ProxyLogon

is Just the Tip of the Iceberg

A New Attack Surface on Microsoft Exchange Server!

🍊 Orange Tsai

DEVCORE

black hat USA 2021

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

Vulnerabilities We Discovered

Vulnerability related to this new attack surface

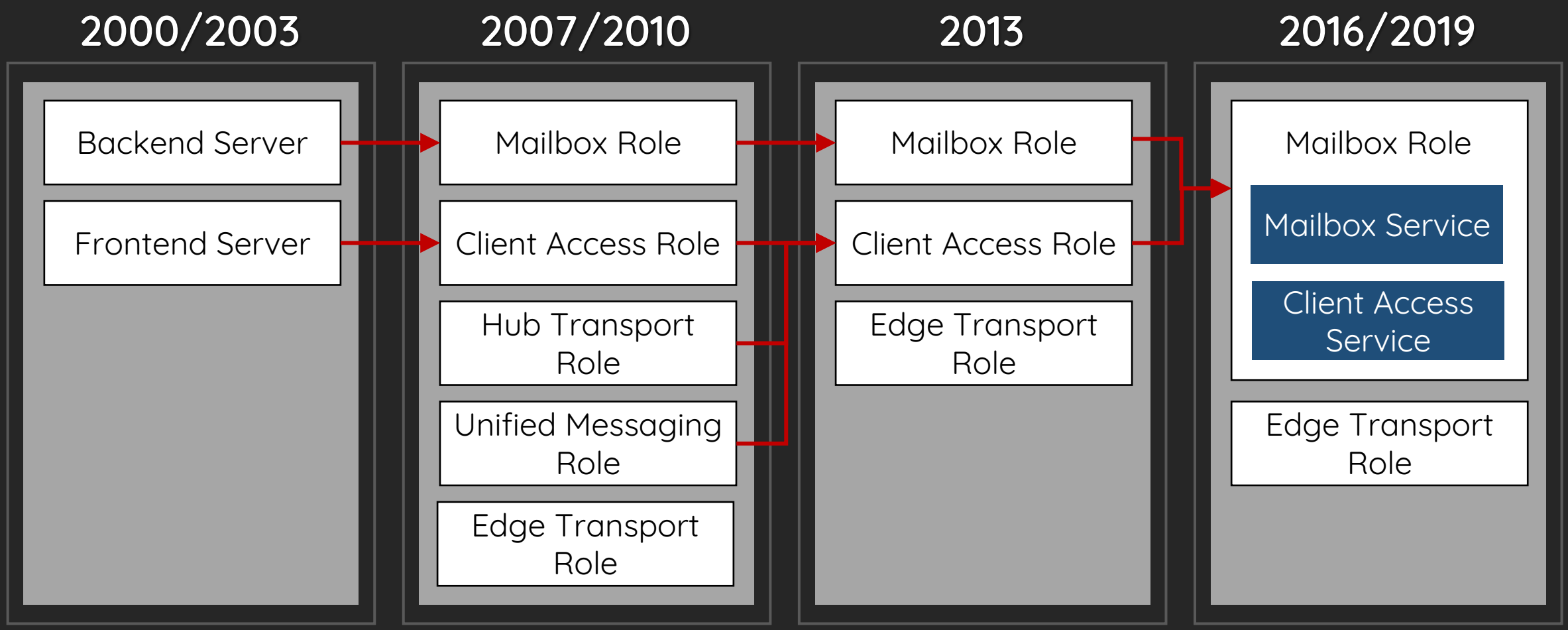
Report Time	Name	CVE	Patch Time	Reported by
Jan 05, 2021	ProxyLogon	CVE-2021-26855	Mar 02, 2021	Orange Tsai, Volexity and MSTIC
Jan 05, 2021	ProxyLogon	CVE-2021-27065	Mar 02, 2021	Orange Tsai, Volexity and MSTIC
Jan 17, 2021	ProxyOracle	CVE-2021-31196	Jul 13, 2021	Orange Tsai
Jan 17, 2021	ProxyOracle	CVE-2021-31195	May 11, 2021	Orange Tsai
Apr 02, 2021	ProxyShell (Pwn2Own Bug)	CVE-2021-34473	Apr 13, 2021	Orange Tsai (Working with ZDI)
Apr 02, 2021	ProxyShell (Pwn2Own Bug)	CVE-2021-34523	Apr 13, 2021	Orange Tsai (Working with ZDI)
Apr 02, 2021	ProxyShell (Pwn2Own Bug)	CVE-2021-31207	May 11, 2021	Orange Tsai (Working with ZDI)
Jun 02, 2021	-	-	-	Orange Tsai

Vulnerabilities Related to This Attack Surface

Vulnerability related to this new attack surface

Dubbed to	CVE	Patch Time	Reported by
HAFNIUM	CVE-2021-26855	Mar 02, 2021	Orange Tsai, Volexity and MSTIC
HAFNIUM	CVE-2021-27065	Mar 02, 2021	Orange Tsai, Volexity and MSTIC
HAFNIUM	CVE-2021-26857	Mar 02, 2021	Dubex and MSTIC
HAFNIUM	CVE-2021-26858	Mar 02, 2021	MSTIC
-	CVE-2021-28480	Apr 13, 2021	NSA
-	CVE-2021-28481	Apr 13, 2021	NSA
-	CVE-2021-28482	Apr 13, 2021	NSA
-	CVE-2021-28483	Apr 13, 2021	NSA

Exchange Architecture

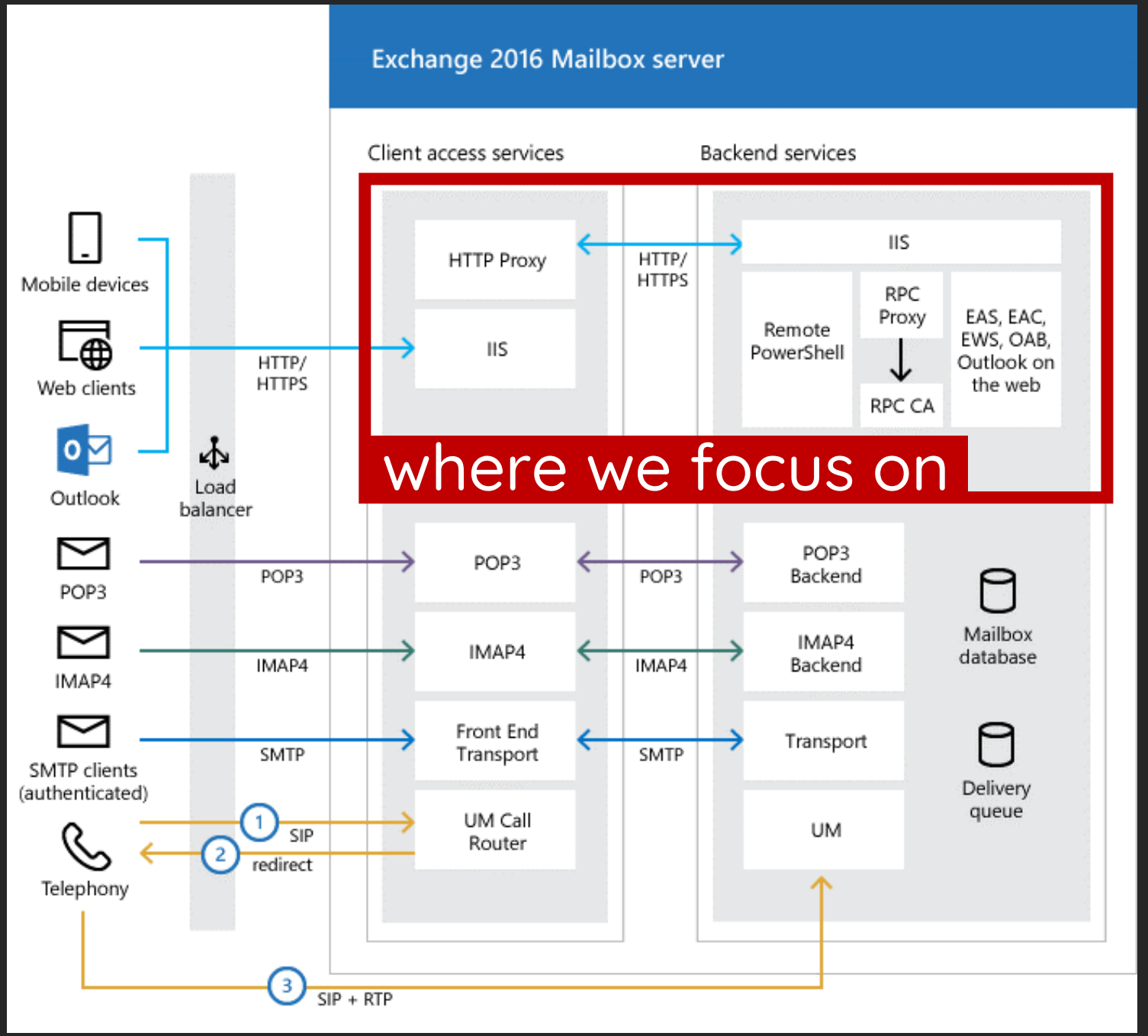


Where to Focus?

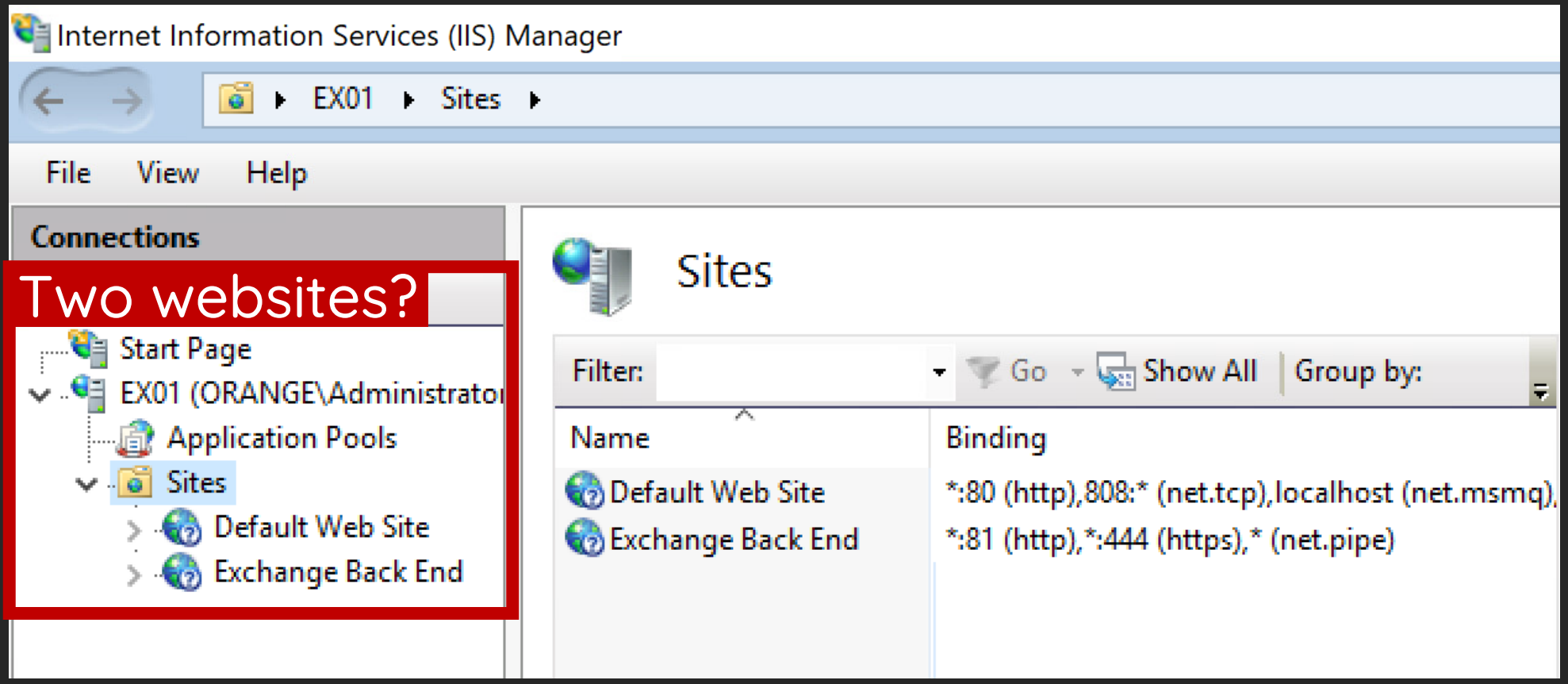
- We focus on the Client Access Service (CAS)
- CAS is a fundamental protocol handler in Microsoft Exchange Server.

  The Microsoft official documentation also indicates:

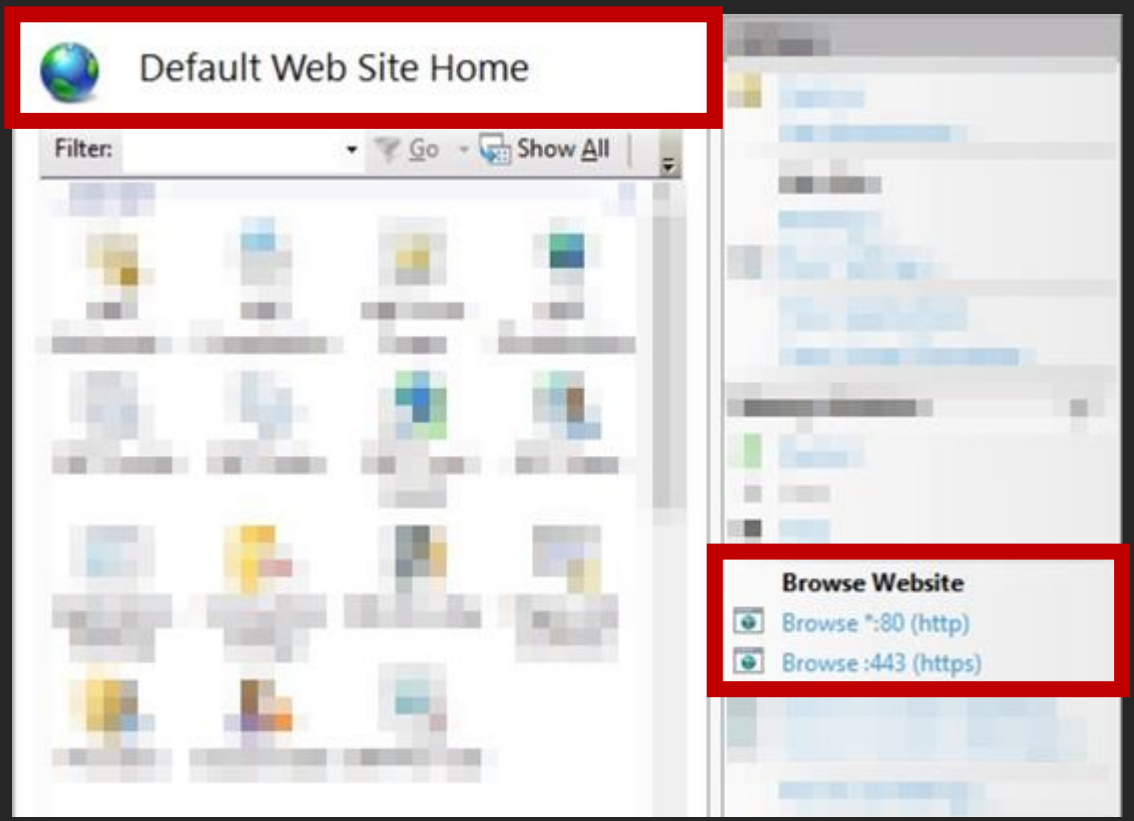
  > "Mailbox servers contain the Client Access Services that accept client connections for all protocols. These frontend services are responsible for routing or proxying connections to the corresponding backend services"

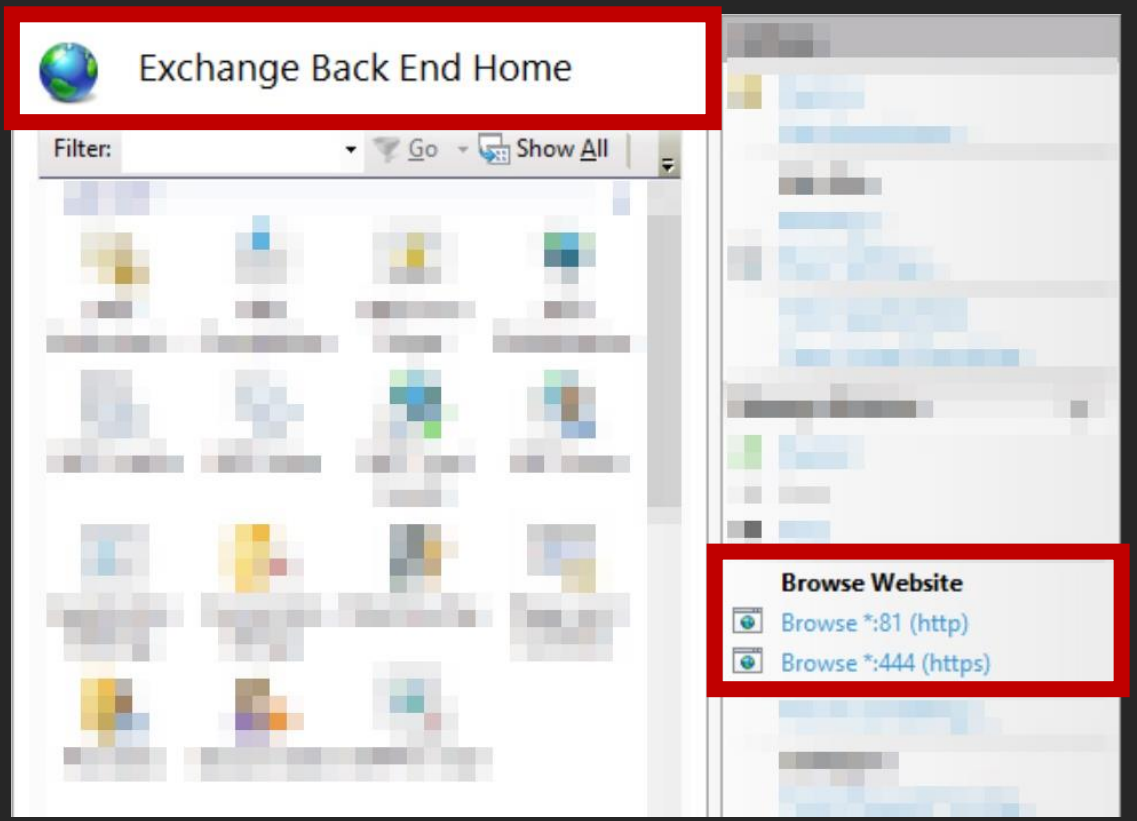

Exchange 2016 Mailbox server
Client access services
Backend services
Mobile devices
Web clients
Outlook
Load balancer
HTTP/ HTTPS
HTTP Proxy
IIS
HTTP/ HTTPS
IIS
Remote PowerShell
RPC Proxy
RPC CA
EAS, EAC, EWS, OAB, Outlook on the web
where we focus on
POP3
POP3
POP3
POP3
POP3 Backend
IMAP4
IMAP4
IMAP4
IMAP4 Backend
Mailbox database
SMTP clients (authenticated)
SMTP
Front End Transport
SMTP
Transport
Delivery queue
Telephony
1
SIP
2
redirect
UM Call Router
UM
3
SIP + RTP

Client Access Service in IIS



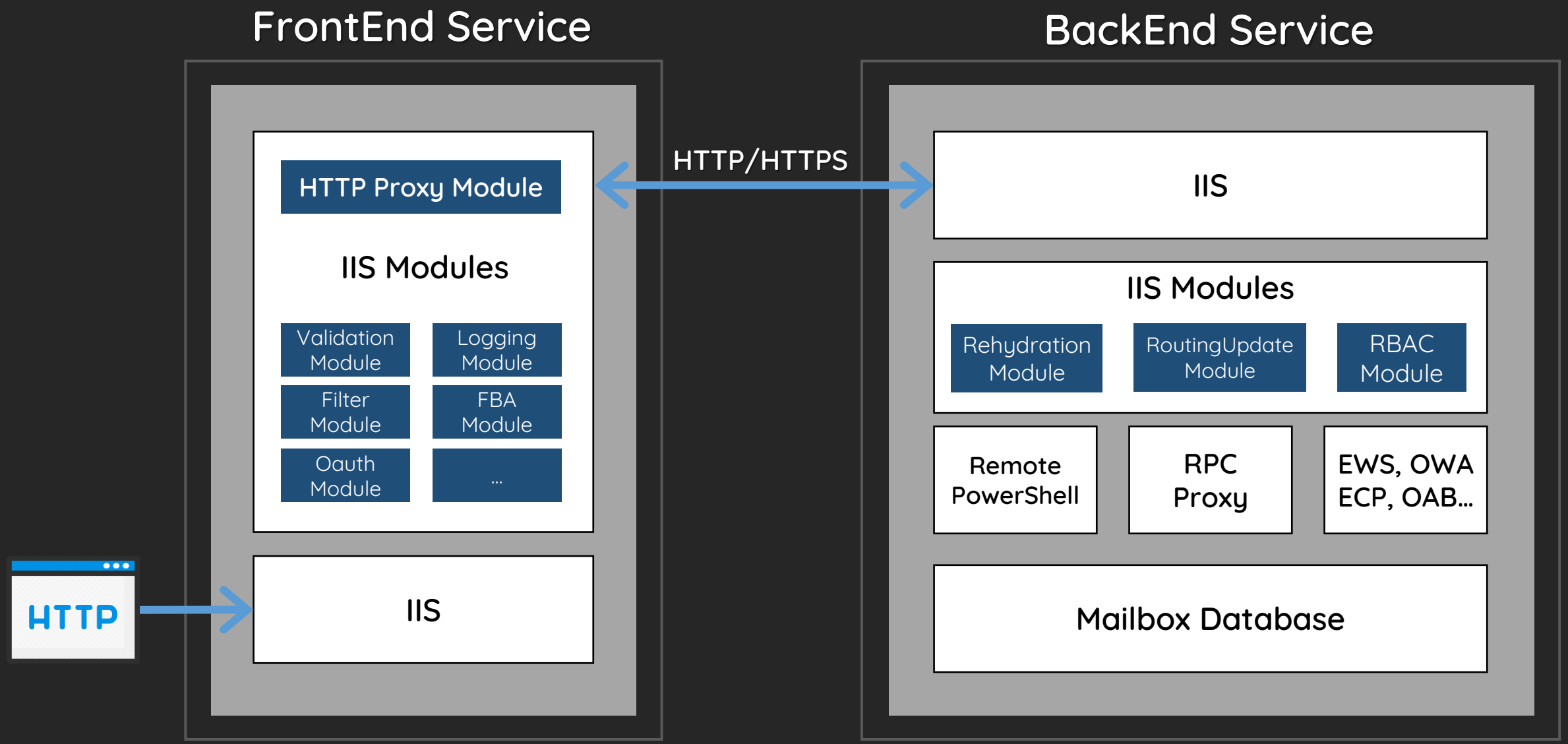
Client Access Service in IIS





Exchange Architecture

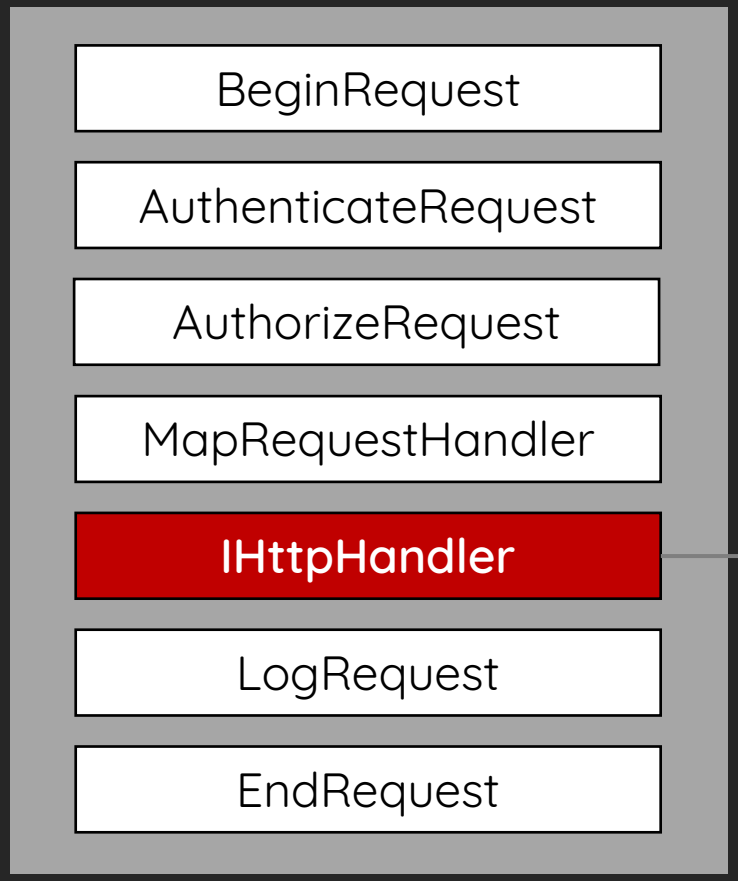
- Applications in Frontend include the ProxyModule
  - Parse incoming HTTP requests, apply protocol specified settings, and forward to the Backend
- Applications in Backend include the BackendRehydrationModule
  - Receive and populate HTTP requests from the Frontend
- Applications synchronizes the internal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


FrontEnd Service
BackEnd Service
HTTP Proxy Module
IIS Modules
Validation Module
Logging Module
Filter Module
FBA Module
Oauth Module
...
HTTP/HTTPS
IIS
HTTP
IIS
IIS Modules
Rehydration Module
RoutingUpdate Module
RBAC Module
Remote PowerShell
RPC Proxy
EWS, OWA ECP, OAB...
Mailbox Database

Our Ideas

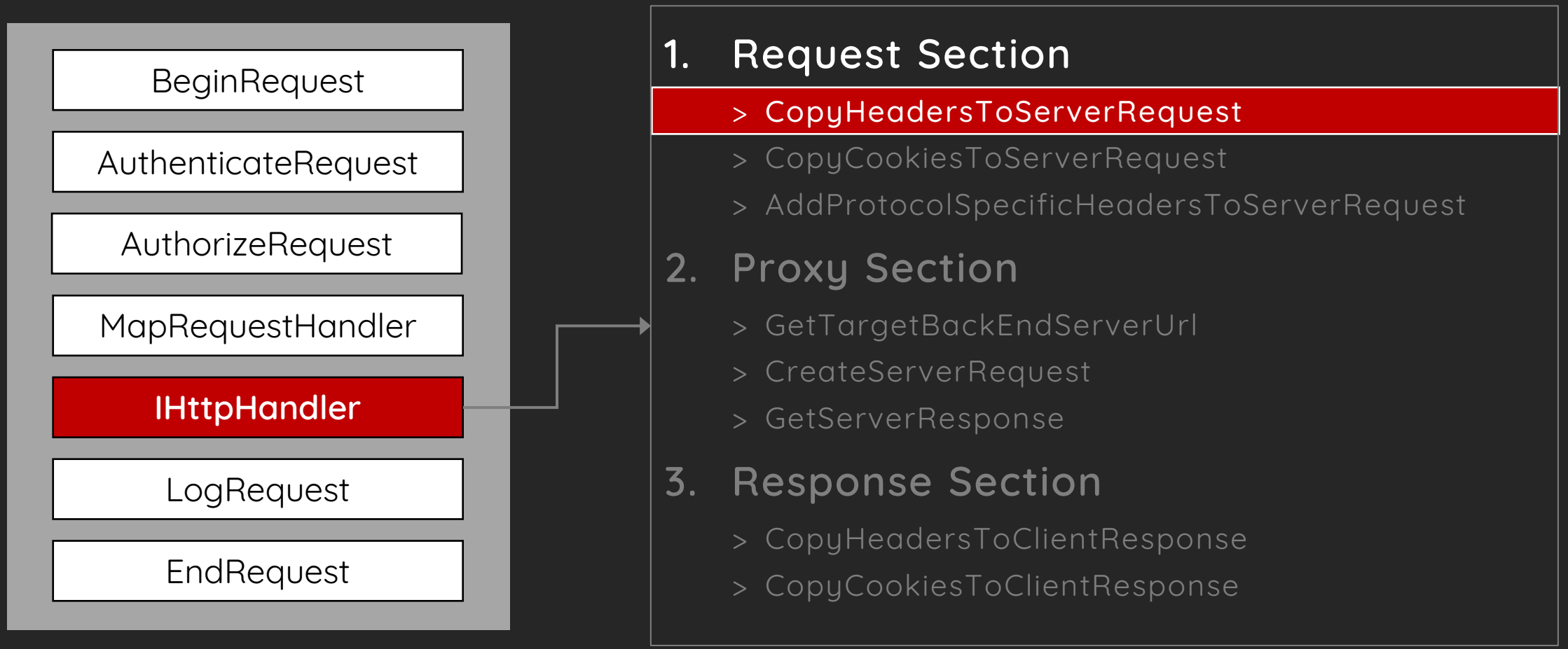
Could we access the Backend intentionally?

\ProxyRequestHandler.cs



1. Request Section
   - > CopyHeadersToServerRequest
   - > CopyCookiesToServerRequest
   - > AddProtocolSpecificHeadersToServerRequest
2. Proxy Section
   - > GetTargetBackEndServerUrl
   - > CreateServerRequest
   - > GetServerResponse
3. Response Section
   - > CopyHeadersToClientResponse
   - > CopyCookiesToClientResponse

Copy Client Headers

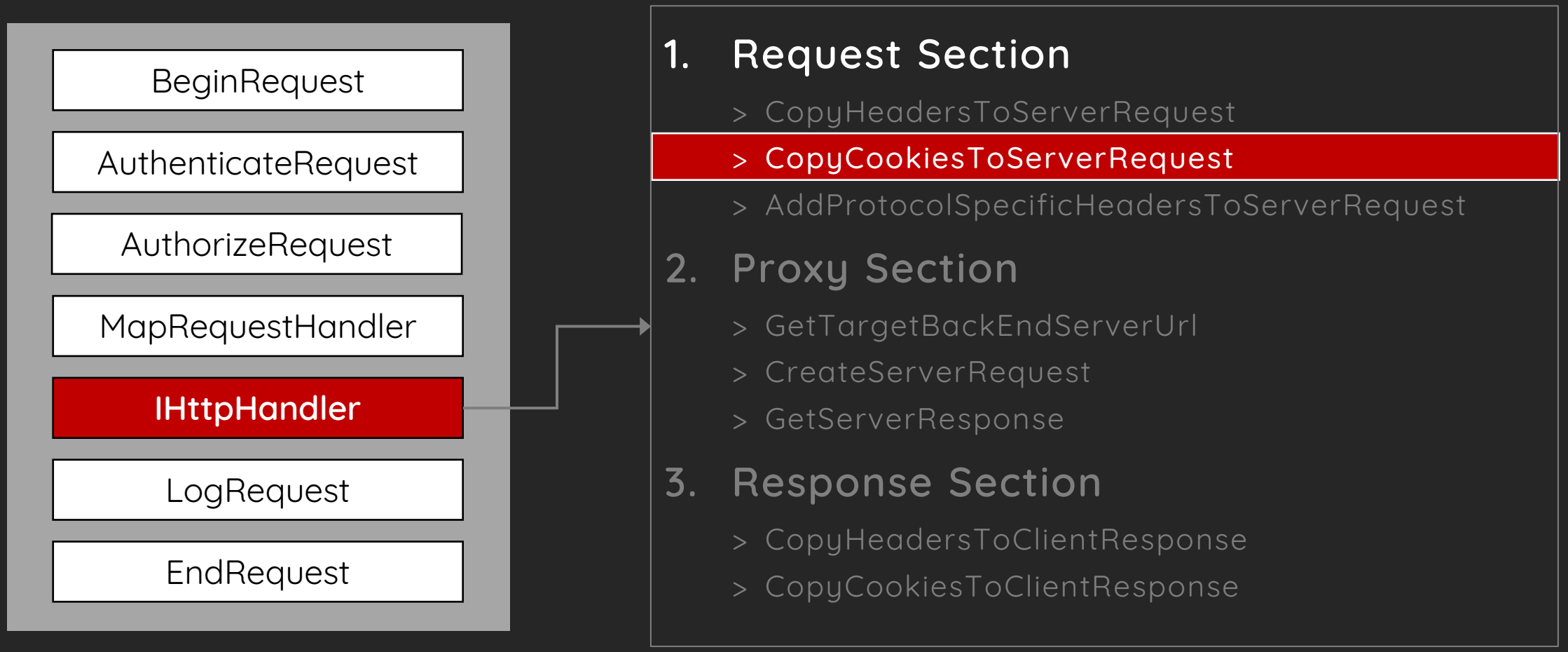


HTTP Header Blacklists

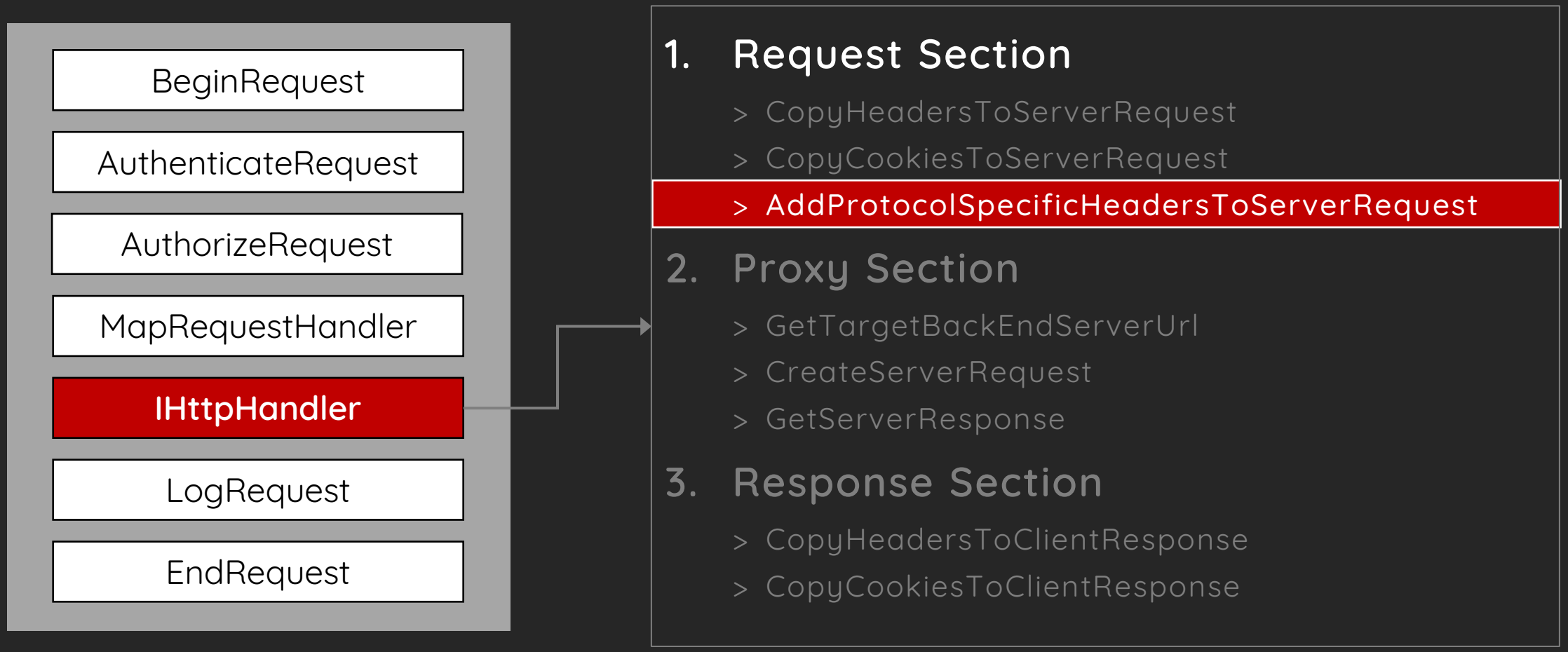
HttpProxy\ProxyRequestHandler.cs

```
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



Add Special Headers

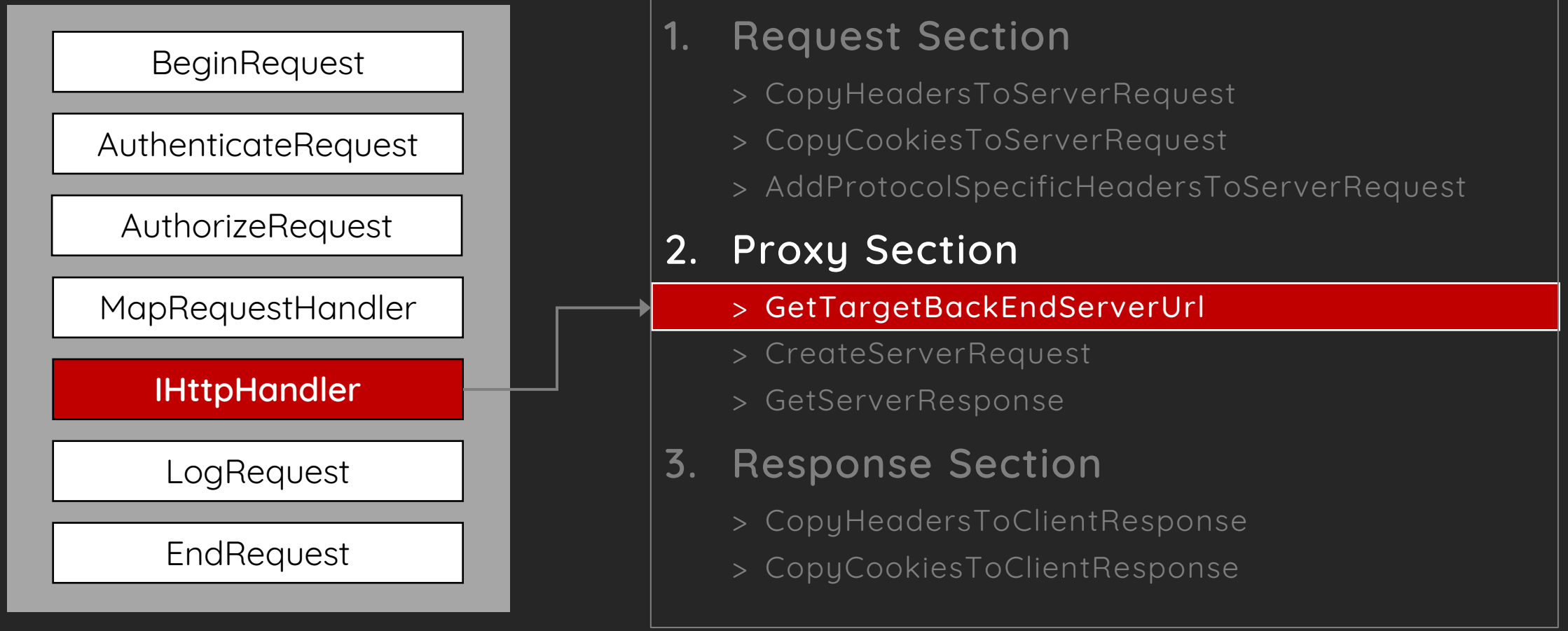


Clone User Identity

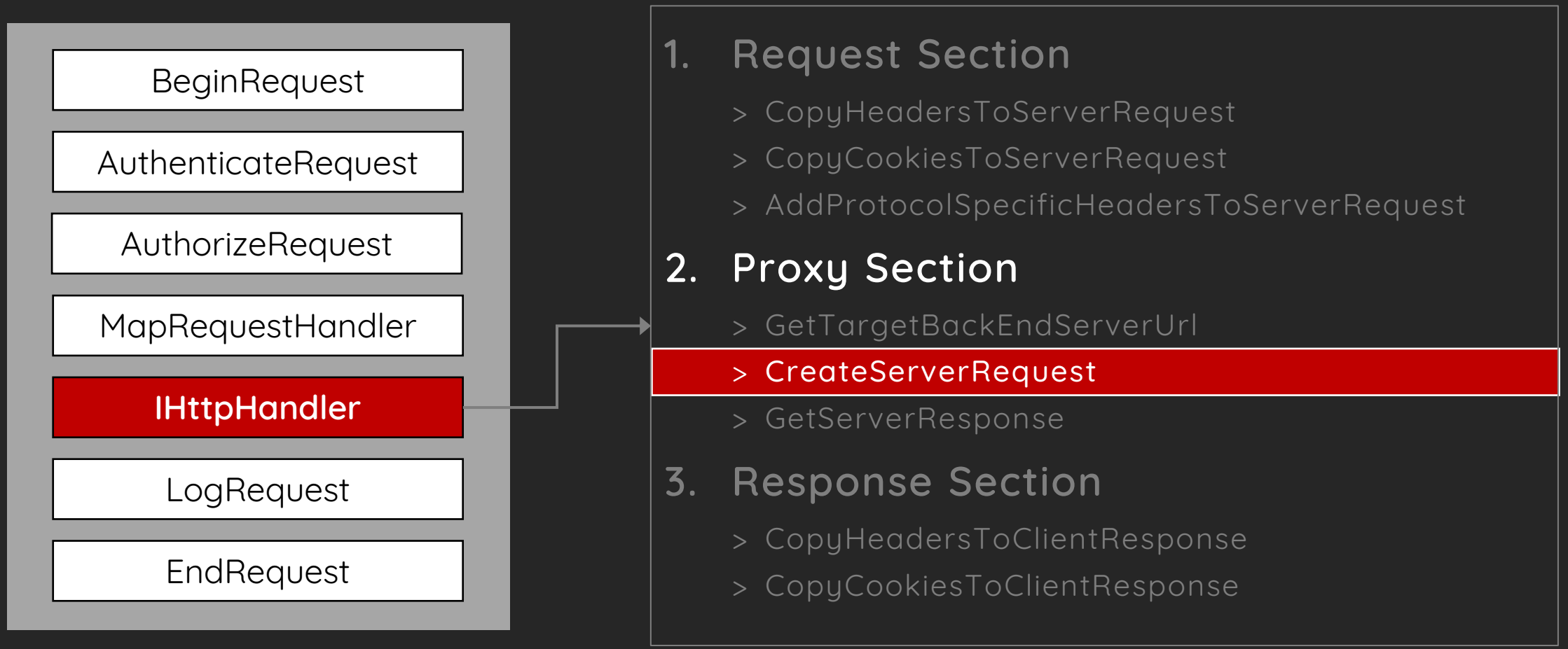
HttpProxy\ProxyRequestHandler.cs

```
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



Create New HTTP Client



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 {
    serverRequest.Headers["Authorization"] = KerberosUtilities.GenerateKerberosAuthHeader(
        serverRequest.Address.Host, this.TraceContext,
        ref this.authenticationContext, ref this.kerberosChallenge);
}
```

Generate Kerberos Ticket

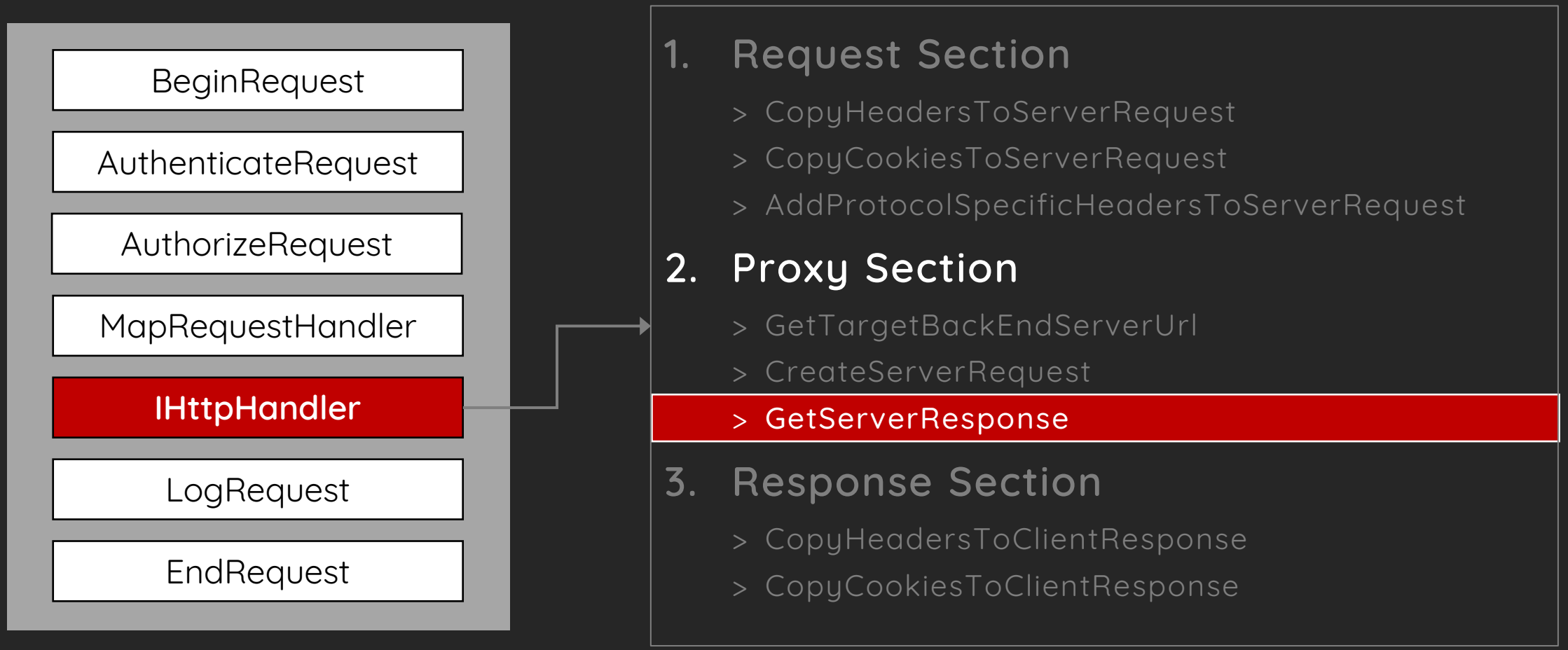
HttpProxy\KerberosUtilities.cs

```
internal static string GenerateKerberosAuthHeader(string host, int traceContext, ref
    AuthenticationContext authenticationContext, ref string kerberosChallenge) {
    // …
    authenticationContext = new AuthenticationContext();
    authenticationContext.InitializeForOutboundNegotiate(AuthenticationMechanism.Kerberos,
        "HTTP/" + host, null, null);

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

    return "Negotiate " + Encoding.ASCII.GetString(bytes);
}
```

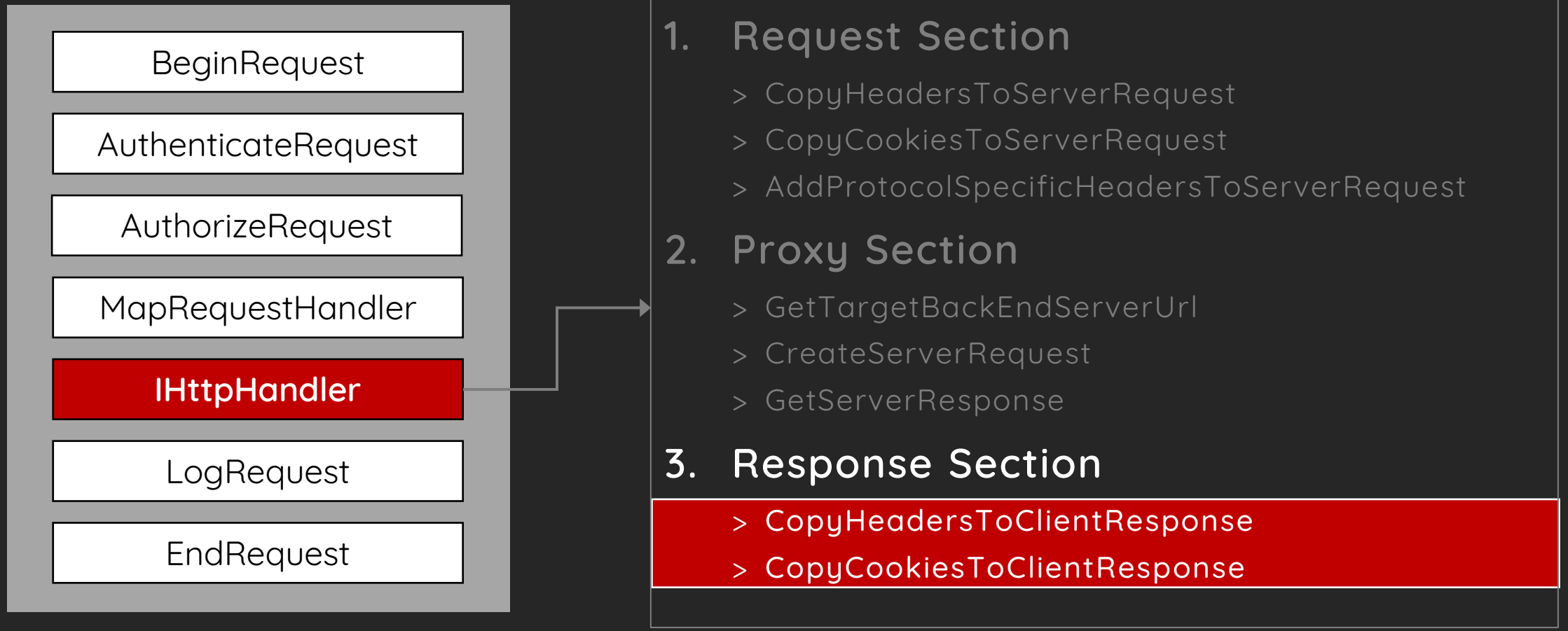
The Actual Request Sent to Backend

```
1  GET /owa/ HTTP/1.1
[redacted]
5  Authorization: Negotiate
   YIIGbAYJKoZIhvcSAQICAQBuggZbMIIGV6ADAgEFoQMCAQ6iBwMFACAAAACjggSSYYIEjjCCBIqgAwIBBaEOGwxP
[redacted]
11 X-CommonAccessToken:
   VgEAVAdXaW5kb3dzQwBBBUJhc2ljTBRPUkFOR0VcQWRtaW5pc3RyYXRvclUsUy0xLTUtMjEtNTQ3MjI2NjU2Q2Nj
[redacted]
```

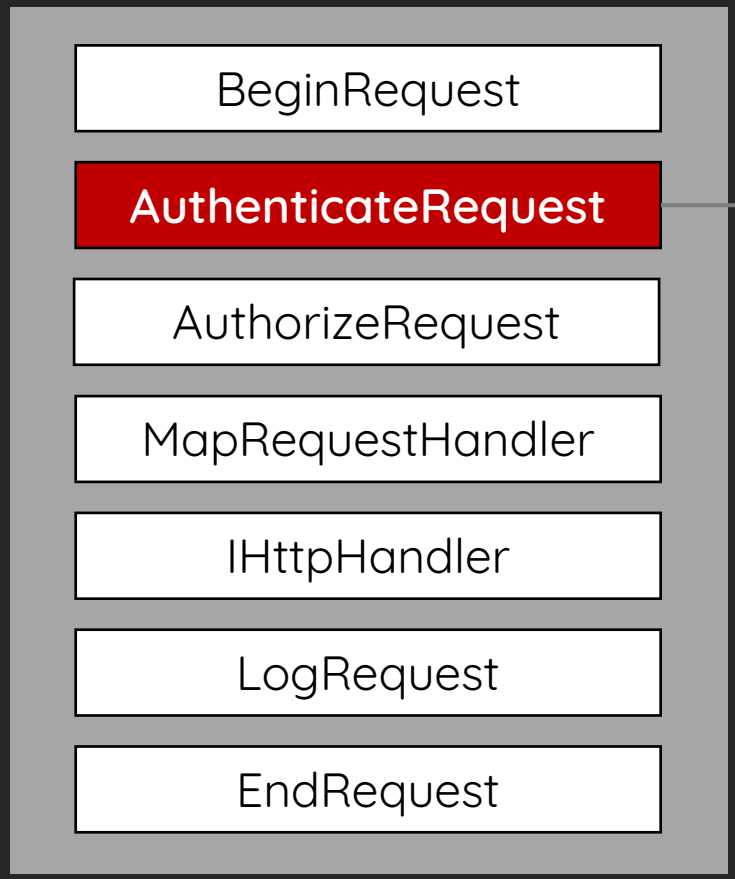
Get Backend Response



Copy Response to Client



Backend Rehydration Module



\BackendRehydrationModule.cs

```
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))
        // …
}
```

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);
    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;
```

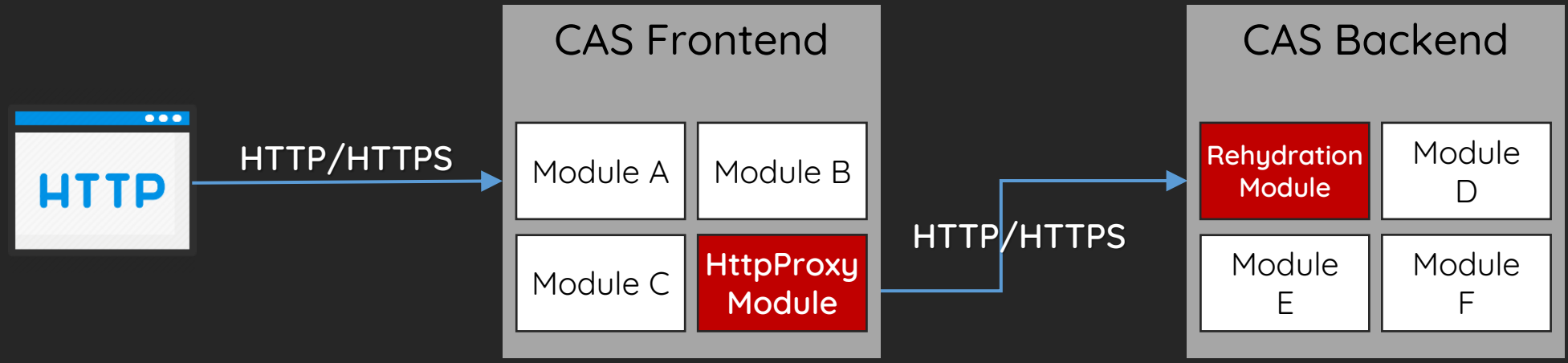
Check AD Extended Rights

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

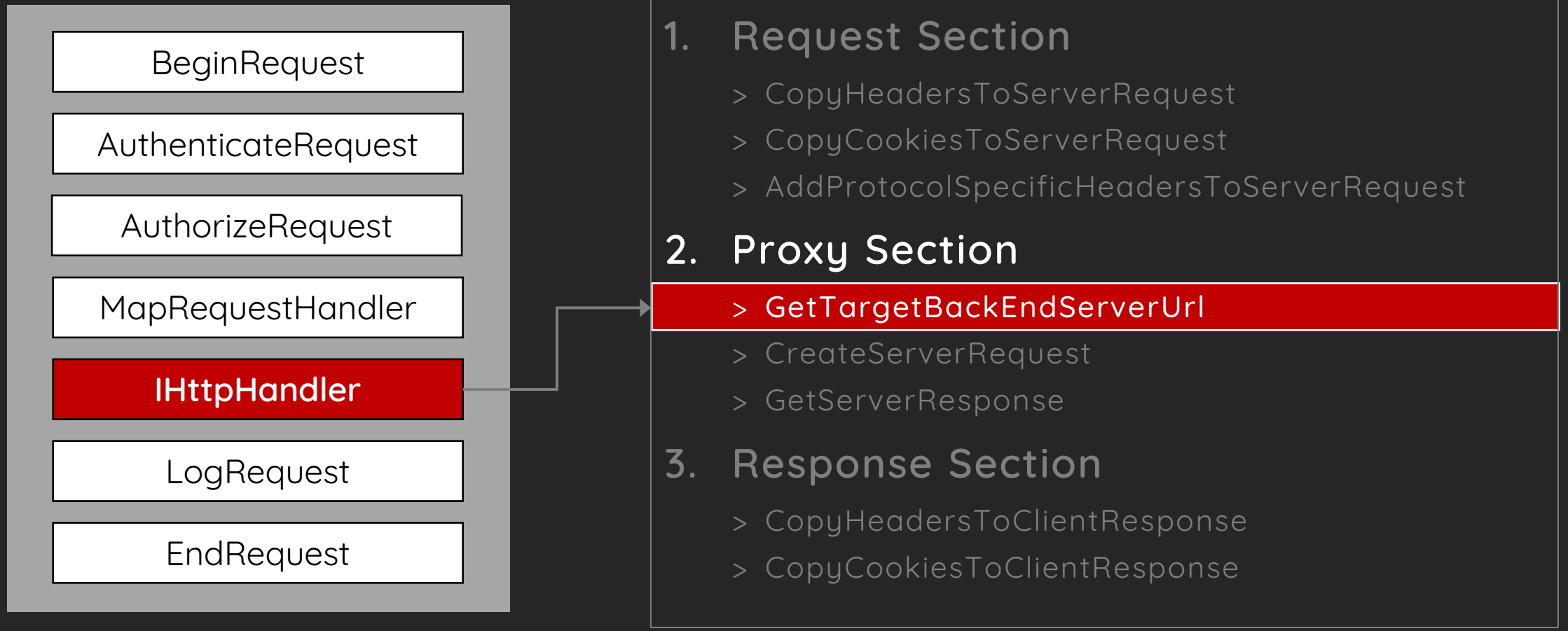


Let's Hack the Planet

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?



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);
}
```

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

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

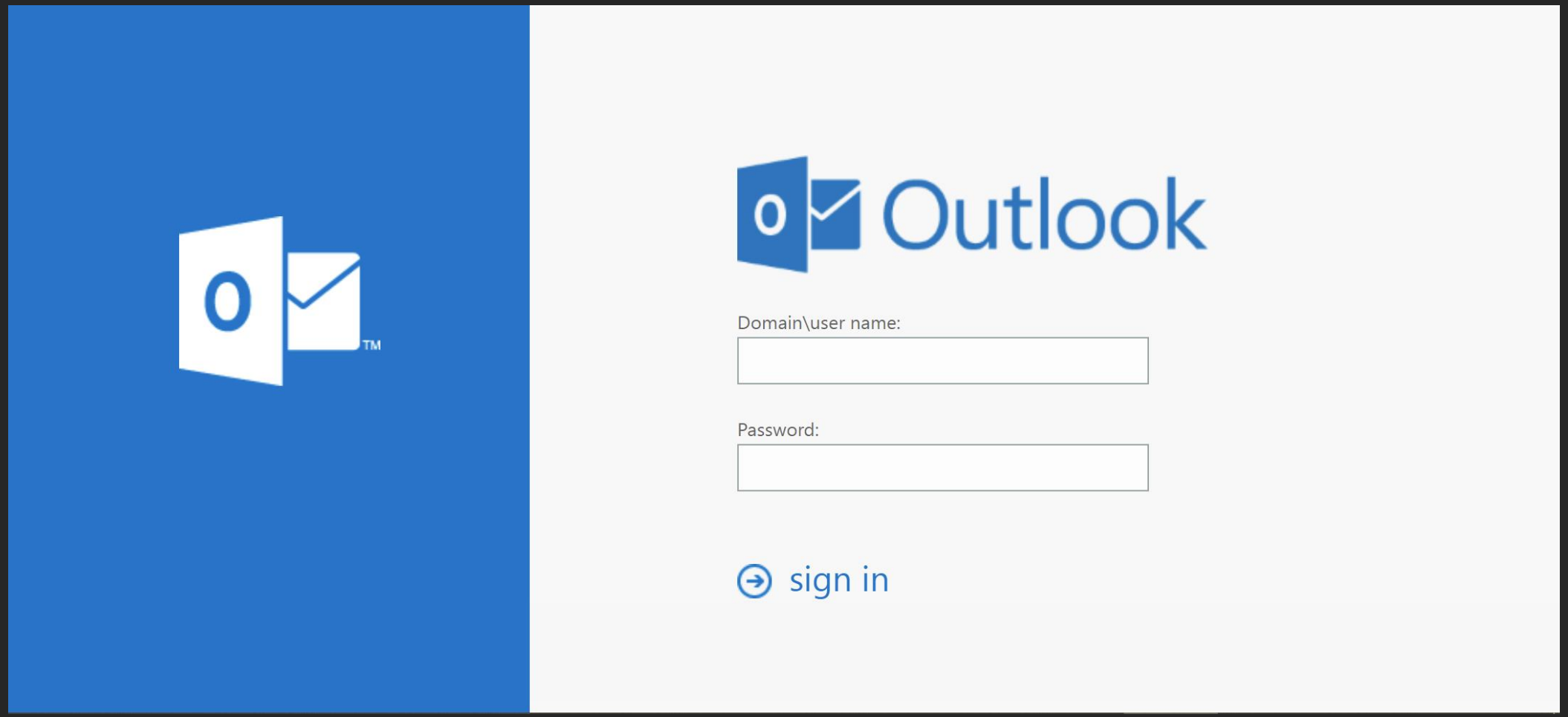
Demo

https://youtu.be/SvjGMo9aMwE

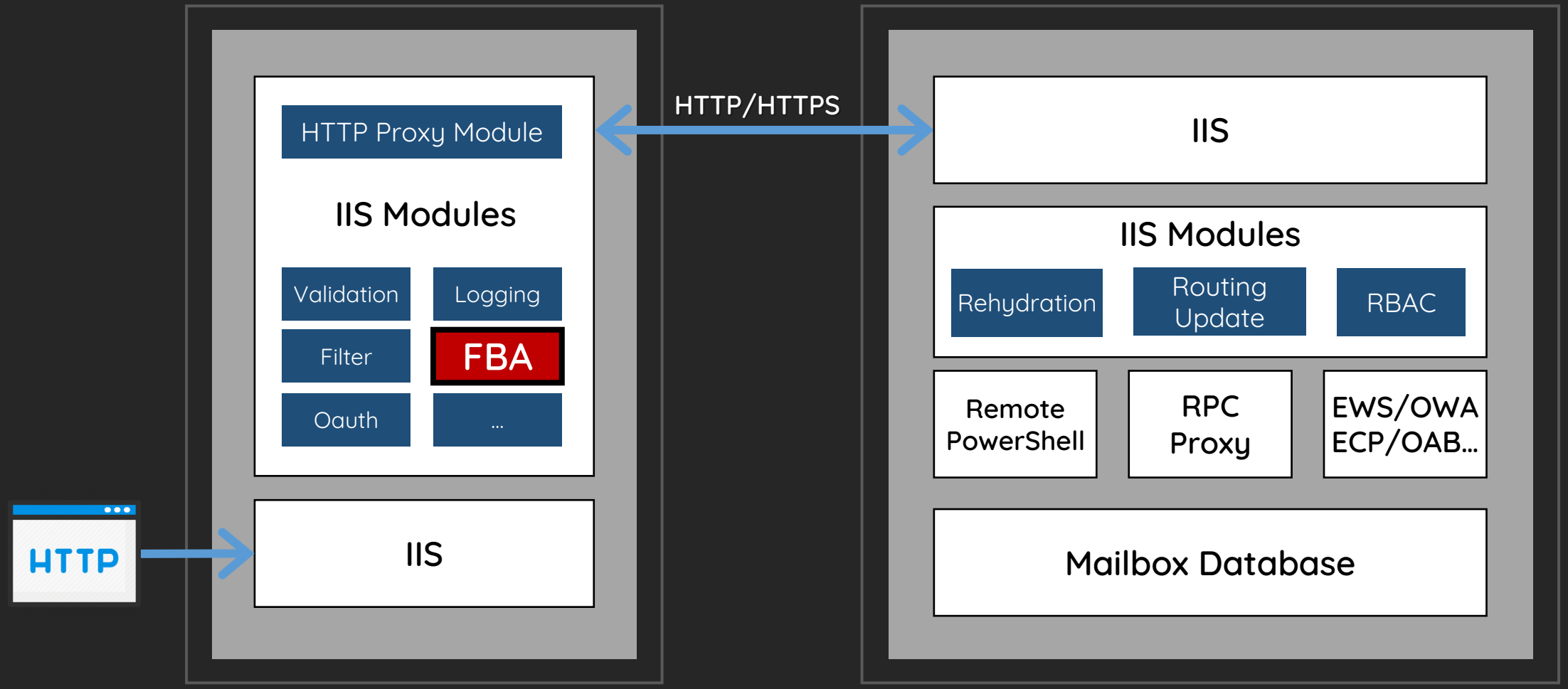
ProxyOracle

- 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

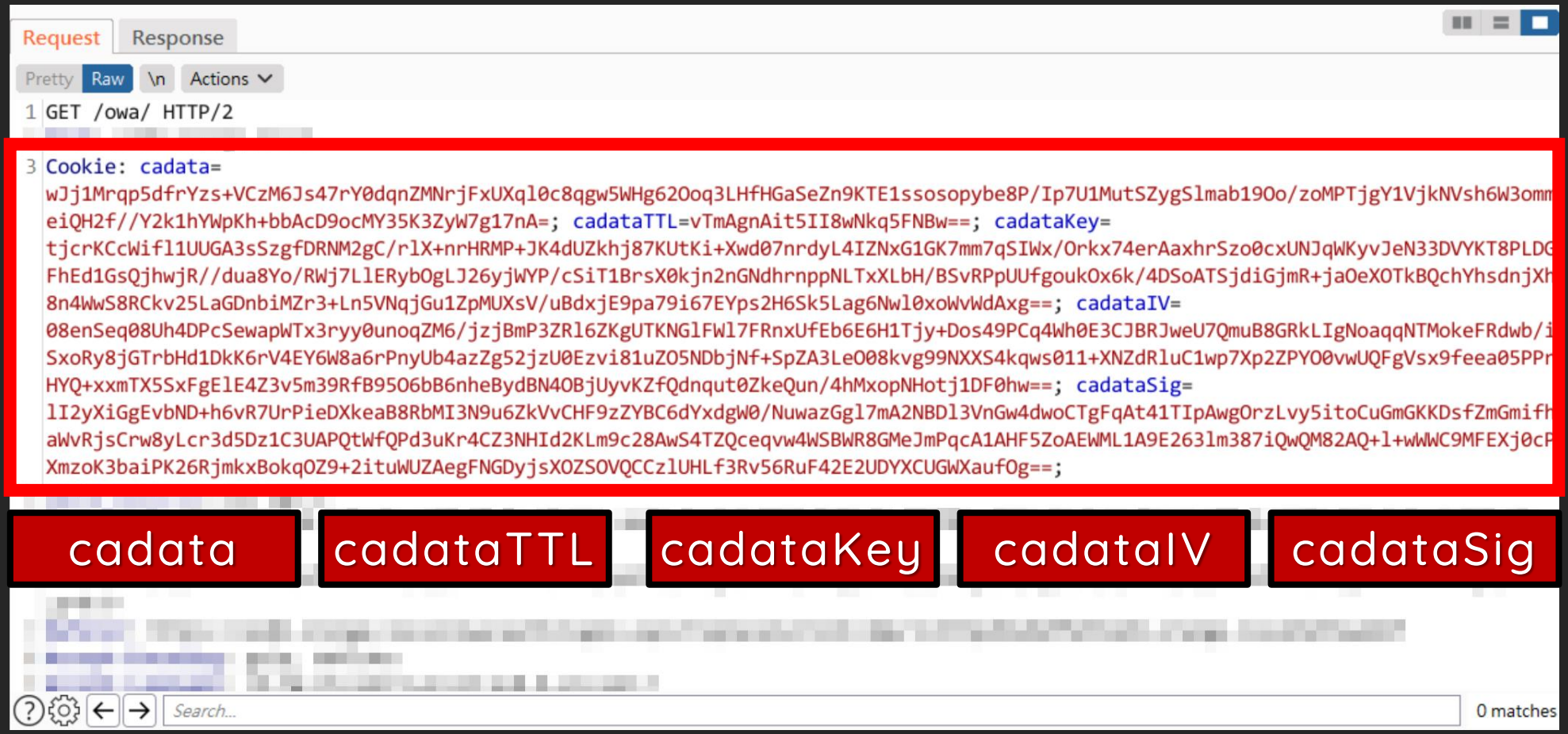
How Users Log-in OWA/ECP?



Form-Based Authentication



How FBA Cookies Looks Like



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)
```

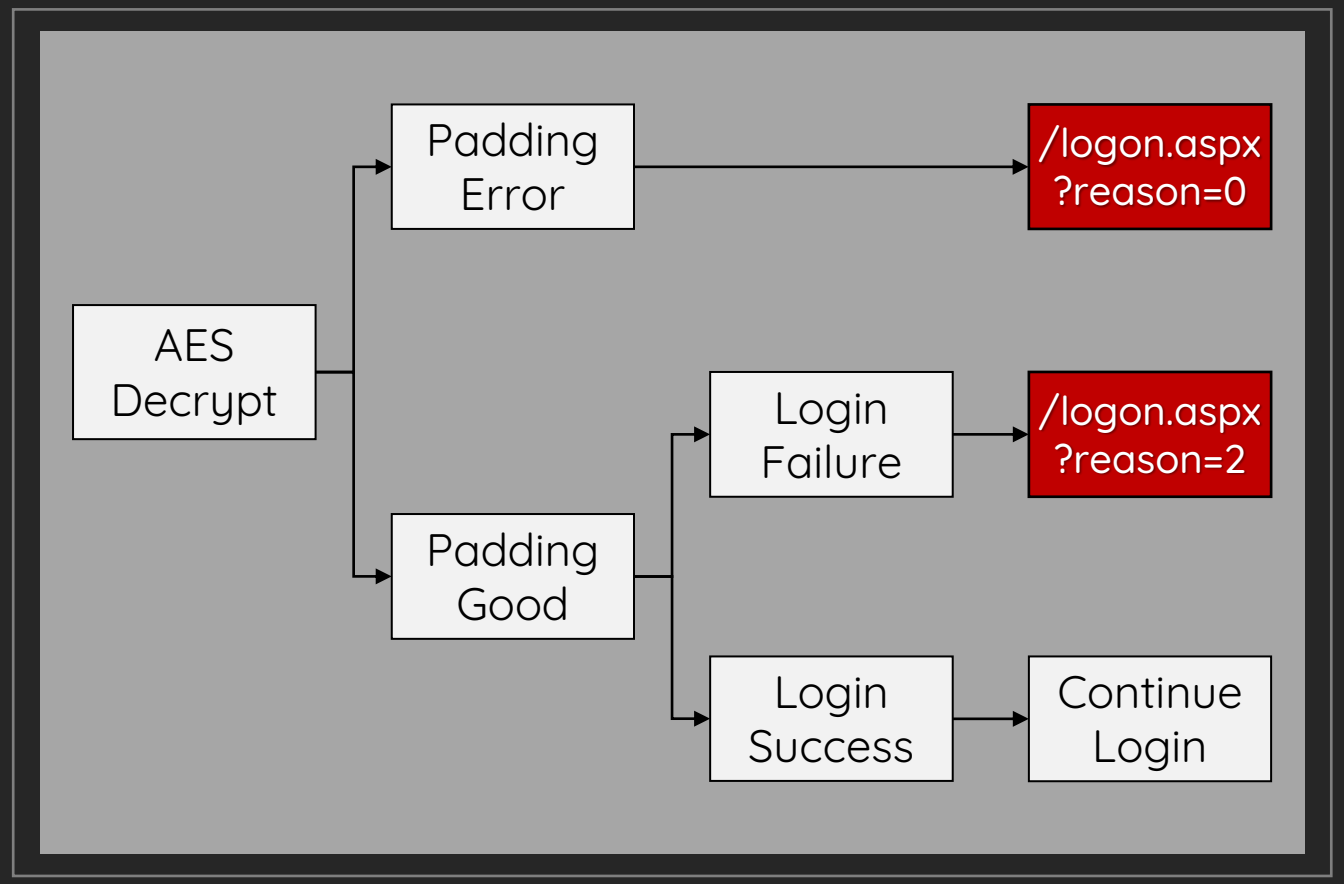

Padding Oracle
Still WORKS

FbaModule Encryption Logic

HttpProxy\FbaModule.cs

```
private void 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;
    }
  }
}
```

The Oracle



\FbaModule.cs

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

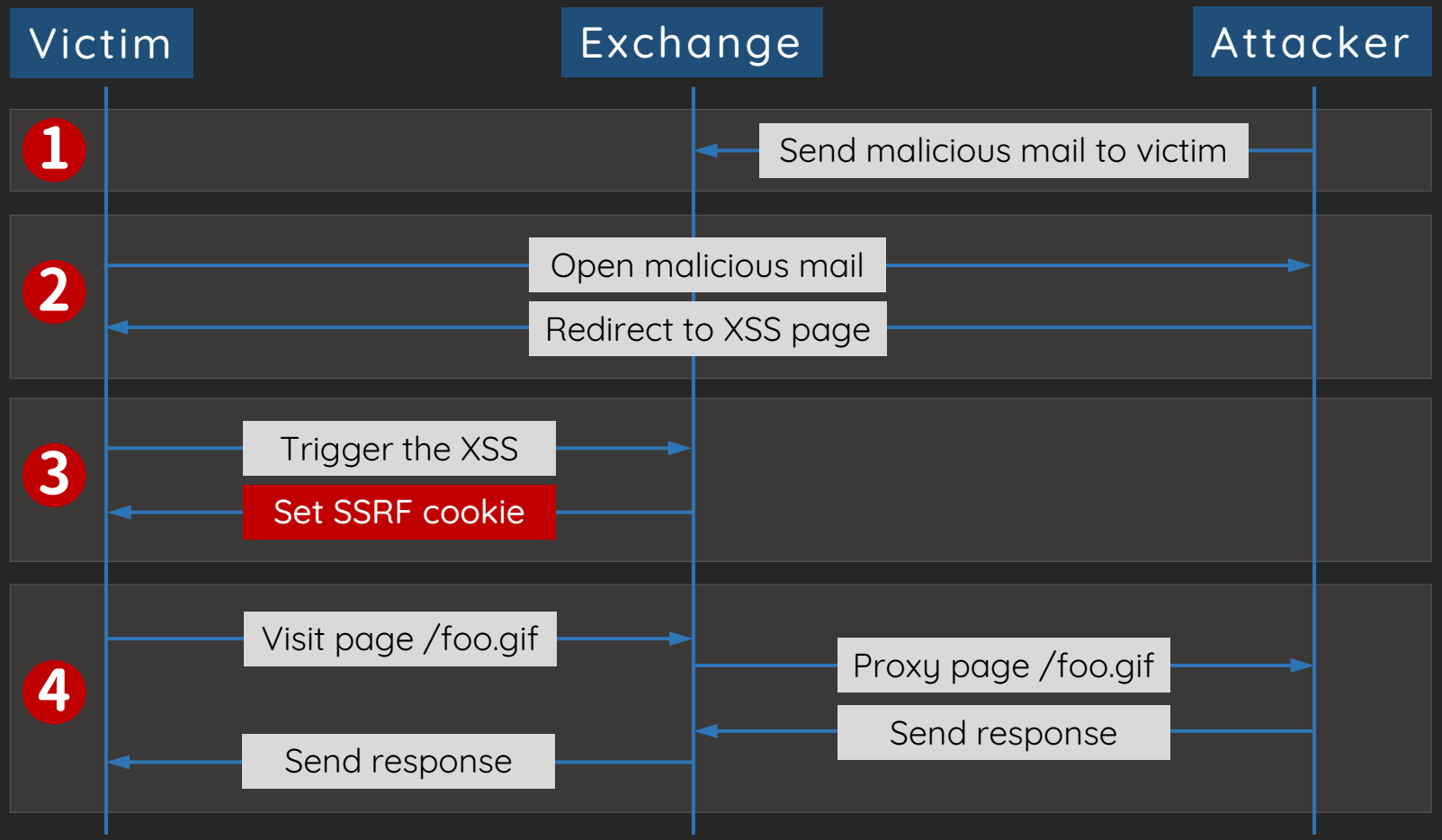
We can decrypt the cookies now

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



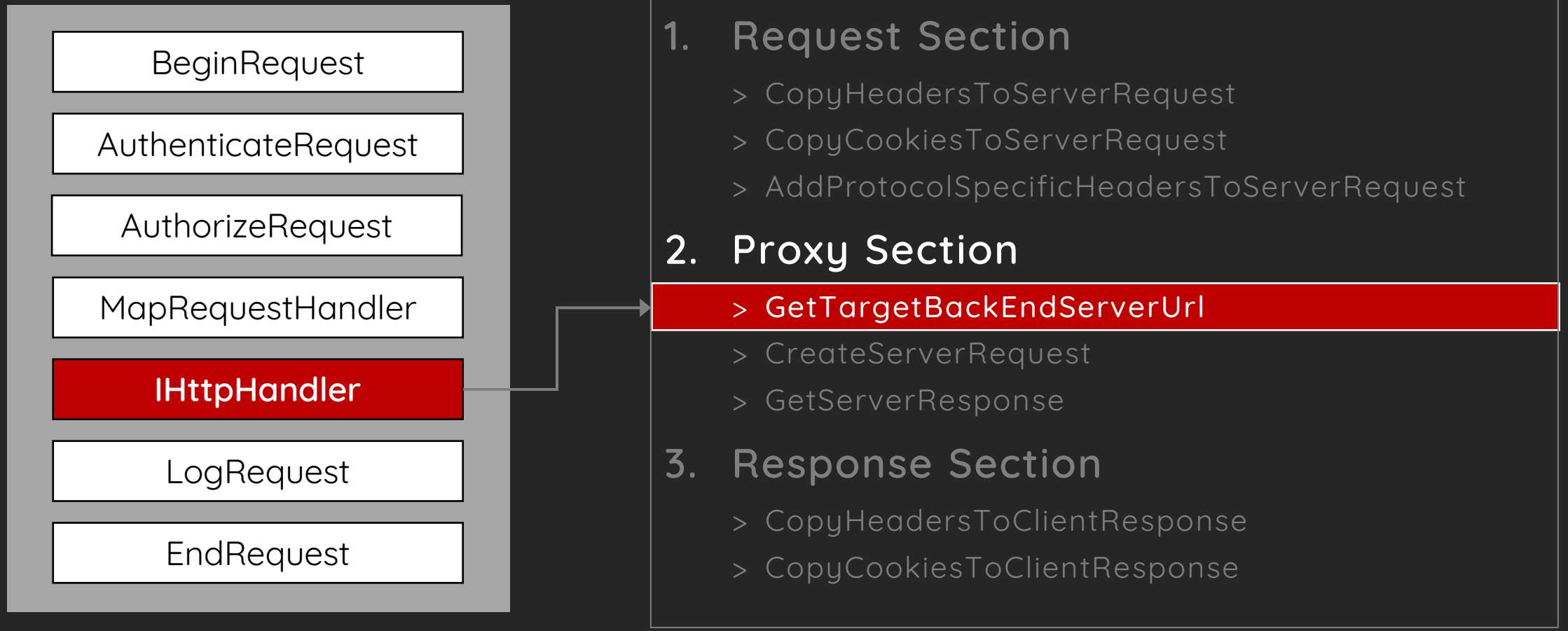
Demo

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?



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

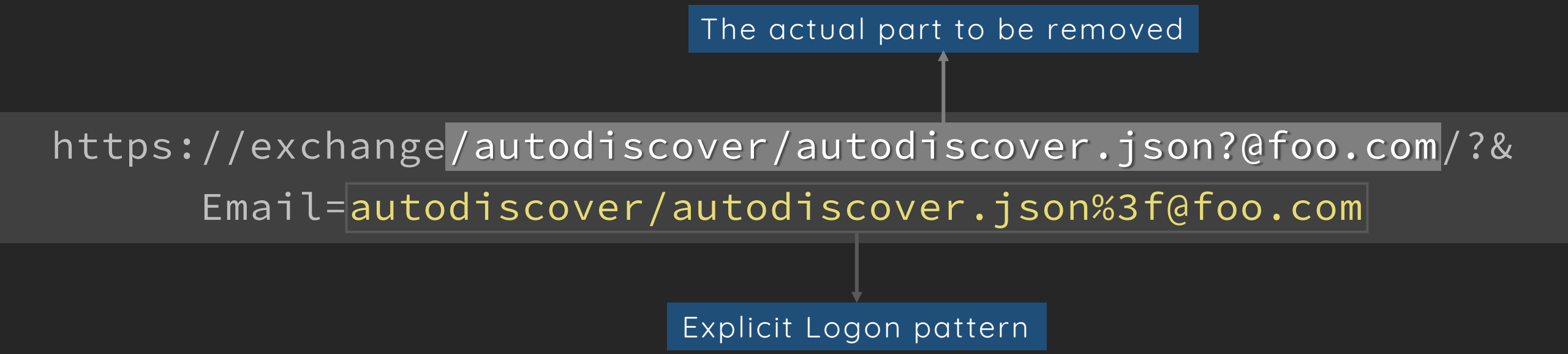
HttpProxy\EwsAutodiscoverProxyRequestHandler.cs

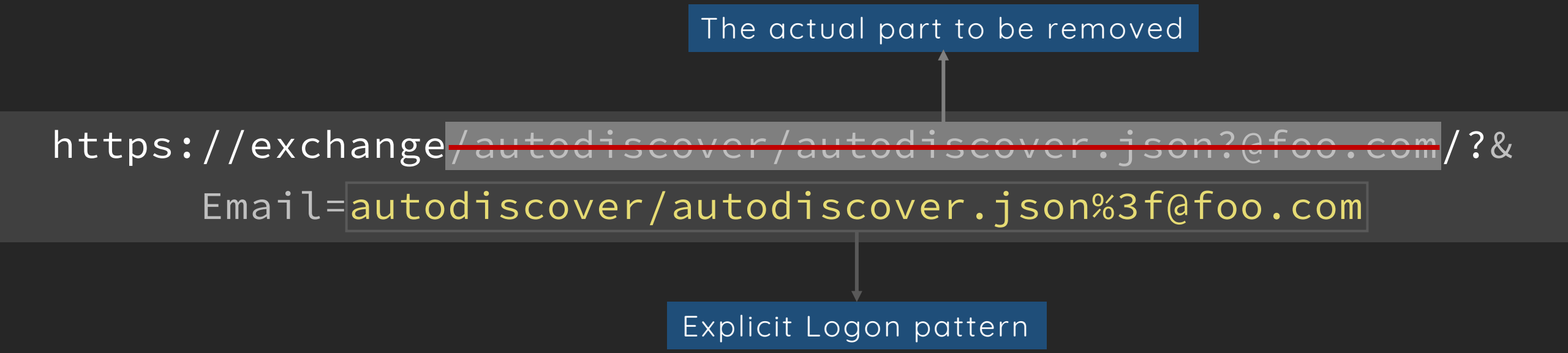
```
protected override AnchorMailbox ResolveAnchorMailbox() {
    if (RequestPathParser.IsAutodiscoverV2PreviewRequest(base.ClientRequest.Url.AbsolutePath))
2       text = base.ClientRequest.Params["Email"];
    // …
    this.isExplicitLogonRequest = true;
    this.explicitLogonAddress = text;
}

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 = 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)
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
```

Arbitrary Backend Access Again!

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

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

DEVCORE

Thanks!

 orange_8361

 orange@chroot.org

 https://blog.orange.tw