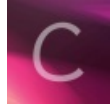


通过DNS重绑定实施SSRF攻击

翻译

NOSEC2019 于 2019-11-18 17:48:56 发布 940 收藏 6

分类专栏: [安全](#) 文章标签: [dns](#) [ssrf](#)



[安全](#) 专栏收录该内容

178 篇文章 2 订阅

订阅专栏



什么是DNS Rebinding?

假设你是一台公网服务器，无数人会给你发来URL，要求你进行内容加载。那么在这海量请求中，会有不怀好意的人发送内网URL，要求你进行加载，这其中的风险难以言喻。


因此，一些聪明的开发人员为了防止这种情况，就发明了各种各样的代码。

`ip_banlist`，你指定的恶意IP列表

`domain`，你指定的域

`getHostname`，这其实是一个函数，它会把一个URL解析为实际的IP地址

```
1. import requests
2. from core_funcs import getHostname
3. from banlists import ip_banlist
4.
5. def secureFetch(domain):
6.     if getHostname(domain) not in ip_banlist:
7.         r = requests.get(domain)
8.         return r.text
```



以上就是DNS重绑定所要解决的问题！

先让我们逐行分析代码。

假设这个函数的参数设置为 `domain='http://wtf.geleta.eu'` 以及 `ip_banlist = ['169.254.169.254', '127.0.0.1']`

在这种情况下，它将对 `wtf.geleta.eu` 进行DNS查询，并得到 `12.34.56.78` 这个结果，它不存在 `ip_banlist` 上，让我们继续！！

此时再神奇的将 `http://wtf.geleta.eu` 的DNS记录改为 `127.0.0.1`

最后服务器会向 `http://wtf.geleta.eu` 发出请求。此时已没有什么能阻拦我向127.0.0.1发出请求了！

以上就是最原始的整体流程。

我们需要做什么

由于我们无法在程序运行时以毫秒为单位手动更改dns记录，因此需要配置一个自定义DNS服务器，并设定好某些域名的解析IP，再将TTL设置为0，这样后端就不会有缓存

一些用来配置域的“接口”——它应该解析什么，解析多少次之类的

足够的运气

而在我使用 `https://lock.cmpxchg8b.com/rebinder.html` 时，发现它的功能并不能满足我的需求。它只能在2个IP之间随机变化，我往往需要发送多个请求才能得到我想要的结果。为此我特意制造了属于自己的工具。

实际漏洞

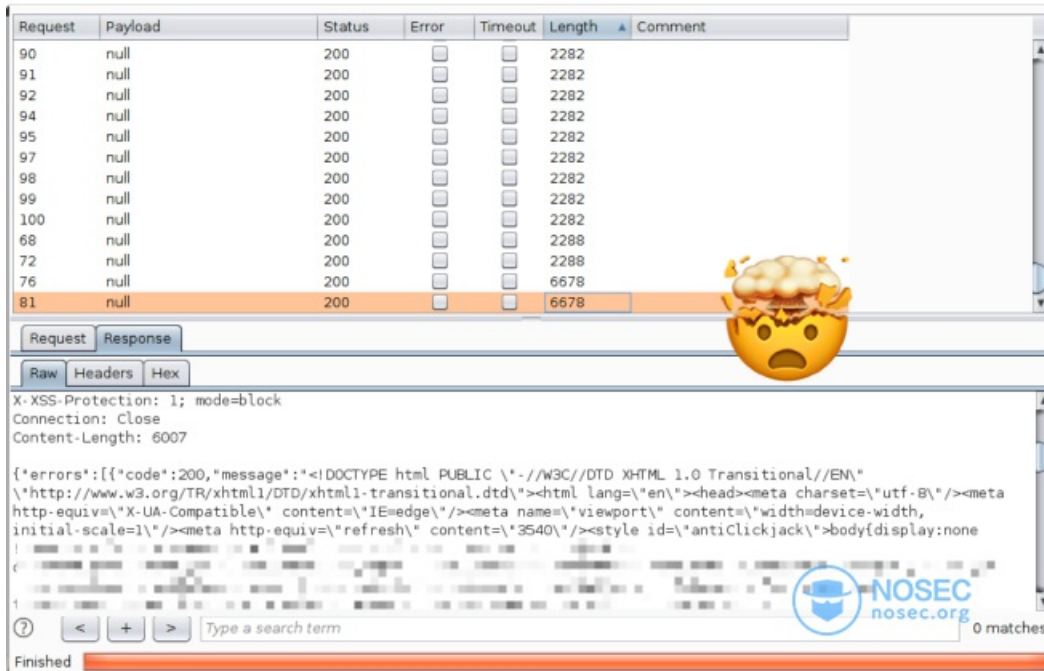
在某次测试时，我偶然发现了一个服务，可以主动发送JSON请求（可以自定义报头等字段）。

当我将url设置为某些内网ip时（比如127.0.0.1），它就会返回如下结果： `The request was blocked`。

我试了几个普通绕过方法后，发现并没有效果。于是我便想起我和我的朋友Jan Masarik所参与的一次CTF竞赛中遇到的一道和DNS重绑定有关的题目。详细信息如下：<https://ctftime.org/writeup/13005>。

这个CTF题是由ELB创作的，我非常感谢他带给我的灵感！

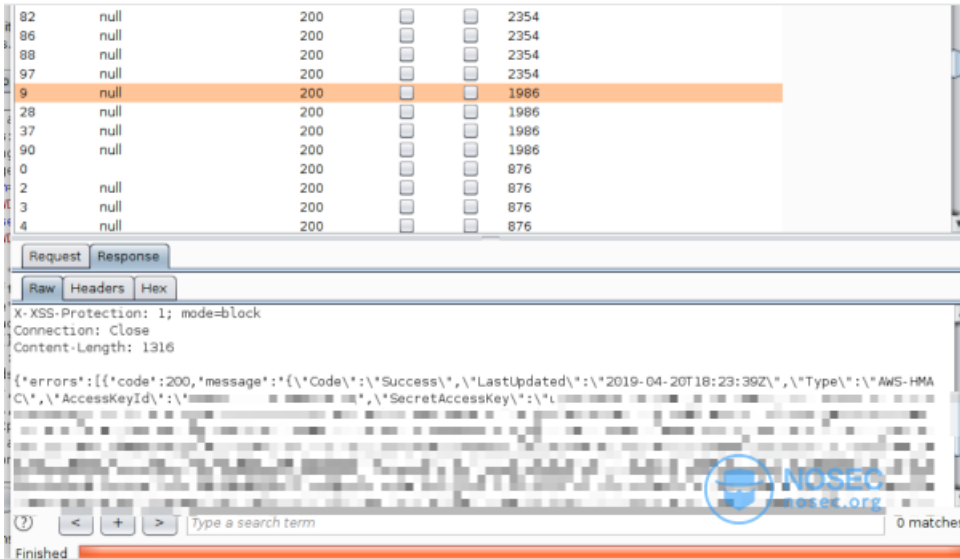
我使用了writeup中的技巧，将谷歌网站的IP解析为127.0.0.1。在利用burp发送了100个请求后，我终于看到了想要的结果！



此时，虽然我成功找到了一个SSRF漏洞，但并不知道接下去该做什么……我以前都只是面对CTF——你拿到flag，一切就结束了。但实际情况并非如此。

在现实生活中效果不是很好

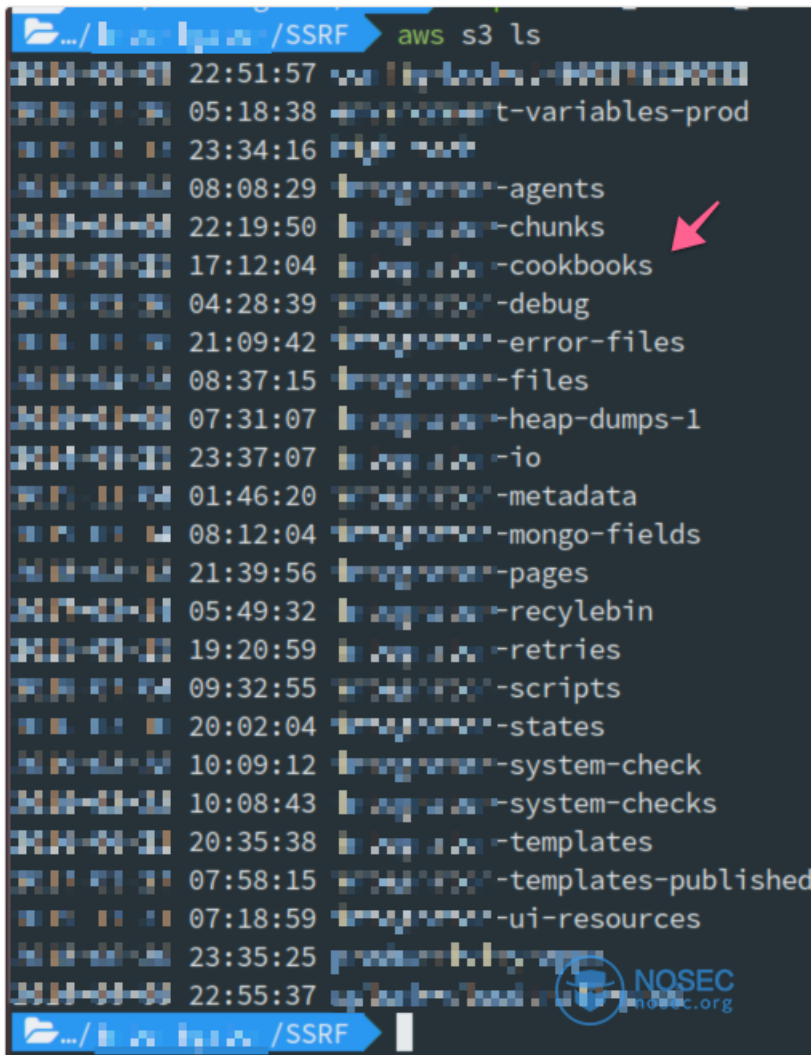
在几个小时毫无头绪后，我向Jan Masarik发出了求助。我告诉他，目标貌似和亚马逊服务有关，于是他发送给我一些ip地址——`169.254.169.254`。他告诉我这是AWS元数据的ip地址，如果我能获取其中的数据，基本上就控制了整个AWS。于是我立刻开始了行动，并很快得到了AWS的key。



回到现实

不幸的是，那些都是没用的key，我几乎不能用它们做任何事.....。

实际上，我只能读写一些存储桶，但这没有任何用处，而那些托管在前端的存储桶都是不可写的。在接下来的几个小时里我试着提升权限以及尝试一些其他操作，但其实能做的并不多。



于是我决定把这个漏洞上报，并持续观察测试，看是否能得到一个RCE。

一周后

我没有收到公司的任何回复，也许我应该把这个漏洞说的更严重一点……，不过这样我有了更多的时间进行测试。

其他利用点

当我向 `127.0.0.1:22` 发出请求时，服务器返回 `bad response: SSH-2.0-OpenSSH_someversion :D`，我试图通过暴力破解得到ssh的root登录密码，但很可惜失败了。

接下来我开始遍历这台机器上的其他端口。

很快我又发现了一个Monit管理界面，由于它存在一个缓冲区漏洞，所以我可以读取到一些内存信息甚至直接关闭整个实例！

最后我把这个新发现也上报了！

一个月后

在等待了一个月后，他们终于修好了漏洞，并给了我一笔赏金，生活还在继续。

后续

回顾整个流程，我觉得在利用DNS这方面缺少一个强悍的界面和日志，于是开始使用Flask api，通过SQL和Redis连接到特殊配置的DNS服务器。

[你可以点击这里](#)查看代码，以及一个现成的可使用的[网站](#)——你可以注册，创建新绑定规则，用它来破解某些东西，查看日志等等。是的，这个网站只使用了http协议，我恨自己，我太懒了，不想去安装https。

具体来说，它的作用是：

我告诉它一个子域，每解析三次到 `1.2.3.4`，就解析一次到 `127.0.0.1`

它会将数据放入数据库，并给出类似 `y198ehiuwqh82319j2139821.ge10.space` 这样的子域名

当我针对这个子域进行解析查询时，DNS服务器会查找数据库中的数据，并加载到redis中，根据给定的规则以更快的速度回复未来的请求。

如果你有一些空闲时间，欢迎看看我的 [\[https://github.com/makuga01/dnsFookup\]](https://github.com/makuga01/dnsFookup) (<https://github.com/makuga01/dnsFookup>)。如果有人能添加一些功能或前端，我会很感激的。

若你想和我联系，可以直接在推特上找到[我](#)。

本文由白帽汇整理并翻译，不代表白帽汇任何观点和立场：<https://nosec.org/home/detail/3185.html>
来源：<https://geleta.eu/2019/my-first-ssrf-using-dns-rebinfing/>