Reproducing The ProxyShell Pwn2Own Exploit

ZDI-21-822	ZDI-CAN-13614	Microsoft	CVE-2021-34523	2021-07-19
(Pwn2Own) Microsoft Exchange Server PowerShell Improper Authentication Remote Code Execution Vulnerability				
ZDI-21-821	ZDI-CAN-13611	Microsoft	CVE-2021-34473	2021-07-19
(Pwn2Own) Microsoft Exchange Server Autodiscover Server Side Request Forgery Authentication Bypass Vulnerability				
ZDI-21-819	ZDI-CAN-13588	Microsoft	CVE-2021-31207	2021-07-19
(Pwn2Own) Microsoft Exchange Server Arbitrary File Write Remote Code Execution Vulnerability				

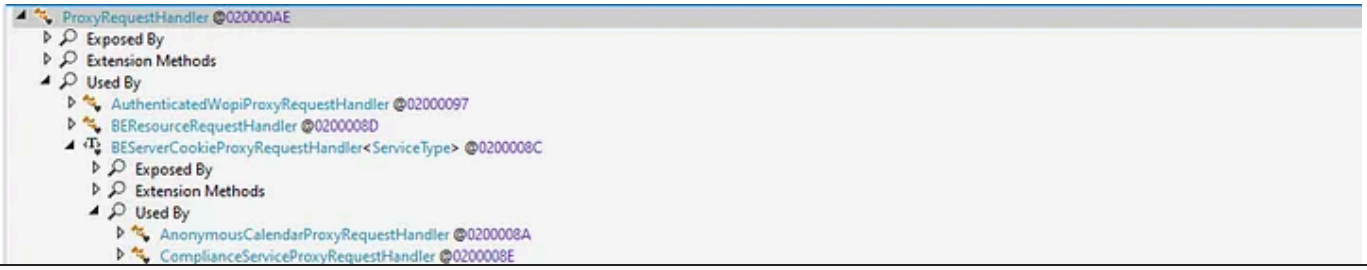
https://www.zerodayinitiative.com/advisories/published/

With these advisories, we can imagine that this chain maybe similar to

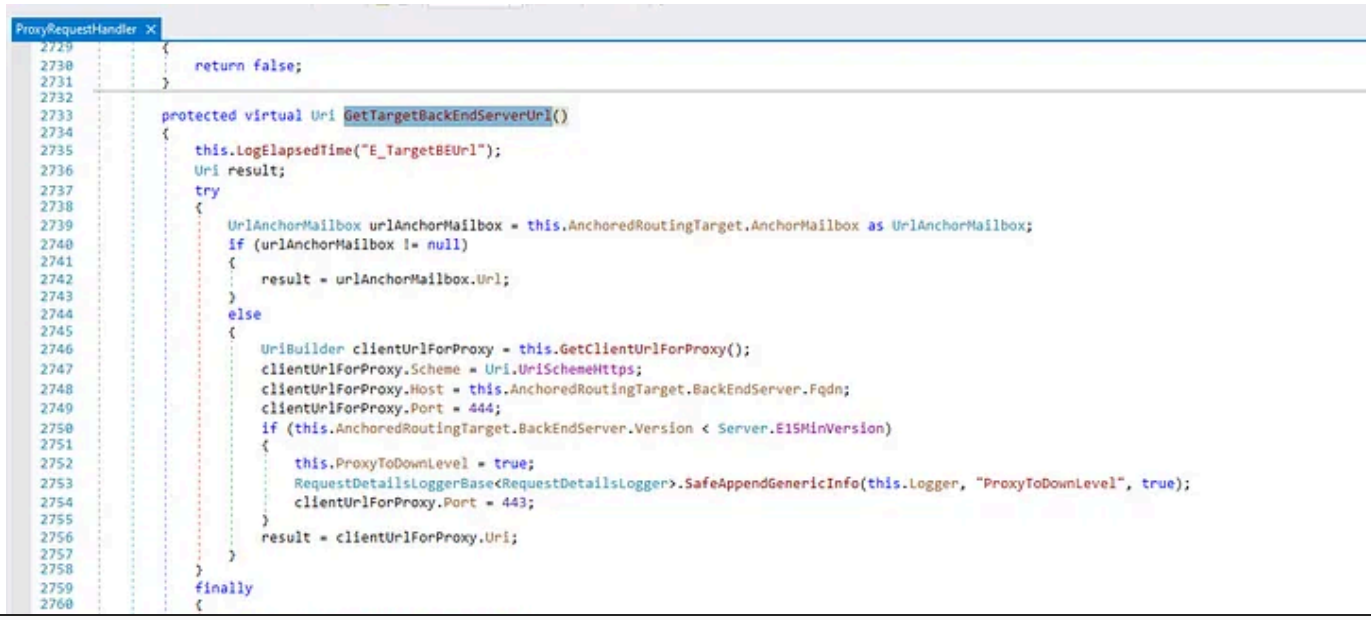
ProxyLogon entry

From ProxyLogon, we know that we can set *AnchoredRoutingTarget* variable from "*X-BEResource*", then Exchange when calculate the target backend URL

With the information from ZDI, pre-auth SSRF come from autodiscover service so we find some class which implement "*Microsoft.Exchange.HttpProxy.ProxyRequestHandler*" and allow for unauthenticated



```
ProxyModule
400     private IHttpHandler SelectHandlerForUnauthenticatedRequest(HttpContext httpContext)
401     {
402         IHttpHandler result;
403         try
404         {
405             if (HttpProxySettings.NeedHandleAsAuthenticatedRequest(httpContext.Request.Headers, httpContext.Request.Cookies, h
406             {
407                 result = this.SelectHandlerForAuthenticatedRequest(httpContext);
408             }
409             else
410             {
411                 UriBuilder uriBuilder = new UriBuilder(httpContext.Request.Url);
412                 string explicitLogonUser = null;
413                 if (Microsoft.Exchange.Clients.Common.UrlUtilities.TryGetExplicitLogonUser(httpContext.Request, out explicitLo
414                 {
415                     uriBuilder.Path = Microsoft.Exchange.Clients.Common.UrlUtilities.GetPathWithExplictLogonHint(httpContext.R
416                 }
417                 IHttpHandler httpHandler = null;
418                 if (HttpProxyGlobals.ProtocolType == ProtocolType.Autodiscover)
419                 {
420                     httpHandler = new AutodiscoverProxyRequestHandler();
421                 }
422                 else if (HttpProxyGlobals.ProtocolType == ProtocolType.Ews)
423                 {
424                     if (RequestPathParser.IsEwsUnauthenticatedRequestProxyHandlerAllowed(httpContext.Request))
425                     {
426                         httpHandler = new EwsProxyRequestHandler();
427                     }
```



```
ProxyRequestHandler
2729        {
2730            return false;
2731        }
2732
2733        protected virtual Uri GetTargetBackEndServerUrl()
2734        {
2735            this.LogElapsedTime("E_TargetBEUrl");
2736            Uri result;
2737            try
2738            {
2739                UrlAnchorMailbox urlAnchorMailbox = this.AnchoredRoutingTarget.AnchorMailbox as UrlAnchorMailbox;
2740                if (urlAnchorMailbox != null)
2741                {
2742                    result = urlAnchorMailbox.Url;
2743                }
2744                else
2745                {
2746                    UriBuilder clientUrlForProxy = this.GetClientUrlForProxy();
2747                    clientUrlForProxy.Scheme = Uri.UriSchemeHttps;
2748                    clientUrlForProxy.Host = this.AnchoredRoutingTarget.BackEndServer.Fqdn;
2749                    clientUrlForProxy.Port = 444;
2750                    if (this.AnchoredRoutingTarget.BackEndServer.Version < Server.E15MinVersion)
2751                    {
2752                        this.ProxyToDownLevel = true;
2753                        RequestDetailsLoggerBase<RequestDetailsLogger>.SafeAppendGenericInfo(this.Logger, "ProxyToDownLevel", true);
2754                        clientUrlForProxy.Port = 443;
2755                    }
2756                    result = clientUrlForProxy.Uri;
2757                }
2758            }
2759            finally
2760            {
```

checking FQDN after ProxyLogon

This is what we need to looking for, if our request IsAutodiscoverV2Request(), it will remove the “explicitLogonAddress” from URI and rebuild the URI.

Focus that this is AbsoluteUri not AbsolutePath

Because *IsAutodiscoverV2PreviewRequest()* check *EndsWith("/autodiscover.json")* and the ***path*** variable is ***AbsoluteUri*** we can make it return ***False*** like

why we can reach some other endpoint without any authentication

The next bug we need to looking for is SSRF into “/powershell endpoint”

We don’t have permission on this endpoint :(

get CommonAccessToken (from Exchange SSRF) there will be an exception and we cannot go through.

WindowsAccessToken

```
A + authenType + L + logonName + U + user SUID
read group SUIDs
G + groupLength + SUIDs of group
```

At this point, I setup socat, change internal Exchange port from 444 to 4443, using socat listening on port 444, then redirect to BurpSuite port (8080) and finally foward into 4443. With this setup, we can capture a “sample”

2. Leaking user SUID via “/mapi/emsmdb”

Now we got SUID, but how about group SUIDs? Check out this cmdlet

```
Get-Group | Format-List Identity,Sid
```

Now, we can craft an admin privilege CommonAccessToken via “X-Rps-CAT” parameter.

We can confirm it again, because the patch only allow some specific extension

But how can we control the data in the mailbox and make it into shell after

Looking back into MS documents, Jang found this one which help us successfully write shell

For myself, I use pypsrp then collect the data while it processing and plug it into our SSRF. To understand more about WinRM you can check this awesome blog

Or you can do the same with Orange's way, implement his own proxy to communicate with WinRM

Our demonstration:

https://www.youtube.com/watch?v=LbIYPFrltdA

Proxy Pwn2own Exchange Rce Reproduce