Operating Systems Lab
Jacobs University Bremen
Dr. Jürgen Schönwälder

Course: 320232
Date: 2016-11-24
Deadline: 2016-11-30

Problem Sheet #12

Problem 12.1: *dns packets munging kernel module* (1+2+3+2+2 = 10 points)

The Domain Name System (DNS) uses a protocol [1] that usually runs over UDP using the well-known port 53. The protocol exchange is very simple: The name resolver of the host sends a query packet to a recursive DNS resolver and the recursive DNS resolver eventually returns a response packet.

Your task is to write a Linux kernel module that hooks into the kernel's netfilter infrastructure. The kernel module should parse DNS packets and decide whether they are moved forward through the netfilter infrastructure and the kernel's networking stack or whether they should be dropped. In addition, the module should allow to modify incoming DNS responses. By passing suitable options to the module, the module may rewrite A and AAAA records returned for a certain DNS name with different addresses. For example, a DNS query for www.google.com may return A and AAAA addresses of www.yahoo.com.

The assignment can be broken down into the following steps:

a) Write code to register hooks into the netfilter system and to identify UDP packets exchanged over IPv4 and IPv6.

b) Write a function to decode DNS packet headers. It should extract the ID, QR, RCODE, QDCOUNT, ANCOUNT, NSCOUNT, and ARCOUNT fields of the DNS header.

c) Write code to read domain names encoded in a DNS packet. The code should handle regular and compressed labels.

d) Write code that iterates over the response records and writes a line for each record to the kernel's logging facility.

e) Write code that attemps to modify A and AAAA records contained in the answer section if the query name matches a certain string.

DNS names are encoded as a sequence of labels where each label is length prefixed. A label can be up to 63 octets long (hence the length octet must contain a value be between 0 and 63). For example, the encoding of `www.google.com` will be (in C notation)

```
unsigned char encoded_name[] = {
    0x03, 'w', 'w', 'w', 0x06, 'g', 'o', 'o', 'g', 'l', 'e', 0x03, 'c', 'o', 'm', 0x00
};
```

DNS also supports message compression. Consult section 4.1.4 of RFC 1035 [1] for the details. In short, if a name www.google.com has been encoded at the offset position 42 relative to the start of the DNS message, then maps.google.com can be encoded as follows (again in C notation):

```
unsigned char encoded_compressed_name[] = {
    0x04, 'm', 'a', 'p', 's', 0xc0, 0x2e
};
```

The two highest bits set to 1 in the value 0xc0 indicate that the byte contains a part of an offset, i.e., a pointer within the DNS message to a place where the name encoding continues. An offset is always encoded using a 16-bit field with the two highest bits set to 1. In the example, the offset is given by the lower 14 bits of the 16-bit value 0xc02e, which is the decimal value 46.

RFC 1035 states that offsets may only ’point’ to labels prior in the message:

> In order to reduce the size of messages, the domain system utilizes a compression scheme which eliminates the repetition of domain names in a message. In this scheme, an entire domain name or a list of labels at the end of a domain name is replaced with a pointer to a prior occurance of the same name.

This, of course, does not prevent any loops. One option to deal with possible loops is to simply count the number of octets in the name and if the name gets longer than the maximum length of a name (255 octets), then there is clearly an error.

References

[1] P. Mockapetris. Domain Names - Implementation and Specification. RFC 1035, ISI, November 1987.