Cloudy with a Chance of APT
Novel Microsoft 365 Attacks in the Wild

Doug Bienstock
Josh Madeley
Doug Bienstock
@doughsec

- Incident Response Manager – 7 years @ Mandiant
- Incident Response and Red Team lead
- Author of adfsdump/spoof, pwnauth
- Lifelong Green Bay Packers fan
Josh Madeley  
@MadeleyJosh

- Manager of professional services – ~6 years @ Mandiant
- Incident Response lead
- Not an author of public tools
- Canadian Ex-Pat that has adopted the Patriots as my team of choice
Last year demonstrated that Apex threat actors have become all stars at abusing Microsoft 365 to achieve their goals.

Large scale espionage campaigns targeted data stored within Microsoft 365.

Novel techniques used to:
- Evade detection
- Automate data theft
- Persistent access beyond credential theft
Avoiding Detection
Disabling Security Features

- Bypass mailbox audit logging
  - **Set-MailboxAuditBypassAssociation**
  - The following scenarios are not logged
    - Mailbox Owner actions by specified users are not logged
    - Delegate actions performed by the users on other mailboxes
    - Admin Actions
- Downgrade licenses to E3
  - Save the target organization some money
  - Disables MailItemsAccessed logging
Mailbox Folder Permission Abuse

If it aint broke, don't fix it.
Mailbox Folder Permissions

- Alternative to Mailbox Delegation
- Mailbox owner, administrator, or an account with full access permissions can grant granular access to specific folders within a mailbox
- Part of Exchange Web Services (EWS)
- Many legitimate use cases will be seen in most environments
  - Sharing calendars
  - “Team” mailboxes
  - Assistants
- First mentioned in red team context by Black Hills in 2017 post

https://www.blackhillsinfosec.com/abusing-exchange-mailbox-permissions-mailsniper
Common Permissions

- Permissions can be assigned as individual permissions or roles
- `ReadItems` grants access to read mail items in a specific folder
- Roles that have the `ReadItems` permission
  - Author
  - Editor
  - NonEditingAuthor
  - Owner
  - PublishingEditor
  - PublishingAuthor
  - Reviewer*
Two Special Users

- Permissions can be assigned to individual users or mail-enabled security group
- Anonymous
  - Any external, unauthenticated users
- Default (aka “Everyone” in certain logs)
  - Any internal, authenticated users
- By default, the access for both special users is set to None.
Abuse

- Neat: Assign the “Default” user “Reviewer” role to allow any authenticated user access to the mailbox folder
  - Permissions do not cascade down from child to parent for existing folders, but newly created folders do inherit
  - Set-MailboxFolderPermission cmdlet OR EWS Managed API calls using a tool like EWSEditor
Detection

- Sign-Ins use EWS to access the modified folders and view email
  - Coded as “non-interactive” sign-ins
  - Non-existent in the Unified Audit Log and must be specifically enabled to forward to SIEM from other MSFT sources
- Unified Audit Log records Set-MailboxPermission events
  - There will be noise from legitimate admin and background EXO activity
- If Mail Items Accessed auditing is enabled look here
  - Throttling concerns
- Enumerate Mailbox Folder Permissions with PowerShell
  - Can be slow and should be targeted towards high value accounts
Hijacking Enterprise Applications and App Registrations
Types of Applications

- Two types of Applications
  - **App Registrations**
    - Initial instance of an application, lives in the tenant that created the app
    - Serves as a "blueprint" to create a service principal in any tenant that uses the application
  - **Enterprise Applications**
    - AKA Service Principals
    - A "copy" of the app registration that lives in the consuming tenant

- Everything in Microsoft 365 uses this model, Microsoft Services like EXO are “first-party” Service Principals
- The term "application" is used to refer to both Enterprise Applications and App Registrations
Two types of permissions can be assigned:

- **Delegated Permissions**: Enable apps to perform API operations on behalf of a user – limited to access data that user has access to. Users consent to the permissions at runtime. The application *acts as* that user.

- **Application Permissions**: Enable apps to perform API operations without a signed in user and access tenant wide data. Requires Admin Consent. The application *acts as* itself.

- Both App Registrations and Enterprise Applications can be assigned permissions
Secrets and Certificates

- Applications can have secrets or certificates associated with them to allow authentication *as the identity of the app*
  - Roughly analogous to API Keys
  - Applications can have multiple secrets or certificates associated with them
- Once created, they cannot be extracted from Azure AD
- Both App Registrations and Enterprise Apps can have secrets assigned to them
  - Enterprise Apps can only have secrets assigned via PowerShell
Enterprise Application Hijacking

- Attackers have modified two key components of existing applications
  - Adding new MS Graph Application Permissions, specifically file.read and mail.read
  - Adding new credentials (both secrets and certificates)
- Access tenant data remotely using the Graph API
  - Conditional Access Policies DO NOT APPLY when authenticating using app secrets
  - Service Principal sign-in logs were not available until mid-2020 and they don’t show in the UAL
- There are dozens of Graph permissions to choose from
  - Domain.ReadWrite.All – Add a rogue IdP
  - Directory.ReadWrite.All
Abuse of App Registrations

- Apps can be created as multi-tenant – customers can “add an app” to their tenant
  - The App Registration is the “master copy” of the app and is linked to all Enterprise Apps in customer tenants
- If we compromise the App Registration we can access data stored in *any* tenant that has the Enterprise App copy
  - All we need is the friendly name (e.g. Microsoft.com) of the tenant we want to access
  - Good luck auditing activities that occur in someone else’s tenant

Caveats

- Permissions in each individual tenant may be different
### App Reg Tenant

<table>
<thead>
<tr>
<th>Basic info</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date</td>
<td>7/21/2021, 1:34:28 PM</td>
</tr>
<tr>
<td>Request ID</td>
<td>fffe5a53-f02d-4970-a214-60a50f70ac01</td>
</tr>
<tr>
<td>Correlation ID</td>
<td>e6b74edd-d4cb-47ae-b1ae-f8f8eb207537</td>
</tr>
<tr>
<td>Status</td>
<td>Success</td>
</tr>
</tbody>
</table>

#### Application

<table>
<thead>
<tr>
<th>Application</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Application ID</td>
<td>6a7d585c-c340-45a6-b72e-96fc7425ba64</td>
</tr>
<tr>
<td>Resource</td>
<td>Microsoft Graph</td>
</tr>
<tr>
<td>Resource ID</td>
<td>00000003-0000-0000-c000-000000000000</td>
</tr>
<tr>
<td>Resource tenant ID</td>
<td></td>
</tr>
<tr>
<td>Home tenant ID</td>
<td></td>
</tr>
<tr>
<td>Service principal ID</td>
<td>9ddf2b4b-f989-4068-b98d-3399ca83517a</td>
</tr>
<tr>
<td>Service principal name</td>
<td>DoughTest</td>
</tr>
</tbody>
</table>

### Target Tenant

<table>
<thead>
<tr>
<th>Basic info</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date</td>
<td>7/21/2021, 1:34:28 PM</td>
</tr>
<tr>
<td>Request ID</td>
<td>fffe5a53-f02d-4970-a214-60a50f70ac01</td>
</tr>
<tr>
<td>Correlation ID</td>
<td>e6b74edd-d4cb-47ae-b1ae-f8f8eb207537</td>
</tr>
<tr>
<td>Status</td>
<td>Success</td>
</tr>
</tbody>
</table>

#### Application

<table>
<thead>
<tr>
<th>Application</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Application ID</td>
<td>6a7d585c-c340-45a6-b72e-96fc7425ba64</td>
</tr>
<tr>
<td>Resource</td>
<td>Microsoft Graph</td>
</tr>
<tr>
<td>Resource ID</td>
<td>00000003-0000-0000-c000-000000000000</td>
</tr>
<tr>
<td>Resource tenant ID</td>
<td></td>
</tr>
<tr>
<td>Home tenant ID</td>
<td></td>
</tr>
<tr>
<td>Service principal ID</td>
<td>9ddf2b4b-f989-4068-b98d-3399ca83517a</td>
</tr>
<tr>
<td>Service principal name</td>
<td>DoughTest</td>
</tr>
</tbody>
</table>
Golden SAML
<t:RequestSecurityTokenResponse
xmlns:t="http://schemas.xmlsoap.org/ws/2005/02/trust">
<saml:Assertion
xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion">
<saml:Attribute
AttributeName="UPN"
AttributeNamespace="http://schemas.xmlsoap.org/claims">
<saml:AttributeValue>robin@doughcorp.co</saml:AttributeValue>
</saml:Attribute>
Decrypting the SigningToken

```csharp
authEncrypt = this.DecodeProtectedBlob(cipherText);
```

<table>
<thead>
<tr>
<th>Groupkey GUID</th>
<th>KDF Algorithm OID</th>
<th>MAC Algorithm OID</th>
</tr>
</thead>
<tbody>
<tr>
<td>3F A6 2E</td>
<td>06</td>
<td>04 02 01</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Encryption Algorithm OID</th>
<th>Encryption IV</th>
<th>Ciphertext</th>
</tr>
</thead>
<tbody>
<tr>
<td>04 01 02</td>
<td>D4 1A 3B C2 B3</td>
<td>82 09 E0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Nonce Value</th>
<th>Ciphertext MAC</th>
</tr>
</thead>
<tbody>
<tr>
<td>04 10 04</td>
<td>D4 1A 3B C2 B3</td>
</tr>
</tbody>
</table>
Key Derivation

```java
public static byte[] DeriveKeySP800_108(HMAC prf, byte[] label, byte[] context, int numberOfBytesToGenerate)
```

- DKM key is not used itself to decrypt Signing Certificate
- Used as initial input for HMAC-SHA256 Key Derivation (NIST SP 800-108)
  - Mostly, but not exactly, follows the standard (because standards are hard ;)
- Context is the Nonce decoded from blob
- Label is the OIDs of the encryption algorithms decoded from blob
- Outputs keys to use for AES encryption as well as SHA256 HMAC for verification of ciphertext
Are we there yet?

- **Claims issuance rules**
  - Determines the claims that will be included in the issued SAML token
  - Order of rules matters
  - Defenders **cannot** see the claims that are put in the token BUT
    - MSFT can, to a degree
    - May be monitoring for tokens that have abnormal or unneeded claims
Are we there yet?

- Token Lifetime
  - Set per Relying Party Trust
  - Default value of 0 == 60 minutes
  - Defenders cannot see the Token Lifetime of submitted SAML tokens BUT
    - Microsoft can
    - May be monitoring for abnormal token lifetimes
  - Spoofed tokens could have a lifetime of years, but will not be valid once the ADFS signing token is rotated after normal 365-day lifespan
Bring your own signing cert

- Why dump the existing signing certificates when we can just use our own?
  - Access to the AD FS server not required
  - Similar to @DrAzureAD AAD Backdoor, but a little stealthier
- `Set-MsolDomainFederationSettings`
  - Global Admin and other privileged roles have access through MSOnline PowerShell
  - Nothing happens on the AD FS server
Nothing to see here, totally normal, nothing was modified

No IP Address recorded

Bring your own signing cert

```json
{
    "CreationTime": "2021-07-20T23:28:45",
    "Operation": "Set federation settings on domain."
```
ADFS Replication
Farmville

- For larger orgs, AD FS servers can exist in a farm configuration
- By default, all farm nodes use the same configuration and secrets
- Nodes are kept in sync by a replication service that runs on the primary AD FS server (the first server that the AD FS role was installed on)
  - It actually runs on all farm nodes, useful for attackers
Replicating

- Replication service uses Windows Communication Framework (WCF)
  - Framework to easily build client server applications
  - Developer can build on top of preset channels (HTTP) and security (WS-Security, Kerberos)
- Endpoint is available at http://sts.acme.com:80/policystoretransfer
  - Kerberos based authentication using WS-TRUST SPNEGO
  - Data payloads are encrypted using shared secret derived from the Kerberos session key
Replicating

```xml
<PolicyStore>
  <AuthorizationPolicy>
    @RuleName = "Permit Service Account"
    => issue(Type = "http://schemas.microsoft.com/authorization/claims/permit", Value = "true");
    @RuleName = "Permit Local Administrators"
    exists([Type = "http://schemas.microsoft.com/ws/2008/06/identity/claims/groupsid", Value = "S-1-5-32-544"])
    => issue(Type = "http://schemas.microsoft.com/authorization/claims/permit", Value = "true");
  </AuthorizationPolicy>
</PolicyStore>
```
Quick and dirty WCF client to interact with the replication service

~150 lines of code, most of it boilerplate WCF initialization
Escalate, persistently

- Edit the ObjectACL for the DKM key to allow domain users read access
- Insert a new Authorization Policy into the AD FS database to permit access for the domain users GroupSID
- Any domain user can obtain the AD FS signing key from anywhere on the internal network
Why?

- AD FS servers expose port 80 to all systems by default
  - The AD FS role creates default firewall rules for us
- Stealth is built in for us 😊
  - Replication events are not logged at all
  - Editing the AD FS configuration database is not logged either
  - Auditing editing domain object ACLs (SACLs) is not often enabled in environments
The End