2021L3HCTF luuuuua Writeup

原创

1mmorta1 于 2021-11-15 16:20:15 发布 5228 收藏

分类专栏： reverse 文章标签： lua app安全

本文链接：https://blog.csdn.net/qq_41866334/article/details/121337114

2021 L3HCTF luuuua Writeup

> AAA：immortal

这题做完以后感觉和ByteCTF2021的language binding很像，这是我之前写的Writeup 。

首先打开java层，发现asserts/res/test.lua里的逻辑是假的。然后通过LoginActivity里的afterTextChanged方法找到关键的b.c.a.b.a.d。 注意它

```
import org.keplerproject.luajava.LuaState;

import org.keplerproject.luajava.LuaStateFactory;
```

所以在网上找到了对应的github:luajava，和一些使用例子:

```
public void invokeFileScript(String scriptFilePath){
 LuaState luaState = LuaStateFactory.newLuaState();
 luaState.openLibs();
 this.initLuaContext(luaState);
 int error = luaState.LdoFile(scriptFilePath);
 if(error!=0){
  Logger.log("Read/Parse lua error. Exit");
  return;
 }
 luaState.close();
}
```

所以我们发现真正的逻辑在logo.jpg上，010editor打开找了一下发现真的有加密了的luac脚本。异或0x3c解密以后得到了luac脚本，unluac尝试反汇编发现报错。和byteCTF那道一样改了opcode的位置，因此需要逆向.so文件找到真正的opcode解析的方法。

```
with open('logo.jpg','rb') as f:
    con = f.read()
con = con[0x3afa2:]
with open('dec.luac','wb') as fs:
    fs.write(b'\x1b')
    for c in con:
        fs.write(bytes([0x3c^c]))
```

使用Android里的ndk工具链重新编译了lua5.3的源码，得到了正确的.so文件。两者对照以后慢慢逆向恢复出正确的opcodes。

然后和ByteCTF一样，魔改一下unluac脚本，反编译出了lua脚本：

```
local base64 = {}
local extract = _G.bit32 and _G.bit32.extract
if not extract then
  if _G.bit then
    local shl, shr, band = _G.bit.lshift, _G.bit.rshift, _G.bit.band
    function extract(v, from, width)
      return band(shr(v, from), shl(1, width) - 1)
    end
  elseif _G._VERSION == "Lua 5.1" then
    function extract(v, from, width)
      local w = 0
      local flag = 2 ^ from
      for i = 0, width - 1 do
        local flag2 = flag + flag
        if flag <= v % flag2 then
          w = w + 2 ^ i
        end
        flag = flag2
      end
      return w
    end
  else
    extract = load([[
return function( v, from, width )
   return ( v >> from ) & ((1 << width) - 1)
  end]])()
  end
end
function base64.makeencoder(s62, s63, spad)
  local encoder = {}
  for b64code, char in pairs({
    [0] = "A",
    "B",
    "C",
    "D",
    "E",
    "F",
    "G",
    "H",
    "I",
    "J",
    "K",
    "L",
    "M",
    "N",
    "O",
    "P",
    "Q",
    "R",
    "S",
    "T",
    "U",
    "V",
    "W",
    "X",
    "Y",
```

```
    "Z",
    "a",
    "b",
    "c",
    "d",
    "e",
    "f",
    "g",
    "h",
    "i",
    "j",
    "k",
    "l",
    "m",
    "n",
    "o",
    "p",
    "q",
    "r",
    "s",
    "t",
    "u",
    "v",
    "w",
    "x",
    "y",
    "z",
    "0",
    "1",
    "2",
    "3",
    "4",
    "5",
    "6",
    "7",
    "8",
    "9",
    s62 or "+",
    s63 or "/",
    spad or "="
  }) do
    encoder[b64code] = char:byte()
  end
  return encoder
end
function base64.makedecoder(s62, s63, spad)
  local decoder = {}
  for b64code, charcode in pairs(base64.makeencoder(s62, s63, spad)) do
    decoder[charcode] = b64code
  end
  return decoder
end
local DEFAULT_ENCODER = base64.makeencoder()
local DEFAULT_DECODER = base64.makedecoder()
local char, concat = string.char, table.concat
function base64.encode(str, encoder, usecaching)
  encoder = encoder or DEFAULT_ENCODER
  local t, k, n = {}, 1, #str
  local lastn = n % 3
```

```
  local cache = {}
  for i = 1, n - lastn, 3 do
    local a, b, c = str:byte(i, i + 2)
    local v = a * 65536 + b * 256 + c
    local s
    if usecaching then
      s = cache[v]
      if not s then
        s = char(encoder[extract(v, 18, 6)], encoder[extract(v, 12, 6)], encoder[extract(v, 6, 6)], encoder[extr
act(v, 0, 6)])
        cache[v] = s
      end
    else
      s = char(encoder[extract(v, 18, 6)], encoder[extract(v, 12, 6)], encoder[extract(v, 6, 6)], encoder[extrac
t(v, 0, 6)])
    end
    t[k] = s
    k = k + 1
  end
  if lastn == 2 then
    local a, b = str:byte(n - 1, n)
    local v = a * 65536 + b * 256
    t[k] = char(encoder[extract(v, 18, 6)], encoder[extract(v, 12, 6)], encoder[extract(v, 6, 6)], encoder[64])
  elseif lastn == 1 then
    local v = str:byte(n) * 65536
    t[k] = char(encoder[extract(v, 18, 6)], encoder[extract(v, 12, 6)], encoder[64], encoder[64])
  end
  return concat(t)
end
function base64.decode(b64, decoder, usecaching)
  decoder = decoder or DEFAULT_DECODER
  local pattern = "[^%w%+%/%=]"
  if decoder then
    local s62, s63
    for charcode, b64code in pairs(decoder) do
      if b64code == 62 then
        s62 = charcode
      elseif b64code == 63 then
        s63 = charcode
      end
    end
    pattern = ("[^%%w%%%s%%%s%%=]"):format(char(s62), char(s63))
  end
  b64 = b64:gsub(pattern, "")
  local cache = usecaching and {}
  local t, k = {}, 1
  local n = #b64
  local padding = b64:sub(-2) == "==" and 2 or b64:sub(-1) == "=" and 1 or 0
  for i = 1, padding > 0 and n - 4 or n, 4 do
    local a, b, c, d = b64:byte(i, i + 3)
    local s
    if usecaching then
      local v0 = a * 16777216 + b * 65536 + c * 256 + d
      s = cache[v0]
      if not s then
        local v = decoder[a] * 262144 + decoder[b] * 4096 + decoder[c] * 64 + decoder[d]
        s = char(extract(v, 16, 8), extract(v, 8, 8), extract(v, 0, 8))
        cache[v0] = s
      end
    else
```

```
    else
      local v = decoder[a] * 262144 + decoder[b] * 4096 + decoder[c] * 64 + decoder[d]
      s = char(extract(v, 16, 8), extract(v, 8, 8), extract(v, 0, 8))
    end
    t[k] = s
    k = k + 1
  end
  if padding == 1 then
    local a, b, c = b64:byte(n - 3, n - 1)
    local v = decoder[a] * 262144 + decoder[b] * 4096 + decoder[c] * 64
    t[k] = char(extract(v, 16, 8), extract(v, 8, 8))
  elseif padding == 2 then
    local a, b = b64:byte(n - 3, n - 2)
    local v = decoder[a] * 262144 + decoder[b] * 4096
    t[k] = char(extract(v, 16, 8))
  end
  return concat(t)
end
local strf = string.format
local byte, char = string.byte, string.char
local spack, sunpack = string.pack, string.unpack
local app, concat = table.insert, table.concat
local stohex = function(s, ln, sep)
  if #s == 0 then
    return ""
  end
  if not ln then
    return (s:gsub(".", function(c)
      return strf("%02x", byte(c))
    end))
  end
  sep = sep or ""
  local t = {}
  for i = 1, #s - 1 do
    t[#t + 1] = strf("%02x%s", s:byte(i), i % ln == 0 and "\n" or sep)
  end
  t[#t + 1] = strf("%02x", s:byte(#s))
  return concat(t)
end
local hextos = function(hs, unsafe)
  local tonumber = tonumber
  if not unsafe then
    hs = string.gsub(hs, "%s+", "")
    if string.find(hs, "[^0-9A-Za-z]") or #hs % 2 ~= 0 then
      error("invalid hex string")
    end
  end
  return hs:gsub("(%x%x)", function(c)
    return char(tonumber(c, 16))
  end)
end
local stx = stohex
local xts = hextos
local ROUNDS = 64
local keysetup = function(key)
  assert(#key == 16)
  local kt = {
    0,
    0,
    0,
```

```
    0
  }
  kt[1], kt[2], kt[3], kt[4] = sunpack(">I4I4I4I4", key)
  local skt0 = {}
  local skt1 = {}
  local sum, delta = 0, 2654435769
  for i = 1, ROUNDS do
    skt0[i] = sum + kt[(sum & 3) + 1]
    sum = sum + delta & 4294967295
    skt1[i] = sum + kt[(sum >> 11 & 3) + 1]
  end
  return {skt0 = skt0, skt1 = skt1}
end
local encrypt_u64 = function(st, bu)
  local skt0, skt1 = st.skt0, st.skt1
  local v0, v1 = bu >> 32, bu & 4294967295
  local sum, delta = 0, 2654435769
  for i = 1, ROUNDS do
    v0 = v0 + ((v1 << 4 ~ v1 >> 5) + v1 ~ skt0[i]) & 4294967295
    v1 = v1 + ((v0 << 4 ~ v0 >> 5) + v0 ~ skt1[i]) & 4294967295
  end
  bu = v0 << 32 | v1
  return bu
end
local enc = function(key, iv, itxt)
  assert(#key == 16, "bad key length")
  assert(#iv == 8, "bad IV length")
  if #itxt == 0 then
    return ""
  end
  local ivu = sunpack("<I8", iv)
  local ot = {}
  local rbn = #itxt
  local ksu, ibu, ob
  local st = keysetup(key)
  for i = 1, #itxt, 8 do
    ksu = encrypt_u64(st, ivu ~ i)
    if rbn < 8 then
      local buffer = string.sub(itxt, i) .. string.rep("\000", 8 - rbn)
      ibu = sunpack("<I8", buffer)
      ob = string.sub(spack("<I8", ibu ~ ksu), 1, rbn)
    else
      ibu = sunpack("<I8", itxt, i)
      ob = spack("<I8", ibu ~ ksu)
      rbn = rbn - 8
    end
    app(ot, ob)
  end
  return concat(ot)
end
function check_login(username, password)
  local encoded = base64.encode(username)
  if encoded ~= "TDNIX1NlYw==" then --L3H_Sec
    return false
  end
  username = username .. "!@#$%^&*("
  local x = base64.encode(enc(username, "1qazxsw2", password))
  if x == "LKq2dSc30DKJo99bsFgTkQM9dor1gLl2rejdnkw2MBpOud+38vFkCCF13qY=" then
    return true
  end
```

```
  end
  return false
end
```

仔细看了一下加密的原理，发现前面虽然很复杂是tea-cbc但其实最后还是异或，所以直接在解出的lua脚本里敲个print，dump出要异或的字节流就可以了。

```
from base64 import b64decode

cmp = b64decode('LKq2dSc30DKJo99bsFgTkQM9dor1gLl2rejdnkw2MBpOud+38vFkCCF13qY=')
print(len(cmp)) # 44
a = [
48256960675354976,
-4890521267755574343+2**64,
4792028846500240997,
9087511422306731418,
7661124995518270764,
-4844959183373908412+2**64
]
ba= list(b''.join(map(lambda x: x.to_bytes(8,"little"),a)))
for i in range(44):
    print(chr(cmp[i]^ba[i]),end='')
```