Bugs

5.4 · 5.3 · 5.2 · 5.1 · 5.0 · 4.0

Here is a list of all bugs found in each release of Lua since 4.0.

5.4: 5.4.4 · 5.4.3 · 5.4.2 · 5.4.1 · 5.4.0
5.3: 5.3.6 · 5.3.5 · 5.3.4 · 5.3.3 · 5.3.2 · 5.3.1 · 5.3.0
5.2: 5.2.4 · 5.2.3 · 5.2.2 · 5.2.1 · 5.2.0
5.1: 5.1.5 · 5.1.4 · 5.1.3 · 5.1.2 · 5.1.1 · 5.1
5.0: 5.0.3 · 5.0.2 · 5.0
4.0: 4.0

Each entry includes a minimal example that exhibits the bug and a patch or solution, when available.

Every Lua release fixes all listed bugs in previous releases, except where noted. Nevertheless, some bugs found in recent releases actually exist since older releases.

If you want to report a bug, please read this first.

❖ Lua 5.4.4 1 · 2 · 3 · 4 · 5 · 6 · 7

1. lua.c assumes that argv has at least one element.
   reported by DoubleF on 27 Jan 2022. existed since 5.0 (at least). fixed in github.

2. Lua can generate wrong code when _ENV is <const>.
   reported by Кныжов Никита on 03 Feb 2022. existed since 5.4.2. fixed in github.

   Example:

```
-- Lua compiled with assertions on
local _ENV <const> = 0
X=0
```

3. Wrong code generation for constants in bitwise operations.
   reported by bmcq on 30 Mar 2022. existed since 5.4.0. fixed in github.

   Example:

```
local _ = {"1","2","3","4","5","6","7","8","9","10","11","12",
"13","14","15","16","17","18","19","20","21","22","23","24","25","26",
"27","28","29","30","31","32","33","34","35","36","37","38",
"39","40","41","42","43","44","45","46","47","48","49","50",
"51","52","53","54","55","56","57","58","59","60","61","62","63","64",
"65","66","67","68","69","70","71","72","73","74","75","76","77","78",
"79","80","81","82","83","84","85","86","87","88","89","90","91","92",
"93","94","95","96","97","98","99","100","101","102","103","104",
"105","106","107","108","109","110","111","112","113","114",
"115","116","117","118","119","120","121","122","123","124",
"125","126","127","128","129","130","131","132","133","134",
"135","136","137","138","139","140","141","142","143","144",
"145","146","147","148","149","150","151","152","153","154",
"155","156","157","158","159","160","161","162","163","164",
"165","166","167","168","169","170","171","172","173","174",
"175","176","177","178","179","180","181","182","183","184",
"185","186","187","188","189","190","191","192","193","194",
"195","196","197","198","199","200","201","202","203","204",
"205","206","207","208","209","210","211","212","213","214",
"215","216","217","218","219","220","221","222","223","224",
"225","226","227","228","229","230","231","232","233","234",
"235","236","237","238","239","240","241","242","243","244",
"245","246","247","248","249","250","251","252","253","254",
"255","256", }
-- should print 3, but prints 2
print (1 | (2 or 3))
```

4. Lua-stack overflow when C stack overflows while handling an error.
   reported by Jinwei Dong on 13 May 2022. existed since 5.4.2. fixed in github.

   Example:

```
print(
    xpcall((0),
        function(...)
            local f
            if d[print(print(print(print(t[...]))))] then
            end
        end
    )
)
```

5. 'lua_settop' may use a pointer to stack invalidated by 'luaF_close'.
   reported by jun louis on 23 May 2022. existed since 5.4.3. fixed in github.

   Example:

```
-- (needs test library)

-- reduce stack size
collectgarbage(); collectgarbage(); collectgarbage()

-- force a stack reallocation
local function loop (n)
  if n < 400 then loop(n + 1) end
end

local o = setmetatable({}, {__close = function () loop(0) end})

local script = [[toclose 2; settop 1; return 1]]

assert(T.testC(script, o) == script)
```

6. 'break' may not properly close variable in a 'for' loop.
   reported by Xmilia Hermit on 18 May 2022. existed since 5.4.0.

   Example:

```
local closed = false

local o1 = setmetatable({}, {__close=function() closed = true end})

local function test()
    for k, v in next, {}, nil, o1 do
      local function f() return k end   -- just to use 'k'
      break
    end
  assert(closed)   -- should be closed here
end

test()
```

7. GC not setting a proper target for next cycle after a full collection in generational mode.
   reported by 最萌 小汐 on 15 Jul 2022. existed since 5.4.3. fixed in github.

❖ Lua 5.4.3 1 · 2 · 3 · 4 · 5 · 6 · 7 · 8 · 9 · 11 · 10 · 11 · 12

1. To-be-closed variables in "for" loops are not avoiding tail calls.
   reported by Xmilia Hermit on 31 Mar 2021. existed since 5.4.3. fixed in github.

   Example:

```
local function iter ()
  return function () return true end, 0, 0,
         setmetatable({}, {__close = print})
end

local function tail() print("tail") end

local function foo ()
  for k in iter() do return tail() end
end

foo()
```

2. C99 comments ("//") are not compatible with C89.
reported by Olexa Bilaniuk on 09 Apr 2021. existed since 5.4.3. fixed in github.

Patch:

```
lvm.c:
@@ -1156,8 +1156,10 @@ void luaV_execute (lua_State *L, CallInfo *ci) {
     Instruction i;  /* instruction being executed */
     StkId ra;  /* instruction's A register */
     vmfetch();
-// low-level line tracing for debugging Lua
-// printf("line: %d\n", luaG_getfuncline(cl->p, pcRel(pc, cl->p)));
+    #if 0
+      /* low-level line tracing for debugging Lua */
+      printf("line: %d\n", luaG_getfuncline(cl->p, pcRel(pc, cl->p)));
+    #endif
     lua_assert(base == ci->func + 1);
     lua_assert(base <= L->top && L->top < L->stack_last);
     /* invalidate top for instructions not expecting it */
```

3. Yielding in a __close metamethod called when returning vararg results mess up the returned values.
reported by Xmilia Hermit on 07 Apr 2021. existed since 5.4.3. fixed in github.

Example:

```
-- The final 'print' should print nothing, as the 'print' inside 'test'
-- returns nothing.
local function test()
    local x <close> = setmetatable({}, {
        __close = coroutine.yield
    })
    return print("Return")
end

local c = coroutine.wrap(test)
c()          -- runs until '__close'
print(c())   -- runs until end
```

Patch:

```
lvm.c:
@@ -847,10 +847,19 @@ void luaV_finishOp (lua_State *L) {
       luaV_concat(L, total);  /* concat them (may yield again) */
       break;
     }
-    case OP_CLOSE:  case OP_RETURN: {  /* yielded closing variables */
+    case OP_CLOSE: {  /* yielded closing variables */
       ci->u.l.savedpc--;  /* repeat instruction to close other vars. */
       break;
     }
+    case OP_RETURN: {  /* yielded closing variables */
+      StkId ra = base + GETARG_A(inst);
+      /* correct top to signal correct number of returns (in case the
+         return is "in top" */
+      L->top = ra + ci->u2.nres;
+      /* repeat instruction to close other vars. and complete the return */
+      ci->u.l.savedpc--;
+      break;
+    }
     default: {
       /* only these other opcodes can yield */
       lua_assert(op == OP_TFORCALL || op == OP_CALL ||
@@ -1672,6 +1681,7 @@ void luaV_execute (lua_State *L, CallInfo *ci) {
           n = cast_int(L->top - ra);  /* get what is available */
         savepc(ci);
         if (TESTARG_k(i)) {  /* may there be open upvalues? */
+          ci->u2.nres = n;  /* save number of returns */
           if (L->top < ci->top)
             L->top = ci->top;
           luaF_close(L, base, CLOSEKTOP, 1);
```

4. Function-statement syntax does not check whether name is a constant.
reported by Halalaluyafail3 on 16 Jun 2021. existed since 5.4.0. fixed in github.

Example:

```
local x<const> = {}
function x() end    -- should raise an error
print(x)
```

5. 'luaL_tolstring' may get confused with negative indices.
   reported by Xmilia Hermit on 22 Jul 2021. existed since 5.4.0. fixed in github.

   Example:

```
-- (must be run in interactive mode)
-- Both prints should show the same result
> debug.debug()
lua_debug> x = setmetatable({}, {__name="TABLE"})
lua_debug> print(x)
lua_debug> error(x)
```

   Patch:

```
lauxlib.c:
@@ -881,6 +881,7 @@ LUALIB_API lua_Integer luaL_len (lua_State *L, int idx) {


 LUALIB_API const char *luaL_tolstring (lua_State *L, int idx, size_t *len) {
+  idx = lua_absindex(L,idx);
   if (luaL_callmeta(L, idx, "__tostring")) {  /* metafield? */
     if (!lua_isstring(L, -1))
       luaL_error(L, "'__tostring' must return a string");
```

6. Negation in macro 'luaV_shiftr' may overflow.
   reported by user 673679 on 22 Jul 2021. existed since 5.3-alpha. fixed in github.

   Example:

```
-- Lua compiled in gcc with option '-fsanitize=undefined'
print(1 >> math.mininteger)
```

   Patch:

```
lvm.c:
@@ -766,7 +766,7 @@ lua_Number luaV_modf (lua_State *L, lua_Number m, lua_Number n) {
 /*
 ** Shift left operation. (Shift right just negates 'y'.)
 */
-#define luaV_shiftr(x,y)       luaV_shiftl(x,-(y))
+#define luaV_shiftr(x,y)       luaV_shiftl(x,intop(-, 0, y))

 lua_Integer luaV_shiftl (lua_Integer x, lua_Integer y) {
   if (y < 0) {  /* shift right? */
```

7. 'coroutine.resume' does not increment counter of C calls when continuing execution after a protected error.
   reported by Jihoi Kim on 29 Oct 2021. existed since 5.4.2. fixed in github.

   Example:

```
local function func()
  pcall(1)
  coroutine.wrap(func)()
end
func()
```

8. Wrong status in coroutine while closing variables during reset.
   reported by MinSeok Kang on 30 Oct 2021. existed since 5.4.1. fixed in github.

   Example:

```
-- Must be run with assertions on, valgrind, or sanitizer
co = coroutine.wrap(
  function()
    local x <close> = setmetatable(
      {}, {__close = function() pcall(co) end}
    )
    error()
  end
)
co()
```

9. Lua stack still active when closing a state.
   reported by Kang woosun on 30 Nov 2021. existed since 5.4.3. fixed in github.

   Example:

```
-- The following chunk may segfault
function v (...)
  return os.exit(0, true)
end

local x <close> = setmetatable({}, {__close = error})

v()
```

10. Finalizers should not be able to invoke the GC.
    reported by 김지회 on 29 Nov 2021. existed since 5.4.0. fixed in github.

    Example:

```
function func1 ()
  local f = setmetatable({}, {__gc = function () collectgarbage("step") end})
  collectgarbage("step" , 1)
end


for i = 0,1000 do
  setmetatable({}, {__gc = func1})
end
```

11. Finalizers can be called with an invalid stack.
    reported by Minseok Kang on 06 Dec 2021. existed since 5.4.3. fixed in github.

    Example:

```
local x = {}; for i=1, 2000 do x[i] = i end

local function f()  end

local function g() return f(table.unpack(x)) end

collectgarbage("step")
setmetatable({}, {__gc = 1})

g()
```

12. Finalizer calling os.exit can corrupt finalization order.
    reported by Xmilia Hermit on 21 Dec 2021. existed since 5.2. fixed in github.

    Example:

```
-- The following code illustrates the problem. If finalizer 3 calls
-- a function from a dynamically loaded C module, the C module
-- will be closed by the time the function is called, generating
-- a seg. fault.

-- should be called last
print("creating 1")
setmetatable({}, {__gc = function () print(1) end})

print("creating 2")
setmetatable({}, {__gc = function ()
  print("2")
  print("creating 3")
  setmetatable({}, {__gc = function () print(3) end})
  os.exit(1, true)
end})
```

❖ Lua 5.4.2 1 · 2 · 3 · 4

1. 'table.sort' does not work for partial orders.
   reported by Egor Skriptunof on 04 Jan 2021. existed since 5.3. fixed in github.

   Example:

```
nan = 0/0
t = {nan, nan, 20, 10}
table.sort(t)
print(table.concat(t, ", "))
  --> -nan, 20, -nan, 10
```

Patch: The manual is deceptive. It is necessary but not sufficient for the sort function to define a partial order.

2. Parameter 'what' of 'debug.getinfo' cannot start with '>'.
reported by Xmilia Hermit on 01 Feb 2021. existed since 5.1. fixed in github.

Example:

```
-- with Lua compiled with option LUA_USE_APICHECK
debug.getinfo(0, ">")
```

Patch:

```
ldblib.c:
@@ -152,6 +152,7 @@ static int db_getinfo (lua_State *L) {
   lua_State *L1 = getthread(L, &arg);
   const char *options = luaL_optstring(L, arg+2, "flnSrtu");
   checkstack(L, L1, 3);
+  luaL_argcheck(L, options[0] != '>', arg + 2, "invalid option '>'");
   if (lua_isfunction(L, arg + 1)) {  /* info about a function? */
     options = lua_pushfstring(L, ">%s", options);  /* add '>' to 'options' */
     lua_pushvalue(L, arg + 1);  /* move function to 'L1' stack */
```

3. Error message in 'string.concat' uses wrong format.
reported by no-n and Andrew Gierth on 14 Feb 2021. existed since 5.3.0. fixed in github.

Example:

```
-- the following call gives an error message with a wrong index
table.concat({}, "", math.maxinteger, math.maxinteger)
```

Patch:

```
ltablib.c:
@@ -146,7 +146,7 @@ static int tmove (lua_State *L) {
 static void addfield (lua_State *L, luaL_Buffer *b, lua_Integer i) {
   lua_geti(L, 1, i);
   if (!lua_isstring(L, -1))
-    luaL_error(L, "invalid value (%s) at index %d in table for 'concat'",
+    luaL_error(L, "invalid value (%s) at index %I in table for 'concat'",
                   luaL_typename(L, -1), i);
   luaL_addvalue(b);
 }
```

4. 'isinstack' wrongly assumes we can work around an undefined behavior.
reported by Yongheng Chen on 21 Feb 2021. existed since 5.3.0. fixed in github.

Example: This bug probably will not cause a failure in flat-memory architectures. We can force it by compiling Lua with the gcc option '-fsanitize=pointer-subtract' (plus what it needs to work) and running the following code:

```
print(setmetatable({}, {__index = 4}).x)
```

❖ Lua 5.4.1 1

1. Key removed from a table during traversal may not be accepted by 'next'.
reported by Xmilia Hermit on 11 Oct 2020. existed since 5.4.0. fixed in 5.4.2. fixed in github.

Example:

```
t = {}
t["no" .. "ref1"] = 1
t["no" .. "ref2"] = 2

for k, v in pairs(t) do
    t[k] = nil
    print(k, v)
    collectgarbage("collect")
end
```

❖ Lua 5.4.0 1 · 2 · 3 · 4 · 5 · 6 · 7 · 8 · 9 · 10 · 11 · 12 · 13

1. Old finalized object may not be visited by GC.
reported by Yongheng Chen on 06 Jul 2020. existed since 5.4.0. fixed in 5.4.1. fixed in github.

Example:

```
-- run this code with some memory checker, such as valgrind
-- or gcc's option -fsanitize=address
local A = {}
A[1] = false     -- create an old anchor for an object

-- obj finalizer
local function gcf (obj)
  A[1] = obj     -- anchor object
  obj = nil      -- remove it from the stack
  collectgarbage("step", 0)   -- do a young collection
  print(getmetatable(A[1]).x)   -- metatable was collected!
end

collectgarbage()   -- make A old
local obj = {}     -- create a new object
collectgarbage("step", 0)   -- make it a survival
setmetatable(obj, {__gc = gcf})   -- create its metatable
obj = nil   -- clear object
collectgarbage("step", 0)   -- will call obj's finalizer
```

Patch:

```
lgc.c:
@@ -1140,7 +1140,7 @@ static void finishgencycle (lua_State *L, global_State *g) {
 static void youngcollection (lua_State *L, global_State *g) {
   GCObject **psurvival;  /* to point to first non-dead survival object */
   lua_assert(g->gcstate == GCSpropagate);
-  markold(g, g->survival, g->reallyold);
+  markold(g, g->allgc, g->reallyold);
   markold(g, g->finobj, g->finobjrold);
   atomic(L);
```

2. Computation of stack limit when entering a coroutine is wrong.
   reported by Yongheng Chen on 06 Jul 2020. existed since 5.4.0. fixed in 5.4.1. fixed in github.

   Example:

```
local lim = 1000
local function stack (n)
  if n > 0 then return stack(n - 1) + 1
  else coroutine.wrap(function ()
         stack(lim)
       end)()
  end
end

print(xpcall(stack, function () return "ok" end, lim))
```

   Patch:

```
ldo.c:
@@ -674,7 +674,7 @@ LUA_API int lua_resume (lua_State *L, lua_State *from, int nargs,
   if (from == NULL)
     L->nCcalls = CSTACKTHREAD;
   else  /* correct 'nCcalls' for this thread */
-    L->nCcalls = getCcalls(from) + from->nci - L->nci - CSTACKCF;
+    L->nCcalls = getCcalls(from) - L->nci - CSTACKCF;
   if (L->nCcalls <= CSTACKERR)
     return resume_error(L, "C stack overflow", nargs);
   luai_userstateresume(L, nargs);
```

3. An emergency collection when handling an error while loading the upvalues of a function can cause a segfault.
   reported by Roberto on 30 Jun 2020. existed since 5.4.0. fixed in 5.4.1. fixed in github.

   Example:

```
-- must compile Lua with option -DHARDMEMTESTS, to force
-- emergency collections
local s = string.dump(function ()
  local x, y, z
  return function () return x + y + z end
end)

for i = 1, #s - 1 do
  assert(not load(string.sub(s, 1, i)))
end
```

Patch:

```
lundump.c:
@@ -205,8 +205,9 @@ static void loadUpvalues (LoadState *S, Proto *f) {
   n = loadInt(S);
   f->upvalues = luaM_newvectorchecked(S->L, n, Upvaldesc);
   f->sizeupvalues = n;
-  for (i = 0; i < n; i++) {
+  for (i = 0; i < n; i++)
     f->upvalues[i].name = NULL;
+  for (i = 0; i < n; i++) {
     f->upvalues[i].instack = loadByte(S);
     f->upvalues[i].idx = loadByte(S);
     f->upvalues[i].kind = loadByte(S);
```

4. 'checkstackp' can run a GC step and destroy a preallocated CallInfo.
   reported by Yongheng Chen on 06 Jul 2020. fixed in 5.4.1. fixed in github.

   Example: See http://lua-users.org/lists/lua-l/2020-07/msg00053.html.

   Patch:

```
ldo.c:
@@ -466,13 +466,13 @@ void luaD_call (lua_State *L, StkId func, int nresults) {
       f = fvalue(s2v(func));
      Cfunc: {
       int n;  /* number of returns */
-      CallInfo *ci = next_ci(L);
+      CallInfo *ci;
       checkstackp(L, LUA_MINSTACK, func);  /* ensure minimum stack size */
+      L->ci = ci = next_ci(L);
       ci->nresults = nresults;
       ci->callstatus = CIST_C;
       ci->top = L->top + LUA_MINSTACK;
       ci->func = func;
-      L->ci = ci;
       lua_assert(ci->top <= L->stack_last);
       if (L->hookmask & LUA_MASKCALL) {
         int narg = cast_int(L->top - func) - 1;
@@ -486,18 +486,18 @@ void luaD_call (lua_State *L, StkId func, int nresults) {
       break;
     }
     case LUA_VLCL: {  /* Lua function */
-      CallInfo *ci = next_ci(L);
+      CallInfo *ci;
       Proto *p = clLvalue(s2v(func))->p;
       int narg = cast_int(L->top - func) - 1;  /* number of real arguments */
       int nfixparams = p->numparams;
       int fsize = p->maxstacksize;  /* frame size */
       checkstackp(L, fsize, func);
+      L->ci = ci = next_ci(L);
       ci->nresults = nresults;
       ci->u.l.savedpc = p->code;  /* starting point */
       ci->callstatus = 0;
       ci->top = func + 1 + fsize;
       ci->func = func;
-      L->ci = ci;
       for (; narg < nfixparams; narg++)
         setnilvalue(s2v(L->top++));  /* complete missing arguments */
       lua_assert(ci->top <= L->stack_last);
```

5. GC after resizing stack can shrink it again.
   reported by Yongheng Chen on 06 Jul 2020. existed since 5.4.0. fixed in 5.4.1. fixed in github.

   Example: See http://lua-users.org/lists/lua-l/2020-07/msg00054.html.

   Patch:

```
ldo.h:
@@ -44,7 +44,7 @@

 /* macro to check stack size and GC */
 #define checkstackGC(L,fsize)  \
-       luaD_checkstackaux(L, (fsize), (void)0, luaC_checkGC(L))
+       luaD_checkstackaux(L, (fsize), luaC_checkGC(L), (void)0)


 /* type of protected functions, to be ran by 'runprotected' */
```

6. Errors in finalizers need a valid 'pc' to produce an error message.
   reported by Rui Zhong on 06 Jul 2020. existed since 5.4.0. fixed in 5.4.1. fixed in github.

   Example: http://lua-users.org/lists/lua-l/2020-07/msg00078.html.

   Patch:

```
lvm.c:
@@ -1104,7 +1104,7 @@ void luaV_finishOp (lua_State *L) {


 #define checkGC(L,c)  \
-       { luaC_condGC(L, L->top = (c),  /* limit of live values */ \
+       { luaC_condGC(L, (savepc(L), L->top = (c)), \
                         updatetrap(ci)); \
            luai_threadyield(L); }

@@ -1792,8 +1792,7 @@ void luaV_execute (lua_State *L, CallInfo *ci) {
         vmbreak;
       }
       vmcase(OP_VARARGPREP) {
-        luaT_adjustvarargs(L, GETARG_A(i), ci, cl->p);
-        updatetrap(ci);
+        ProtectNT(luaT_adjustvarargs(L, GETARG_A(i), ci, cl->p));
         if (trap) {
           luaD_hookcall(L, ci);
           L->oldpc = pc + 1;  /* next opcode will be seen as a "new" line */
```

7. 'popen' can crash if called with an invalid mode.
   reported by Viacheslav Usov on 06 Jul 2020. existed since 5.1 (at least). fixed in 5.4.1. fixed in github.

   Example: (system dependent)

   Patch:

```
liolib.c:
@@ -279,6 +279,8 @@ static int io_popen (lua_State *L) {
   const char *filename = luaL_checkstring(L, 1);
   const char *mode = luaL_optstring(L, 2, "r");
   LStream *p = newprefile(L);
+  luaL_argcheck(L, ((mode[0] == 'r' || mode[0] == 'w') && mode[1] == '\0'),
+                   2, "invalid mode");
   p->f = l_popen(L, filename, mode);
   p->closef = &io_pclose;
   return (p->f == NULL) ? luaL_fileresult(L, 0, filename) : 1;
```

8. Field 'L->oldpc' is not always updated when returning to a function.
   reported by Yongheng Chen on 09 Jul 2020. existed since 5.4.0. fixed in 5.4.1. fixed in github.

   Example:

```
-- run this code under valgrind. (Error depends on details of dynamic
-- addresses.)
function foo ()
  local f = load[[io.write('+');
                  for i = 1, 10000 do local a = {}; debug.sethook(nil) end
                  io.write'-']]

  local u = setmetatable({},
    {__gc = assert(load[[debug.sethook(print, "l");
                  error('err');
                  ]])})

  u = nil
  f()
end

for i = 1, 200 do
  foo()
end
```

   Patch:

```
lgc.c:
@@ -856,6 +856,8 @@ static void GCTM (lua_State *L) {
     if (unlikely(status != LUA_OK)) {  /* error while running __gc? */
       luaE_warnerror(L, "__gc metamethod");
       L->top--;  /* pops error object */
+      if (isLua(L->ci))
```

```
+        L->oldpc = L->ci->u.l.savedpc;  /* update 'oldpc' */
      }
    }
  }
```

9. C stack overflow (again).
   reported by Yongheng Chen on 15 Jul 2020. fixed in 5.4.1. fixed in github.

   Example:

```
function errfunc ()
  return 10 + xpcall(nil, errfunc)
end

coroutine.resume(coroutine.create(function() xpcall(nil, errfunc) end))
```

10. Barriers cannot be active during sweep phase, even in generational mode.
    reported by Yongheng Chen on 15 Jul 2020. existed since 5.4.0. fixed in 5.4.1. fixed in github.

    Example:

```
-- The following chunk, under a memory checker like valgrind,
-- produces a memory access violation.
local old = {10}
collectgarbage()   -- make 'old' old
local co = coroutine.create(
  function ()
    local x = nil
    local f = function ()
                return x[1]
              end
    x = coroutine.yield(f)
    coroutine.yield()
  end
)
local _, f = coroutine.resume(co)     -- create closure over x in thread
collectgarbage("step", 0)   -- make upvalue a survival
old[1] = {"hello"}    -- 'old' go to grayagain as 'touched1'
coroutine.resume(co, {123})     -- its value will be new
co = nil
-- next minor collection hits the barrier between upvalue and its
-- conent while sweeping the thread. This will mix the lists 'gray'
-- and 'grayagain' and will remove 'old' from 'grayagain'.
collectgarbage("step", 0)
assert(f() == 123 and old[1][1] == "hello")   -- still ok
collectgarbage("step", 0)   -- run the collector once more
-- now, as 'old' was not in 'grayagain', 'old[1]' was deleted
assert(f() == 123 and old[1][1] == "hello")
```

11. Negation overflow in getlocal/setlocal.
    reported by Yongheng Chen on 24 Jul 2020. existed since 5.2. fixed in 5.4.1. fixed in github.

    Example:

```
print(debug.getlocal(1, 2^31))
```

12. Access to debug information in line hook of stripped function.
    reported by Yongheng Chen on 24 Jul 2020. existed since 5.4.0. fixed in 5.4.1. fixed in github.

    Example:

```
local function foo ()
   local a = 1
   local b = 2
   local c = 3
end

local s = load(string.dump(foo, true))
local line
debug.sethook(function (e, l) line = l end, "l"); s(); debug.sethook(nil)
print(line)
```

13. Long string can be collected while its contents is being read when loading a binary file.
    reported by Payo Nel on 15 Aug 2020. existed since 5.3. fixed in 5.4.1. fixed in github.

    Example:

```
-- run this code under some memory checker
local function myload (s)
  return load(function ()
    if s == "" then return nil
    else
      local c = string.sub(s, 1, 1)
      s = string.sub(s, 2)
      collectgarbage()
      return c
    end
  end)
end

local y = string.dump(function ()
  return '01234567890123456789012345678901234567890123456789'
end)
y = myload(y)
assert(y() == '01234567890123456789012345678901234567890123456789')
```

❖ Lua 5.3.6

That is probably the last release of Lua 5.3. Bugs reported later are probably fixed in Lua 5.4.

❖ Lua 5.3.5 1 · 2 · 3 · 4

1. 'LUA_USE_READLINE' defined twice in FreeBSD recipe.
   reported by Russell Haley on 13 Jul 2018.

   Patch:

   ```
   src/Makefile:
   @@ -104,2 +104,2 @@
    freebsd:
   -       $(MAKE) $(ALL) SYSCFLAGS="-DLUA_USE_LINUX -DLUA_USE_READLINE -I/usr/include/edit" SYSLIBS="-Wl,-E -ledit" CC="c
   +       $(MAKE) $(ALL) SYSCFLAGS="-DLUA_USE_LINUX -I/usr/include/edit" SYSLIBS="-Wl,-E -ledit" CC="cc"
   ```

2. Revision number is not updated in Makefile.
   reported by milly on 8 Aug 2018.

   Patch:

   ```
   Makefile:
   @@ -49 +49 @@
   -R= $V.4
   +R= $V.5
   ```

3. Long brackets with a huge number of '=' overflow some internal buffer arithmetic.
   reported by Marco on 12 Dec 2018. existed since 5.1.

   Example:

   ```
   local eqs = string.rep("=", 0x3ffffffe)
   local code = "return [" .. eqs .. "[a]" .. eqs .. "]"
   print(#assert(load(code))())
   ```

4. Joining an upvalue with itself can cause a use-after-free crash.
   reported by Fady Othman on 10 Jan 2019. existed since 5.3.

   Example:

   ```
   -- the next code may crash the machine
   f=load(function() end)
   interesting={}
   interesting[0]=string.rep("A",512)
   debug.upvaluejoin(f,1,f,1)
   ```

   Patch:

   ```
   lapi.c:
   @@ -1289,6 +1289,8 @@
   LUA_API void lua_upvaluejoin (lua_State *L, int fidx1, int n1,
      LClosure *f1;
      UpVal **up1 = getupvalref(L, fidx1, n1, &f1);
      UpVal **up2 = getupvalref(L, fidx2, n2, NULL);
   +  if (*up1 == *up2)
   ```

```
+    return;
   luaC_upvdeccount(L, *up1);
   *up1 = *up2;
   (*up1)->refcount++;
```

❖ Lua 5.3.4 1 · 2 · 3 · 4 · 5 · 6 · 7

1. Wrong code generated for a 'goto' followed by a label inside an 'if'.
   reported by 云风 on 13 Apr 2017. existed since 5.2.

   Example:

```
-- should print 32323232..., but prints only '3'
if true then
  goto LBL
  ::loop::
  print(2)
  ::LBL::
  print(3)
  goto loop
end
```

   Patch:

```
lparser.c:
@@ -1392,7 +1392,7 @@
     luaK_goiffalse(ls->fs, &v);  /* will jump to label if condition is true */
     enterblock(fs, &bl, 0);  /* must enter block before 'goto' */
     gotostat(ls, v.t);  /* handle goto/break */
-    skipnoopstat(ls);  /* skip other no-op statements */
+    while (testnext(ls, ';')) {}  /* skip semicolons */
     if (block_follow(ls, 0)) {  /* 'goto' is the entire block? */
       leaveblock(fs);
       return;  /* and that is it */
```

2. Lua crashes when building sequences with more than 2^30 elements.
   reported by Viacheslav Usov on 11 May 2017.

   Example:

```
-- crashes if machine has enough memory
local t = {}
for i = 1, 0x7fffffff do
  t[i] = i
end
```

   Patch:

```
ltable.c:
@@ -223,7 +223,9 @@
   unsigned int na = 0;  /* number of elements to go to array part */
   unsigned int optimal = 0;  /* optimal size for array part */
   /* loop while keys can fill more than half of total size */
-  for (i = 0, twotoi = 1; *pna > twotoi / 2; i++, twotoi *= 2) {
+  for (i = 0, twotoi = 1;
+       twotoi > 0 && *pna > twotoi / 2;
+       i++, twotoi *= 2) {
     if (nums[i] > 0) {
       a += nums[i];
       if (a > twotoi/2) {  /* more than half elements present? */
```

3. Table length computation overflows for sequences larger than 2^31 elements.
   reported by Viacheslav Usov on 12 May 2017.

   Example:

```
-- on a machine with enough memory
local t = {}
for i = 1, 2147483681 do
  t[i] = i
end
print(#t)
```

   Patch:

```
ltable.h:
@@ -56,3 +56,3 @@
```

```
 LUAI_FUNC int luaH_next (lua_State *L, Table *t, StkId key);
-LUAI_FUNC int luaH_getn (Table *t);
+LUAI_FUNC lua_Unsigned luaH_getn (Table *t);

ltable.c:
@@ -614,4 +614,4 @@

-static int unbound_search (Table *t, unsigned int j) {
-  unsigned int i = j;  /* i is zero or a present index */
+static lua_Unsigned unbound_search (Table *t, lua_Unsigned j) {
+  lua_Unsigned i = j;  /* i is zero or a present index */
   j++;
@@ -620,3 +620,3 @@
     i = j;
-    if (j > cast(unsigned int, MAX_INT)/2) {  /* overflow? */
+    if (j > l_castS2U(LUA_MAXINTEGER) / 2) {  /* overflow? */
       /* table was built with bad purposes: resort to linear search */
@@ -630,3 +630,3 @@
   while (j - i > 1) {
-    unsigned int m = (i+j)/2;
+    lua_Unsigned m = (i+j)/2;
     if (ttisnil(luaH_getint(t, m))) j = m;
@@ -642,3 +642,3 @@
 */
-int luaH_getn (Table *t) {
+lua_Unsigned luaH_getn (Table *t) {
   unsigned int j = t->sizearray;
```

4. Lua does not check GC when creating error messages.
   reported by Viacheslav Usov on 06 Jul 2017. existed since 5.3.2.

   Example:

```
function test()
  bob.joe.larry = 23
end

-- memory will grow steadly
for i = 1, math.huge do
  pcall(test)
  if i % 100000 == 0 then
    io.write(collectgarbage'count'*1024, "\n")
  end
end
```

   Patch:

```
ldebug.c:
@@ -653,6 +653,7 @@
   CallInfo *ci = L->ci;
   const char *msg;
   va_list argp;
+  luaC_checkGC(L);  /* error message uses memory */
   va_start(argp, fmt);
   msg = luaO_pushvfstring(L, fmt, argp);  /* format message */
   va_end(argp);
```

5. Dead keys with nil values can stay in weak tables.
   reported by 云风 Cloud Wu on 15 Aug 2017. existed since 5.2.

   Example:

```
-- The following chunk, under a memory checker like valgrind, produces a memory access violation.

local a = setmetatable({}, {__mode = 'kv'})

a['ABCDEFGHIJKLMNOPQRSTUVWXYZ' .. 'abcdefghijklmnopqrstuvwxyz'] = {}
a[next(a)] = nil
collectgarbage()
print(a['BCDEFGHIJKLMNOPQRSTUVWXYZ' .. 'abcdefghijklmnopqrstuvwxyz'])
```

   Patch:

```
lgc.c:
@@ -643,8 +643,9 @@
     for (n = gnode(h, 0); n < limit; n++) {
       if (!ttisnil(gval(n)) && (iscleared(g, gkey(n)))) {
         setnilvalue(gval(n));  /* remove value ... */
-        removeentry(n);  /* and remove entry from table */
       }
```

```
+      if (ttisnil(gval(n)))  /* is entry empty? */
+        removeentry(n);  /* remove entry from table */
     }
   }
 }
```

6. `lua_pushcclosure` should not call the garbage collector when `n` is zero.
   reported by Andrew Gierth on 05 Dec 2017. existed since 5.3.3.

   Patch:

```
lapi.c:
@@ -533,6 +533,7 @@
   lua_lock(L);
   if (n == 0) {
     setfvalue(L->top, fn);
+    api_incr_top(L);
   }
   else {
     CClosure *cl;
@@ -546,9 +547,9 @@
       /* does not need barrier because closure is white */
     }
     setclCvalue(L, L->top, cl);
+    api_incr_top(L);
+    luaC_checkGC(L);
   }
-  api_incr_top(L);
-  luaC_checkGC(L);
   lua_unlock(L);
 }
```

7. Memory-allocation error when resizing a table can leave it in an inconsistent state..
   reported by Roberto on 08 Dec 2017. existed since 5.0.

   Example:

```
local a = {x = 1, y = 1, z = 1}
a[1] = 10   -- goes to the hash part (which has 4 slots)
print(a[1])   --> 10

-- assume that the 2nd memory allocation from now fails
pcall(rawset, a, 2, 20)   -- forces a rehash

-- a[1] now exists both in the array part (because the array part
-- grew) and in the hash part (because the allocation of the hash
-- part failed, keeping it as it was).
-- This makes the following traversal goes forever...
for k,v in pairs(a) do print(k,v) end
```

   Patch:

```
ltable.c:
@@ -332,17 +332,34 @@
 }


+typedef struct {
+  Table *t;
+  unsigned int nhsize;
+} AuxsetnodeT;
+
+
+static void auxsetnode (lua_State *L, void *ud) {
+  AuxsetnodeT *asn = cast(AuxsetnodeT *, ud);
+  setnodevector(L, asn->t, asn->nhsize);
+}
+
+
 void luaH_resize (lua_State *L, Table *t, unsigned int nasize,
                                           unsigned int nhsize) {
   unsigned int i;
   int j;
+  AuxsetnodeT asn;
   unsigned int oldasize = t->sizearray;
   int oldhsize = allocsizenode(t);
   Node *nold = t->node;  /* save old hash ... */
   if (nasize > oldasize)  /* array part must grow? */
     setarrayvector(L, t, nasize);
   /* create new hash part with appropriate size */
```

```
-  setnodevector(L, t, nhsize);
+  asn.t = t; asn.nhsize = nhsize;
+  if (luaD_rawrunprotected(L, auxsetnode, &asn) != LUA_OK) {  /* mem. error? */
+    setarrayvector(L, t, oldasize);  /* array back to its original size */
+    luaD_throw(L, LUA_ERRMEM);  /* rethrow memory error */
+  }
   if (nasize < oldasize) {  /* array part must shrink? */
     t->sizearray = nasize;
     /* re-insert elements from vanishing slice */
```

❖ Lua 5.3.3 1 · 2 · 3 · 4

1. Expression list with four or more expressions in a 'for' loop can crash the interpreter.
   reported by Marco Schöpl on 17 Jun 2016. existed since 5.2. fixed in 5.3.4.

   Example:

   ```
   -- the next loop will probably crash the interpreter
   repeat until load "for _ in _,_,_,_ do local function _() end"
   ```

   Patch:

   ```
   lparser.c:
   @@ -323,6 +323,8 @@
          luaK_nil(fs, reg, extra);
        }
      }
   +  if (nexps > nvars)
   +    ls->fs->freereg -= nexps - nvars;  /* remove extra values */
    }


   @@ -1160,11 +1162,8 @@
        int nexps;
        checknext(ls, '=');
        nexps = explist(ls, &e);
   -    if (nexps != nvars) {
   +    if (nexps != nvars)
          adjust_assign(ls, nvars, nexps, &e);
   -      if (nexps > nvars)
   -        ls->fs->freereg -= nexps - nvars;  /* remove extra values */
   -    }
        else {
          luaK_setoneret(ls->fs, &e);  /* close last expression */
          luaK_storevar(ls->fs, &lh->v, &e);
   ```

2. Checking a format for `os.date` may read pass the format string.
   reported by Nagaev Boris on 10 Jul 2016. existed since 5.3.3. fixed in 5.3.4.

   Example: This bug does not seem to happen with regular compilers. It needs an "interceptor" 'memcmp' function that continues reading memory after a difference is found.

   Patch:

   ```
   loslib.c:
   263c263,264
   <   for (option = LUA_STRFTIMEOPTIONS; *option != '\0'; option += oplen) {
   ---
   >   int convlen = (int)strlen(conv);
   >   for (option = LUA_STRFTIMEOPTIONS; *option != '\0' && oplen <= convlen; option += oplen) {
   ```

3. Lua can generate wrong code in functions with too many constants.
   reported by Marco Schöpl on 17 Jul 2016. existed since 5.3.3. fixed in 5.3.4.

   Example: See http://lua-users.org/lists/lua-l/2016-07/msg00303.html.

   Patch:

   ```
   lcode.c:
   @@ -1018,8 +1018,8 @@
    */
    static void codebinexpval (FuncState *fs, OpCode op,
                               expdesc *e1, expdesc *e2, int line) {
   -  int rk1 = luaK_exp2RK(fs, e1);  /* both operands are "RK" */
   -  int rk2 = luaK_exp2RK(fs, e2);
   +  int rk2 = luaK_exp2RK(fs, e2);  /* both operands are "RK" */
   +  int rk1 = luaK_exp2RK(fs, e1);
   ```

```
   freeexps(fs, e1, e2);
   e1->u.info = luaK_codeABC(fs, op, 0, rk1, rk2);  /* generate opcode */
   e1->k = VRELOCABLE;  /* all those operations are relocatable */
```

4. When a coroutine tries to resume a non-suspended coroutine, it can do some mess (and break C assertions) before detecting the error.
   reported by Marco Schöpl on 20 Jul 2016. fixed in 5.3.4.

   Example:

```
-- with C assertions on
A = coroutine.running()
B = coroutine.create(function() coroutine.resume(A) end)
coroutine.resume(B)
-- or
A = coroutine.wrap(function() pcall(A, _) end)
A()
```

❖ Lua 5.3.2 1 · 2 · 3

1. Metatable may access its own deallocated field when it has a self reference in __newindex.
   reported by actboy168 on 01 Jan 2016. existed since 5.3.2. fixed in 5.3.3.

   Example:

```
local mt = {}
mt.__newindex = mt
local t = setmetatable({}, mt)
t[1] = 1     -- will segfault on some machines
```

   Patch:

```
lvm.c:
@@ -190,18 +190,19 @@
   for (loop = 0; loop < MAXTAGLOOP; loop++) {
     const TValue *tm;
     if (oldval != NULL) {
-      lua_assert(ttistable(t) && ttisnil(oldval));
+      Table *h = hvalue(t);  /* save 't' table */
+      lua_assert(ttisnil(oldval));
       /* must check the metamethod */
-      if ((tm = fasttm(L, hvalue(t)->metatable, TM_NEWINDEX)) == NULL &&
+      if ((tm = fasttm(L, h->metatable, TM_NEWINDEX)) == NULL &&
          /* no metamethod; is there a previous entry in the table? */
          (oldval != luaO_nilobject ||
          /* no previous entry; must create one. (The next test is
             always true; we only need the assignment.) */
-         (oldval = luaH_newkey(L, hvalue(t), key), 1))) {
+         (oldval = luaH_newkey(L, h, key), 1))) {
         /* no metamethod and (now) there is an entry with given key */
         setobj2t(L, cast(TValue *, oldval), val);
-        invalidateTMcache(hvalue(t));
-        luaC_barrierback(L, hvalue(t), val);
+        invalidateTMcache(h);
+        luaC_barrierback(L, h, val);
         return;
       }
       /* else will try the metamethod */
```

2. Label between local definitions can mix-up their initializations.
   reported by Karel Tuma on 01 Mar 2016. existed since 5.2. fixed in 5.3.3.

   Example:

```
do
  local k = 0
  local x
  ::foo::
  local y       -- should be reset to nil after goto, but it is not
  assert(not y)
  y = true
  k = k + 1
  if k < 2 then goto foo end
end
```

   Patch:

```
lparser.c:
@@ -1226,7 +1226,7 @@
   checkrepeated(fs, ll, label);  /* check for repeated labels */
   checknext(ls, TK_DBCOLON);  /* skip double colon */
   /* create new entry for this label */
-  l = newlabelentry(ls, ll, label, line, fs->pc);
+  l = newlabelentry(ls, ll, label, line, luaK_getlabel(fs));
   skipnoopstat(ls);  /* skip other no-op statements */
   if (block_follow(ls, 0)) {  /* label is last no-op statement in the block? */
     /* assume that locals are already out of scope */
```

3. `gmatch` iterator fails when called from a coroutine different from the one that created it.
   reported by Nagaev Boris on 18 Mar 2016. existed since 5.3.2. fixed in 5.3.3.

   Example:

```
local f = string.gmatch("1 2 3 4 5", "%d+")
print(f())     --> 1
co = coroutine.wrap(f)
print(co())    --> ??? (should be 2)
```

   Patch:

```
lstrlib.c:
@@ -688,6 +688,7 @@
 static int gmatch_aux (lua_State *L) {
   GMatchState *gm = (GMatchState *)lua_touserdata(L, lua_upvalueindex(3));
   const char *src;
+  gm->ms.L = L;
   for (src = gm->src; src <= gm->ms.src_end; src++) {
     const char *e;
     reprepstate(&gm->ms);
```

❖ Lua 5.3.1 1

1. `io.lines` does not check maximum number of options.
   reported by Patrick Donnell on 10 Jul 2015. existed since 5.3.0. fixed in 5.3.2.

   Example:

```
-- can crash in some machines
t ={}; for i = 1, 253 do t[i] = 1 end
io.lines("someexistingfile", table.unpack(t))()
```

   Patch:

```
liolib.c:
@@ -318,8 +318,15 @@
 static int io_readline (lua_State *L);


+/*
+** maximum number of arguments to 'f:lines'/'io.lines' (it + 3 must fit
+** in the limit for upvalues of a closure)
+*/
+#define MAXARGLINE     250
+
 static void aux_lines (lua_State *L, int toclose) {
   int n = lua_gettop(L) - 1;  /* number of arguments to read */
+  luaL_argcheck(L, n <= MAXARGLINE, MAXARGLINE + 2, "too many arguments");
   lua_pushinteger(L, n);  /* number of arguments to read */
   lua_pushboolean(L, toclose);  /* close/not close file when finished */
   lua_rotate(L, 2, 2);  /* move 'n' and 'toclose' to their positions */
```

❖ Lua 5.3.0 1 · 2 · 3 · 4

1. `string.format("%f")` can cause a buffer overflow (only when 'lua_Number' is long double!).
   reported by Roberto on 13 Jan 2015. existed since 5.3. fixed in 5.3.1.

   Example:

```
string.format("%.99f", 1e4000)    -- when floats are long double
```

   Patch:

```
lstrlib.c:
@@ -800,3 +800,4 @@
 /* maximum size of each formatted item (> len(format('%99.99f', -1e308))) */
-#define MAX_ITEM       512
+#define MAX_ITEM  \
+  (sizeof(lua_Number) <= 4 ? 150 : sizeof(lua_Number) <= 8 ? 450 : 5050)
```

2. `debug.getlocal` on a coroutine suspended in a hook can crash the interpreter.
   reported by 云风 on 11 Feb 2015. existed since 5.2. fixed in 5.3.1.

   Example: See http://lua-users.org/lists/lua-l/2015-02/msg00146.html.

   Patch:

```
ldebug.c:
@@ -49,4 +49,14 @@


+static void swapextra (lua_State *L) {
+  if (L->status == LUA_YIELD) {
+    CallInfo *ci = L->ci;  /* get function that yielded */
+    StkId temp = ci->func;  /* exchange its 'func' and 'extra' values */
+    ci->func = restorestack(L, ci->extra);
+    ci->extra = savestack(L, temp);
+  }
+}
+
+
 /*
 ** this function can be called asynchronous (e.g. during a signal)
@@ -145,4 +155,5 @@
   const char *name;
   lua_lock(L);
+  swapextra(L);
   if (ar == NULL) {  /* information about non-active function? */
     if (!isLfunction(L->top - 1))  /* not a Lua function? */
@@ -159,4 +170,5 @@
     }
   }
+  swapextra(L);
   lua_unlock(L);
   return name;
@@ -166,10 +178,13 @@
 LUA_API const char *lua_setlocal (lua_State *L, const lua_Debug *ar, int n) {
   StkId pos = 0;  /* to avoid warnings */
-  const char *name = findlocal(L, ar->i_ci, n, &pos);
+  const char *name;
   lua_lock(L);
+  swapextra(L);
+  name = findlocal(L, ar->i_ci, n, &pos);
   if (name) {
     setobjs2s(L, pos, L->top - 1);
     L->top--;  /* pop value */
   }
+  swapextra(L);
   lua_unlock(L);
   return name;
@@ -271,4 +286,5 @@
   StkId func;
   lua_lock(L);
+  swapextra(L);
   if (*what == '>') {
     ci = NULL;
@@ -289,4 +305,5 @@
     api_incr_top(L);
   }
+  swapextra(L);
   if (strchr(what, 'L'))
     collectvalidlines(L, cl);
```

3. Suspended __le metamethod can give wrong result.
   reported by Eric Zhong on 07 Apr 2015. existed since 5.2. fixed in 5.3.1.

   Example:

```
mt = {__le = function (a,b) coroutine.yield("yield"); return a.x <= b.x end}
t1 = setmetatable({x=1}, mt)
t2 = {x=2}
co = coroutine.wrap(function (a,b) return t2 <= t1 end)
```

```
co()
print(co())   --> true  (should be false)
```

Patch:

```
lstate.h:
@@ -94,6 +94,7 @@
 #define CIST_YPCALL    (1<<4)  /* call is a yieldable protected call */
 #define CIST_TAIL      (1<<5)  /* call was tail called */
 #define CIST_HOOKYIELD (1<<6)  /* last hook called yielded */
+#define CIST_LEQ       (1<<7)  /* using __lt for __le */

 #define isLua(ci)      ((ci)->callstatus & CIST_LUA)

lvm.c:
@@ -292,9 +292,14 @@
     return l_strcmp(tsvalue(l), tsvalue(r)) <= 0;
   else if ((res = luaT_callorderTM(L, l, r, TM_LE)) >= 0)  /* first try 'le' */
     return res;
-  else if ((res = luaT_callorderTM(L, r, l, TM_LT)) < 0)  /* else try 'lt' */
-    luaG_ordererror(L, l, r);
-  return !res;
+  else {  /* try 'lt': */
+    L->ci->callstatus |= CIST_LEQ;  /* mark it is doing 'lt' for 'le' */
+    res = luaT_callorderTM(L, r, l, TM_LT);
+    L->ci->callstatus ^= CIST_LEQ;  /* clear mark */
+    if (res < 0)
+      luaG_ordererror(L, l, r);
+    return !res;  /* result is negated */
+  }
 }


@@ -553,11 +558,11 @@
     case OP_LE: case OP_LT: case OP_EQ: {
       int res = !l_isfalse(L->top - 1);
       L->top--;
-      /* metamethod should not be called when operand is K */
-      lua_assert(!ISK(GETARG_B(inst)));
-      if (op == OP_LE &&  /* "<=" using "<" instead? */
-          ttisnil(luaT_gettmbyobj(L, base + GETARG_B(inst), TM_LE)))
-        res = !res;  /* invert result */
+      if (ci->callstatus & CIST_LEQ) {  /* "<=" using "<" instead? */
+        lua_assert(op == OP_LE);
+        ci->callstatus ^= CIST_LEQ;  /* clear mark */
+        res = !res;  /* negate result */
+      }
       lua_assert(GET_OPCODE(*ci->u.l.savedpc) == OP_JMP);
       if (res != GETARG_A(inst))  /* condition failed? */
         ci->u.l.savedpc++;  /* skip jump instruction */
```

4. Return hook may not see correct values for active local variables when function returns.
   reported by Philipp Janda and Peng Yi on 19 May 2015. existed since 5.0. fixed in 5.3.1.

   Example: See http://lua-users.org/lists/lua-l/2015-05/msg00376.html.

❖ Lua 5.2.4

That was the last release of Lua 5.2. Bugs reported later are probably fixed in Lua 5.3.

❖ Lua 5.2.3 1 · 2 · 3

1. Compiler can optimize away overflow check in `table.unpack`.
   reported by Paige DePol on 30 Mar 2014. existed since 5.1. fixed in 5.3.0 and 5.2.4.

   Example:

```
unpack({}, 0, 2^31 - 1) -- crashes on some platforms with some compiler options
```

   Patch:

```
ltablib.c:
@@ -134,13 +135,14 @@


 static int unpack (lua_State *L) {
-  int i, e, n;
```

```
+  int i, e;
+  unsigned int n;
   luaL_checktype(L, 1, LUA_TTABLE);
   i = luaL_optint(L, 2, 1);
   e = luaL_opt(L, luaL_checkint, 3, luaL_len(L, 1));
   if (i > e) return 0;  /* empty range */
-  n = e - i + 1;  /* number of elements */
-  if (n <= 0 || !lua_checkstack(L, n))  /* n <= 0 means arith. overflow */
+  n = (unsigned int)e - (unsigned int)i;  /* number of elements minus 1 */
+  if (n > (INT_MAX - 10) || !lua_checkstack(L, ++n))
     return luaL_error(L, "too many results to unpack");
   lua_rawgeti(L, 1, i);  /* push arg[i] (avoiding overflow problems) */
   while (i++ < e)  /* push arg[i + 1...e] */
```

2. Ephemeron table can wrongly collect entry with strong key.
   reported by Jörg Richter on 22 Aug 2014. existed since 5.2.0. fixed in 5.3.0 and 5.2.4.

   Example: This bug is very hard to reproduce, because it depends on a specific interleaving of events between the incremental collector and the program.

   Patch:

```
lgc.c:
@@ -403,7 +403,7 @@
       reallymarkobject(g, gcvalue(gval(n)));  /* mark it now */
     }
   }
-  if (prop)
+  if (g->gcstate != GCSatomic || prop)
     linktable(h, &g->ephemeron);  /* have to propagate again */
   else if (hasclears)  /* does table have white keys? */
     linktable(h, &g->allweak);  /* may have to clean white keys */
```

3. Chunk with too many lines may crash Lua.
   reported by Roberto on 14 Nov 2014. existed since 5.1 at least. fixed in 5.3.0 and 5.2.4.

   Example: The cause of the bug is the use of an unitialized variable, so it cannot be reproduced reliably.

```
local s = string.rep("\n", 2^24)
print(load(function () return s end))
```

   Patch:

```
llex.c:
@@ -153,5 +153,5 @@
     next(ls);  /* skip `\n\r' or `\r\n' */
   if (++ls->linenumber >= MAX_INT)
-    luaX_syntaxerror(ls, "chunk has too many lines");
+    lexerror(ls, "chunk has too many lines", 0);
 }
```

❖ Lua 5.2.2 1 · 2 · 3 · 4 · 5 · 6 · 7 · 8

1. Stack overflow in vararg functions with many fixed parameters called with few arguments.
   reported by 云风 on 17 Apr 2013. existed since 5.1. fixed in 5.2.3.

   Example:

```
function f(p1, p2, p3, p4, p5, p6, p7, p8, p9, p10,
           p11, p12, p13, p14, p15, p16, p17, p18, p19, p20,
           p21, p22, p23, p24, p25, p26, p27, p28, p29, p30,
           p31, p32, p33, p34, p35, p36, p37, p38, p39, p40,
           p41, p42, p43, p44, p45, p46, p48, p49, p50, ...)
  local a1, a2, a3, a4, a5, a6, a7, a8, a9, a10, a11, a12, a13, a14
end

f()   -- crashes on some machines
```

   Patch:

```
ldo.c:
@@ -324,7 +324,7 @@
     case LUA_TLCL: {  /* Lua function: prepare its call */
       StkId base;
       Proto *p = clLvalue(func)->p;
-      luaD_checkstack(L, p->maxstacksize);
```

```
+       luaD_checkstack(L, p->maxstacksize + p->numparams);
        func = restorestack(L, funcr);
        n = cast_int(L->top - func) - 1;  /* number of real arguments */
        for (; n < p->numparams; n++)
```

2. Garbage collector can trigger too many times in recursive loops.
   reported by Roberto on 25 Apr 2013. existed since 5.2.2. fixed in 5.2.3.

   Example:

```
function f() f() end
f()   -- it takes too long before a "stack overflow" error
```

   Patch:

```
lgc.c:
@@ -495,2 +495,3 @@
 static lu_mem traversestack (global_State *g, lua_State *th) {
+  int n = 0;
   StkId o = th->stack;
@@ -505,3 +506,9 @@
   }
-  return sizeof(lua_State) + sizeof(TValue) * th->stacksize;
+  else {  /* count call infos to compute size */
+    CallInfo *ci;
+    for (ci = &th->base_ci; ci != th->ci; ci = ci->next)
+      n++;
+  }
+  return sizeof(lua_State) + sizeof(TValue) * th->stacksize +
+         sizeof(CallInfo) * n;
 }
```

3. Wrong assert when reporting concatenation errors (manifests only when Lua is compiled in debug mode).
   reported by Roberto on 05 May 2013. existed since 5.2.0. fixed in 5.2.3.

   Example:

```
-- only with Lua compiled in debug mode
print({} .. 2)
```

   Patch:

```
ldebug.c:
@@ -519,5 +519,5 @@
 l_noret luaG_concaterror (lua_State *L, StkId p1, StkId p2) {
   if (ttisstring(p1) || ttisnumber(p1)) p1 = p2;
-  lua_assert(!ttisstring(p1) && !ttisnumber(p2));
+  lua_assert(!ttisstring(p1) && !ttisnumber(p1));
   luaG_typeerror(L, p1, "concatenate");
 }
```

4. Wrong error message in some short-cut expressions.
   reported by Egor Skriptunoff on 10 May 2013. existed since 5.0. fixed in 5.2.3.

   Example:

```
a,b,c = true,true,true
(a and b or c)('', '')
--> stdin:1: attempt to call a boolean value (global 'c')
--  it should be global 'b' instead of 'c'
```

   Patch:

```
ldebug.c:
@@ -327,12 +327,20 @@
 }


+static int filterpc (int pc, int jmptarget) {
+  if (pc < jmptarget)  /* is code conditional (inside a jump)? */
+    return -1;  /* cannot know who sets that register */
+  else return pc;  /* current position sets that register */
+}
+
+
 /*
 ** try to find last instruction before 'lastpc' that modified register 'reg'
 */
```

```
 static int findsetreg (Proto *p, int lastpc, int reg) {
   int pc;
   int setreg = -1;  /* keep last instruction that changed 'reg' */
+  int jmptarget = 0;  /* any code before this address is conditional */
   for (pc = 0; pc < lastpc; pc++) {
     Instruction i = p->code[pc];
     OpCode op = GET_OPCODE(i);
@@ -341,33 +349,38 @@
       case OP_LOADNIL: {
         int b = GETARG_B(i);
         if (a <= reg && reg <= a + b)  /* set registers from 'a' to 'a+b' */
-          setreg = pc;
+          setreg = filterpc(pc, jmptarget);
         break;
       }
       case OP_TFORCALL: {
-        if (reg >= a + 2) setreg = pc;  /* affect all regs above its base */
+        if (reg >= a + 2)  /* affect all regs above its base */
+          setreg = filterpc(pc, jmptarget);
         break;
       }
       case OP_CALL:
       case OP_TAILCALL: {
-        if (reg >= a) setreg = pc;  /* affect all registers above base */
+        if (reg >= a)  /* affect all registers above base */
+          setreg = filterpc(pc, jmptarget);
         break;
       }
       case OP_JMP: {
         int b = GETARG_sBx(i);
         int dest = pc + 1 + b;
         /* jump is forward and do not skip `lastpc'? */
-        if (pc < dest && dest <= lastpc)
-          pc += b;  /* do the jump */
+        if (pc < dest && dest <= lastpc) {
+          if (dest > jmptarget)
+            jmptarget = dest;  /* update 'jmptarget' */
+        }
         break;
       }
       case OP_TEST: {
-        if (reg == a) setreg = pc;  /* jumped code can change 'a' */
+        if (reg == a)  /* jumped code can change 'a' */
+          setreg = filterpc(pc, jmptarget);
         break;
       }
       default:
         if (testAMode(op) && reg == a)  /* any instruction that set A */
-          setreg = pc;
+          setreg = filterpc(pc, jmptarget);
         break;
     }
   }
```

5. luac listings choke on long strings.
   reported by Ashwin Hirschi on 03 Jul 2013. existed since 5.2.1. fixed in 5.2.3.

   Example:

   ```
   -- When you call 'luac -l' over this chunk, it chokes the output
   s="Lorem ipsum dolor sit amet, consectetur, "
   ```

   Patch:

   ```
   luac.c:
   @@ -251,7 +251,7 @@
    static void PrintConstant(const Proto* f, int i)
    {
     const TValue* o=&f->k[i];
   - switch (ttype(o))
   + switch (ttypenv(o))
     {
      case LUA_TNIL:
           printf("nil");
   ```

6. GC can collect a long string still in use during parser.
   reported by Roberto on 30 Aug 2013. existed since 5.2.0. fixed in 5.2.3.

   Example: This bug is very difficult to happen (and to reproduce), because it depends on the GC running in a very specific way when parsing a source code with long (larger than 40 characters) identifiers.

Patch:

```
ltable.h:
@@ -18,4 +18,8 @@
 #define invalidateTMcache(t)   ((t)->flags = 0)

+/* returns the key, given the value of a table entry */
+#define keyfromval(v) \
+  (gkey(cast(Node *, cast(char *, (v)) - offsetof(Node, i_val))))
+

 LUAI_FUNC const TValue *luaH_getint (Table *t, int key);

llex.c:
@@ -134,4 +134,7 @@
     luaC_checkGC(L);
   }
+  else {  /* string already present */
+    ts = rawtsvalue(keyfromval(o));  /* re-use value previously stored */
+  }
   L->top--;  /* remove string from stack */
   return ts;
```

7. Call to macro `luai_userstateclose` should be done only after the calls to __gc methods.
   reported by Jean-Luc Jumpertz on 02 Sep 2013. fixed in 5.2.3.

   Patch:

```
lstate.c:
@@ -194,2 +194,4 @@
   g->gcrunning = 1;  /* allow gc */
+  g->version = lua_version(NULL);
+  luai_userstateopen(L);
 }
@@ -224,2 +226,4 @@
   luaC_freeallobjects(L);  /* collect all objects */
+  if (g->version)  /* closing a fully built state? */
+    luai_userstateclose(L);
   luaM_freearray(L, G(L)->strt.hash, G(L)->strt.size);
@@ -289,3 +293,3 @@
   g->panic = NULL;
-  g->version = lua_version(NULL);
+  g->version = NULL;
   g->gcstate = GCSpause;
@@ -308,4 +312,2 @@
   }
-  else
-    luai_userstateopen(L);
   return L;
@@ -317,3 +319,2 @@
   lua_lock(L);
-  luai_userstateclose(L);
   close_state(L);
```

8. Resuming the running coroutine makes it unyieldable.
   reported by Florian Nücke on 28 Oct 2013. existed since 5.2.0. fixed in 5.2.3.

   Example:

```
-- should print 'true'
print(coroutine.resume(coroutine.create(function()
  coroutine.resume(coroutine.running())
  coroutine.yield()
end)))
```

   Patch:

```
ldo.c:
@@ -536,2 +536,3 @@
   int status;
+  int oldnny = L->nny;  /* save 'nny' */
   lua_lock(L);
@@ -557,3 +558,3 @@
   }
-  L->nny = 1;  /* do not allow yields */
+  L->nny = oldnny;  /* restore 'nny' */
   L->nCcalls--;
```

❖ Lua 5.2.1 1 · 2 · 3 · 4

1. Some patterns can overflow the C stack, due to recursion.
   reported by Tim Starling on 08 Jul 2012. existed since 2.5. fixed in 5.2.2.

   Example:

```
print(string.find(string.rep("a", 2^20), string.rep(".?", 2^20)))
```

2. `pcall` may not restore previous error function when inside coroutines.
   reported by Alexander Gavrilov on 12 Jun 2012. existed since 5.2.0. fixed in 5.2.2.

   Example:

```
function errfunc(x)
  return 'errfunc'
end

function test(do_yield)
  print(do_yield and "yielding" or "not yielding")
  pcall(function() -- this pcall sets errfunc back to none
    if do_yield then
      coroutine.yield() -- stops errfunc from being restored
    end
  end)
  error('fail!')
end

coro = coroutine.wrap(function()
  print(xpcall(test, errfunc, false))
  print(xpcall(test, errfunc, true))
  print(xpcall(test, errfunc, false))
end)

coro()
--> not yielding
--> false       errfunc
--> yielding
coro()
--> false       temp:12: fail!       <<<< should be 'errfunc' too
--> not yielding
--> false       errfunc
```

   Patch:

```
ldo.c:
@@ -403,7 +403,11 @@
   int n;
   lua_assert(ci->u.c.k != NULL);  /* must have a continuation */
   lua_assert(L->nny == 0);
-  /* finish 'lua_callk' */
+  if (ci->callstatus & CIST_YPCALL) {  /* was inside a pcall? */
+    ci->callstatus &= ~CIST_YPCALL;  /* finish 'lua_pcall' */
+    L->errfunc = ci->u.c.old_errfunc;
+  }
+  /* finish 'lua_callk'/'lua_pcall' */
   adjustresults(L, ci->nresults);
   /* call continuation function */
   if (!(ci->callstatus & CIST_STAT))  /* no call status? */
```

3. Check for garbage collector in function calls does not cover all paths.
   reported by Roberto on 15 Aug 2012. existed since 5.2.1. fixed in 5.2.2.

   Example: See http://lua-users.org/lists/lua-l/2012-08/msg00149.html.

   Patch:

```
ldo.c:
@@ -311,6 +311,7 @@
       ci->top = L->top + LUA_MINSTACK;
       lua_assert(ci->top <= L->stack_last);
       ci->callstatus = 0;
+      luaC_checkGC(L);  /* stack grow uses memory */
       if (L->hookmask & LUA_MASKCALL)
         luaD_hook(L, LUA_HOOKCALL, -1);
       lua_unlock(L);
@@ -338,6 +339,7 @@
       ci->u.l.savedpc = p->code;  /* starting point */
       ci->callstatus = CIST_LUA;
```

```
       L->top = ci->top;
+      luaC_checkGC(L);  /* stack grow uses memory */
       if (L->hookmask & LUA_MASKCALL)
         callhook(L, ci);
       return 0;
@@ -393,7 +395,6 @@
     luaV_execute(L);  /* call it */
   if (!allowyield) L->nny--;
   L->nCcalls--;
-  luaC_checkGC(L);
 }
```

4. `load` and `loadfile` return wrong result when given an environment for a binary chunk with no upvalues.
   reported by Vladimir Strakh on 28 Nov 2012. existed since 5.2.0. fixed in 5.2.2.

   Example:

```
f = load(string.dump(function () return 1 end), nil, "b", {})
print(type(f))   --> table      (should be function)
```

   Patch:

```
lbaselib.c:
@@ -244,5 +244,11 @@

-static int load_aux (lua_State *L, int status) {
-  if (status == LUA_OK)
+static int load_aux (lua_State *L, int status, int envidx) {
+  if (status == LUA_OK) {
+    if (envidx != 0) {  /* 'env' parameter? */
+      lua_pushvalue(L, envidx);  /* environment for loaded function */
+      if (!lua_setupvalue(L, -2, 1))  /* set it as 1st upvalue */
+        lua_pop(L, 1);  /* remove 'env' if not used by previous call */
+    }
     return 1;
+  }
   else {
@@ -258,9 +264,5 @@
   const char *mode = luaL_optstring(L, 2, NULL);
-  int env = !lua_isnone(L, 3);  /* 'env' parameter? */
+  int env = (!lua_isnone(L, 3) ? 3 : 0);  /* 'env' index or 0 if no 'env' */
   int status = luaL_loadfilex(L, fname, mode);
-  if (status == LUA_OK && env) {  /* 'env' parameter? */
-    lua_pushvalue(L, 3);
-    lua_setupvalue(L, -2, 1);  /* set it as 1st upvalue of loaded chunk */
-  }
-  return load_aux(L, status);
+  return load_aux(L, status, env);
 }
@@ -309,5 +311,5 @@
   size_t l;
-  int top = lua_gettop(L);
   const char *s = lua_tolstring(L, 1, &l);
   const char *mode = luaL_optstring(L, 3, "bt");
+  int env = (!lua_isnone(L, 4) ? 4 : 0);  /* 'env' index or 0 if no 'env' */
   if (s != NULL) {  /* loading a string? */
@@ -322,7 +324,3 @@
   }
-  if (status == LUA_OK && top >= 4) {  /* is there an 'env' argument */
-    lua_pushvalue(L, 4);  /* environment for loaded function */
-    lua_setupvalue(L, -2, 1);  /* set it as 1st upvalue */
-  }
-  return load_aux(L, status);
+  return load_aux(L, status, env);
 }
```

❖ Lua 5.2.0 1 · 2 · 3 · 4 · 5 · 6

1. Memory hoarding when creating Lua hooks for coroutines.
   reported by Arseny Vakhrushev on 16 Jan 2012. existed since 5.1. fixed in 5.2.1.

   Example:

```
collectgarbage(); print(collectgarbage'count' * 1024)

for i = 1, 100 do
  local co = coroutine.create(function () end)
  local x = {}
  for j=1,1000 do x[j] = j end
```

```
  debug.sethook(co, function () return x end, 'l')
end

collectgarbage(); print(collectgarbage'count' * 1024)
-- value should back to near the original level
```

Patch:

```
ldblib.c:
@@ -253,14 +253,15 @@
 }


-#define gethooktable(L)        luaL_getsubtable(L, LUA_REGISTRYINDEX, HOOKKEY);
+#define gethooktable(L)        luaL_getsubtable(L, LUA_REGISTRYINDEX, HOOKKEY)


 static void hookf (lua_State *L, lua_Debug *ar) {
   static const char *const hooknames[] =
     {"call", "return", "line", "count", "tail call"};
   gethooktable(L);
-  lua_rawgetp(L, -1, L);
+  lua_pushthread(L);
+  lua_rawget(L, -2);
   if (lua_isfunction(L, -1)) {
     lua_pushstring(L, hooknames[(int)ar->event]);
     if (ar->currentline >= 0)
@@ -306,10 +307,15 @@
     count = luaL_optint(L, arg+3, 0);
     func = hookf; mask = makemask(smask, count);
   }
-  gethooktable(L);
+  if (gethooktable(L) == 0) {  /* creating hook table? */
+    lua_pushstring(L, "k");
+    lua_setfield(L, -2, "__mode");  /** hooktable.__mode = "k" */
+    lua_pushvalue(L, -1);
+    lua_setmetatable(L, -2);  /* setmetatable(hooktable) = hooktable */
+  }
+  lua_pushthread(L1); lua_xmove(L1, L, 1);
   lua_pushvalue(L, arg+1);
-  lua_rawsetp(L, -2, L1);  /* set new hook */
-  lua_pop(L, 1);  /* remove hook table */
+  lua_rawset(L, -3);  /* set new hook */
   lua_sethook(L1, func, mask, count);  /* set hooks */
   return 0;
 }
@@ -325,7 +331,8 @@
     lua_pushliteral(L, "external hook");
   else {
     gethooktable(L);
-    lua_rawgetp(L, -1, L1);   /* get hook */
+    lua_pushthread(L1); lua_xmove(L1, L, 1);
+    lua_rawget(L, -2);   /* get hook */
     lua_remove(L, -2);  /* remove hook table */
   }
   lua_pushstring(L, unmakemask(mask, buff));
```

2. Lexical gets confused with some combination of arithmetic operators and hexadecimal numbers.
   reported by Alexandra Barros on 17 Jan 2012. existed since 5.2.0. fixed in 5.2.1.

Example:

```
print(0xE+1)
```

Patch:

```
llex.c:
@@ -223,12 +223,19 @@

 /* LUA_NUMBER */
 static void read_numeral (LexState *ls, SemInfo *seminfo) {
+  const char *expo = "Ee";
+  int first = ls->current;
   lua_assert(lisdigit(ls->current));
-  do {
-    save_and_next(ls);
-    if (check_next(ls, "EePp"))  /* exponent part? */
+  save_and_next(ls);
+  if (first == '0' && check_next(ls, "Xx"))  /* hexadecimal? */
+    expo = "Pp";
```

```
+  for (;;) {
+    if (check_next(ls, expo))  /* exponent part? */
       check_next(ls, "+-");  /* optional exponent sign */
-  } while (lislalnum(ls->current) || ls->current == '.');
+    if (lisxdigit(ls->current) || ls->current == '.')
+      save_and_next(ls);
+    else  break;
+  }
   save(ls, '\0');
   buffreplace(ls, '.', ls->decpoint);  /* follow locale for decimal point */
   if (!buff2d(ls->buff, &seminfo->r))  /* format error? */
```

3. Finalizers may call functions from a dynamic library after the library has been unloaded.
   reported by Josh Haberman on 08 Apr 2012. existed since 5.1. fixed in 5.2.1.

   Example:

```
local u = setmetatable({}, {__gc = function () foo() end})
local m = require 'mod'   -- 'mod' may be any dynamic library written in C
foo = m.foo     -- 'foo' may be any function from 'mod'
-- end program; it crashes
```

   Patch:

```
loadlib.c:
95c95
< #define LIBPREFIX     "LOADLIB: "
---
> #define CLIBS         "_CLIBS"
251,266c251,256
<
< static void **ll_register (lua_State *L, const char *path) {
<   void **plib;
<   lua_pushfstring(L, "%s%s", LIBPREFIX, path);
<   lua_gettable(L, LUA_REGISTRYINDEX);  /* check library in registry? */
<   if (!lua_isnil(L, -1))  /* is there an entry? */
<     plib = (void **)lua_touserdata(L, -1);
<   else {  /* no entry yet; create one */
<     lua_pop(L, 1);  /* remove result from gettable */
<     plib = (void **)lua_newuserdata(L, sizeof(const void *));
<     *plib = NULL;
<     luaL_setmetatable(L, "_LOADLIB");
<     lua_pushfstring(L, "%s%s", LIBPREFIX, path);
<     lua_pushvalue(L, -2);
<     lua_settable(L, LUA_REGISTRYINDEX);
<   }
---
> static void *ll_checkclib (lua_State *L, const char *path) {
>   void *plib;
>   lua_getfield(L, LUA_REGISTRYINDEX, CLIBS);
>   lua_getfield(L, -1, path);
>   plib = lua_touserdata(L, -1);  /* plib = CLIBS[path] */
>   lua_pop(L, 2);  /* pop CLIBS table and 'plib' */
270a261,270
> static void ll_addtoclib (lua_State *L, const char *path, void *plib) {
>   lua_getfield(L, LUA_REGISTRYINDEX, CLIBS);
>   lua_pushlightuserdata(L, plib);
>   lua_pushvalue(L, -1);
>   lua_setfield(L, -3, path);  /* CLIBS[path] = plib */
>   lua_rawseti(L, -2, luaL_len(L, -2) + 1);  /* CLIBS[#CLIBS + 1] = plib */
>   lua_pop(L, 1);  /* pop CLIBS table */
> }
>
>
272,273c272,273
< ** __gc tag method: calls library's `ll_unloadlib' function with the lib
< ** handle
---
> ** __gc tag method for CLIBS table: calls 'll_unloadlib' for all lib
> ** handles in list CLIBS
276,278c276,281
<   void **lib = (void **)luaL_checkudata(L, 1, "_LOADLIB");
<   if (*lib) ll_unloadlib(*lib);
<   *lib = NULL;  /* mark library as closed */
---
>   int n = luaL_len(L, 1);
>   for (; n >= 1; n--) {  /* for each handle, in reverse order */
>     lua_rawgeti(L, 1, n);  /* get handle CLIBS[n] */
>     ll_unloadlib(lua_touserdata(L, -1));
>     lua_pop(L, 1);  /* pop handle */
>   }
```

```
284,286c287,292
<   void **reg = ll_register(L, path);
<   if (*reg == NULL) *reg = ll_load(L, path, *sym == '*');
<   if (*reg == NULL) return ERRLIB;  /* unable to load library */
---
>   void *reg = ll_checkclib(L, path);  /* check loaded C libraries */
>   if (reg == NULL) {  /* must load library? */
>     reg = ll_load(L, path, *sym == '*');
>     if (reg == NULL) return ERRLIB;  /* unable to load library */
>     ll_addtoclib(L, path, reg);
>   }
292c298
<     lua_CFunction f = ll_sym(L, *reg, sym);
---
>     lua_CFunction f = ll_sym(L, reg, sym);
675,676c681,683
<   /* create new type _LOADLIB */
<   luaL_newmetatable(L, "_LOADLIB");
---
>   /* create table CLIBS to keep track of loaded C libraries */
>   luaL_getsubtable(L, LUA_REGISTRYINDEX, CLIBS);
>   lua_createtable(L, 0, 1);  /* metatable for CLIBS */
678a686
>   lua_setmetatable(L, -2);
```

4. Wrong handling of `nCcalls` in coroutines.
   reported by Alexander Gavrilov on 18 Apr 2012. existed since 5.2.0. fixed in 5.2.1.

   Example:

```
coroutine.wrap(function()
  print(pcall(pcall,pcall,pcall,pcall,pcall,error,3))
end)()
```

   Patch:

```
ldo.c:
@@ -402,8 +402,6 @@
   int n;
   lua_assert(ci->u.c.k != NULL);  /* must have a continuation */
   lua_assert(L->nny == 0);
-  /* finish 'luaD_call' */
-  L->nCcalls--;
   /* finish 'lua_callk' */
   adjustresults(L, ci->nresults);
   /* call continuation function */
@@ -513,7 +511,6 @@
         api_checknelems(L, n);
         firstArg = L->top - n;  /* yield results come from continuation */
       }
-      L->nCcalls--;  /* finish 'luaD_call' */
       luaD_poscall(L, firstArg);  /* finish 'luaD_precall' */
     }
     unroll(L, NULL);
```

5. Internal Lua values may escape through the debug API.
   reported by Dan Tull on 20 Apr 2012. existed since 5.1. fixed in 5.2.1.

   Example:

```
local firsttime = true
local function foo ()
  if firsttime then
    firsttime = false
    return "a = 1"
  else
    for i = 1, 10 do
      print(debug.getlocal(2, i))
    end
  end
end

print(load(foo))   -- prints some lines and then crashes
```

6. Problems when yielding from debug hooks.
   reported by Erik Cassel on 05 Jun 2012. existed since 5.2.0. fixed in 5.2.1.

   Example: In C, set a line hook that simply yields, and then call any Lua function. You get an infinite loop of yields.

❖ Lua 5.1.5 1 · 2

That was the last release of Lua 5.1. Bugs reported later are probably fixed in Lua 5.2.

1. Comments in `src/Makefile` are not portable.
   reported by Lorenzo Donati on 09 Mar 2012. existed since 5.1.5. fixed in 5.2.0.

   Example: This glitch happens when compiling Lua on some mingw systems; it does not seem to affect other systems.

   Patch:

```
src/Makefile:
@@ -50,3 +50,3 @@
 $(LUA_A): $(CORE_O) $(LIB_O)
-       $(AR) $@ $(CORE_O) $(LIB_O)     # DLL needs all object files
+       $(AR) $@ $(CORE_O) $(LIB_O)
        $(RANLIB) $@
```

2. When loading a file, Lua may call the reader function again after it returned end of input .
   reported by Chris Howie on 05 Jun 2013. existed since 5.1. fixed in 5.2.0.

   Example:

```
load(function () print("called"); return nil end)
--> called
--> called             (should be called only once!)
```

   Patch:

```
lzio.h:
@@ -59,6 +59,7 @@
   lua_Reader reader;
   void* data;                  /* additional data */
   lua_State *L;                        /* Lua state (for reader) */
+  int eoz;                     /* true if reader has no more data */
 };


lzio.c:
@@ -22,10 +22,14 @@
   size_t size;
   lua_State *L = z->L;
   const char *buff;
+  if (z->eoz) return EOZ;
   lua_unlock(L);
   buff = z->reader(L, z->data, &size);
   lua_lock(L);
-  if (buff == NULL || size == 0) return EOZ;
+  if (buff == NULL || size == 0) {
+    z->eoz = 1;  /* avoid calling reader function next time */
+    return EOZ;
+  }
   z->n = size - 1;
   z->p = buff;
   return char2int(*(z->p++));
@@ -51,6 +55,7 @@
   z->data = data;
   z->n = 0;
   z->p = NULL;
+  z->eoz = 0;
 }
```

❖ Lua 5.1.4 1 · 2 · 3 · 4 · 5 · 6 · 7 · 8 · 9 · 10 · 11

1. Maliciously crafted precompiled code can crash Lua.
   reported by Peter Cawley on 01 Sep 2008.

   Solution: To avoid running precompiled code from untrusted sources, raise an error if the first byte in the stream is the escape character (decimal 27).

2. Smart use of varargs may create functions that return too many arguments and overflow the stack of C functions.
   reported by Patrick Donnelly on 10 Dec 2008. fixed in 5.1.5.

   Example:

```
function lunpack(i, ...)
  if i == 0 then
    return ...
  else
    return lunpack(i-1, 1, ...)
  end
end
```

Now, if C calls `lunpack(n)` with a huge n, it may end with too many values in its stack and confuse its stack indices.

3. Wrong code generation for some particular boolean expressions. (see also 9)
   reported by Brian Kelley on 15 Apr 2009. existed since 5.0. fixed in 5.1.5.

   Example:

```
print(((1 or false) and true) or false)   --> 1, but should be 'true'
```

   Patch: (partial solution; see also 9)

```
lcode.c:
@@ -544,15 +544,18 @@
       pc = NO_JUMP;  /* always true; do nothing */
       break;
     }
-    case VFALSE: {
-      pc = luaK_jump(fs);  /* always jump */
-      break;
-    }
     case VJMP: {
       invertjump(fs, e);
       pc = e->u.s.info;
       break;
     }
+    case VFALSE: {
+      if (!hasjumps(e)) {
+        pc = luaK_jump(fs);  /* always jump */
+        break;
+      }
+      /* else go through */
+    }
     default: {
       pc = jumponcond(fs, e, 0);
       break;
@@ -572,14 +575,17 @@
       pc = NO_JUMP;  /* always false; do nothing */
       break;
     }
-    case VTRUE: {
-      pc = luaK_jump(fs);  /* always jump */
-      break;
-    }
     case VJMP: {
       pc = e->u.s.info;
       break;
     }
+    case VTRUE: {
+      if (!hasjumps(e)) {
+        pc = luaK_jump(fs);  /* always jump */
+        break;
+      }
+      /* else go through */
+    }
     default: {
       pc = jumponcond(fs, e, 1);
       break;
```

4. `luaV_settable` may invalidate a reference to a table and try to reuse it.
   reported by Mark Feldman on 27 Jun 2009. existed since 5.0. fixed in 5.1.5.

   Example:

```
grandparent = {}
grandparent.__newindex = function(s,_,_) print(s) end

parent = {}
parent.__newindex = parent
setmetatable(parent, grandparent)
```

```
child = setmetatable({}, parent)
child.foo = 10      --> (crash on some machines)
```

Patch:

```
lvm.c:
@@ -133,6 +133,7 @@

 void luaV_settable (lua_State *L, const TValue *t, TValue *key, StkId val) {
   int loop;
+  TValue temp;
   for (loop = 0; loop < MAXTAGLOOP; loop++) {
     const TValue *tm;
     if (ttistable(t)) {  /* `t' is a table? */
@@ -152,7 +153,9 @@
       callTM(L, tm, t, key, val);
       return;
     }
-    t = tm;  /* else repeat with `tm' */
+    /* else repeat with `tm' */
+    setobj(L, &temp, tm);  /* avoid pointing inside table (may rehash) */
+    t = &temp;
   }
   luaG_runerror(L, "loop in settable");
 }
```

5. `debug.getfenv` does not check whether it has an argument.
   reported by Patrick Donnelly on 30 Jul 2009. existed since 5.1. fixed in 5.1.5.

   Example:

   ```
   debug.getfenv()   -- should raise an error
   ```

   Patch:

   ```
   ldblib.c:
   @@ -45,6 +45,7 @@


    static int db_getfenv (lua_State *L) {
   +  luaL_checkany(L, 1);
      lua_getfenv(L, 1);
      return 1;
    }
   ```

6. GC may get stuck during parsing and avoids proper resizing of the string table, making its lists grow too much and degrading performance.
   reported by Sean Conner on 10 Nov 2009. existed since 5.1. fixed in 5.1.5.

   Example: See http://lua-users.org/lists/lua-l/2009-11/msg00463.html.

   Patch:

   ```
   llex.c:
   @@ -118,8 +118,10 @@
      lua_State *L = ls->L;
      TString *ts = luaS_newlstr(L, str, l);
      TValue *o = luaH_setstr(L, ls->fs->h, ts);  /* entry for `str' */
   -  if (ttisnil(o))
   +  if (ttisnil(o)) {
        setbvalue(o, 1);  /* make sure `str' will not be collected */
   +    luaC_checkGC(L);
   +  }
      return ts;
    }
   ```

7. `string.format` may get buffer as an argument when there are missing arguments and format string is too long.
   reported by Roberto on 12 Apr 2010. existed since 5.0. fixed in 5.1.5.

   Example:

   ```
   x = string.rep("x", 10000) .. "%d"
   print(string.format(x))    -- gives wrong error message
   ```

   Patch:

```
lstrlib.c:
@@ -754,6 +754,7 @@


 static int str_format (lua_State *L) {
+  int top = lua_gettop(L);
   int arg = 1;
   size_t sfl;
   const char *strfrmt = luaL_checklstring(L, arg, &sfl);
@@ -768,7 +769,8 @@
     else { /* format item */
       char form[MAX_FORMAT];  /* to store the format (`%...') */
       char buff[MAX_ITEM];  /* to store the formatted item */
-      arg++;
+      if (++arg > top)
+        luaL_argerror(L, arg, "no value");
       strfrmt = scanformat(L, strfrmt, form);
       switch (*strfrmt++) {
         case 'c': {
```

8. `io.read("*n","*n")` may return garbage if second read fails.
   reported by Roberto on 12 Apr 2010. existed since 5.0. fixed in 5.1.5.

   Example:

```
print(io.read("*n", "*n"))   --<< enter "10   hi"
--> file (0x884420)     nil
```

   Patch:

```
liolib.c:
@@ -276,7 +276,10 @@
     lua_pushnumber(L, d);
     return 1;
   }
-  else return 0;  /* read fails */
+  else {
+    lua_pushnil(L);  /* "result" to be removed */
+    return 0;  /* read fails */
+  }
 }
```

9. Wrong code generation for some particular boolean expressions.
   reported by Thierry Van Elsuwe on 20 Jan 2011. existed since 5.0. fixed in 5.1.5.

   Example:

```
print((('hi' or true) and true) or true)
--> hi     (should be true)
print(((nil and nil) or false) and true)
--> nil    (should be false)
```

   Patch: (to be applied after the patch in 3)

```
lcode.c:
@@ -549,13 +549,6 @@
       pc = e->u.s.info;
       break;
     }
-    case VFALSE: {
-      if (!hasjumps(e)) {
-        pc = luaK_jump(fs);  /* always jump */
-        break;
-      }
-      /* else go through */
-    }
     default: {
       pc = jumponcond(fs, e, 0);
       break;
@@ -579,13 +572,6 @@
       pc = e->u.s.info;
       break;
     }
-    case VTRUE: {
-      if (!hasjumps(e)) {
-        pc = luaK_jump(fs);  /* always jump */
-        break;
```

```
-      }
-      /* else go through */
-    }
     default: {
       pc = jumponcond(fs, e, 1);
       break;
```

10. Newindex metamethod may not work if metatable is its own metatable.
reported by Cuero Bugot on 09 Aug 2011. existed since 5.1. fixed in 5.1.5.

Example:

```
meta={}
setmetatable(meta, meta)
meta.__newindex = function(t, key, value) print("set") end
o = setmetatable({}, meta)
o.x = 10    -- should print 'set'
```

Patch:

```
lvm.c:
@@ -142,6 +142,7 @@
       if (!ttisnil(oldval) ||  /* result is no nil? */
           (tm = fasttm(L, h->metatable, TM_NEWINDEX)) == NULL) { /* or no TM? */
         setobj2t(L, oldval, val);
+        h->flags = 0;
         luaC_barriert(L, h, val);
         return;
       }
```

11. Parser may collect a prototype while building it.
reported by Ingo van Lil on 13 Oct 2011. existed since 5.1.4 (caused by patch 5.1.4-6). fixed in 5.1.5.

Patch:

```
lparser.c:
@@ -374,9 +374,9 @@
   lua_assert(luaG_checkcode(f));
   lua_assert(fs->bl == NULL);
   ls->fs = fs->prev;
-  L->top -= 2;  /* remove table and prototype from the stack */
   /* last token read was anchored in defunct function; must reanchor it */
   if (fs) anchor_token(ls);
+  L->top -= 2;  /* remove table and prototype from the stack */
 }
```

❖ Lua 5.1.3 1 · 2 · 3 · 4 · 5 · 6 · 7 · 8 · 9 · 10 · 11 · 12

1. `LUAI_MAXCSTACK` must be smaller than `-LUA_REGISTRYINDEX`.
reported by Patrick Donnelly on 11 Feb 2008. existed since 5.1.3. fixed in 5.1.4.

Example:

```
j = 1e4
co = coroutine.create(function()
       t = {}
       for i = 1, j do t[i] = i end
       return unpack(t)
end)
print(coroutine.resume(co))
```

Patch:

```
luaconf.h:
443c443,444
< ** functions to consume unlimited stack space.
---
> ** functions to consume unlimited stack space. (must be smaller than
> ** -LUA_REGISTRYINDEX)
445,446c446
< #define LUAI_MCS_AUX  ((int)(INT_MAX / (4*sizeof(LUA_NUMBER))))
< #define LUAI_MAXCSTACK        (LUAI_MCS_AUX > SHRT_MAX ? SHRT_MAX : LUAI_MCS_AUX)
---
> #define LUAI_MAXCSTACK        8000
```

2. `coroutine.resume` pushes element without ensuring stack size.
   reported on 11 Feb 2008. existed since 5.0. fixed in 5.1.4.

   Example: This bug cannot be detected without internal assertions.

   Patch:

```
lbaselib.c:
@@ -526,7 +526,7 @@
   status = lua_resume(co, narg);
   if (status == 0 || status == LUA_YIELD) {
     int nres = lua_gettop(co);
-    if (!lua_checkstack(L, nres))
+    if (!lua_checkstack(L, nres + 1))
       luaL_error(L, "too many results to resume");
     lua_xmove(co, L, nres);  /* move yielded values */
     return nres;
```

3. `lua_checkstack` may have arithmetic overflow for large 'size'.
   reported by Patrick Donnelly on 12 Feb 2008. existed since 5.0. fixed in 5.1.4.

   Example:

```
print(unpack({1,2,3}, 0, 2^31-3))
```

   Patch:

```
lapi.c:
@@ -93,15 +93,14 @@


 LUA_API int lua_checkstack (lua_State *L, int size) {
-  int res;
+  int res = 1;
   lua_lock(L);
-  if ((L->top - L->base + size) > LUAI_MAXCSTACK)
+  if (size > LUAI_MAXCSTACK || (L->top - L->base + size) > LUAI_MAXCSTACK)
     res = 0;  /* stack overflow */
-  else {
+  else if (size > 0) {
     luaD_checkstack(L, size);
     if (L->ci->top < L->top + size)
       L->ci->top = L->top + size;
-    res = 1;
   }
   lua_unlock(L);
   return res;
```

4. `unpack` with maximum indices may crash due to arithmetic overflow.
   reported by Patrick Donnelly on 12 Feb 2008. existed since 5.1. fixed in 5.1.4.

   Example:

```
print(unpack({1,2,3}, 2^31-1, 2^31-1))
```

   Patch:

```
lbaselib.c:
@@ -344,10 +344,12 @@
   luaL_checktype(L, 1, LUA_TTABLE);
   i = luaL_optint(L, 2, 1);
   e = luaL_opt(L, luaL_checkint, 3, luaL_getn(L, 1));
+  if (i > e) return 0;  /* empty range */
   n = e - i + 1;  /* number of elements */
-  if (n <= 0) return 0;  /* empty range */
-  luaL_checkstack(L, n, "table too big to unpack");
-  for (; i<=e; i++)  /* push arg[i...e] */
+  if (n <= 0 || !lua_checkstack(L, n))  /* n <= 0 means arith. overflow */
+    return luaL_error(L, "too many results to unpack");
+  lua_rawgeti(L, 1, i);  /* push arg[i] (avoiding overflow problems) */
+  while (i++ < e)  /* push arg[i + 1...e] */
     lua_rawgeti(L, 1, i);
   return n;
 }
```

5. Maliciously crafted precompiled code can crash Lua.
   reported by Peter Cawley on 24 Mar 2008. existed since 5.0. fixed in 5.1.4.

Example:

```
a = string.dump(function()return;end)
a = a:gsub(string.char(30,37,122,128), string.char(34,0,0), 1)
loadstring(a)()
```

Patch:

```
ldebug.c:
@@ -275,12 +275,12 @@

 static int precheck (const Proto *pt) {
   check(pt->maxstacksize <= MAXSTACK);
-  lua_assert(pt->numparams+(pt->is_vararg & VARARG_HASARG) <= pt->maxstacksize);
-  lua_assert(!(pt->is_vararg & VARARG_NEEDSARG) ||
+  check(pt->numparams+(pt->is_vararg & VARARG_HASARG) <= pt->maxstacksize);
+  check(!(pt->is_vararg & VARARG_NEEDSARG) ||
                (pt->is_vararg & VARARG_HASARG));
   check(pt->sizeupvalues <= pt->nups);
   check(pt->sizelineinfo == pt->sizecode || pt->sizelineinfo == 0);
-  check(GET_OPCODE(pt->code[pt->sizecode-1]) == OP_RETURN);
+  check(pt->sizecode > 0 && GET_OPCODE(pt->code[pt->sizecode-1]) == OP_RETURN);
   return 1;
 }

@@ -363,7 +363,11 @@
     }
     switch (op) {
       case OP_LOADBOOL: {
-        check(c == 0 || pc+2 < pt->sizecode);  /* check its jump */
+        if (c == 1) {  /* does it jump? */
+          check(pc+2 < pt->sizecode);  /* check its jump */
+          check(GET_OPCODE(pt->code[pc+1]) != OP_SETLIST ||
+                GETARG_C(pt->code[pc+1]) != 0);
+        }
         break;
       }
       case OP_LOADNIL: {
@@ -428,7 +432,10 @@
       }
       case OP_SETLIST: {
         if (b > 0) checkreg(pt, a + b);
-        if (c == 0) pc++;
+        if (c == 0) {
+          pc++;
+          check(pc < pt->sizecode - 1);
+        }
         break;
       }
       case OP_CLOSURE: {
```

6. Maliciously crafted precompiled code can blow the C stack.
   reported by Greg Falcon on 25 Mar 2008. existed since 5.0. fixed in 5.1.4.

Example:

```
function crash(depth)
  local init = '\27\76\117\97\81\0\1\4\4\4\8\0\7\0\0\0\61\115\116' ..
               '\100\105\110\0\1\0\0\0\1\0\0\0\0\0\0\2\2\0\0\0\36' ..
               '\0\0\0\30\0\128\0\0\0\0\0\1\0\0\0\0\0\0\0\1\0\0\0' ..
               '\1\0\0\0\0\0\0\2'
  local mid = '\1\0\0\0\30\0\128\0\0\0\0\0\0\0\0\1\0\0\0\1\0\0\0\0'
  local fin = '\0\0\0\0\0\0\0\2\0\0\0\1\0\0\0\1\0\0\0\1\0\0\0\2\0' ..
              '\0\0\97\0\1\0\0\0\1\0\0\0\0\0\0\0'
  local lch = '\2\0\0\0\36\0\0\0\30\0\128\0\0\0\0\0\1\0\0\0\0\0\0' ..
              '\0\1\0\0\0\1\0\0\0\0\0\0\2'
  local rch = '\0\0\0\0\0\0\0\2\0\0\0\1\0\0\0\1\0\0\0\1\0\0\0\2\0' ..
              '\0\0\97\0\1\0\0\0\1'
  for i=1,depth do lch,rch = lch..lch,rch..rch end
  loadstring(init .. lch .. mid .. rch .. fin)
end
for i=1,25 do print(i); crash(i) end
```

Patch:

```
lundump.c:
@@ -161,7 +160,9 @@

 static Proto* LoadFunction(LoadState* S, TString* p)
 {
```

```
- Proto* f=luaF_newproto(S->L);
+ Proto* f;
+ if (++S->L->nCcalls > LUAI_MAXCCALLS) error(S,"code too deep");
+ f=luaF_newproto(S->L);
  setptvalue2s(S->L,S->L->top,f); incr_top(S->L);
  f->source=LoadString(S); if (f->source==NULL) f->source=p;
  f->linedefined=LoadInt(S);
@@ -175,6 +176,7 @@
  LoadDebug(S,f);
  IF (!luaG_checkcode(f), "bad code");
  S->L->top--;
+ S->L->nCcalls--;
  return f;
 }
```

7. Code validator may reject (maliciously crafted) correct code.
   reported by Greg Falcon on 26 Mar 2008. existed since 5.0. fixed in 5.1.4.

   Example:

```
z={}
for i=1,27290 do z[i]='1,' end
z = 'if 1+1==2 then local a={' .. table.concat(z) .. '} end'
func = loadstring(z)
print(loadstring(string.dump(func)))
```

   Patch:

```
ldebug.c:
@@ -346,9 +346,18 @@
           int dest = pc+1+b;
           check(0 <= dest && dest < pt->sizecode);
           if (dest > 0) {
-            /* cannot jump to a setlist count */
-            Instruction d = pt->code[dest-1];
-            check(!(GET_OPCODE(d) == OP_SETLIST && GETARG_C(d) == 0));
+            int j;
+            /* check that it does not jump to a setlist count; this
+               is tricky, because the count from a previous setlist may
+               have the same value of an invalid setlist; so, we must
+               go all the way back to the first of them (if any) */
+            for (j = 0; j < dest; j++) {
+              Instruction d = pt->code[dest-1-j];
+              if (!(GET_OPCODE(d) == OP_SETLIST && GETARG_C(d) == 0)) break;
+            }
+            /* if 'j' is even, previous value is not a setlist (even if
+               it looks like one) */
+            check((j&1) == 0);
           }
         }
         break;
```

8. Maliciously crafted precompiled code can inject invalid boolean values into Lua code.
   reported by Greg Falcon on 27 Mar 2008. existed since 5.0. fixed in 5.1.4.

   Example:

```
maybe = string.dump(function() return ({[true]=true})[true] end)
maybe = maybe:gsub('\1\1','\1\2')
maybe = loadstring(maybe)()
assert(type(maybe) == "boolean" and maybe ~= true and maybe ~= false)
```

   Patch:

```
lundump.c:
@@ -115,7 +115,7 @@
        setnilvalue(o);
        break;
    case LUA_TBOOLEAN:
-       setbvalue(o,LoadChar(S));
+       setbvalue(o,LoadChar(S)!=0);
        break;
    case LUA_TNUMBER:
        setnvalue(o,LoadNumber(S));
```

9. `string.byte` gets confused with some out-of-range negative indices.
   reported by Mike Pall on 03 Jun 2008. existed since 5.1. fixed in 5.1.4.

   Example:

```
print(string.byte("abc", -5))   --> 97   98   99   (should print nothing)
```

Patch:

```
lstrlib.c:
@@ -35,7 +35,8 @@

 static ptrdiff_t posrelat (ptrdiff_t pos, size_t len) {
   /* relative string position: negative means back from end */
-  return (pos>=0) ? pos : (ptrdiff_t)len+pos+1;
+  if (pos < 0) pos += (ptrdiff_t)len + 1;
+  return (pos >= 0) ? pos : 0;
 }
```

10. User-requested GC step may loop forever.
reported by Makoto Hamanaka on 01 Jul 2008. existed since 5.1. fixed in 5.1.4.

Example:

```
collectgarbage("setpause", 100) -- small value
collectgarbage("setstepmul", 2000) -- large value
collectgarbage("step",0)
```

Patch:

```
lapi.c:
@@ -929,10 +929,13 @@
         g->GCthreshold = g->totalbytes - a;
       else
         g->GCthreshold = 0;
-      while (g->GCthreshold <= g->totalbytes)
+      while (g->GCthreshold <= g->totalbytes) {
         luaC_step(L);
-      if (g->gcstate == GCSpause)  /* end of cycle? */
-        res = 1;  /* signal it */
+        if (g->gcstate == GCSpause) {  /* end of cycle? */
+          res = 1;  /* signal it */
+          break;
+        }
+      }
       break;
     }
     case LUA_GCSETPAUSE: {
```

11. `module` may change the environment of a C function.
reported by Peter Cawley on 16 Jul 2008. existed since 5.1. fixed in 5.1.4.

Example:

```
pcall(module, "xuxu")
assert(debug.getfenv(pcall) == xuxu)
```

Patch:

```
loadlib.c:
@@ -506,8 +506,11 @@

 static void setfenv (lua_State *L) {
   lua_Debug ar;
-  lua_getstack(L, 1, &ar);
-  lua_getinfo(L, "f", &ar);
+  if (lua_getstack(L, 1, &ar) == 0 ||
+      lua_getinfo(L, "f", &ar) == 0 ||  /* get calling function */
+      lua_iscfunction(L, -1))
+    luaL_error(L, "function " LUA_QL("module")
+                  " not called from a Lua function");
   lua_pushvalue(L, -2);
   lua_setfenv(L, -2);
   lua_pop(L, 1);
```

12. Internal macro `svalue` is wrong.
reported by Martijn van Buul on 04 Aug 2008. existed since 5.1. fixed in 5.1.4.

Example:

```
/* in luaconf.h */
#define LUAI_USER_ALIGNMENT_T   union { char b[32]; }
```

Patch:

```
lobject.h:
@@ -210,3 +210,3 @@
 #define getstr(ts)     cast(const char *, (ts) + 1)
-#define svalue(o)       getstr(tsvalue(o))
+#define svalue(o)       getstr(rawtsvalue(o))
```

❖ Lua 5.1.2 1 · 2 · 3 · 4 · 5 · 6 · 7 · 8 · 9 · 10 · 11 · 12 · 13

1. Lua may close standard files, which then may be used by C.
   reported by David Manura on 17 Apr 2007. fixed in 5.1.3.

2. Code generated for `-nil`, `-true`, and `-false` is wrong.
   reported by David Manura and Rici Lake on 29 Apr 2007. existed since 5.1. fixed in 5.1.3.

   Example:

   ```
   print(-nil)
   ```

   Patch:

   ```
   lcode.c:
   @@ -699,7 +699,7 @@
      e2.t = e2.f = NO_JUMP; e2.k = VKNUM; e2.u.nval = 0;
      switch (op) {
        case OPR_MINUS: {
   -      if (e->k == VK)
   +      if (!isnumeral(e))
            luaK_exp2anyreg(fs, e);  /* cannot operate on non-numeric constants */
          codearith(fs, OP_UNM, e, &e2);
          break;
   ```

3. Count hook may be called without being set.
   reported by Mike Pall in May 2007. fixed in 5.1.3.

   Patch:

   ```
   lvm.c:
   @@ -61,11 +61,9 @@
      lu_byte mask = L->hookmask;
      const Instruction *oldpc = L->savedpc;
      L->savedpc = pc;
   -  if (mask > LUA_MASKLINE) {  /* instruction-hook set? */
   -    if (L->hookcount == 0) {
   -      resethookcount(L);
   -      luaD_callhook(L, LUA_HOOKCOUNT, -1);
   -    }
   +  if ((mask & LUA_MASKCOUNT) && L->hookcount == 0) {
   +    resethookcount(L);
   +    luaD_callhook(L, LUA_HOOKCOUNT, -1);
      }
      if (mask & LUA_MASKLINE) {
        Proto *p = ci_func(L->ci)->l.p;
   ```

4. Recursive coroutines may overflow C stack.

   Example:

   ```
   a = function(a) coroutine.wrap(a)(a) end
   a(a)
   ```

   Patch: The 'nCcalls' counter should be shared by all threads. (That is, it should be declared in the 'global_State' structure, not in 'lua_State'.)

5. Wrong error message in some concatenations.
   reported by Alex Davies in May 2007. existed since 5.1.2. fixed in 5.1.3.

   Example:

   ```
   a = nil; a = (1)..a
   ```

Patch:

```
ldebug.c:
@@ -563,8 +563,8 @@


 void luaG_concaterror (lua_State *L, StkId p1, StkId p2) {
-  if (ttisstring(p1)) p1 = p2;
-  lua_assert(!ttisstring(p1));
+  if (ttisstring(p1) || ttisnumber(p1)) p1 = p2;
+  lua_assert(!ttisstring(p1) && !ttisnumber(p1));
   luaG_typeerror(L, p1, "concatenate");
 }
```

6. Very small numbers all collide in the hash function. (This creates only performance problems; the behavior is correct.).
   reported on 18 Apr 2007. existed since Lua 5.0. fixed in 5.1.3.

   Patch:

```
ltable.c:
87,88c87,88
<   n += 1;  /* normalize number (avoid -0) */
<   lua_assert(sizeof(a) <= sizeof(n));
---
>   if (luai_numeq(n, 0))  /* avoid problems with -0 */
>     return gnode(t, 0);
```

7. Too many variables in an assignment may cause a C stack overflow.
   reported by Mike Pall on 31 Jul 2007. existed since 5.0. fixed in 5.1.3.

   Example:

```
$ ulimit -s 1024       # Reduce C stack to 1MB for quicker results
$ lua -e 'local s = "a,"; for i=1,18 do s = s..s end print(loadstring("local a;"..s.."a=nil", ""))'
```

   Patch:

```
lparser.c:
@@ -938,6 +938,8 @@
     primaryexp(ls, &nv.v);
     if (nv.v.k == VLOCAL)
       check_conflict(ls, lh, &nv.v);
+    luaY_checklimit(ls->fs, nvars, LUAI_MAXCCALLS - ls->L->nCcalls,
+                    "variable names");
     assignment(ls, &nv, nvars+1);
   }
   else {  /* assignment -> `=' explist1 */
```

8. An error in a module loaded through the '-l' option shows no traceback.
   reported by David Manura on 25 Aug 2007. existed since 5.1. fixed in 5.1.3.

   Example:

```
lua -ltemp    (assuming temp.lua has an error)
```

   Patch:

```
lua.c:
@@ -144,7 +144,7 @@
 static int dolibrary (lua_State *L, const char *name) {
   lua_getglobal(L, "require");
   lua_pushstring(L, name);
-  return report(L, lua_pcall(L, 1, 0, 0));
+  return report(L, docall(L, 1, 1));
 }
```

9. `gsub` may go wild when wrongly called without its third argument and with a large subject.
   reported by Florian Berger on 26 Oct 2007. existed since 5.1. fixed in 5.1.3.

   Example:

```
x = string.rep('a', 10000) .. string.rep('b', 10000)
print(#string.gsub(x, 'b'))
```

   Patch:

```
lstrlib.c:
@@ -631,6 +631,2 @@
     }
-    default: {
-      luaL_argerror(L, 3, "string/function/table expected");
-      return;
-    }
   }
@@ -650,2 +646,3 @@
   const char *p = luaL_checkstring(L, 2);
+  int  tr = lua_type(L, 3);
   int max_s = luaL_optint(L, 4, srcl+1);
@@ -655,2 +652,5 @@
   luaL_Buffer b;
+  luaL_argcheck(L, tr == LUA_TNUMBER || tr == LUA_TSTRING ||
+                   tr == LUA_TFUNCTION || tr == LUA_TTABLE, 3,
+                      "string/function/table expected");
   luaL_buffinit(L, &b);
```

10. `table.remove` removes last element of a table when given an out-of-bound index.
    reported by Patrick Donnelly on 13 Nov 2007. existed since 5.0 at least. fixed in 5.1.3.

    Example:

```
a = {1,2,3}
table.remove(a, 4)
print(a[3])   --> nil   (should be 3)
```

    Patch:

```
ltablib.c:
@@ -118,7 +118,8 @@
 static int tremove (lua_State *L) {
   int e = aux_getn(L, 1);
   int pos = luaL_optint(L, 2, e);
-  if (e == 0) return 0;  /* table is `empty' */
+  if (!(1 <= pos && pos <= e))  /* position is outside bounds? */
+   return 0;  /* nothing to remove */
   luaL_setn(L, 1, e - 1);  /* t.n = n-1 */
   lua_rawgeti(L, 1, pos);  /* result = t[pos] */
   for ( ;pos&lt;e; pos++) {
```

11. `lua_setfenv` may crash if called over an invalid object.
    reported by Mike Pall on 28 Nov 2007. existed since 5.1. fixed in 5.1.3.

    Example:

```
> debug.setfenv(3, {})
```

    Patch:

```
lapi.c:
@@ -749,7 +749,7 @@
       res = 0;
       break;
   }
-  luaC_objbarrier(L, gcvalue(o), hvalue(L->top - 1));
+  if (res) luaC_objbarrier(L, gcvalue(o), hvalue(L->top - 1));
   L->top--;
   lua_unlock(L);
   return res;
```

12. Stand-alone interpreter shows incorrect error message when the "message" is a coroutine.
    reported by Patrick Donnelly on 17 Dec 2007. existed since 5.1. fixed in 5.1.3.

    Example:

```
> error(coroutine.create(function() end))
```

    Patch:

```
lua.c:
@@ -74,6 +74,8 @@


 static int traceback (lua_State *L) {
+  if (!lua_isstring(L, 1))  /* 'message' not a string? */
+    return 1;  /* keep it intact */
```

```
   lua_getfield(L, LUA_GLOBALSINDEX, "debug");
   if (!lua_istable(L, -1)) {
     lua_pop(L, 1);
```

13. `debug.sethook/gethook` may overflow the thread's stack.
    reported by Ivko Stanilov on 04 Jan 2008. existed since 5.1. fixed in 5.1.3.

    Example:

```
a = coroutine.create(function() yield() end)
coroutine.resume(a)
debug.sethook(a)      -- may overflow the stack of 'a'
```

    Patch:

```
ldblib.c:
@@ -268,12 +268,11 @@
     count = luaL_optint(L, arg+3, 0);
     func = hookf; mask = makemask(smask, count);
   }
-  gethooktable(L1);
-  lua_pushlightuserdata(L1, L1);
+  gethooktable(L);
+  lua_pushlightuserdata(L, L1);
   lua_pushvalue(L, arg+1);
-  lua_xmove(L, L1, 1);
-  lua_rawset(L1, -3);  /* set new hook */
-  lua_pop(L1, 1);  /* remove hook table */
+  lua_rawset(L, -3);  /* set new hook */
+  lua_pop(L, 1);  /* remove hook table */
   lua_sethook(L1, func, mask, count);  /* set hooks */
   return 0;
 }
@@ -288,11 +287,10 @@
   if (hook != NULL && hook != hookf)  /* external hook? */
     lua_pushliteral(L, "external hook");
   else {
-    gethooktable(L1);
-    lua_pushlightuserdata(L1, L1);
-    lua_rawget(L1, -2);   /* get hook */
-    lua_remove(L1, -2);  /* remove hook table */
-    lua_xmove(L1, L, 1);
+    gethooktable(L);
+    lua_pushlightuserdata(L, L1);
+    lua_rawget(L, -2);   /* get hook */
+    lua_remove(L, -2);  /* remove hook table */
   }
   lua_pushstring(L, unmakemask(mask, buff));
   lua_pushinteger(L, lua_gethookcount(L1));
```

❖ Lua 5.1.1 1 · 2 · 3 · 4 · 5 · 6 · 7 · 8 · 9

1. List constructors have wrong limit.
   reported by Norman Ramsey on 5 Jun 2006. existed since Lua 5.1. fixed in 5.1.2.

   Example:

```
a = {}
a[1] = "x={1"
for i = 2, 2^20 do
  a[i] = 1
end
a[#a + 1] = "}"
s = table.concat(a, ",")
assert(loadstring(s))()
print(#x)
```

   Patch:

```
lparser.c:
@@ -489,7 +489,7 @@

 static void listfield (LexState *ls, struct ConsControl *cc) {
   expr(ls, &cc->v);
-  luaY_checklimit(ls->fs, cc->na, MAXARG_Bx, "items in a constructor");
+  luaY_checklimit(ls->fs, cc->na, MAX_INT, "items in a constructor");
   cc->na++;
```

```
   cc->tostore++;
 }
```

2. Wrong message error in some cases involving closures.
   reported by Shmuel Zeigerman on 8 Jul 2006. existed since Lua 5.1. fixed in 5.1.2.

   Example:

```
local Var
local function main()
  NoSuchName (function() Var=0 end)
end
main()
```

   The error message is *attempt to call upvalue 'Var' (a nil value)*. It should be *attempt to call global 'NoSuchName' (a nil value)*.

   Patch:

```
ldebug.c:
@@ -435,14 +435,16 @@
         break;
       }
       case OP_CLOSURE: {
-        int nup;
+        int nup, j;
         check(b < pt->sizep);
         nup = pt->p[b]->nups;
         check(pc + nup < pt->sizecode);
-        for (; nup>0; nup--) {
-          OpCode op1 = GET_OPCODE(pt->code[pc+nup]);
+        for (j = 1; j <= nup; j++) {
+          OpCode op1 = GET_OPCODE(pt->code[pc + j]);
           check(op1 == OP_GETUPVAL || op1 == OP_MOVE);
         }
+        if (reg != NO_REG)  /* tracing? */
+          pc += nup;  /* do not 'execute' these pseudo-instructions */
         break;
       }
       case OP_VARARG: {
```

3. `string.format("%")` may read past the string.
   reported by Roberto in Sep 2006. existed since 5.0 at least. fixed in 5.1.2.

   Example:

```
print(string.format("%"))
```

   Patch:

```
lstrlib.c:
@@ -723,7 +723,7 @@

 static const char *scanformat (lua_State *L, const char *strfrmt, char *form) {
   const char *p = strfrmt;
-  while (strchr(FLAGS, *p)) p++;  /* skip flags */
+  while (*p != '\0' && strchr(FLAGS, *p) != NULL) p++;  /* skip flags */
   if ((size_t)(p - strfrmt) >= sizeof(FLAGS))
     luaL_error(L, "invalid format (repeated flags)");
   if (isdigit(uchar(*p))) p++;  /* skip width */
```

4. `os.date` throws an error when result is the empty string.
   reported by Nick Gammon on 15 Sep 2006. existed since 4.0. fixed in 5.1.2.

   Example:

```
print(os.date(""))
```

   Patch:

```
loslib.c:
@@ -148,7 +148,18 @@
   else {
-    char b[256];
-    if (strftime(b, sizeof(b), s, stm))
-      lua_pushstring(L, b);
-    else
-      return luaL_error(L, LUA_QL("date") " format too long");
+    char cc[3];
```

```
+    luaL_Buffer b;
+    cc[0] = '%'; cc[2] = '\0';
+    luaL_buffinit(L, &b);
+    for (; *s; s++) {
+      if (*s != '%' || *(s + 1) == '\0')  /* no conversion specifier? */
+        luaL_addchar(&b, *s);
+      else {
+        size_t reslen;
+        char buff[200];  /* should be big enough for any conversion result */
+        cc[1] = *(++s);
+        reslen = strftime(buff, sizeof(buff), cc, stm);
+        luaL_addlstring(&b, buff, reslen);
+      }
+    }
+    luaL_pushresult(&b);
   }
```

5. `setfenv` accepts invalid first argument.
   reported by Doug Rogers in 8 Feb 2007. existed since 5.0. fixed in 5.1.2.

   Example:

```
setfenv(nil, {})   -- should throw an error
```

   Patch:

```
lbaselib.c:
@@ -116,3 +116,3 @@

-static void getfunc (lua_State *L) {
+static void getfunc (lua_State *L, int opt) {
   if (lua_isfunction(L, 1)) lua_pushvalue(L, 1);
@@ -120,3 +120,3 @@
     lua_Debug ar;
-    int level = luaL_optint(L, 1, 1);
+    int level = opt ? luaL_optint(L, 1, 1) : luaL_checkint(L, 1);
     luaL_argcheck(L, level >= 0, 1, "level must be non-negative");
@@ -133,3 +133,3 @@
 static int luaB_getfenv (lua_State *L) {
-  getfunc(L);
+  getfunc(L, 1);
   if (lua_iscfunction(L, -1))  /* is a C function? */
@@ -144,3 +144,3 @@
   luaL_checktype(L, 2, LUA_TTABLE);
-  getfunc(L);
+  getfunc(L, 0);
   lua_pushvalue(L, 2);
```

6. Wrong code generated for arithmetic expressions in some specific scenarios.
   reported by Thierry Grellier on 19 Jan 2007. existed since 5.1. fixed in 5.1.2.

   Example:

```
-- use a large number of names (almost 256)
v1=1; v2=1; v3=1; v4=1; v5=1; v6=1; v7=1; v8=1; v9=1;
v10=1; v11=1; v12=1; v13=1; v14=1; v15=1; v16=1; v17=1;
v18=1; v19=1; v20=1; v21=1; v22=1; v23=1; v24=1; v25=1;
v26=1; v27=1; v28=1; v29=1; v30=1; v31=1; v32=1; v33=1;
v34=1; v35=1; v36=1; v37=1; v38=1; v39=1; v40=1; v41=1;
v42=1; v43=1; v44=1; v45=1; v46=1; v47=1; v48=1; v49=1;
v50=1; v51=1; v52=1; v53=1; v54=1; v55=1; v56=1; v57=1;
v58=1; v59=1; v60=1; v61=1; v62=1; v63=1; v64=1; v65=1;
v66=1; v67=1; v68=1; v69=1; v70=1; v71=1; v72=1; v73=1;
v74=1; v75=1; v76=1; v77=1; v78=1; v79=1; v80=1; v81=1;
v82=1; v83=1; v84=1; v85=1; v86=1; v87=1; v88=1; v89=1;
v90=1; v91=1; v92=1; v93=1; v94=1; v95=1; v96=1; v97=1;
v98=1; v99=1; v100=1; v101=1; v102=1; v103=1; v104=1; v105=1;
v106=1; v107=1; v108=1; v109=1; v110=1; v111=1; v112=1; v113=1;
v114=1; v115=1; v116=1; v117=1; v118=1; v119=1; v120=1; v121=1;
v122=1; v123=1; v124=1; v125=1; v126=1; v127=1; v128=1; v129=1;
v130=1; v131=1; v132=1; v133=1; v134=1; v135=1; v136=1; v137=1;
v138=1; v139=1; v140=1; v141=1; v142=1; v143=1; v144=1; v145=1;
v146=1; v147=1; v148=1; v149=1; v150=1; v151=1; v152=1; v153=1;
v154=1; v155=1; v156=1; v157=1; v158=1; v159=1; v160=1; v161=1;
v162=1; v163=1; v164=1; v165=1; v166=1; v167=1; v168=1; v169=1;
v170=1; v171=1; v172=1; v173=1; v174=1; v175=1; v176=1; v177=1;
v178=1; v179=1; v180=1; v181=1; v182=1; v183=1; v184=1; v185=1;
v186=1; v187=1; v188=1; v189=1; v190=1; v191=1; v192=1; v193=1;
v194=1; v195=1; v196=1; v197=1; v198=1; v199=1; v200=1; v201=1;
v202=1; v203=1; v204=1; v205=1; v206=1; v207=1; v208=1; v209=1;
```

```
v210=1; v211=1; v212=1; v213=1; v214=1; v215=1; v216=1; v217=1;
v218=1; v219=1; v220=1; v221=1; v222=1; v223=1; v224=1; v225=1;
v226=1; v227=1; v228=1; v229=1; v230=1; v231=1; v232=1; v233=1;
v234=1; v235=1; v236=1; v237=1; v238=1; v239=1; v240=1; v241=1;
v242=1; v243=1; v244=1; v245=1; v246=1; v247=1; v248=1; v249=1;
v250=1;
v251={k1 = 1};
v252=1;
print(2 * v251.k1, v251.k1 * 2);   -- 2 2, OK
v253=1;
print(2 * v251.k1, v251.k1 * 2);   -- 1 2, ???
```

Patch:

```
lcode.c:
@@ -657,10 +657,16 @@
   if (constfolding(op, e1, e2))
     return;
   else {
-    int o1 = luaK_exp2RK(fs, e1);
     int o2 = (op != OP_UNM && op != OP_LEN) ? luaK_exp2RK(fs, e2) : 0;
-    freeexp(fs, e2);
-    freeexp(fs, e1);
+    int o1 = luaK_exp2RK(fs, e1);
+    if (o1 > o2) {
+      freeexp(fs, e1);
+      freeexp(fs, e2);
+    }
+    else {
+      freeexp(fs, e2);
+      freeexp(fs, e1);
+    }
     e1->u.s.info = luaK_codeABC(fs, op, 0, o1, o2);
     e1->k = VRELOCABLE;
   }
@@ -718,10 +724,15 @@
       luaK_exp2nextreg(fs, v);  /* operand must be on the `stack' */
       break;
     }
-    default: {
+    case OPR_ADD: case OPR_SUB: case OPR_MUL: case OPR_DIV:
+    case OPR_MOD: case OPR_POW: {
       if (!isnumeral(v)) luaK_exp2RK(fs, v);
       break;
     }
+    default: {
+      luaK_exp2RK(fs, v);
+      break;
+    }
   }
 }
```

7. Assignment of nil to parameter may be optimized away.
   reported by Thomas Lauer on 21 Mar 2007. existed since 5.1. fixed in 5.1.2.

Example:

```
function f (a)
  a=nil
  return a
end

print(f("test"))
```

Patch:

```
lcode.c:
@@ -35,16 +35,20 @@
 void luaK_nil (FuncState *fs, int from, int n) {
   Instruction *previous;
   if (fs->pc > fs->lasttarget) {  /* no jumps to current position? */
-    if (fs->pc == 0)  /* function start? */
-      return;  /* positions are already clean */
-    previous = &fs->f->code[fs->pc-1];
-    if (GET_OPCODE(*previous) == OP_LOADNIL) {
-      int pfrom = GETARG_A(*previous);
-      int pto = GETARG_B(*previous);
-      if (pfrom <= from && from <= pto+1) {  /* can connect both? */
-        if (from+n-1 > pto)
-          SETARG_B(*previous, from+n-1);
```

```
-        return;
+    if (fs->pc == 0) {  /* function start? */
+      if (from >= fs->nactvar)
+        return;  /* positions are already clean */
+    }
+    else {
+      previous = &fs->f->code[fs->pc-1];
+      if (GET_OPCODE(*previous) == OP_LOADNIL) {
+        int pfrom = GETARG_A(*previous);
+        int pto = GETARG_B(*previous);
+        if (pfrom <= from && from <= pto+1) {  /* can connect both? */
+          if (from+n-1 > pto)
+            SETARG_B(*previous, from+n-1);
+          return;
+        }
       }
     }
   }
```

8. Concat metamethod converts numbers to strings.
   reported by Paul Winwood on 24 Dec 2006. existed since 5.0. fixed in 5.1.2.

   Example:

   ```
   a = {}
   setmetatable(a, {__concat = function (a,b) print(type(a), type(b)) end})
   a = 4 .. a
   ```

   Patch:

   ```
   lvm.c:
   @@ -281,10 +281,12 @@
      do {
        StkId top = L->base + last + 1;
        int n = 2;  /* number of elements handled in this pass (at least 2) */
   -    if (!tostring(L, top-2) || !tostring(L, top-1)) {
   +    if (!(ttisstring(top-2) || ttisnumber(top-2)) || !tostring(L, top-1)) {
          if (!call_binTM(L, top-2, top-1, top-2, TM_CONCAT))
            luaG_concaterror(L, top-2, top-1);
   -    } else if (tsvalue(top-1)->len > 0) {  /* if len=0, do nothing */
   +    } else if (tsvalue(top-1)->len == 0)  /* second op is empty? */
   +      (void)tostring(L, top - 2);  /* result is first op (as string) */
   +    else {
          /* at least two string values; get as many as possible */
          size_t tl = tsvalue(top-1)->len;
          char *buffer;
   ```

9. `loadlib.c` is a library and should not access Lua internals (via `lobject.h`).
   reported by Jérôme Vuarand on 25 Mar 2007. existed since 5.0 at least. fixed in 5.1.2.

   Example: The bug has no effect on external behavior.

   Patch: In loadlib.c, change all ocurrences of `luaO_pushfstring` to `lua_pushfstring`.

❖ Lua 5.1 1 · 2 · 3 · 4 · 5 · 6 · 7 · 8 · 9 · 10

1. In 16-bit machines, and/or expressions with numeric constants as the right operand may result in weird values.
   reported by Andreas Stenius on 15 Mar 2006. fixed in 5.1.1.

   Example:

   ```
   print(false or 0)   -- on 16-bit machines
   ```

   Patch:

   ```
   lcode.c:
   @@ -731,17 +731,15 @@
        case OPR_AND: {
          lua_assert(e1->t == NO_JUMP);  /* list must be closed */
          luaK_dischargevars(fs, e2);
   -      luaK_concat(fs, &e1->f, e2->f);
   -      e1->k = e2->k; e1->u.s.info = e2->u.s.info;
   -      e1->u.s.aux = e2->u.s.aux; e1->t = e2->t;
   +      luaK_concat(fs, &e2->f, e1->f);
   +      *e1 = *e2;
          break;
        }
   ```

```
     case OPR_OR: {
       lua_assert(e1->f == NO_JUMP);  /* list must be closed */
       luaK_dischargevars(fs, e2);
-      luaK_concat(fs, &e1->t, e2->t);
-      e1->k = e2->k; e1->u.s.info = e2->u.s.info;
-      e1->u.s.aux = e2->u.s.aux; e1->f = e2->f;
+      luaK_concat(fs, &e2->t, e1->t);
+      *e1 = *e2;
       break;
     }
```

2. `luaL_checkudata` may produce wrong error message.
   reported by Greg Falcon on 21 Mar 2006. fixed in 5.1.1.

   Example:

```
getmetatable(io.stdin).__gc()
--> bad argument #1 to '__gc' (FILE* expected, got table)
```

   Patch:

```
lauxlib.c:
@@ -123,11 +123,17 @@

 LUALIB_API void *luaL_checkudata (lua_State *L, int ud, const char *tname) {
   void *p = lua_touserdata(L, ud);
-  lua_getfield(L, LUA_REGISTRYINDEX, tname);  /* get correct metatable */
-  if (p == NULL || !lua_getmetatable(L, ud) || !lua_rawequal(L, -1, -2))
-    luaL_typerror(L, ud, tname);
-  lua_pop(L, 2);  /* remove both metatables */
-  return p;
+  if (p != NULL) {  /* value is a userdata? */
+    if (lua_getmetatable(L, ud)) {  /* does it have a metatable? */
+      lua_getfield(L, LUA_REGISTRYINDEX, tname);  /* get correct metatable */
+      if (lua_rawequal(L, -1, -2)) {  /* does it have the correct mt? */
+        lua_pop(L, 2);  /* remove both metatables */
+        return p;
+      }
+    }
+  }
+  luaL_typerror(L, ud, tname);  /* else error */
+  return NULL;  /* to avoid warnings */
 }
```

3. Windows applications that use both Lua and DirectX may present erractic behavior. *THIS IS NOT A BUG IN Lua!* The problem is that DirectX violates an ABI that Lua depends on.

   Patch: The simplest solution is to use DirectX with the `D3DCREATE_FPU_PRESERVE` flag. Otherwise, you can change the definition of `lua_number2int` in luaconf.h to this one:

```
#define lua_number2int(i,d)   __asm fld d   __asm fistp i
```

4. Option '%q' in `string.formatE` does not handle '\r' correctly.
   reported by FleetCommand on 1 Apr 2006. fixed in 5.1.1.

   Example:

```
local s = "a string with \r and \n and \r\n and \n\r"
local c = string.format("return %q", s)
assert(assert(loadstring(c))() == s)
```

   Patch:

```
lstrlib.c:
@@ -703,6 +703,10 @@
         luaL_addchar(b, *s);
         break;
       }
+      case '\r': {
+        luaL_addlstring(b, "\\r", 2);
+        break;
+      }
       case '\0': {
         luaL_addlstring(b, "\\000", 4);
         break;
```

5. `luaL_dofile` and `luaL_dostring` should return all values returned by the chunk.
   reported by mos on 11 Apr 2006. fixed in 5.1.1.

   Patch:

```
lauxlib.h:
@@ -108,9 +108,11 @@

 #define luaL_typename(L,i)     lua_typename(L, lua_type(L,(i)))

-#define luaL_dofile(L, fn)     (luaL_loadfile(L, fn) || lua_pcall(L, 0, 0, 0))
+#define luaL_dofile(L, fn) \
+       (luaL_loadfile(L, fn) || lua_pcall(L, 0, LUA_MULTRET, 0))

-#define luaL_dostring(L, s)    (luaL_loadstring(L, s) || lua_pcall(L, 0, 0, 0))
+#define luaL_dostring(L, s) \
+       (luaL_loadstring(L, s) || lua_pcall(L, 0, LUA_MULTRET, 0))

 #define luaL_getmetatable(L,n) (lua_getfield(L, LUA_REGISTRYINDEX, (n)))
```

6. Garbage collector does not compensate enough for finalizers.
   reported by Roberto in May 2006. fixed in 5.1.1.

   Patch:

```
lgc.c:
@@ -322,4 +322,6 @@

-static void propagateall (global_State *g) {
-  while (g->gray) propagatemark(g);
+static size_t propagateall (global_State *g) {
+  size_t m = 0;
+  while (g->gray) m += propagatemark(g);
+  return m;
 }
@@ -542,3 +544,3 @@
   marktmu(g);  /* mark `preserved' userdata */
-  propagateall(g);  /* remark, to propagate `preserveness' */
+  udsize += propagateall(g);  /* remark, to propagate `preserveness' */
   cleartable(g->weak);  /* remove collected objects from weak tables */
@@ -592,2 +594,4 @@
         GCTM(L);
+        if (g->estimate > GCFINALIZECOST)
+          g->estimate -= GCFINALIZECOST;
```

7. Debug hooks may get wrong when mixed with coroutines.
   reported by Ivko Stanilov on 3 Jun 2006. fixed in 5.1.1.

   Example:

```
co = coroutine.create(function () coroutine.yield() end)
debug.sethook(co, function() end, "lr")
coroutine.resume(co)
coroutine.resume(co)
```

   Patch:

```
ldo.c:
@@ -389,6 +389,7 @@
       return;
   }
   else {  /* resuming from previous yield */
+    L->status = 0;
     if (!f_isLua(ci)) {  /* `common' yield? */
       /* finish interrupted execution of `OP_CALL' */
       lua_assert(GET_OPCODE(*((ci-1)->savedpc - 1)) == OP_CALL ||
@@ -399,7 +400,6 @@
     else  /* yielded inside a hook: just continue its execution */
       L->base = L->ci->base;
   }
-  L->status = 0;
   luaV_execute(L, cast_int(L->ci - L->base_ci));
 }
```

8. List constructors have wrong limit.

9. Wrong message error in some cases involving closures.

10. Wrong code generated for arithmetic expressions in some specific scenarios.

❖ Lua 5.0.3

That was the last release of Lua 5.0. Bugs reported later are probably fixed in Lua 5.1.

❖ Lua 5.0.2 1 · 2 · 3 · 4 · 5 · 6 · 7 · 8

1. String concatenation may cause arithmetic overflow, leading to a buffer overflow.
   reported by Rici Lake on 20 May 2004. fixed in 5.1 and 5.0.3.

   Example:

```
longs = string.rep("\0", 2^25)
function catter(i)
    return assert(loadstring(
      string.format("return function(a) return a%s end",
                    string.rep("..a", i-1))))()
end
rep129 = catter(129)
rep129(longs)
```

   Patch:

```
lvm.c:
@@ -321,15 +321,15 @@
         luaG_concaterror(L, top-2, top-1);
      } else if (tsvalue(top-1)->tsv.len > 0) {  /* if len=0, do nothing */
        /* at least two string values; get as many as possible */
-       lu_mem tl = cast(lu_mem, tsvalue(top-1)->tsv.len) +
-                   cast(lu_mem, tsvalue(top-2)->tsv.len);
+       size_t tl = tsvalue(top-1)->tsv.len;
        char *buffer;
        int i;
-       while (n < total && tostring(L, top-n-1)) {  /* collect total length */
-         tl += tsvalue(top-n-1)->tsv.len;
-         n++;
+       /* collect total length */
+       for (n = 1; n < total && tostring(L, top-n-1); n++) {
+         size_t l = tsvalue(top-n-1)->tsv.len;
+         if (l >= MAX_SIZET - tl) luaG_runerror(L, "string length overflow");
+         tl += l;
        }
-       if (tl > MAX_SIZET) luaG_runerror(L, "string size overflow");
        buffer = luaZ_openspace(L, &G(L)->buff, tl);
        tl = 0;
        for (i=n; i>0; i--) {  /* concat all strings */
```

2. `lua_getupvalue` and `lua_setupvalue` do not check for index too small.
   reported by Mike Pall on 6 Jun 2004. fixed in 5.1 and 5.0.3.

   Example:

```
debug.getupvalue(function() end, 0)
```

   Patch:

```
lapi.c
941c941
<     if (n > f->c.nupvalues) return NULL;
---
>     if (!(1 <= n && n <= f->c.nupvalues)) return NULL;
947c947
<     if (n > p->sizeupvalues) return NULL;
---
>     if (!(1 <= n && n <= p->sizeupvalues)) return NULL;
```

3. Values held in open upvalues of suspended threads may be incorrectly collected.
   reported by Spencer Schumann on 31 Dec 2004. fixed in 5.1 and 5.0.3.

   Example:

```
local thread_id = 0
local threads = {}

function fn(thread)
    thread_id = thread_id + 1
    threads[thread_id] = function()
                             thread = nil
```

```
                        end
    coroutine.yield()
end

while true do
    local thread = coroutine.create(fn)
    coroutine.resume(thread, thread)
end
```

Patch:

```
lgc.c:
221,224c221,222
<       if (!u->marked) {
<         markobject(st, &u->value);
<         u->marked = 1;
<       }
---
>       markobject(st, u->v);
>       u->marked = 1;
```

4. `rawset` and `rawget` do not ignore extra arguments.
   reported by Romulo Bahiense on 11 Mar 2005. fixed in 5.1 and 5.0.3.

   Example:

```
a = {}
rawset(a, 1, 2, 3)
print(a[1], a[2])    -- should be 2 and nil
```

   Patch:

```
lbaselib.c:
175a176
>   lua_settop(L, 2);
183a185
>   lua_settop(L, 3);
```

5. Weak tables that survive one collection are never collected.
   reported by Chromix on 2 Jan 2006. fixed in 5.1 and 5.0.3.

   Example:

```
a = {}
print(gcinfo())
for i = 1, 10000 do
  a[i] = setmetatable({}, {__mode = "v"})
end
collectgarbage()
a = nil
collectgarbage()
print(gcinfo())
```

   Patch:

```
lgc.c
@@ -366,7 +366,7 @@
   GCObject *curr;
   int count = 0;  /* number of collected items */
   while ((curr = *p) != NULL) {
-    if (curr->gch.marked > limit) {
+    if ((curr->gch.marked & ~(KEYWEAK | VALUEWEAK)) > limit) {
       unmark(curr);
       p = &curr->gch.next;
     }
```

6. Garbage collector does not compensate enough for finalizers.

7. Some "not not" expressions may not result in boolean values.
   reported by Aaron Brown on 30 Jun 2005. existed since 4.0. fixed in 5.0.3.

   Example:

```
-- should print false, but prints nil
print(not not (nil and 4))
```

8. On some machines, closing a "piped file" (created with `io.popen`) may crash Lua.
   reported by Mike Pall on 23 May 2005. existed since 5.0. fixed in 5.0.3.

   Example:

```
f = io.popen("ls")
f:close()
```

❖ Lua 5.0 1 · 2 · 3 · 4 · 5 · 6 · 7 · 8 · 9 · 10 · 11 · 12 · 13 · 14 · 15 · 16 · 17

1. `lua_closethread` exists only in the manual.
   reported by Nguyen Binh on 28 Apr 2003. fixed in 5.0.2.

   Solution: The manual is wrong: threads are subject to garbage collection.

2. Attempt to resume a running coroutine crashes Lua.
   reported by Alex Bilyk on 9 May 2003. fixed in 5.0.2.

   Example:

```
function co_func (current_co)
   coroutine.resume(co)
end
co = coroutine.create(co_func)
coroutine.resume(co)
coroutine.resume(co)     --> crash
```

   Patch:

```
ldo.c:
325,326c325
<     if (nargs >= L->top - L->base)
<       luaG_runerror(L, "cannot resume dead coroutine");
---
>     lua_assert(nargs < L->top - L->base);
329c328,329
<   else if (ci->state & CI_YIELD) {  /* inside a yield? */
---
>   else {  /* inside a yield */
>     lua_assert(ci->state & CI_YIELD);
344,345d343
<   else
<     luaG_runerror(L, "cannot resume non-suspended coroutine");
351a350,358
> static int resume_error (lua_State *L, const char *msg) {
>   L->top = L->ci->base;
>   setsvalue2s(L->top, luaS_new(L, msg));
>   incr_top(L);
>   lua_unlock(L);
>   return LUA_ERRRUN;
> }
>
>
355a363,368
>   if (L->ci == L->base_ci) {
>     if (nargs >= L->top - L->base)
>       return resume_error(L, "cannot resume dead coroutine");
>   }
>   else if (!(L->ci->state & CI_YIELD))  /* not inside a yield? */
>     return resume_error(L, "cannot resume non-suspended coroutine");
```

3. `file:close` cannot be called without a file (results a crash).
   reported by Tuomo Valkonen on 27 May 2003. fixed in 5.0.2.

   Example:

```
> io.stdin.close()    -- correct call should be io.stdin:close()
```

   Patch:

```
liolib.c:
161c161
<   if (lua_isnone(L, 1)) {
---
>   if (lua_isnone(L, 1) && lua_type(L, lua_upvalueindex(1)) == LUA_TTABLE) {
```

4. C functions may have stacks larger than current top.
   reported by Alex Bilyk on 9 Jun 2003. fixed in 5.0.2.

   Example: Must recompile Lua with a change in `lua.c` and with `lua_assert` defined:

```
lua.c:
381a382
>   lua_checkstack(l, 1000);
```

   Patch:

```
lgc.c:
247c247
<     if (!(ci->state & CI_C) && lim < ci->top)
---
>     if (lim < ci->top)
```

5. 'pc' address is invalidated when a coroutine is suspended.
   reported by Nick Trout on 7 Jul 2003. fixed in 5.0.2.

   Example:

```
function g(x)
    coroutine.yield(x)
end

function f (i)
  debug.sethook(print, "l")
  for j=1,1000 do
    g(i+j)
  end
end

co = coroutine.wrap(f)
co(10)
pcall(co)
pcall(co)
```

   Patch:

```
lvm.c:
402d401
<   L->ci->u.l.pc = &pc;
405a405
>   L->ci->u.l.pc = &pc;
676,678c676
<           lua_assert(ci->u.l.pc == &pc &&
<                      ttisfunction(ci->base - 1) &&
<                      (ci->state & CI_SAVEDPC));
---
>           lua_assert(ttisfunction(ci->base - 1) && (ci->state & CI_SAVEDPC));
```

6. Userdata to be collected still counts into new GC threshold, increasing memory consumption.
   reported by Roberto on 25 Jul 2003. fixed in 5.0.2.

   Example:

```
a = newproxy(true)
getmetatable(a).__gc = function () end
for i=1,10000000 do
  newproxy(a)
  if math.mod(i, 10000) == 0 then print(gcinfo()) end
end
```

   Patch:

```
lgc.h:
18c18
< void luaC_separateudata (lua_State *L);
---
> size_t luaC_separateudata (lua_State *L);

lgc.c:
113c113,114
< void luaC_separateudata (lua_State *L) {
---
> size_t luaC_separateudata (lua_State *L) {
>   size_t deadmem = 0;
127a129
```

```
>       deadmem += sizeudata(gcotou(curr)->uv.len);
136a139
>   return deadmem;
390c393
< static void checkSizes (lua_State *L) {
---
> static void checkSizes (lua_State *L, size_t deadmem) {
400c403
<   G(L)->GCthreshold = 2*G(L)->nblocks;  /* new threshold */
---
>   G(L)->GCthreshold = 2*G(L)->nblocks - deadmem;  /* new threshold */
454c457,458
< static void mark (lua_State *L) {
---
> static size_t mark (lua_State *L) {
>   size_t deadmem;
467c471
<   luaC_separateudata(L);  /* separate userdata to be preserved */
---
>   deadmem = luaC_separateudata(L);  /* separate userdata to be preserved */
475a480
>   return deadmem;
480c485
<   mark(L);
---
>   size_t deadmem = mark(L);
482c487
<   checkSizes(L);
---
>   checkSizes(L, deadmem);
```

7. IBM AS400 (OS400) has sizeof(void *)==16, and a '%p' may generate up to 60 characters in a 'printf', causing a buffer overflow in `tostring`.
   reported by David Burgess on 25 Aug 2003. fixed in 5.0.2.

   Example:

   ```
   print{}  -- on an AS400 machine
   ```

   Patch:

   ```
   liolib.c:
   178c178
   <   char buff[32];
   ---
   >   char buff[128];

   lbaselib.c:
   327c327
   <   char buff[64];
   ---
   >   char buff[128];
   ```

8. Syntax `local function` does not increment stack size.
   reported by Rici Lake on 26 Sep 2003. fixed in 5.0.2.

   Example:

   ```
   -- must run this with precompiled code
   local a,b,c
   local function d () end
   ```

   Patch:

   ```
   lparser.c:
   1143a1144
   >   FuncState *fs = ls->fs;
   1145c1146,1147
   <   init_exp(&v, VLOCAL, ls->fs->freereg++);
   ---
   >   init_exp(&v, VLOCAL, fs->freereg);
   >   luaK_reserveregs(fs, 1);
   1148c1150,1152
   <   luaK_storevar(ls->fs, &v, &b);
   ---
   >   luaK_storevar(fs, &v, &b);
   >   /* debug information will only see the variable after this point! */
   >   getlocvar(fs, fs->nactvar - 1).startpc = fs->pc;
   ```

9. Count hook may be called without being set.
   reported by Andreas Stenius on 6 Oct 2003. fixed in 5.0.2.

   Example:
   Set your hooks as below. It is weird to use a positive count without setting the count hook, but it is not wrong.

   ```
   lua_sethook(L, my_hook, LUA_MASKLINE | LUA_MASKRET, 1);
   ```

   Patch:

   ```
   lvm.c:
   69c69
   <   if (mask > LUA_MASKLINE) {  /* instruction-hook set? */
   ---
   >   if (mask & LUA_MASKCOUNT) {  /* instruction-hook set? */
   ```

10. `dofile` eats one return value when called without arguments.
    reported by Frederico Abraham on 15 Jan 2004. fixed in 5.0.2.

    Example:

    ```
    a,b = dofile()   --< here you enter `return 1,2,3 <eof>'
    print(a,b)   --> 2   3   (should be 1 and 2)
    ```

    Patch:

    ```
    lbaselib.c:
    313a314
    >   int n = lua_gettop(L);
    317c318
    <   return lua_gettop(L) - 1;
    ---
    >   return lua_gettop(L) - n;
    ```

11. String concatenation may cause arithmetic overflow, leading to a buffer overflow.
12. `lua_getupvalue` and `lua_setupvalue` do not check for index too small.
13. `rawset` and `rawget` do not ignore extra arguments.
14. Weak tables that survive one collection are never collected.
15. Garbage collector does not compensate enough for finalizers.
16. Some "not not" expressions may not result in boolean values.
17. On some machines, closing a "piped file" (created with io.popen) may crash Lua.

❖ Lua 4.0

Most bugs in Lua 4.0 have been fixed in Lua 4.0.1 and later versions.

1. Parser did not accept a ';' after a 'return'.
   reported by lhf on 29 Nov 2000. fixed in 4.0.1.
2. When 'read' fails it must return nil (and not no value).
   reported by Carlos Cassino on 22 Dec 2000. fixed in 5.0.
3. `lua_pushuserdata(L,NULL)` does not work.
   reported by Edgar Toernig on 1 Feb 2001. fixed in 4.0.1.
4. `while 1 dostring[[print('hello\n')]] end` never reclaims memory.
   reported by Andrew Paton on 2 Feb 2001. fixed in 4.0.1.
5. ESC (which starts precompiled code) in C is \33, not \27.
   reported by Edgar Toernig on 6 Feb 2001. fixed in 4.0.1.
6. Error message for '%a' gave wrong line number.
   reported by Leonardo Constantino on 10 Jul 2001. fixed in 4.0.1.
7. Crash when rawget/rawset get extra arguments.
   reported by Eric Mauger on 21 Dec 2001. fixed in 4.0.1.

8. Line hook gets wrong 'ar'.
   reported by Daniel Sinclair on 19 Jun 2002. fixed in 4.0.1.

9. 'protectedparser' may run GC and then collect 'filename' (in 'parse_file').
   reported by Alex Bilyk on 19 Jun 2002. fixed in 4.0.1.

10. `ULONG_MAX>>10` may not fit into an `int`.
    reported by Jeff Petkau on 21 Nov 2002. fixed in 5.0.

last update: thu aug 25 10:50:28 utc 2022