Fine penetration tests for fine websites

Dr.-Ing. Mario Heiderich, Cure53
Rudolf Reusch Str. 33
D 10367 Berlin
cure53.de · mario@cure53.de

Pentest-Report dnsmasq 05.-06.2016

Cure53, Dr.-Ing. Mario Heiderich, Mike Wege, Dario Weißer

Index

Introduction

“Dnsmasq provides network infrastructure for small networks: DNS, DHCP, router advertisement and network boot. It is designed to be lightweight and have a small footprint, suitable for resource constrained routers and firewalls.

*It has also been widely used for tethering on smartphones and portable hotspots, and to support virtual networking in virtualisation frameworks. Supported platforms include Linux (with glibc and uclibc), Android, *BSD, and Mac OS X. Dnsmasq is included in most Linux distributions and the ports systems of FreeBSD, OpenBSD and NetBSD. Dnsmasq provides full IPv6 support.”*

From http://www.thekelleys.org.uk/dnsmasq/doc.html

This report documents the findings of the penetration test and security assessment conducted by the Cure53 team against the dnsmasq server. The project was generously funded from the SOS programme run by Mozilla. Carried out over a period of two weeks in May and June 2016, this assignment involved three members of the Cure53 team.

Dr.-Ing. Mario Heiderich, Cure53
Rudolf Reusch Str. 33
D 10367 Berlin
cure53.de · mario@cure53.de

Regarding the approach and scope, the test was performed as a tool-assisted code audit. Getting full coverage of the available dnsmasq sources was attempted and the tests were aimed at several specific attacks and vulnerability patterns, namely buffer overflows, parsing errors, information leaks, cryptographic errors and usage of weak functions.

As for the results, the tests have only uncovered six issues, with three among them classified as actual security vulnerabilities. It has to be underscored that neither issues with “Critical” nor with “High” severity rankings were found, denoting that the discoveries are not particularly salient in terms of their impact, scope, and the risk they pose.

Scope

- **dnsmasq Sources**
  - http://thekelleys.org.uk/gitweb/?p=dnsmasq.git;a=summary

Identified Vulnerabilities

The following sections list both vulnerabilities and implementation issues spotted during the testing period. Note that findings are listed in a chronological order rather than by their degree of severity and impact. The aforementioned severity rank is simply given in brackets following the title heading for each vulnerability. Each vulnerability is additionally given a unique identifier (e.g. *DM-01-001*) for the purpose of facilitating any future follow-up correspondence.

DM-01-001 Uninitialized buffer leads to memory leakage *(Medium)*

It was discovered that the application does not properly initialize the buffer for incoming and outgoing packets. This leads to a partial disclosure of the previously received or sent packets. Taking advantage of this issue made it possible to leak up to [*28+tftp_prefix_length*] bytes from a packet that another party had sent to the server. The attack is limited in scope and impact since only the network data can be leaked. It is further constrained by the condition that the packets must be greater than 530 bytes in size. However, it cannot be excluded that an attacker could work around these limitations by using other services, although this option has not been investigated further during this test.

In the following example one can observe a setup comprising the server, a victim and an attacker. The victim sends a long packet (around 550 bytes) to the server. Since this is only a proof of concept the content is inconsequential here.

Dr.-Ing. Mario Heiderich, Cure53
Rudolf Reusch Str. 33
D 10367 Berlin
cure53.de · mario@cure53.de

Victim → Server:

```
perl -e 'print "LeakMe"x91' | nc -u 192.168.1.1 53
```

Having completed sending assignments, the *daemon->packet* buffer is filled with the data from the client's request. In essence, the attacker can now run their attack aimed at leaking a few bytes. The packet that is used here is a *TFTP*[1] request to read a file.

Attacker → Server:

```
perl -e 'print "\x00\x01".("A"x500)."\x00octet\x00"' | nc -u 192.168.1.1 69
```

The file is not accessible and the server responds with the corresponding error message. Since the buffer holding the packet is not flushed, it still holds the contents from the previous communication with the victim.

Server → Attacker:

```
17:39:33.558152 IP (tos 0x0, ttl 64, id 65025, offset 0, flags [none], proto UDP
(17), length 570)
    192.168.1.1.47255 > 192.168.1.3.50423: [udp sum ok] UDP, length 542
       0x0000:  0800 27d0 e35c 0800 2788 9fcd 0800 4500  ..'..\..'.....E.
       0x0010:  023a fe01 0000 4011 f75c c0a8 0101 c0a8  .:....@..\......
       0x0020:  0103 b897 c4f7 0226 9f78 0005 0000 6361  .......&.x....ca
       0x0030:  6e6e 6f74 2072 6561 6420 2f6c 656c 2f41  nnot.read./lel/A
       0x0040:  4141 4141 4141 4141 4141 4141 4141 4141  AAAAAAAAAAAAAAAA
       ...
       0x0210:  4141 4141 4141 4141 4141 4141 4141 4141  AAAAAAAAAAAAAAAA
       0x0220:  4100 6374 6574 0065 4c65 616b 4d65 4c65  A.ctet.eLeakMeLe
       0x0230:  616b 4d65 4c65 616b 4d65 4c65 616b 4d65  akMeLeakMeLeakMe
       0x0240:  4c65 616b 4d65 4c65                      LeakMeLe
```

It is evident that the message sent to the attacker should by no means contain data sent by the victim. The vulnerability is caused by two different flaws described next.

1. No initialization of buffers:

The packet buffer "*packet*" located in the "*daemon*" structure is used to store requests and responses for different protocols. The server does not fork when UDP packets are received, signifying that the buffer becomes reusable for different connections from different hosts. Because the memory is not cleared between the connections, the buffer might technically contain old data.

[1] https://en.wikipedia.org/wiki/Trivial_File_Transfer_Protocol

Dr.-Ing. Mario Heiderich, Cure53
Rudolf Reusch Str. 33
D 10367 Berlin
cure53.de · mario@cure53.de

2. Incorrect use of snprintf():

It can be inferred from the existing schemes that the author of the code probably believed that the return value of *snprintf()*[2] equals the number of bytes written into the destination buffer. In fact this is only the case if the source string is smaller or equal to the limit value. *Snprintl* returns the number of bytes that could have been written if there was no limitation in place.

The *tftp_request* function is called frequently in order to check for new requests on the *tftp* socket. Basically, the file requests are handled here but, prior to a file being available for reading, the accessibility needs to be verified. After checking the permissions and potentially reading the requested file, a response of size *len* is sent to the client.

Affected File:
/src/tftp.c

Affected Code:

```
void tftp_request(struct listener *listen, time_t now)
{
…
       if ((transfer->file = check_tftp_fileperm(&len, prefix))) {
              ...
       }

       while (sendto(transfer->sockfd, packet, len, 0,
                           (struct sockaddr *)&peer, sa_len(&peer)) == -1
                                 && errno == EINTR);
```

Check_tftp_fileperm performs a permission check and tries to open the file. If it turns out impossible then an error message is generated and written into the packet buffer.

```
static struct tftp_file *check_tftp_fileperm(ssize_t *len, char *prefix)
{
       …
       if ((fd = open(namebuff, O_RDONLY)) == -1)
       {
              if (errno == ENOENT)
              {
                     *len = tftp_err(ERR_FNF, packet,
                            _("file %s not found"), namebuff);
                     return NULL;
              }
              else if (errno == EACCES)
```

[2] https://en.wikipedia.org/wiki/C_file_input/output#sprintf

Dr.-Ing. Mario Heiderich, Cure53
Rudolf Reusch Str. 33
D 10367 Berlin
cure53.de · mario@cure53.de

```
                goto perm;
        else
                goto oops;
}
…
oops:
*len =  tftp_err_oops(packet, namebuff);
if (fd != -1)
        close(fd);
return NULL;
```

The problem occurs in *tftp_err* where the error message is written into the message buffer. The return value of *snprintf()* is used to determine the size of the final message that is sent to the user. While the maximum number of bytes *snprintf* can write is in fact 500, the return value equals the length of the string before it is truncated at 500 bytes. Thus more data than what is actually in the buffer is sent to the user.

```
static ssize_t tftp_err_oops(char *packet, char *file)
{
        /* May have >1 refs to file, so potentially mangle a copy of the name */
        strcpy(daemon->namebuff, file);
        return tftp_err(ERR_NOTDEF, packet,
                _("cannot read %s: %s"), daemon->namebuff);
}

static ssize_t tftp_err(int err, char *packet, char *message, char *file)
{
        struct errmess {
                unsigned short op, err;
                char message[];
        } *mess = (struct errmess *)packet;
        ...
        ssize_t ret = 4;
        ret += (snprintf(mess->message, 500,  message, file, errstr) + 1);
        ...
        return  ret;
}
```

It is recommended to overwrite buffers with zero, both before receiving data and after sending it. Furthermore *tftp_err* needs to be adjusted to return the correct size.

Dr.-Ing. Mario Heiderich, Cure53
Rudolf Reusch Str. 33
D 10367 Berlin
cure53.de · mario@cure53.de

DM-01-003 Makefile lacks security parameters for gcc *(Low)*

Position-independant executable[3] (PIE) is a security mechanism that makes exploitation of vulnerabilities much more difficult. Binaries without the PIE are always mapped to the same address when being executed. An attacker who already has found a vulnerability can use this information to read data from a known address or to create a ROP payload[4]. A binary compiled with PIE is mapped to a random memory location when being executed and eliminates all predictable addresses.

By default, the Executables are mapped to the address *0x400000* on the 64bit Linux systems.

Memory maps without PIE:

```
00400000-0044f000 r-xp 00000000 08:01 271825
/root/dnsmasq/src/dnsmasq
0064f000-00653000 rw-p 0004f000 08:01 271825
/root/dnsmasq/src/dnsmasq
00872000-00893000 rw-p 00000000 00:00 0                                  [heap]
```

With PIE enabled, the Executable no longer remains at *0x400000* but has a random address which is chosen when the program is started.

Memory maps with PIE:

```
7f2292f5f000-7f2292fb1000 r-xp 00000000 08:01 271827
/root/dnsmasq/src/dnsmasq
7f22931a5000-7f22931a9000 rw-p 00000000 00:00 0
7f22931ae000-7f22931b1000 rw-p 00000000 00:00 0
7f22931b1000-7f22931b5000 rw-p 00052000 08:01 271827
/root/dnsmasq/src/dnsmasq
7f2293616000-7f2293637000 rw-p 00000000 00:00 0                          [heap]
```

PIE should be enabled by default on all systems that support this feature. Furthermore, adding subsequent compiler security flags should be considered. Some of them are described in the *debian* wiki[5].

[3] https://en.wikipedia.org/wiki/Position-independent_code#PIE
[4] https://en.wikipedia.org/wiki/Return-oriented_programming
[5] https://wiki.debian.org/Hardening#gcc_-Wformat_-Wformat-security

Dr.-Ing. Mario Heiderich, Cure53
Rudolf Reusch Str. 33
D 10367 Berlin
cure53.de · mario@cure53.de

DM-01-006 Allocated memory is not cleared *(Low)*

The application uses *malloc()*[6] to allocate memory for buffers and data structures. This function returns a pointer to the allocated space but leaves its contents untouched. In other words, the previously freed data can remain in the area in which it will then be contained by the buffer. This can lead to memory disclosure if this type of buffer is sent via network without being filled properly.

The problem can be avoided by overwriting buffers with 0's immediately after allocation. It is recommended to use *calloc()* instead of the current function in order to avoid the pitfalls of *malloc()*. The buffers that are allocated on the stack must be nulled manually. As already described in DM-01-001, it is important to clear buffers prior to any re-usage.

Miscellaneous Issues

This section covers those noteworthy findings that did not lead to an exploit but might aid an attacker in achieving their malicious goals in the future. Most of these results are vulnerable code snippets that did not provide an easy way to be called. Conclusively, while a vulnerability is present, an exploit might not always be possible.

DM-01-002 Unchecked return value can lead to NULL pointer dereference (*Low*)

The functions *send_ra_alias* and *periodic_slaac* do not check the return value of *expand.* Therefore, they might lead to a possible read from 0. *Expand* resizes a buffer and returns the pointer on success and, if *malloc* fails to allocate enough memory, *NULL* is returned. The result would be an invalid dereference call which effectively crashes the program.

Affected File:
/src/slaac.c

Affected Code:

```
ping = expand(sizeof(struct ping_packet));
ping->type = ICMP6_ECHO_REQUEST;
ping->code = 0;
```

Affected File:
/src/radv.c

[6] https://en.wikipedia.org/wiki/C_dynamic_memory_allocation

Dr.-Ing. Mario Heiderich, Cure53
Rudolf Reusch Str. 33
D 10367 Berlin
cure53.de · mario@cure53.de

Affected Code:

```
ra = expand(sizeof(struct ra_packet));

ra->type = ND_ROUTER_ADVERT;
ra->code = 0;
ra->hop_limit = hop_limit;
```

It is recommended to check the return values of *Expand* and to make sure that the operation is aborted in case of the 0 value..

DM-01-004 Wrong assumption about return value of snprintf()/vsnprintf() (*Low*)

As already discussed, particularly in the realm of DM-01-001, the return value of *snprintf()/vsnprintf()* is inappropriately used across various locations. While this can fortunately result only in a mangled output in the cases described below, it is nevertheless recommended to fix this oversight.

Affected File:
/src/log.c

Affected Code:

```
len = p - entry->payload;
va_start(ap, format);
len += vsnprintf(p, MAX_MESSAGE - len, format, ap) + 1; /* include zero-
terminator */
va_end(ap);
entry->length = len > MAX_MESSAGE ? MAX_MESSAGE : len;
```

Affected File:
/src/rfc2131.c

Affected Code:

```
char *s = option_string(AF_INET, req_options[i], NULL, 0, NULL, 0);
q += snprintf(q, MAXDNAME - (q - daemon->namebuff),
       "%d%s%s%s",
       req_options[i],
       strlen(s) != 0 ? ":" : "",
       s,
       req_options[i+1] == OPTION_END ? "" : ", ")
```

Affected File:
/src/rfc3315.c

Dr.-Ing. Mario Heiderich, Cure53
Rudolf Reusch Str. 33
D 10367 Berlin
cure53.de · mario@cure53.de

Affected Code:

```
char *s = option_string(AF_INET6, opt6_uint(oro, i, 2), NULL, 0, NULL, 0);
q += snprintf(q, MAXDNAME - (q - daemon->namebuff),
       "%d%s%s%s",
       opt6_uint(oro, i, 2),
       strlen(s) != 0 ? ":" : "",
       s,
       (i > opt6_len(oro) - 3) ? "" : ", ");
```

A recommendation stemming from the above is to use the return value according to *RETURN VALUE* and *NOTES* in the respective man-page or, alternatively, have it properly implemented in */src/cache.c*.

DM-01-005 Hardcoded values in fscanf() format strings with aliased buffers (*Low*)

The problem described here concerns the usage of hard-coded string length values. More specifically, it is the values with calls to *fscanf()* with borrowed buffers that can easily lead to buffer overflow in the event of code changes being deployed. This does not currently lead to any buffer overflows because the buffer sizes are large enough. Moreover the referenced names of the aliased buffers are specific to the original case and not to the reuse or more general case.

Affected File:
/src/lease.c

Affected Code:

```
/* client-id max length is 255 which is 255*2 digits + 254 colons
       borrow DNS packet buffer which is always larger than 1000 bytes */
if (leasestream)
       while (fscanf(leasestream, "%255s %255s",
              daemon->dhcp_buff3, daemon->dhcp_buff2) == 2)
       {
```

Affected Code:

```
ei = atol(daemon->dhcp_buff3);

if (fscanf(leasestream, " %64s %255s %764s",
              daemon->namebuff, daemon->dhcp_buff,
                     daemon->packet) != 3)
       break;
```

It is recommended to use the allocated buffer size constants *modulo* the maximum expected lengths from the respective header file and double-apply the *stringify* operator to embed the resulting values in the format strings. The downside to this solution is that it will likely not increase the readability of the code.

Dr.-Ing. Mario Heiderich, Cure53
Rudolf Reusch Str. 33
D 10367 Berlin
cure53.de · mario@cure53.de

Conclusion

Describing the results of the spring 2016 security assessment of the dnsmasq server carried out by the Cure53 team, this report highlights the overall positive impression that the state of security at the dnsmasq makes. At the same time, some attention should be given to the fact that six issues were still found. Three of the discoveries constitute actual security vulnerabilities and should be addressed as soon as possible.

Looking at all findings of the audit, which was notably sponsored by the Mozilla SOS Program, one has to underline that only one issue, namely the DM-01-001 revealing a memory leak, was eventually classified as “Medium” in terms of severity. Consequently, the remaining problems were less impactful, effectively suggesting not only the robustness of the code, but also pointing to the predicted relative ease of implementing fixes.

The overall impression of the code quality shared by the Cure53 testing team was positive, with further notes being made about the code’s mature feel. At the same time, some of the numerous hand hand-crafted parsers and serializers/deserializers were somewhat questionable, thus leaving room for further investigations of the dnsmasq server. The main problem appears to be the largely missing documentation of the great majority of the existing code. As a result, it can be forecasted that future maintainers of either the main branch or the potential forks of the codebase, will have considerable difficulty in fully understanding the implementation in place. This needs to be discussed and reflected upon when the long-term perspective of the dnsmasq evolution is considered.

One general technical recommendation for improving the security of the dnsmasq is to replace or further restrict unbounded string handling calls. This applies to *strcat(), strcpy(), strlen(), sprintf(), strtok()* with their respective *n-limited* functions. At any rate, the proper usage of the signed length values needs to be ensured as it otherwise leads to a potential for buffer under- and over- flows. The application of an internal random number generator over the *system-rng/prng* has not left the Cure53 overly confident about its suitability and aptness. However, no obviously problematic usage of this mechanism for the cryptographically relevant operations could be identified during the tests.

Cure53 would like to thank Gervase Markham and Chris Riley of Mozilla for their excellent project coordination, support and assistance, both before and during this assignment.