luanti-org / luanti Public

Code Issues 1.3k Pull requests 92 Actions Projects Security and qu

Commit 53cef18

SmallJoker authored and **sfan5** committed yesterday

Lua API: Sanitize the environment of safe functions (#16985)

stable-5 · 5.15.2

1 parent 4308f18 commit 53cef18

1 file changed +105 -42 lines changed

Top

Filter files…

src/script/cpp_api

s_security.cpp

1 file changed +105 -42 lines changed

Search within code

src/script/cpp_api/s_security.cpp

```
@@ -23,29 +23,63 @@
23 23      lua_setfield(L, -2, #name);
24 24
25 25
26    - static inline void copy_safe(lua_State *L, const char *list[], unsigned len,
      - int from=-2, int to=-1)
   26 + // Do not use directly. This is a helper function for `copy_safe`!
   27 + // Note: Circular references are not handled. `t_global` must be absolute.
   28 + static void recursive_copy(lua_State *L, int idx, int t_global)
29 29   {
30    -     if (from < 0) from = lua_gettop(L) + from + 1;
31    -     if (to   < 0) to   = lua_gettop(L) + to   + 1;
32    -     for (unsigned i = 0; i < (len / sizeof(list[0])); i++) {
33    -         lua_getfield(L, from, list[i]);
```

```
32      -         lua_setfield(L, to,   list[i]);
        30  +     if (idx < 0) idx = lua_gettop(L) + idx + 1;
        31  +
        32  +     switch (lua_type(L, idx)) {
        33  +     case LUA_TTABLE:
        34  +         {
        35  +             lua_newtable(L);
        36  +             const int to = lua_gettop(L);
        37  +             for (lua_pushnil(L); lua_next(L, idx); lua_pop(L, 1)) {
        38  +                 // Ensure the key can stay unmodified
        39  +                 switch (lua_type(L, -2)) {
        40  +                 case LUA_TNUMBER:
        41  +                 case LUA_TSTRING:
        42  +                     // accepted
        43  +                     break;
        44  +                 default:
        45  +                     luaL_error(L, "unhandled type in recursive_copy");
        46  +                     break;
        47  +                 }
        48  +
        49  +                 recursive_copy(L, -1, t_global); // value
        50  +
        51  +                 lua_pushvalue(L, -2);
        52  +                 lua_pushvalue(L, -2);
        53  +                 lua_rawset(L, to);
        54  +             }
        55  +             lua_replace(L, idx);
        56  +         }
        57  +         break;
        58  +     case LUA_TFUNCTION:
        59  +         lua_pushvalue(L, t_global);
        60  +         lua_setfenv(L, idx);
        61  +         break;
        62  +     default:
        63  +         // keep value as-is
        64  +         break;
33      65          }
34      66      }
35      67
36      -   static void shallow_copy_table(lua_State *L, int from=-2, int to=-1)
```

```
      68 + static inline void copy_safe(lua_State *L, const char *list[], unsigned len,
         int from=-2, int to=-1)
37    69  {
38    70      if (from < 0) from = lua_gettop(L) + from + 1;
39    71      if (to   < 0) to   = lua_gettop(L) + to   + 1;
40        -   lua_pushnil(L);
41        -   while (lua_next(L, from) != 0) {
42        -       assert(lua_type(L, -1) != LUA_TTABLE);
43        -       // duplicate key and value for lua_rawset
44        -       lua_pushvalue(L, -2);
45        -       lua_pushvalue(L, -2);
46        -       lua_rawset(L, to);
47        -       lua_pop(L, 1);
      72 +
      73 +   lua_pushvalue(L, LUA_GLOBALSINDEX);
      74 +   const int t_sec = lua_gettop(L);
      75 +
      76 +   for (unsigned i = 0; i < (len / sizeof(list[0])); i++) {
      77 +       lua_getfield(L, from, list[i]);
      78 +       recursive_copy(L, -1, t_sec);
      79 +       lua_setfield(L, to, list[i]);
48    80      }
      81 +
      82 +   lua_pop(L, 1);
49    83  }
50    84
51    85  // Pushes the original version of a library function on the stack, from the old
         version
          @@ -59,6 +93,11 @@ static inline void push_original(lua_State *L, const char
          *lib, const char *func
59    93      lua_remove(L, -2);  // Remove lib
60    94  }
61    95
      96 + static int l_nop(lua_State *L)
      97 + {
      98 +   return 0;
      99 + }
     100 +
62   101
63   102  void ScriptApiSecurity::initializeSecurity()
```

```
64  103     {
        @@ -88,8 +127,9 @@ void ScriptApiSecurity::initializeSecurity()
88  127             "unpack",
89  128             "_VERSION",
90  129             "xpcall",
91      -       };
92      -       static const char *whitelist_tables[] = {
    130 +
    131 +           // ******** Tables whitelist ********
    132 +
93  133             // These libraries are completely safe BUT we need to duplicate their
        table
94  134             // to ensure the sandbox can't affect the insecure env
95  135             "coroutine",
        @@ -130,7 +170,6 @@ void ScriptApiSecurity::initializeSecurity()
130 170             "path",
131 171             "searchpath",
132 172         };
133     - #if USE_LUAJIT
134 173         static const char *jit_whitelist[] = {
135 174             "arch",
136 175             "flush",
        @@ -142,23 +181,57 @@ void ScriptApiSecurity::initializeSecurity()
142 181             "version",
143 182             "version_num",
144 183         };
145     - #endif
    184 +
146 185         m_secure = true;
147 186
148 187         lua_State *L = getStack();
    188 +       const int sanity_check_top = lua_gettop(L);
149 189
150     -       // Backup globals to the registry
151     -       lua_getglobal(L, "_G");
152     -       lua_rawseti(L, LUA_REGISTRYINDEX, CUSTOM_RIDX_GLOBALS_BACKUP);
    190 +       /*
    191 +           This function creates a secondary table, only accessible through
```

```
192 + `core.request_insecure_environment` containing all insecure Lua functions.
193 +
194 + 1. Create a new table for the secure env.
195 + 2. Replace the main thread env (LUA_GLOBALSINDEX) to use the secure env.
196 + 3. Replace the environment of secure C functions (LUA_ENVIRONINDEX).
197 + */
198 +
199 + // Insecure env (stack + 1)
200 + lua_pushvalue(L, LUA_GLOBALSINDEX);
201 + const int old_globals = lua_gettop(L);
202 + {
203 + // For later use in secured functions and the insecure env
204 + lua_pushvalue(L, old_globals);
205 + lua_rawseti(L, LUA_REGISTRYINDEX, CUSTOM_RIDX_GLOBALS_BACKUP);
206 + }
153 207
154 - // Replace the global environment with an empty one
155 - int thread = getThread(L);
208 + // Secure env
156 209 createEmptyEnv(L);
157 - setLuaEnv(L, thread);
210 + {
211 + // Perform a full switch of the Lua state to the secure env
212 + const int idx_secure = lua_gettop(L);
158 213
159 - // Get old globals
160 - lua_rawgeti(L, LUA_REGISTRYINDEX, CUSTOM_RIDX_GLOBALS_BACKUP);
161 - int old_globals = lua_gettop(L);
214 + int thread = getThread(L);
215 + lua_pushvalue(L, idx_secure);
216 + setLuaEnv(L, thread);
217 + }
218 + lua_pop(L, 1);
219 +
220 + {
221 + // Security check
222 + lua_rawgeti(L, LUA_REGISTRYINDEX, CUSTOM_RIDX_GLOBALS_BACKUP);
223 + luaL_checktype(L, -1, LUA_TTABLE);
```

```
                +
                +         lua_getglobal(L, "_G");
                +         luaL_checktype(L, -1, LUA_TTABLE);
                +         FATAL_ERROR_IF(lua_rawequal(L, -1, -2), "insecure _G");
                +
                +         lua_pushcfunction(L, l_nop);
                +         lua_getfenv(L, -1);
                +         FATAL_ERROR_IF(lua_rawequal(L, -1, -4), "insecure env");
                +
                +         lua_pop(L, 4);
                +     }
162   235
163   236
164   237         // Copy safe base functions
                  @@ -174,17 +247,6 @@ void ScriptApiSecurity::initializeSecurity()
174   247         lua_pop(L, 1);
175   248
176   249
177         -     // Copy safe libraries
178         -     for (const char *libname : whitelist_tables) {
179         -         lua_getfield(L, old_globals, libname);
180         -         lua_newtable(L);
181         -         shallow_copy_table(L);
182         -
183         -         lua_setglobal(L, libname);
184         -         lua_pop(L, 1);
185         -     }
186         -
187         -
188   250         // Copy safe IO functions
189   251         lua_getfield(L, old_globals, "io");
190   252         lua_newtable(L);
                  @@ -233,7 +295,7 @@ void ScriptApiSecurity::initializeSecurity()
233   295         lua_setglobal(L, "package");
234   296         lua_pop(L, 1);  // Pop old package
235   297
236         - #if USE_LUAJIT
        298   +
237   299         // Copy safe jit functions, if they exist
```

```
238  300         lua_getfield(L, -1, "jit");
239  301         if (!lua_isnil(L, -1)) {
     @@ -242,7 +304,7 @@ void ScriptApiSecurity::initializeSecurity()
242  304             lua_setglobal(L, "jit");
243  305         }
244  306         lua_pop(L, 1);  // Pop old jit
245       - #endif
     307 +
246  308
247  309         // Get rid of 'core' in the old globals, we don't want anyone thinking it's
248  310         // safe or even usable.
     @@ -263,6 +325,8 @@ void ScriptApiSecurity::initializeSecurity()
263  325         lua_setfield(L, -2, "__index");
264  326         lua_setmetatable(L, -2);
265  327         lua_pop(L, 1); // Pop empty string
     328 +
     329 +     FATAL_ERROR_IF(sanity_check_top != lua_gettop(L), "unbalanced stack");
266  330     }
267  331
268  332     #if CHECK_CLIENT_BUILD()
     @@ -275,7 +339,6 @@ void ScriptApiSecurity::initializeSecurityClient()
275  339             "collectgarbage",
276  340             "DIR_DELIM",
277  341             "error",
278       -         "getfenv",
279  342             "ipairs",
280  343             "next",
281  344             "pairs",
```

Comments 0

Please sign in to comment.