O365 ...
Pitting the Theory Against the Practice

FIRST & AfricaCERT Symposium
Open-source Tools and CSIRT Success Stories
Context

• **Business Email Compromise in O365**

• **Toolbox**: all open-source tools

• **To collect**
  • Azure Powershell modules and script
  • ANSSI framework

• **To parse and automate**
  • Jq (json)
  • Csvcut (... csv)
Objective of the incident response ...

- Root cause
  - MFA bypassed
  - Malicious application registered, for which user can give consent, with too much permissions
  - Guest/partner access abused
  - Anti-spoofing/anti-spam policy bypassed (simple spoofing, IA)
  - And all we don’t know yet 😃

- Extent of the compromise
  - New inbox rule created, forwarding to attacker
  - Access by rogue application via tokens
  - Access to other services (Azure resources, SharePoint, Teams, Azure AD, ...)

- Containment
  - Identify impacted users
  - Identify rogue roles/accounts
  - Revoke tokens/change password (not that easy)
... Versus the complexity of the platform

- Number of **admin consoles** (more than 15 !)

- **Licensing** impacts logs retention and features (Identity Protection, policies, cmdlets)

- **Variety of logs** (sign-ins, audit logs, activity logs, risky users, risky sign-ins, ...)
  - And **variety of results** (GUI versus Powershell): limitations, fields, latency, corrupted logs

- More than web services
  - Legacy/basic authentication protocols (**MFA bypass**)
  - OAuth2 applications (called “registered applications”): **new form of phishing**
  - **Guest/partner** access
  - **Add-in** (additional applications, no logs)
  - **Sharing** documents
## Logs of interest

<table>
<thead>
<tr>
<th>Microsoft name</th>
<th>Description</th>
<th>Retention</th>
<th>How to collect</th>
</tr>
</thead>
<tbody>
<tr>
<td>Azure AD sign-ins</td>
<td>All logins (Azure AD, O365 apps, admin)</td>
<td>1 week/1 month</td>
<td>GUI/powershell</td>
</tr>
<tr>
<td>Risky users/risky sign-ins</td>
<td>Abnormal logins/unusual behavior reports</td>
<td>N/A</td>
<td>GUI</td>
</tr>
<tr>
<td>Azure AD audit logs</td>
<td>Tenant management (users, groups, applications registered, admin operations, ...)</td>
<td>1 week/1 month</td>
<td>GUI/powershell</td>
</tr>
<tr>
<td>O365 audit logs</td>
<td>Operations on apps (mailbox, office, teams, OneDrive, ...)</td>
<td>3 months</td>
<td>GUI/powershell</td>
</tr>
<tr>
<td>Message Trace Reports</td>
<td>Emails sent and received</td>
<td>3 months</td>
<td>GUI</td>
</tr>
<tr>
<td>Registered applications</td>
<td>List and permissions</td>
<td>N/A</td>
<td>GUI/powershell</td>
</tr>
</tbody>
</table>

- **Powershell instead of GUI**
  - **Scriptable**
  - **Homogeneity** of the results
  - Better **retention**
  - **Not corrupted**
  - No latency between user’ actions and availability in logs
Azure AD sign-ins

• Extract
  • Json output

```powershell
Import-Module DFIR-O365RC
Senddate = get-date
$Startdate = $senddate.adddays(-30)
Get-AADLogs -startdate $startdate -enddate $senddate
```

• Parse
  • `jq -r '.[].createdDateTime, .userPrincipalName, .ipAddress, .appDisplayName, .appId, .resourceId, .deviceDetail.operatingSystem, .deviceDetail.browser, .location.city, .location.countryOrRegion, .appliedConditionalAccessPolicies[].displayName, .appliedConditionalAccessPolicies[].result' | @csv`
  • “appId” = source / “resourceId” = destination
Azure AD audit logs

• Extract
  • Json output
  • Same command as sign-ins: it collects all in one

```
Import-Module DFIR-0365RC
$enddate = get-date
$startdate = $enddate.adddays(-30)
Get-AADLogs -StartDate $startdate -Enddate $enddate
```

• Parse
  • **Overview** of who did what
    ```
    jq -r '.[].initiatedBy.user.userPrincipalName, .activityDisplayName' | @csv' | sort -u
    ```
  
  • **Timeline** with more details (typically who gave consent to an application)
    ```
    jq -r 'select(.activityDisplayName == "Consent to application") | [.activityDateTime, .initiatedBy.user.ipAddress, .initiatedBy.user.userPrincipalName, .targetResources[].displayName] | @csv'
    ```
  
  • **Generic form** to extract a timeline related to the activity “X” (fieldN of interest for this activity)
    ```
    jq -r 'select(.activityDisplayName == "X") | [.activityDateTime, .<field1>, .<field2>] | @csv'
    ```
Applications registered: list and permissions

- **List applications**
  - Module AzureADPreview, cmdlet Get-AzureADServicePrincipal
  - Json output

- **List permissions**
  - Get-AzureADPSPermissions.ps1: [https://gist.github.com/pignoret/41793f8c6211d2df5051d77ca3728c09](https://gist.github.com/pignoret/41793f8c6211d2df5051d77ca3728c09)
  - Csv output

- **Audit logs limited to activity involving applications**
  - DFIR-O365RC
O365 audit logs

- Extract
  - Json output

- Parse
  - **Overview** of who did what
    
    \[ \text{jq} \ -r \ '(.\ .Operation, .Workload, .UserId) | @csv' | sort -u \]

  - **Examples of Operations**
    - Add-MailboxPermission, MailboxLogin, MailItemsAccessed, Create, Sent, New-InboxRule, Set-InboxRule, UpdateInboxRules
    - AnonymousLinkCreated, FileAccessed, FileCopied, FileDeleted, FileModified, FileMoved, FileDownloaded, FileUploaded

  - **Timeline** for the operation “X”
    - \[ \text{jq} \ -r 'select(.Operation == "X") | [.eventDateTime, .field1, .field2, .field3, ..., .fieldN] | @csv' \]
Message Trace Reports

• Extract
  • GUI
  • Asynchronous process
  • Choose “Extended Report”

• Caution
  • Up to 24 hours to see last emails
  • Not exhaustive: official answer from Microsoft “blocked emails not included until explicitly requested in the filter”
  • Body spoofing difficult to detect (sender = body, not envelop)

• Parse
  • Timeline
csvcut -c date_time_utc,original_client_ip,client_hostname,server_ip,server_hostname,message_id,reference,directionality,sender_address,return_path,recipient_address,message_subject,total_bytes

  • Spoofed email: “client_hostname”, “server_hostname”, “server_ip”, “directionality”, “sender” will point inconsistency
  • Email headers (SPF/DKIM/DMARC/Spam checks): fields “message_info”, “custom_data”
In a “nutshell”

• Data acquisition
  • PowerShell to collect logs, [https://github.com/ANSSI-FR/DFIR-O365RC](https://github.com/ANSSI-FR/DFIR-O365RC)
    • It works and handles token refresh, API throttling, limited number of results per query
    • Ensure content stability (structure, fields, field names) to then automate parsing
  • PowerShell to collect configuration (application IDs, application permissions)
  • Collect Message Trace Reports from GUI (beware of latency)
  • For other configurations: ... screenshots (user consent, policies)

• Logs analysis
  • Fields vary with Operations/Activity, but automation possible thanks to stable logs collection
  • jq: [https://stedolan.github.io/jq/](https://stedolan.github.io/jq/)

• Biggest known caveats
  • Web logins: hard to identify the primary connection of the attacker (1 login = ~30 lines of logs)
  • Application ID puzzle (~30% of IDs neither in Tenant list, nor documented by Microsoft)
  • Message Trace Report latency and still, not exhaustive
Thank you

contact: cert@excellium-services.com