CVE-2014-5461 Analysis

Author : Minseok Kang

1. Overview

Crash type : heap-buffer-overflow

Version : v5.2.2 (git commit hash : 1294b09d8eff59a5fa00a43a2c462d338546da1f)

2. PoC code

```
function f(p1, p2, p3, p4, p5, p6, p7, p8, p9, p10,
           p11, p12, p13, p14, p15, p16, p17, p18, p19, p20,
           p21, p22, p23, p24, p25, p26, p27, p28, p29, p30,
           p31, p32, p33, p34, p35, p36, p37, p38, p39, p40,
           p41, p42, p43, p44, p45, p46, p48, p49, p50, ...)
  local a1, a2, a3, a4, a5, a6, a7, a8, a9, a10, a11, a12, a13, a14
end

f()   -- crashes on some machines
```

3. Root Cause Analysis

Following log is observed by executing PoC code in Lua compiled with address sanitizer applied.

```
=================================================================
==6242==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x61a000001770 at pc 0x562fdfe409fb
WRITE of size 8 at 0x61a000001770 thread T0
    #0 0x562fdfe409fa in adjust_varargs /home/user/cve-2014-5461/lua/ldo.c:266
    #1 0x562fdfe41f88 in luaD_precall /home/user/cve-2014-5461/lua/ldo.c:332
    #2 0x562fdfe82b17 in luaV_execute /home/user/cve-2014-5461/lua/lvm.c:759
    #3 0x562fdfe42b82 in luaD_call /home/user/cve-2014-5461/lua/ldo.c:395
    #4 0x562fdfe3300c in f_call /home/user/cve-2014-5461/lua/lapi.c:927
    #5 0x562fdfe3f1a5 in luaD_rawrunprotected /home/user/cve-2014-5461/lua/ldo.c:131
    #6 0x562fdfe44ee2 in luaD_pcall /home/user/cve-2014-5461/lua/ldo.c:595
    #7 0x562fdfe33661 in lua_pcallk /home/user/cve-2014-5461/lua/lapi.c:953
    #8 0x562fdfe20708 in docall /home/user/cve-2014-5461/lua/lua.c:179
    #9 0x562fdfe215eb in handle_script /home/user/cve-2014-5461/lua/lua.c:337
    #10 0x562fdfe22513 in pmain /home/user/cve-2014-5461/lua/lua.c:465
```

Lua supports vararg in function. Lua function executed through calling luaD_precall function internally, which is implemented as below.

```
/*
** returns true if function has been executed (C function)
*/
int luaD_precall (lua_State *L, StkId func, int nresults) {
  lua_CFunction f;
  CallInfo *ci;
  int n;  /* number of arguments (Lua) or returns (C) */
  ptrdiff_t funcr = savestack(L, func);
  switch (ttype(func)) {
  ...
    case LUA_TLCL: {  /* Lua function: prepare its call */
      StkId base;
      Proto *p = clLvalue(func)->p;
      luaD_checkstack(L, p->maxstacksize);
      func = restorestack(L, funcr);
      n = cast_int(L->top - func) - 1;  /* number of real arguments */
      for (; n < p->numparams; n++)
        setnilvalue(L->top++);  /* complete missing arguments */
      base = (!p->is_vararg) ? func + 1 : adjust_varargs(L, p, n);

      /* removed */

      return 0;
    }
  ...
  }
}
```

If Lua function is declared with vararg, adjust_varargs function is called to handle extra arguments. adjust_varargs function(ldo.c:257) is implemented as below.

```
static StkId adjust_varargs (lua_State *L, Proto *p, int actual) {
  int i;
  int nfixargs = p->numparams;
  StkId base, fixed;
  lua_assert(actual >= nfixargs);
  /* move fixed parameters to final position */
  fixed = L->top - actual;  /* first fixed argument */
  base = L->top;  /* final position of first argument */
  for (i=0; i<nfixargs; i++) {
    setobjs2s(L, L->top++, fixed + i);
    setnilvalue(fixed + i);
  }
  return base;
}
```

In line 8, setobjs2s macro, which copies values from 3rd argument to 2nd argument is called repetitively. However, no process checks whether the size of the Lua stack is sufficient. In PoC code, buffer overflow occurs as calling setobjs2s macro exceeds the Lua stack.

4. Patch

There is luaD_checkstack macro in Lua. This macro checks whether Lua stack reserves enough space for operations, otherwise extend it by reallocating the stack. The macro was added to adjust_varargs function to realloc Lua stack if it is needed.

```
static StkId adjust_varargs (lua_State *L, Proto *p, int actual) {
  int i;
  int nfixargs = p->numparams;
  StkId base, fixed;
  lua_assert(actual >= nfixargs);
  /* move fixed parameters to final position */
  luaD_checkstack(L, p->maxstacksize);  /* check again for new 'base' */
  fixed = L->top - actual;  /* first fixed argument */
  base = L->top;  /* final position of first argument */
  for (i=0; i<nfixargs; i++) {
    setobjs2s(L, L->top++, fixed + i);
    setnilvalue(fixed + i);
  }
  return base;
}
```

5. Reference

http://www.lua.org/bugs.html#5.2.2-1

https://github.com/lua/lua/commit/fa3b126a23f42134e6c9cc1ae2ba9f8d2df97967