luanti-org / luanti Public

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

Commit 8a929df

SmallJoker authored last month · 19 / 19 · Verified

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

master (#16985)

1 parent b3de906 commit 8a929df

2 files changed +105 -45 lines changed

Top

Filter files…

- doc
  - sscsm_api.md
- src/script/cpp_api
  - s_security.cpp

2 files changed +105 -45 lines changed

Search within code

doc/sscsm_api.md

```
@@ -150,7 +150,6 @@ Functions that take or return paths always use virtual paths.
150 150   * `assert`
151 151   * `collectgarbage`
152 152   * `error`
153       - * `getfenv`
154 153   * `ipairs`
155 154   * `next`
156 155   * `pairs`
@@ -160,7 +159,6 @@ Functions that take or return paths always use virtual paths.
```

```
160 159   * `rawget`
161 160   * `rawset`
162 161   * `select`
163       - * `setfenv`
164 162   * `getmetatable`
165 163   * `setmetatable`
166 164   * `tonumber`
```

src/script/cpp_api/s_security.cpp

```
          @@ -25,29 +25,63 @@
25  25        lua_setfield(L, -2, #name);
26  26
27  27
28        - static inline void copy_safe(lua_State *L, const char *list[], unsigned len,
          int from=-2, int to=-1)
      28  + // Do not use directly. This is a helper function for `copy_safe`!
      29  + // Note: Circular references are not handled. `t_global` must be absolute.
      30  + static void recursive_copy(lua_State *L, int idx, int t_global)
29  31    {
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]);
34        -         lua_setfield(L, to,   list[i]);
      32  +     if (idx < 0) idx = lua_gettop(L) + idx + 1;
      33  +
      34  +     switch (lua_type(L, idx)) {
      35  +     case LUA_TTABLE:
      36  +         {
      37  +             lua_newtable(L);
      38  +             const int to = lua_gettop(L);
      39  +             for (lua_pushnil(L); lua_next(L, idx); lua_pop(L, 1)) {
      40  +                 // Ensure the key can stay unmodified
      41  +                 switch (lua_type(L, -2)) {
      42  +                 case LUA_TNUMBER:
      43  +                 case LUA_TSTRING:
      44  +                     // accepted
      45  +                     break;
      46  +                 default:
```

```
+                    luaL_error(L, "unhandled type in recursive_copy");
+                    break;
+                }
+
+                recursive_copy(L, -1, t_global); // value
+
+                lua_pushvalue(L, -2);
+                lua_pushvalue(L, -2);
+                lua_rawset(L, to);
+            }
+            lua_replace(L, idx);
+        }
+        break;
+    case LUA_TFUNCTION:
+        lua_pushvalue(L, t_global);
+        lua_setfenv(L, idx);
+        break;
+    default:
+        // keep value as-is
+        break;
     }
 }

-static void shallow_copy_table(lua_State *L, int from=-2, int to=-1)
+static inline void copy_safe(lua_State *L, const char *list[], unsigned len,
int from=-2, int to=-1)
 {
     if (from < 0) from = lua_gettop(L) + from + 1;
     if (to   < 0) to   = lua_gettop(L) + to   + 1;
-    lua_pushnil(L);
-    while (lua_next(L, from) != 0) {
-        assert(lua_type(L, -1) != LUA_TTABLE);
-        // duplicate key and value for lua_rawset
-        lua_pushvalue(L, -2);
-        lua_pushvalue(L, -2);
-        lua_rawset(L, to);
-        lua_pop(L, 1);
+
+    lua_pushvalue(L, LUA_GLOBALSINDEX);
+    const int t_sec = lua_gettop(L);
```

```
                77 +
                78 +     for (unsigned i = 0; i < (len / sizeof(list[0])); i++) {
                79 +         lua_getfield(L, from, list[i]);
                80 +         recursive_copy(L, -1, t_sec);
                81 +         lua_setfield(L, to, list[i]);
50              82           }
                83 +
                84 +     lua_pop(L, 1);
51              85       }
52              86
53              87       // Pushes the original version of a library function on the stack, from the old
                         version
                         @@ -61,6 +95,11 @@ static inline void push_original(lua_State *L, const char
                         *lib, const char *func
61              95           lua_remove(L, -2);  // Remove lib
62              96       }
63              97
                98 + static int l_nop(lua_State *L)
                99 + {
               100 +     return 0;
               101 + }
               102 +
64             103
65             104       void ScriptApiSecurity::initializeSecurity()
66             105       {
                         @@ -90,8 +129,9 @@ void ScriptApiSecurity::initializeSecurity()
90             129           "unpack",
91             130           "_VERSION",
92             131           "xpcall",
93                   -     };
94                   -     static const char *whitelist_tables[] = {
               132 +
               133 +         // ******** Tables whitelist ********
               134 +
95             135           // These libraries are completely safe BUT we need to duplicate their
                         table
96             136           // to ensure the sandbox can't affect the insecure env
97             137           "coroutine",
                         @@ -132,7 +172,6 @@ void ScriptApiSecurity::initializeSecurity()
```

```diff
 			"path",
 			"searchpath",
 		};
-#if USE_LUAJIT
 		static const char *jit_whitelist[] = {
 			"arch",
 			"flush",
@@ -144,23 +183,57 @@ void ScriptApiSecurity::initializeSecurity()
 			"version",
 			"version_num",
 		};
-#endif
+
 	m_secure = true;
 
 	lua_State *L = getStack();
+	const int sanity_check_top = lua_gettop(L);
 
-	// Backup globals to the registry
-	lua_getglobal(L, "_G");
-	lua_rawseti(L, LUA_REGISTRYINDEX, CUSTOM_RIDX_GLOBALS_BACKUP);
+	/*
+		This function creates a secondary table, only accessible through
+		`core.request_insecure_environment` containing all insecure Lua functions.
+
+		1. Create a new table for the secure env.
+		2. Replace the main thread env (LUA_GLOBALSINDEX) to use the secure env.
+		3. Replace the environment of secure C functions (LUA_ENVIRONINDEX).
+	*/
+
+	// Insecure env (stack + 1)
+	lua_pushvalue(L, LUA_GLOBALSINDEX);
+	const int old_globals = lua_gettop(L);
+	{
+		// For later use in secured functions and the insecure env
+		lua_pushvalue(L, old_globals);
+		lua_rawseti(L, LUA_REGISTRYINDEX, CUSTOM_RIDX_GLOBALS_BACKUP);
```

```
                208 +     }
155             209
156             -     // Replace the global environment with an empty one
157             -     int thread = getThread(L);
                210 +     // Secure env
158             211       createEmptyEnv(L);
159             -     setLuaEnv(L, thread);
                212 +     {
                213 +         // Perform a full switch of the Lua state to the secure env
                214 +         const int idx_secure = lua_gettop(L);
160             215
161             -     // Get old globals
162             -     lua_rawgeti(L, LUA_REGISTRYINDEX, CUSTOM_RIDX_GLOBALS_BACKUP);
163             -     int old_globals = lua_gettop(L);
                216 +         int thread = getThread(L);
                217 +         lua_pushvalue(L, idx_secure);
                218 +         setLuaEnv(L, thread);
                219 +     }
                220 +     lua_pop(L, 1);
                221 +
                222 +     {
                223 +         // Security check
                224 +         lua_rawgeti(L, LUA_REGISTRYINDEX, CUSTOM_RIDX_GLOBALS_BACKUP);
                225 +         luaL_checktype(L, -1, LUA_TTABLE);
                226 +
                227 +         lua_getglobal(L, "_G");
                228 +         luaL_checktype(L, -1, LUA_TTABLE);
                229 +         FATAL_ERROR_IF(lua_rawequal(L, -1, -2), "insecure _G");
                230 +
                231 +         lua_pushcfunction(L, l_nop);
                232 +         lua_getfenv(L, -1);
                233 +         FATAL_ERROR_IF(lua_rawequal(L, -1, -4), "insecure env");
                234 +
                235 +         lua_pop(L, 4);
                236 +     }
164             237
165             238
166             239       // Copy safe base functions
        @@ -176,17 +249,6 @@ void ScriptApiSecurity::initializeSecurity()
176             249       lua_pop(L, 1);
```

```
177  250
178  251
179       -      // Copy safe libraries
180       -      for (const char *libname : whitelist_tables) {
181       -          lua_getfield(L, old_globals, libname);
182       -          lua_newtable(L);
183       -          shallow_copy_table(L);
184       -
185       -          lua_setglobal(L, libname);
186       -          lua_pop(L, 1);
187       -      }
188       -
189       -
190  252          // Copy safe IO functions
191  253          lua_getfield(L, old_globals, "io");
192  254          lua_newtable(L);
          @@ -235,7 +297,7 @@ void ScriptApiSecurity::initializeSecurity()
235  297          lua_setglobal(L, "package");
236  298          lua_pop(L, 1);  // Pop old package
237  299
238       - #if USE_LUAJIT
       300 +
239  301          // Copy safe jit functions, if they exist
240  302          lua_getfield(L, -1, "jit");
241  303          if (!lua_isnil(L, -1)) {
          @@ -244,7 +306,7 @@ void ScriptApiSecurity::initializeSecurity()
244  306              lua_setglobal(L, "jit");
245  307          }
246  308          lua_pop(L, 1);  // Pop old jit
247       - #endif
       309 +
248  310
249  311          // Get rid of 'core' in the old globals, we don't want anyone thinking it's
250  312          // safe or even usable.
          @@ -265,6 +327,8 @@ void ScriptApiSecurity::initializeSecurity()
265  327          lua_setfield(L, -2, "__index");
266  328          lua_setmetatable(L, -2);
267  329          lua_pop(L, 1); // Pop empty string
       330 +
```

```
     331 +      FATAL_ERROR_IF(sanity_check_top != lua_gettop(L), "unbalanced stack");
268  332    }
269  333
270  334    #if CHECK_CLIENT_BUILD()
          @@ -277,7 +341,6 @@ void ScriptApiSecurity::initializeSecurityClient()
277  341            "collectgarbage",
278  342            "DIR_DELIM",
279  343            "error",
280       -          "getfenv",
281  344            "ipairs",
282  345            "next",
283  346            "pairs",
          @@ -404,7 +467,6 @@ void ScriptApiSecurity::initializeSecuritySSCSM()
404  467            "collectgarbage",
405  468            "DIR_DELIM",
406  469            "error",
407       -          "getfenv",
408  470            "ipairs",
409  471            "next",
410  472            "pairs",
```

Comments 0

Please sign in to comment.