CVE-2020-15889 Analysis

Author : Minseok Kang

1. Overview

Crash type : heap-based buffer over-read

Version : v5.4.0 (git commit hash : c33b1728aeb7dfeec4013562660e07d32697aa6b)

2. PoC code

```
function errfunc()
        setmetatable({}, {__gc = 1})
        errfunc()
end

errfunc()
```

3. Root Cause Analysis

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

```
user@ubuntu20:~/cve-2020-15889$ ./lua/lua poc.lua
=================================================================
==5245==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x60400000100c at pc 0x55ef590ca577
READ of size 4 at 0x60400000100c thread T0
    #0 0x55ef590ca576 in findsetreg (/home/user/cve-2020-15889/lua/lua+0x28576)
    #1 0x55ef590ca9e4 in getobjname (/home/user/cve-2020-15889/lua/lua+0x289e4)
    #2 0x55ef590cb651 in varinfo (/home/user/cve-2020-15889/lua/lua+0x29651)
    #3 0x55ef590cb736 in luaG_typeerror (/home/user/cve-2020-15889/lua/lua+0x29736)
    #4 0x55ef590cf18c in luaD_tryfuncTM (/home/user/cve-2020-15889/lua/lua+0x2d18c)
    #5 0x55ef590d0abc in luaD_call (/home/user/cve-2020-15889/lua/lua+0x2eabc)
    #6 0x55ef590d0b5e in luaD_callnoyield (/home/user/cve-2020-15889/lua/lua+0x2eb5e)
    #7 0x55ef590dbb59 in dothecall (/home/user/cve-2020-15889/lua/lua+0x39b59)
    #8 0x55ef590cd6bf in luaD_rawrunprotected (/home/user/cve-2020-15889/lua/lua+0x2b6bf)
    #9 0x55ef590d23db in luaD_pcall (/home/user/cve-2020-15889/lua/lua+0x303db)
    #10 0x55ef590dc152 in GCTM (/home/user/cve-2020-15889/lua/lua+0x3a152)
```

PoC code recursively calls errfunc, which sets erroneous finalizer(number value 1) to empty table. When a new object is created, Lua internally checks memory usage to determine whether garbage collection is needed. In PoC code, garbage collection functions are called when the empty table is

created in luaV_execute function.

- luaV_execute function(lvm.c:1335)

```
vmcase(OP_NEWTABLE) {
        int b = GETARG_B(i);  /* log2(hash size) + 1 */
        int c = GETARG_C(i);  /* array size */
        Table *t;

        /*생략*/

        checkGC(L, ra + 1);
        vmbreak;
      }
```

During the process, the finalizer that was set previously is called by callallpendingfinalizers function(lgc.c:879). However, as the finalizer is not a callable value(number 1), Lua starts to call auxiliary functions to handle the error. luaD_tryfuncTM -> luaG_typeerror ->varinfo ->getobjname are called in order, and to get information from the object that has created error, getobjname function is called by varinfo function.

- varinfo function(lgc.c:684)

```
static const char *varinfo (lua_State *L, const TValue *o) {
  const char *name = NULL;  /* to avoid warnings */
  CallInfo *ci = L->ci;
  const char *kind = NULL;
  if (isLua(ci)) {
    kind = getupvalname(ci, o, &name);  /* check whether 'o' is an upvalue */
    if (!kind && isinstack(ci, o))  /* no? try a register */
      kind = getobjname(ci_func(ci)->p, currentpc(ci),
                        cast_int(cast(StkId, o) - (ci->func + 1)), &name);
  }
  return (kind) ? luaO_pushfstring(L, " (%s '%s')", kind, name) : "";
}
```

currentpc(ci), which is a second argument to getobjname function is implemented as below.

- currentpc function(ldebug.c:45)

```
static int currentpc (CallInfo *ci) {
  lua_assert(isLua(ci));
  return pcRel(ci->u.l.savedpc, ci_func(ci)->p);
}
```

pcRel is a macro function that subtracts the second argument from the first argument and returns the result – 1. Basically, Callinfo structure saves current pc to u.l.savedpc variable before calling another function, to handle some possible errors happening from callee. However, in PoC code, there is no updating process of u.l.savedpc, so two parameters in pcRel has same value, eventually returns value of -1. This causes out of index error when findsetreg function(line 4) is called.

- findsetreg function(ldebug.c:472)

```
static int findsetreg (const Proto *p, int lastpc, int reg) {
  int pc;
  int setreg = -1;  /* keep last instruction that changed 'reg' */
  int jmptarget = 0;  /* any code before this address is conditional */
  if (testMMMode(GET_OPCODE(p->code[lastpc])))
    lastpc--;  /* previous instruction was not actually executed */

  /*생략*/

  return setreg;
}
```

4. Patch

CVE reference says youngcollection function insufficiently marks objects during garbage collection process, and to fix the problem, youngcollection function was patched to mark more objects. However, this patch doesn't seem to be related to the root cause of the crash, and it may fix another bug in Lua. Instead, github link below shows a correct patch of the problem. This patch saves pc value before garbage collection is called.

https://github.com/lua/lua/commit/31b8c2d4380a762d1ed6a7faee74a1d107f86014

5. Reference

https://github.com/lua/lua/commit/127e7a6c8942b362aa3c6627f44d660a4fb75312

http://lua-users.org/lists/lua-l/2020-07/msg00078.html

http://lua-users.org/lists/lua-l/2020-12/msg00157.html

https://github.com/lua/lua/commit/31b8c2d4380a762d1ed6a7faee74a1d107f86014