CVE-2020-24342 Analysis

Author : Sunghun Oh

1. Overview

Crash type : stack overflow

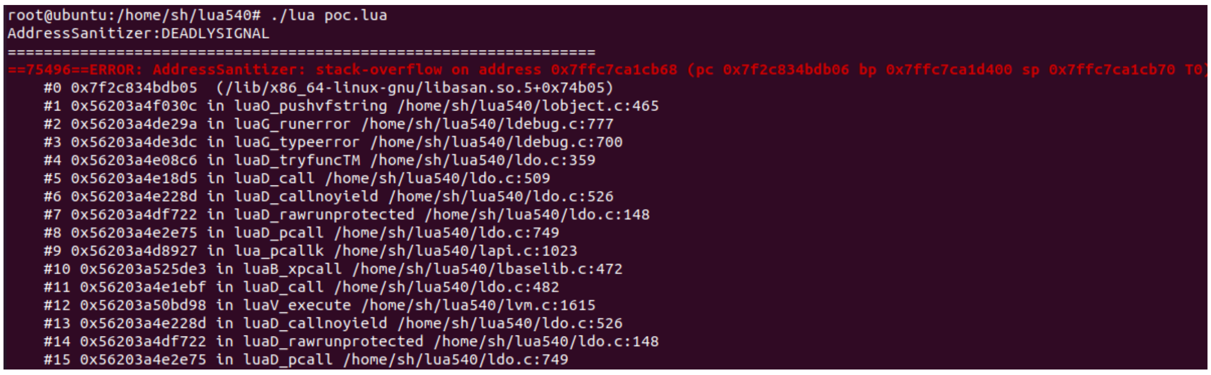
Version : Lua 5.4.0 (git commit hash : c33b1728aeb7dfeec4013562660e07d32697aa6b)

2. PoC Code

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

3. Root Cause Analysis

Lua through 5.4.0 allows a stack redzone cross in 'luaO_pushvfstring' because a protection me chanism wrongly calls luaD_callnoyield twice in a row. Therefore, this crash is caused by insuff icient error processing of stack overflow. When the PoC code is executed with the Lua interpr eter to which the Address sanitizer is applied, the following logs can be checked.



```
root@ubuntu:/home/sh/lua540# ./lua poc.lua
AddressSanitizer:DEADLYSIGNAL
=================================================================
==75496==ERROR: AddressSanitizer: stack-overflow on address 0x7ffc7ca1cb68 (pc 0x7f2c834bdb06 bp 0x7ffc7ca1d400 sp 0x7ffc7ca1cb70 T0
    #0 0x7f2c834bdb05  (/lib/x86_64-linux-gnu/libasan.so.5+0x74b05)
    #1 0x56203a4f030c in luaO_pushvfstring /home/sh/lua540/lobject.c:465
    #2 0x56203a4de29a in luaG_runerror /home/sh/lua540/ldebug.c:777
    #3 0x56203a4de3dc in luaG_typeerror /home/sh/lua540/ldebug.c:700
    #4 0x56203a4e08c6 in luaD_tryfuncTM /home/sh/lua540/ldo.c:359
    #5 0x56203a4e18d5 in luaD_call /home/sh/lua540/ldo.c:509
    #6 0x56203a4e228d in luaD_callnoyield /home/sh/lua540/ldo.c:526
    #7 0x56203a4df722 in luaD_rawrunprotected /home/sh/lua540/ldo.c:148
    #8 0x56203a4e2e75 in luaD_pcall /home/sh/lua540/ldo.c:749
    #9 0x56203a4d8927 in lua_pcallk /home/sh/lua540/lapi.c:1023
    #10 0x56203a525de3 in luaB_xpcall /home/sh/lua540/lbaselib.c:472
    #11 0x56203a4e1ebf in luaD_call /home/sh/lua540/ldo.c:482
    #12 0x56203a50bd98 in luaV_execute /home/sh/lua540/lvm.c:1615
    #13 0x56203a4e228d in luaD_callnoyield /home/sh/lua540/ldo.c:526
    #14 0x56203a4df722 in luaD_rawrunprotected /home/sh/lua540/ldo.c:148
    #15 0x56203a4e2e75 in luaD_pcall /home/sh/lua540/ldo.c:749
```

In the 'luaD_callnoyield' function, if there is a possibility of stack overflow, the number of Call Info is set to zero because it is forcibly inspected when calling the function. However, if "func tion" is not a function, the code causes an error before checking the stack. Then, since the e rror handling call calls again luaD_callnoyield and nCalls decrease again to pass through the s tack red zone(limit) without causing a stack overflow error, stack overflow check logic internall y occurs through bypass.

The function 'luaD_callnoyield' is as follows.

```c
void luaD_callnoyield (lua_State *L, StkId func, int nResults) {
    incXCcalls(L);
    if (getCcalls(L) <= CSTACKERR) /* possible stack overflow? */
        luaE_freeCI(L);
    luaD_call(L, func, nResults);
    decXCcalls(L);}
```

ldo.c:522 - luaD_callnoyield

The function 'luaE_freeCI' is called in the if statement checking Stack overflow, and the function in formation is as follows.

```c
void luaE_freeCI (lua_State *L) {
    CallInfo *ci = L->ci;
    CallInfo *next = ci->next;
    ci->next = NULL;
    L->nCcalls += L->nci; /* add removed elements back to 'nCcalls' */
    while ((ci = next) != NULL) {
        next = ci->next;
        luaM_free(L, ci);
        L->nci--;
    }
    L->nCcalls -= L->nci; /* adjust result */
}
```

lstate.c:174 - luaE_freeCI

Since the stack overflow is forcibly inspected in this function, the number of CallInfo is forcibl y zeroed. Also, Since error handler is a function in the while statement of this function, L→nci is reduced only once, but it is enough to skip the red zone.

4. Patch

The code that checks the existing stack overflow in the 'luaD_callnoyield' function called the 'l uaE_freeCI' function to free CallInfo and to decrease L→nci as soon as the stack overflow occur s. However, this function was identified as the cause of the problem and was patched to inv oke the 'luaE_enterCall' function that properly checks the 'luaE_exitCall' function and stack ove rflow to recover the reduction of L→nci in the next function.

Detailed code patches can be found in the link below.

https://github.com/lua/lua/commit/34affe7a63fc5d842580a9f23616d057e17dfe27

5. Reference

http://lua-users.org/lists/lua-l/2020-07/msg00052.html
https://github.com/lua/lua/commit/34affe7a63fc5d842580a9f23616d057e17dfe27