

## Heap Based Buffer Over-Read Vulnerability in Rldns-1.3

Written by : Antonius (w1sdom)

Web : [www.bluedragonsec.com](http://www.bluedragonsec.com)

*First of all, I apologize if this write-up feels a bit "robotic." As a roboticist and electronic engineer who spends most of my time building hardware, my writing style tends to be quite literal.*

### Vulnerability Discovery

During a fuzzing session with rldns-1.3, I discovered a **heap-based out-of-bounds read vulnerability**. I used **honggfuzz 2.6** for this session.

**What is Rldns?** Rldns is an open-source DNS server project that I maintain.

You can find the archives here : [https://github.com/bluedragonsecurity/rldns\\_archives](https://github.com/bluedragonsecurity/rldns_archives)

and the source codes for latest rldns here : <https://github.com/bluedragonsecurity/rldns>

Currently, the latest version is rldns-1.4.

In February 2026, I finally had some spare time to fuzz the server. Through this process, I identified several vulnerabilities in version 1.3, including **null pointer dereferences** and **heap-based buffer over-reads**—which was quite alarming since I am the developer!

### Fuzzing Methodology & Environment

To detect memory corruption, I relied on **AddressSanitizer (ASan)**. I used **honggfuzz** specifically to generate malformed DNS packets to trigger crashes.

I used the following CFLAGS and LDFLAGS to ensure ASan would capture the corruption details:

```
CFLAGS = -O1 -g -fno-omit-frame-pointer -fno-optimize-sibling-calls  
-D_FORTIFY_SOURCE=0 -fsanitize=address,undefined  
LDFLAGS = -fsanitize=address,undefined
```

Full Makefile url :

```
https://raw.githubusercontent.com/bluedragonsecurity/rldns/refs/heads/main/  
Makefile\_for\_honggfuzz
```

To enable ASan logging, I created a shell script named `run.sh` to execute rldns

```
#!/bin/bash  
pkill -9 rldns  
echo "" > asan_log.txt
```

```

ASAN_PATH=$(gcc -print-file-name=libasan.so)
if [[ "$ASAN_PATH" == "libasan.so" ]]; then
  ASAN_PATH=$(ldd ./rldns | grep libasan | awk '{print $3}')
fi
export LD_PRELOAD="$ASAN_PATH"
export ASAN_OPTIONS="log_path=stderr:abort_on_error=1:detect_leaks=0"
./rldns &>> asan_log.txt &

```

then I run rldns as root using that script :

```
./run.sh
```

Confirming the dns server is running :

```

(root@robohax-20bws2ng00)-[~]
# ps aux | grep rldns
rldns      60487  0.0  0.1 21474884560 14256 pts/3 S   08:22   0:00 ./rldns
root      61593  0.0  0.0  6748   2412 pts/3  S+  08:24   0:00 grep --color=auto rldns

(root@robohax-20bws2ng00)-[~]
# dig @127.0.0.1 ringlayer.net

; <<>> DiG 9.20.15-2-Debian <<>> @127.0.0.1 ringlayer.net
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->HEADER<- opcode: QUERY, status: NOERROR, id: 25767
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags;; udp: 4096
;; QUESTION SECTION:
;ringlayer.net.                IN      A
;

;; ANSWER SECTION:
ringlayer.net.                600     IN      A      160.202.119.34

;; Query time: 87 msec
;; SERVER: 127.0.0.1#53(127.0.0.1) (UDP)
;; WHEN: Tue Feb 24 08:24:38 WIB 2026
;; MSG SIZE rcvd: 58

(root@robohax-20bws2ng00)-[~]
#

```

I have prepared dns corpus for honggfuzz, which can be downloaded here :

[https://github.com/bluedragonsecurity/corpus\\_for\\_honggfuzz](https://github.com/bluedragonsecurity/corpus_for_honggfuzz)

Next step, I used honggfuzz to generate various malformed dns packets for rldns :

```
honggfuzz -i dns_corpus -s -- /bin/nc -u -n -w 1 127.0.0.1 53
```

```
[ 0 days 00 hrs 00 mins 17 secs ]
Iterations : 32
Mode [1/3] : Feedback Driven Dry Run [10/12]
Target : /bin/nc -u -n -w 1 127.0.0.1 53
Threads : 2, CPUs: 4, CPU%: 25% [6%/CPU]
Speed : 0/sec [avg: 1]
Crashes : 0 [unique: 0, blacklist: 0, verified: 0]
Timeouts : 30 [1 sec]
Corpus Size : 0, max: 8,192 bytes, init: 12 files
Cov Update : 0 days 00 hrs 00 mins 17 secs ago
Coverage : edge: 0/0 [0%] pc: 0 cmp: 0

[ LOGS ] / honggfuzz 2.6 /-
2026-02-24T08:32:58+0700][W][66028] subproc_checkTimeLimit():532 pid=66062 took too much time (limit 1 s). Killing it with SIGKILL
[2026-02-24T08:32:59+0700][W][66027] subproc_checkTimeLimit():532 pid=66078 took too much time (limit 1 s). Killing it with SIGKILL
[2026-02-24T08:32:59+0700][W][66028] subproc_checkTimeLimit():532 pid=66079 took too much time (limit 1 s). Killing it with SIGKILL
[2026-02-24T08:33:00+0700][W][66028] subproc_checkTimeLimit():532 pid=66094 took too much time (limit 1 s). Killing it with SIGKILL
[2026-02-24T08:33:00+0700][W][66027] subproc_checkTimeLimit():532 pid=66093 took too much time (limit 1 s). Killing it with SIGKILL
[2026-02-24T08:33:02+0700][W][66027] subproc_checkTimeLimit():532 pid=66104 took too much time (limit 1 s). Killing it with SIGKILL
[2026-02-24T08:33:02+0700][W][66028] subproc_checkTimeLimit():532 pid=66103 took too much time (limit 1 s). Killing it with SIGKILL
[2026-02-24T08:33:03+0700][W][66027] subproc_checkTimeLimit():532 pid=66115 took too much time (limit 1 s). Killing it with SIGKILL
[2026-02-24T08:33:03+0700][W][66028] subproc_checkTimeLimit():532 pid=66116 took too much time (limit 1 s). Killing it with SIGKILL
[2026-02-24T08:33:04+0700][W][66027] subproc_checkTimeLimit():532 pid=66126 took too much time (limit 1 s). Killing it with SIGKILL
[2026-02-24T08:33:04+0700][W][66028] subproc_checkTimeLimit():532 pid=66125 took too much time (limit 1 s). Killing it with SIGKILL
[2026-02-24T08:33:05+0700][W][66028] subproc_checkTimeLimit():532 pid=66135 took too much time (limit 1 s). Killing it with SIGKILL
[2026-02-24T08:33:05+0700][W][66027] subproc_checkTimeLimit():532 pid=66136 took too much time (limit 1 s). Killing it with SIGKILL
[2026-02-24T08:33:06+0700][W][66027] subproc_checkTimeLimit():532 pid=66147 took too much time (limit 1 s). Killing it with SIGKILL
[2026-02-24T08:33:06+0700][W][66028] subproc_checkTimeLimit():532 pid=66146 took too much time (limit 1 s). Killing it with SIGKILL
[2026-02-24T08:33:07+0700][W][66028] subproc_checkTimeLimit():532 pid=66157 took too much time (limit 1 s). Killing it with SIGKILL
[2026-02-24T08:33:07+0700][W][66027] subproc_checkTimeLimit():532 pid=66156 took too much time (limit 1 s). Killing it with SIGKILL
[2026-02-24T08:33:08+0700][W][66028] subproc_checkTimeLimit():532 pid=66166 took too much time (limit 1 s). Killing it with SIGKILL
[2026-02-24T08:33:08+0700][W][66027] subproc_checkTimeLimit():532 pid=66167 took too much time (limit 1 s). Killing it with SIGKILL
[2026-02-24T08:33:09+0700][W][66027] subproc_checkTimeLimit():532 pid=66185 took too much time (limit 1 s). Killing it with SIGKILL
[2026-02-24T08:33:09+0700][W][66028] subproc_checkTimeLimit():532 pid=66184 took too much time (limit 1 s). Killing it with SIGKILL
[2026-02-24T08:33:10+0700][W][66027] subproc_checkTimeLimit():532 pid=66194 took too much time (limit 1 s). Killing it with SIGKILL
[2026-02-24T08:33:10+0700][W][66028] subproc_checkTimeLimit():532 pid=66195 took too much time (limit 1 s). Killing it with SIGKILL
[2026-02-24T08:33:11+0700][W][66028] subproc_checkTimeLimit():532 pid=66204 took too much time (limit 1 s). Killing it with SIGKILL
[2026-02-24T08:33:11+0700][W][66027] subproc_checkTimeLimit():532 pid=66205 took too much time (limit 1 s). Killing it with SIGKILL
```

## Crash Analysis

Once rldns crashed, the ASan log reported a **READ of size 16**:

**READ of size 16 at 0x7bff60fe0a7f thread T0**

**#0 0x7fdf62c7e4ce in strlen**

**.././././src/libsanitizer/sanitizer\_common/sanitizer\_common\_interceptors.inc:425**

**#1 0x564dea55f3fa in \_count\_octet\_length src/main.c:1021**

**#2 0x564dea55f3fa in parse\_enumerate\_and\_fetch\_each\_octet\_and\_just\_return\_a\_name src/main.c:667**

**#3 0x564dea561d80 in rldns\_main src/main.c:1943**

**#4 0x564dea568cb7 in main src/main.c:1882**

**#5 0x7fdf62029f67 in \_\_libc\_start\_call\_main ../sysdeps/nptl/libc\_start\_call\_main.h:58**

**#6 0x7fdf6202a024 in \_\_libc\_start\_main\_impl ../csu/libc-start.c:360**

**#7 0x564dea552690 in \_start (/home/robohax/Desktop/CVE/rldns Remote Heap-Based Buffer Over-read/rldns-1.3/rldns+0x2f690) (BuildId: b895cc58067deb351b9d579ecbaa4f52e8b2b4f1)**

The vulnerability is triggered because the `strlen` function encounters an out-of-bounds read. Specifically, if `a_name` contains invalid bytes (like `0xff`), `strlen` may result in a segmentation fault.

The bug originates in the function `parse_enumerate_and_fetch_each_octet_and_just_return_a_name` at `src/main.c:667`. The routines responsible for building the name from a DNS query do not properly null-terminate the string. When a malformed packet ends with a non-null byte (e.g., `0xff`), the subsequent call to `strlen(a_name)` reads past the boundary of the allocated heap buffer.

## Capturing the Malformed Packet

Since honggfuzz does not automatically log the specific packet that causes a crash, I created a shared object (`dns_logger.c`) to hook the `recvfrom` function and log the last received packet to `/tmp/last_packet.dns`

```
#define GNU_SOURCE
#include <dlfcn.h>
#include <stdio.h>
#include <sys/socket.h>
#include <unistd.h>

ssize_t recvfrom(int sockfd, void *buf, size_t len, int flags,
                 struct sockaddr *src_addr, socklen_t *addrlen) {
    static ssize_t (*real_recvfrom)(int, void*, size_t, int, struct sockaddr*, socklen_t*) = NULL;
    if (!real_recvfrom) real_recvfrom = dlsym(RTLD_NEXT, "recvfrom");

    ssize_t n = real_recvfrom(sockfd, buf, len, flags, src_addr, addrlen);
    if (n > 0) {
        FILE *f = fopen("/tmp/last_packet.dns", "wb");
        if (f) {
            fwrite(buf, 1, n, f);
            fflush(f);
            fsync(fileno(f));
            fclose(f);
        }
    }
    return n;
}
```

Next, I compile it as shared object :

```
gcc -fPIC -shared -o dns_logger.so dns_logger.c -ldl
```

Since this shared object can not be preloaded when ASan activated, I use the original Makefile from `rldns-1.3` :

```
OBJ = obj
SRC = src
```

**all : rldns**

**rldns : main.o**

**gcc -Wall -fPIC -pthread -fstack-protector-all -o rldns \$(OBJ)/main.o**

**main.o : \$(SRC)/main.c \$(SRC)/vars.h \$(SRC)/structs.h \$(SRC)/oops.h \$(SRC)/headers.h**

**gcc -Wall -fPIC -pthread -fstack-protector-all -c \$(SRC)/main.c -o \$(OBJ)/main.o**

**clean:**

**rm -f \$(OBJ)/\***

**install:**

**rm -rf /usr/local/rldns;mkdir /usr/local/rldns**

**mkdir /usr/local/rldns/zones**

**cp rldns /usr/local/rldns**

**cp rldns.conf /usr/local/rldns**

**cp zones/\* /usr/local/rldns/zones**

**cp docs/rldns.1 /usr/share/man/man1**

Recompile to test and run the rldns with LD\_PRELOAD:

**make clean && make**

**cp dns\_logger.so /tmp/dns\_logger.so**

**LD\_PRELOAD=/tmp/dns\_logger.so ./rldns**

Next, I execute honggfuzz :

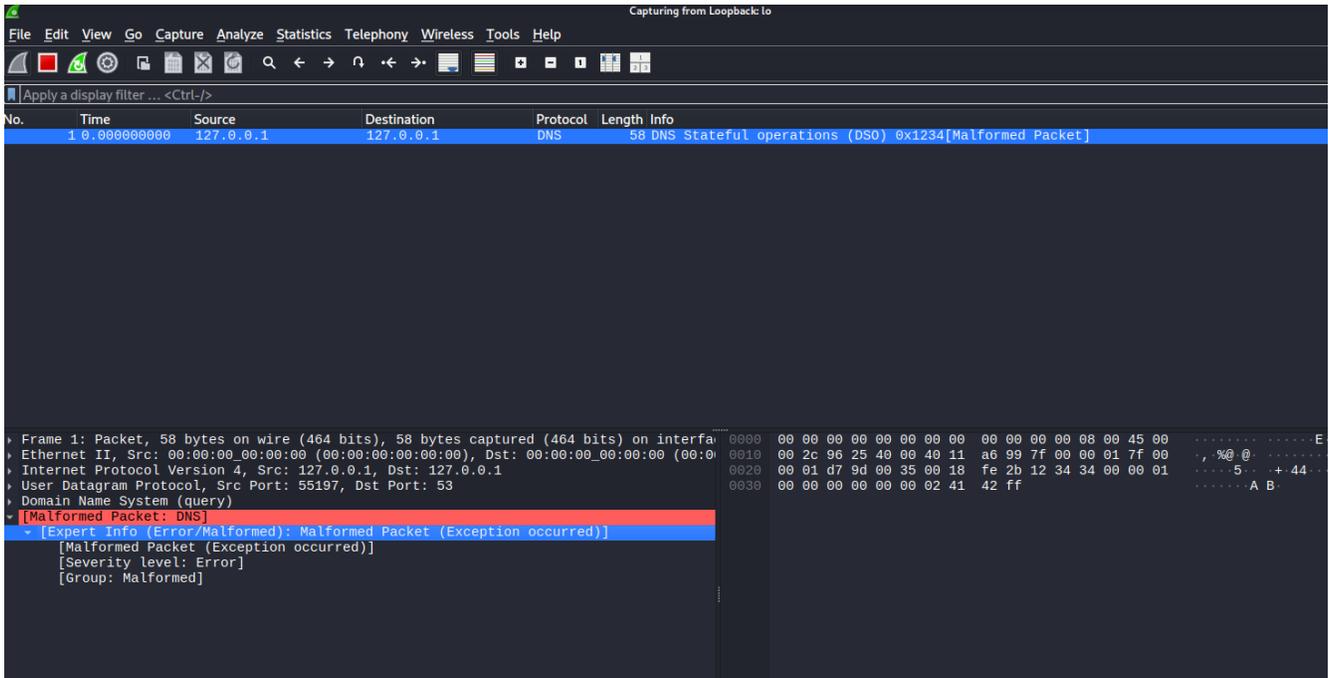
**honggfuzz -i dns\_corpus -s -- /bin/nc -u -n -w 1 127.0.0.1 53**

Once rldns-1.3 crashes, I got the last dns packet from honggfuzz at **/tmp/last\_packet.dns**

```
(root@robohax-20bws2ng00)-[~/tmp]
# xxd last_packet.dns
00000000: 1234 0100 0001 0000 0000 0000 0241 42ff  .4. ... ..AB.

(root@robohax-20bws2ng00)-[~/tmp]
#
```

Using xxd, we know that it's a malformed dns packet, since it contains only 16 bytes and ended with 0xff.



## The POC

The PoC that crashes rldns-1.3 is available here:

<https://github.com/bluedragonsecurity/rldns-1.3-heap-out-of-bounds-vulnerability-fixed-in-rldns-1.4/>

I expect to be busy over the next three years developing intelligence/hacking devices and hunting for 0-days. If you discover any vulnerabilities in rldns, please report them to me at:

**bluedragonsec2023@gmail.com.**