternetMessageId = tostring(record.Folders_0_FolderItems_0_InternetMessageId)
| project InternetMessageId
        , ClientIPAddress
        , MailboxOwnerUPN
        , timestamp
| join ( EmailEvents
        | project InternetMessageId
        , SenderIPv4
        , SenderFromAddress
        , RecipientEmailAddress
        , Subject
        , Timestamp
    )
on $left.InternetMessageId == $right.InternetMessageId
```

Done (5.577 s)

1,696 records

InternetMessageId	ClientIPAddress	MailboxOwnerUPN	timestamp	InternetMessageId1	SenderIPv4	Sender...	Recipient...	Subject	Timestamp
<SN7PI[redacted]	174.2C[redacted]	[redacted]	2024-03-14 13:29:55.0000	<SN7P[redacted]	12[redacted]	[redacted]	[redacted]	[redacted]	2024-03-13 18:19:36.0000
<znP-U[redacted]	174.2C[redacted]	[redacted]	2024-03-14 13:30:26.0000	<znP-U[redacted]	14[redacted]	[redacted]	[redacted]	[redacted]	2024-03-14 13:30:23.0000
<CH2PI[redacted]	174.2C[redacted]	[redacted]	2024-03-14 13:38:45.0000	<CH2P[redacted]	12[redacted]	[redacted]	[redacted]	[redacted]	2024-03-13 15:23:37.0000
<CH2PI[redacted]	174.2C[redacted]	[redacted]	2024-03-14 13:38:45.0000	<CH2P[redacted]	12[redacted]	[redacted]	[redacted]	[redacted]	2024-03-13 15:23:36.0000
<1c0ac[redacted]	174.2C[redacted]	[redacted]	2024-03-14 13:48:49.0000	<1c0ac[redacted]	20[redacted]	[redacted]	[redacted]	[redacted]	2024-03-14 13:48:45.0000
<1B.4D[redacted]	174.2C[redacted]	[redacted]	2024-03-14 14:03:15.0000	<1B.4D[redacted]	14[redacted]	[redacted]	[redacted]	[redacted]	2024-03-14 14:03:12.0000
<73606[redacted]	174.2C[redacted]	[redacted]	2024-03-14 14:12:24.0000	<7360€[redacted]	12[redacted]	[redacted]	[redacted]	[redacted]	2024-03-14 14:12:22.0000
<73606[redacted]	174.2C[redacted]	[redacted]	2024-03-14 14:12:24.0000	<7360€[redacted]	12[redacted]	[redacted]	[redacted]	[redacted]	2024-03-14 14:12:22.0000
<73606[redacted]	174.2C[redacted]	[redacted]	2024-03-14 14:12:24.0000	<7360€[redacted]	12[redacted]	[redacted]	[redacted]	[redacted]	2024-03-14 14:12:02.0000
<73606[redacted]	174.2C[redacted]	[redacted]	2024-03-14 14:12:24.0000	<7360€[redacted]	12[redacted]	[redacted]	[redacted]	[redacted]	2024-03-14 14:12:22.0000
<73606[redacted]	174.2C[redacted]	[redacted]	2024-03-14 14:12:24.0000	<7360€[redacted]	12[redacted]	[redacted]	[redacted]	[redacted]	2024-03-14 14:12:22.0000

To sum up:

- We knew the attacker’s IP based on MFA logs.
- We can search the audit logs for that IP address and cross-reference message IDs with email logs to see which emails were accessed.
- We can now report which mail items were accessed from the attacker’s IP address and effectively reduce the breach scope.

To sum up:

- In this case the number of records accessed did not trigger mailbox throttling.
- If it did, our SOC would immediately be notified thanks to the custom detection we’ve built in our SIEM.

5.
Closing

Kids, DO try this at home

Jim’s idea is awesome:

- We can now show exactly which emails were accessed by a hacker!
- We have a way to export these logs to any platform!
- We can leverage the IsThrottled indicator for SIEM alerts!

Where to find out more

- The JavaScript code is available on our GitHub:
  - https://github.com/SecurityRiskAdvisors/azure-security-tools/
- There’s a full write-up on our blog: https://sra.io/blog/unlocking-microsofts-audit-logs-a-comprehensive-guide-to-enhanced-security-and-risk-mitigation/