再谈Lua字节码攻击

董宇骏 Nu1L Team

1．背景介绍

Lua是一个轻量级、可扩展的脚本语言。

在软件开发领域，Lua常常作为嵌入式脚本语言引擎。

游戏开发：主流游戏引擎支持的脚本语言、游戏模组语言

嵌入式系统：网络设备web后端

应用扩展：定制程序功能，应用场景如Redis，OpenResty

2．威胁模型

Lua脚本强保护

执行环境弱保护、高权限

攻击面：应用程序对执行脚本的验证逻辑。Lua任意执行=完整RCE

例子：在服务端执行的Lua插件

Lua脚本弱保护

执行环境强保护、低权限

攻击面：Lua引擎。

Lua任意执行≠完整RCE

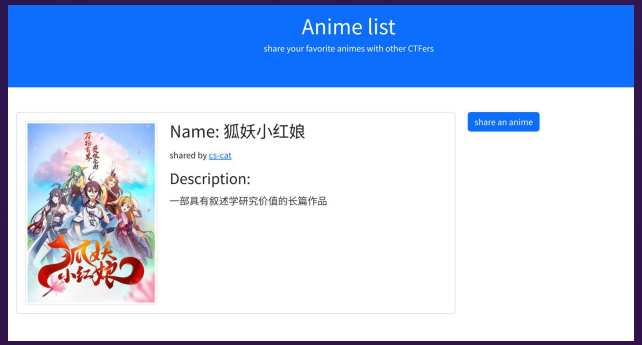
例子：Redis的Lua脚本功能

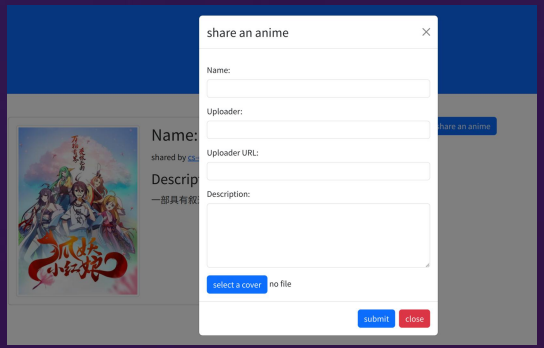
京麒 网络安全大会

3．题目分析

架构：nginx+Lua CGI，对外表现为Web服务器

目标：实现任意代码执行，执行/readflag





3．题目分析

上传新的项目：`cgi/additem.lua`

```
print(cgi.setdata(cgi.getdata()..cgi.postbody().."\n"))
```

获得已上传的项目：`cgi/getlist.lua`

```
print(cgi.getdata())
```

上传封面图片：`cgi/uploadimg.lua`

```
local fname=cgi.querystr()..".jpg"
if #fname>5 and #fname<32 then
    print(cgi.saveto(fname))
end
```

京麒 网络安全大会 2023 LCTFCON

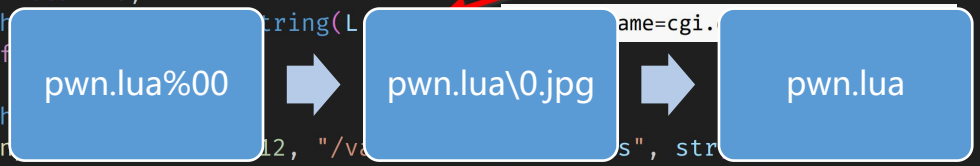
3. 题目分析

```
static int saveFile(lua_State *L) {
  if (strcmp(getenv("REQUEST_METHOD"), "POST") != 0)
    return 0;
  int clen = atoi(getenv("CONTENT_LENGTH"));
  if (clen > 200 * 1024)
    return 0;
  if (!lua_isstring(L, -1))
    return 0;
  char *str = lua_tostring(L, -1);
  if(strchr(str,'/'))
    return 0;
  char pathbuf[512];
  snprintf(pathbuf, 512, "/var/www/html/data/%s", str);
  int fd = open(pathbuf, O_WRONLY | O_TRUNC | O_CREAT, S_IRUSR | S_IWUSR);
  while (clen)
    clen -= splice(0, NULL, fd, NULL, clen, 0);
  close(fd);
  lua_pop(L, 1);
  lua_pushboolean(L, 1);
  return 1;
}
```

```
local fname=cgi.querystr()..".jpg"
```

3. 题目分析

```
static int saveFile(lua_State *L) {
  if (strcmp(getenv("REQUEST_METHOD"), "POST") != 0)
    return 0;
  int clen = atoi(getenv("CONTENT_LENGTH"));
  if (clen > 200 * 1024)
    return 0;
  if (!lua_isstring(L, -1))
    return 0;
  ch                tring(L
  if
  ch
  sn                12, "/va                s", str
  int fd = open(pathbuf, O_WRONLY | O_TRUNC | O_CREAT, S_IRUSR | S_IWUSR);
  while (clen)
    clen -= splice(0, NULL, fd, NULL, clen, 0);
  close(fd);
  lua_pop(L, 1);
  lua_pushboolean(L, 1);
  return 1;
}
```



3．题目分析

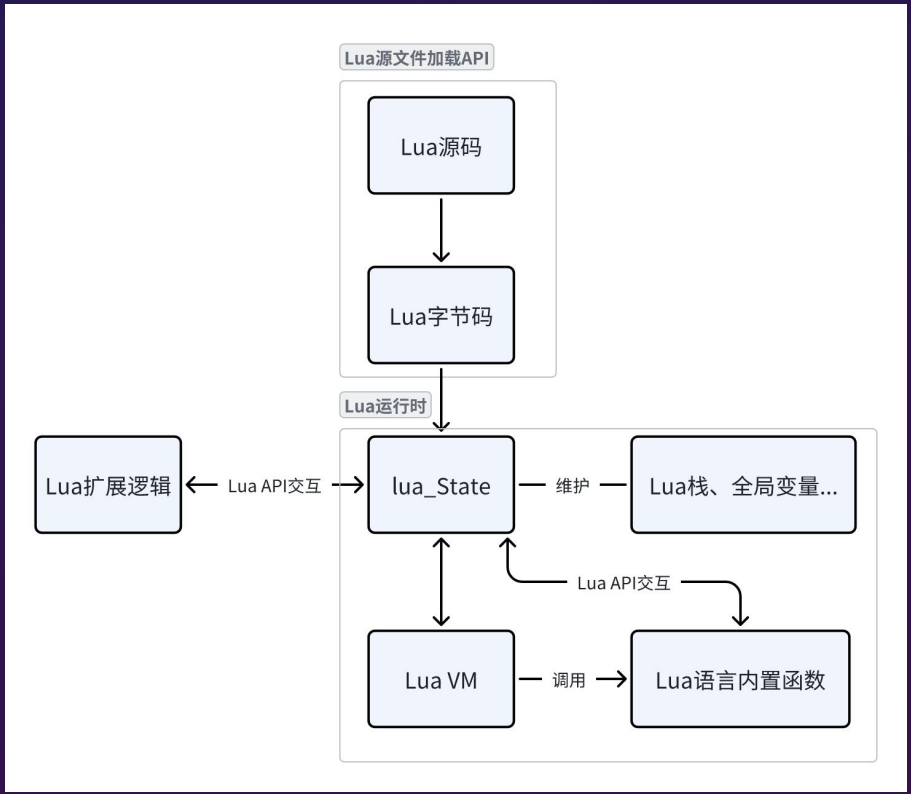
题目环境Lua为裁剪后的静态编译Lua。

任意Lua脚本执行无法产生执行/readflag的效果。

需要通过Lua沙盒逃逸实现任意shellcode执行。

预期攻击方式：Lua字节码攻击

4.1 Lua引擎简介


Lua源文件加载API
Lua源码
Lua字节码
Lua运行时
Lua扩展逻辑
Lua API交互
lua_State
维护
Lua栈、全局变量...
Lua API交互
Lua VM
调用
Lua语言内置函数

京麒 网络安全大会

CTFCON 2023

4.1 Lua引擎简介

```
; Lua version 5.4.4, x64, time: 0.006936073303222656, render: 0.001
lfile: Lua bytecode executable, version 5.4 (click to expand)

main <input-file.lua:0,0> (5 instructions at b77ee6c9_3bbb0579)
0+ params, 2 slots, 1 upvalue, 0 locals, 2 constants, 0 functions
function main(...) --line 1 through 1
    1      VARARGPREP      0
    2      GETTABUP        0 0 0      ; _ENV "print"
    3      LOADK           1 1        ; "ctfcon 2023"
    4      CALL            0 2 1      ; 1 in 0 out
    5      RETURN          0 1 1      ; 0 out

    upvalues (1)
        index  name      instack   idx       kind
        0      _ENV      true      0         VDKREG (regular)
    constants (2)
        index  type      value
        0      string    "print"
        1      string    "ctfcon 2023"
end
```

https://luac.nl/s/b26f2b28598a7076223792baa2

4.1 Lua引擎简介

Lua字节码序列化：luac程序 / string.dump函数

Lua字节码反序列化：load函数 / **Lua源文件加载API自动识别**

```
> string.dump(function() print("hello") end)
uaT
xV(w@stdin
          G print hello ENV
> load(string.dump(function() print("hello") end))()
hello
>
```

4.1Lua引擎简介

Lua沙盒往往会禁用load函数，从而避免加载恶意字节码。

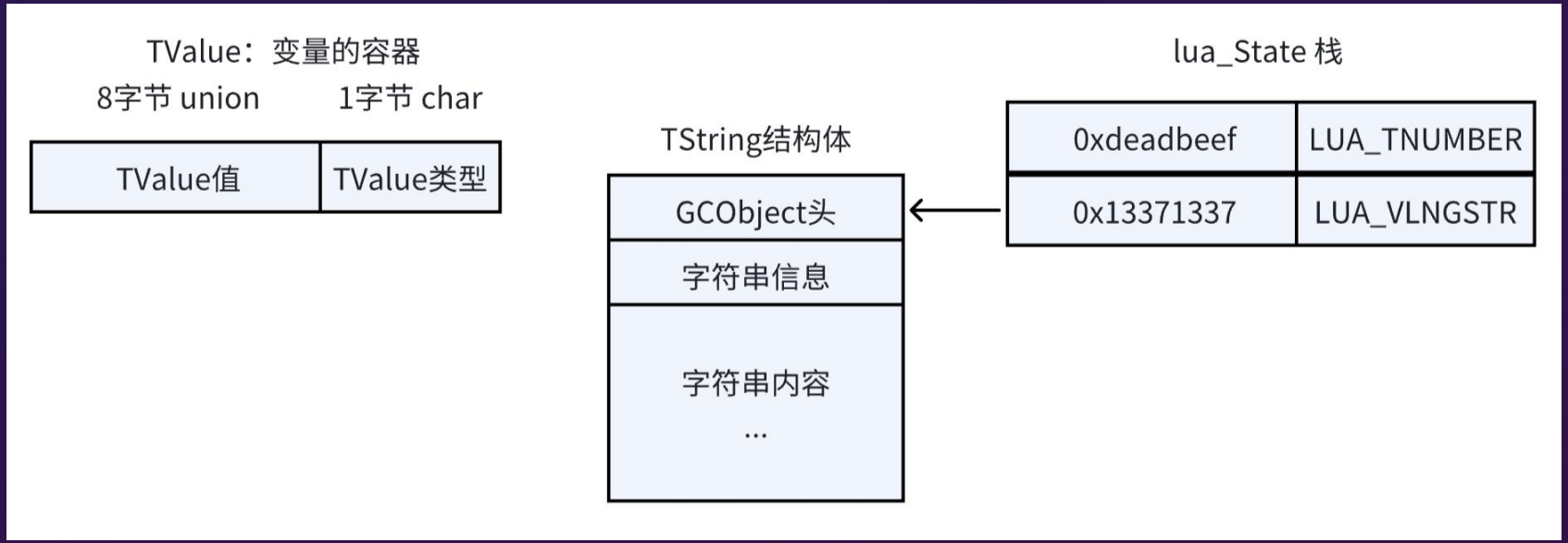
“It is safe to load malformed binary chunks; load signals an appropriate error. However, Lua does not check the consistency of the code inside binary chunks; running maliciously crafted bytecode can crash the interpreter. “

4.1 Lua引擎简介

但Lua沙盒加载Lua内容时，可能由于API误用而加载恶意字节码。

API名	功能	是否允许加载字节码
luaL_loadfile	从文件加载Lua chunk	允许
luaL_dofile	从文件加载Lua chunk并执行	允许
luaL_loadfilex	从文件加载Lua chunk	可选
luaL_loadbuffer	从缓冲区加载Lua chunk	允许
luaL_dostring	从字符串加载Lua chunk并执行	允许（但难以构造字节码）
luaL_loadbufferx	从缓冲区加载Lua chunk	可选

4.1 Lua引擎简介


TValue：变量的容器
8字节 union
1字节 char
TValue值
TValue类型
TString结构体
GCObject头
字符串信息
字符串内容
...
lua_State 栈
0xdeadbeef
LUA_TNUMBER
0x13371337
LUA_VLNGSTR

4.2 Lua5.4字节码攻击

沙盒逃逸三个重要原语：

addrof原语 -> 任意读原语 -> 任意写原语

三个原语到任意shellcode执行：

攻击语言引擎数据结构、攻击libc数据结构、泄漏栈地址进行ROP

4.2.1 实现addrof原语

通过tostring泄漏函数地址和table地址。**局限性：不能泄漏string地址**

```
> tostring({})
table: 0x555ffbdf8d60
> tostring(print)
function: 0x555ffa9d8220
```

4.2.1 实现addrof原语

通过string.format泄漏变量地址

```
> string.format("%p","a")
0x555ffbdf9400
> string.format("%p",print)
0x555ffa9d8220
> string.format("%p",{})
0x555ffbdf98a0
```

4.2.1 实现addrof原语

通过FORLOOP字节码类型混淆实现addrof原语

```
vmcase(OP_FORLOOP) {
  StkId ra = RA(i);
  if (ttisinteger(s2v(ra + 2))) {  /* integer loop? */
    lua_Unsigned count = l_castS2U(ivalue(s2v(ra + 1)));
    if (count > 0) {  /* still more iterations? */
      lua_Integer step = ivalue(s2v(ra + 2));
      lua_Integer idx = ivalue(s2v(ra));  /* internal index */
      chgivalue(s2v(ra + 1), count - 1);  /* update counter */
      idx = intop(+, idx, step);  /* add step to index */
      chgivalue(s2v(ra), idx);  /* update internal index */
      setivalue(s2v(ra + 3), idx);  /* and control variable */
      pc -= GETARG_Bx(i);  /* jump back */
    }
  }
```

京麒 网络安全大会 2023 LCTFCON

4.2.1 实现addrof原语

```
if(isInt(ra+2)) {
    unsigned cnt = load(ra+1);
    if(cnt > 0) {
        int step = load(ra+2);
        int idx = load(ra);
        store(ra+1,cnt-1);
        store(ra,idx+step);
        store_as_int(ra+3,idx+step);
    }
}
```

通过FORLOOP字节码类型混淆实现addrof原语，
获取RA对应变量的地址，以整数形式写入RA+3

RA	0x13371337(目标)	LUA_VLNGSTR
RA+1	1	LUA_VNUMINT
RA+2	0	LUA_VNUMINT
RA+3	0(任意值均可)	LUA_VNUMINT

FORLOOP →

RA	0x13371337(目标)	LUA_VLNGSTR
RA+1	0	LUA_VNUMINT
RA+2	0	LUA_VNUMINT
RA+3	0x13371337(目标)	LUA_VNUMINT

4.2.2 实现任意读原语

```
if(isInt(ra+2)) {
    unsigned cnt = load(ra+1);
    if(cnt > 0) {
        int step = load(ra+2);
        int idx = load(ra);
        store(ra+1,cnt-1);
        store(ra,idx+step);
        store_as_int(ra+3,idx+step);
    }
}
```

通过FORLOOP字节码类型混淆实现TValue值的任意构造，
将类型为LUA_VLNGSTR的TValue的值改写到要读取的地址附近，
相当于在要读取的地址附近伪造一个TString

RA	addr1	LUA_VLNGSTR	FORLOOP →	RA	target	LUA_VLNGSTR
RA+1	1	LUA_VNUMINT		RA+1	0	LUA_VNUMINT
RA+2	target-addr1	LUA_VNUMINT		RA+2	target-addr1	LUA_VNUMINT
RA+3	0(任意值均可)	LUA_VNUMINT		RA+3	target	LUA_VNUMINT

4.2.2 实现任意读原语

在target-16处伪造TString，

通过lnglen字段配合OP_LEN实现任意地址8字节读

```
typedef struct TString {
  CommonHeader;
  lu_byte extra;  /* reserved words for short strings; "has hash" for longs */
  lu_byte shrlen;  /* length for short strings */
  unsigned int hash;
  union {
    size_t lnglen;  /* length for long strings */
    struct TString *hnext;  /* linked list for hash table */
  } u;
  char contents[1];
} TString;
```

4.2.3 实现任意写原语

Lua中没有数组的概念，Table同时具备哈希表和数组的功能

通过伪造Table，在目标地址附近伪造TValue数组，再向数组写入内容

```
typedef struct Table {
  CommonHeader;
  lu_byte flags;  /* 1<<p means tagmethod(p) is not present */
  lu_byte lsizenode;  /* log2 of size of 'node' array */
  unsigned int alimit;  /* "limit" of 'array' array */
  TValue *array;  /* array part */
  Node *node;
  Node *lastfree;  /* any free position is before this position */
  struct Table *metatable;
  GCObject *gclist;
} Table;
```

4.2.3 实现任意写原语

在Lua字符串中伪造Table结构体。通过addrof原语获得伪造结构体的地址。

将伪造的地址作为目标Table，向下标0写入一个Lua整数，即可实现任意地址写。

会破坏目标地址+8处的一个字节（TValue类型标签）

```
local function write(a,v)
    -- write p64(v) to a
    -- will spoil a byte @ a+8
    local tbl=string.pack("I4j",256,a)
    write_tbl(addrof(tbl)+24-12,v)
end
```

4.2.3 实现任意写原语

OP_SETLIST实现任意值作为Table指针 + 数组形式写访问

```
vmcase(OP_SETLIST) {
  StkId ra = RA(i);
  int n = GETARG_B(i);
  unsigned int last = GETARG_C(i);
  Table *h = hvalue(s2v(ra));
  if (n == 0)
    n = cast_int(L->top.p - ra) - 1;  /* get up to the top */
  else
    L->top.p = ci->top.p;  /* correct top in case of emergency GC */
  last += n;
  if (TESTARG_k(i)) {···
  if (last > luaH_realasize(h))  /* needs more space? */
    luaH_resizearray(L, h, last);  /* preallocate it at once */
  for (; n > 0; n--) {
    TValue *val = s2v(ra + n);
    setobj2t(L, &h->array[last - 1], val);
    last--;
    luaC_barrierback(L, obj2gco(h), val);
  }
  vmbreak;
}
```

将RA作为Table（不检查类型），RA+1开始的n个TValue依次写入Table的数组部分，下标从last开始。

last=0

n=1

RA+1为待写入的值

4.3 实现任意shellcode执行

1. addrof (tostring) 泄漏ELF基址

2. 任意地址读，读取environ变量，泄漏栈地址

3. 在栈上用任意地址写布置ROP

题目环境及exp：
https://github.com/Nu1LCTF/n1ctf-2023/tree/main/pwn/anime

5．总结

允许加载任意Lua字节码很危险

Redis CVE-2015-4335

Roblox Studio Arbitrary Code Execution (2022.10.26 静默修复)

5．总结

缓解措施：

1. Lua沙箱禁止load函数或强制使用文本模式加载
2. 在沙箱内不使用允许加载字节码的Lua函数

API名	功能	是否允许加载字节码
luaL_loadfile	从文件加载Lua chunk	允许
luaL_dofile	从文件加载Lua chunk并执行	允许
luaL_loadfilex	从文件加载Lua chunk	可选
luaL_loadbuffer	从缓冲区加载Lua chunk	允许
luaL_dostring	从字符串加载Lua chunk并执行	允许（但难以构造字节码）
luaL_loadbufferx	从缓冲区加载Lua chunk	可选


京麒
网络安全大会
2023
CTFCON
THANKS