Lua in WAFs

An Examination of Lua's Use in ModSecurity
And Other Web Application Firewalls

Overview

- Review - WAFs

- FOSS Implementations
  - ModSecurity
    - DSL
    - Features
    - Lua Extensibility
  - Nginx/OpenResty
    - WAF projects
    - Ideologies

Overview - WAF

- Web Application Firewall
  - Inspect Layer 7 / HTTP(S) traffic
  - Block/log malicious transactions
  - Manipulate request/response content
  - Deep inspection into client behavior

Overview – WAF Features

- “Required” Features
  - Access/manipulate request + response headers + body
  - Flexible, extensible configuration
  - Tunable (false-positives, app-specific use cases)
  - Anti-evasion mechanism
  - Persistent data storage
  - Turing-complete configuration language/extensions
  - Detailed, performant audit logging
    - != debug logging

ModSecurity

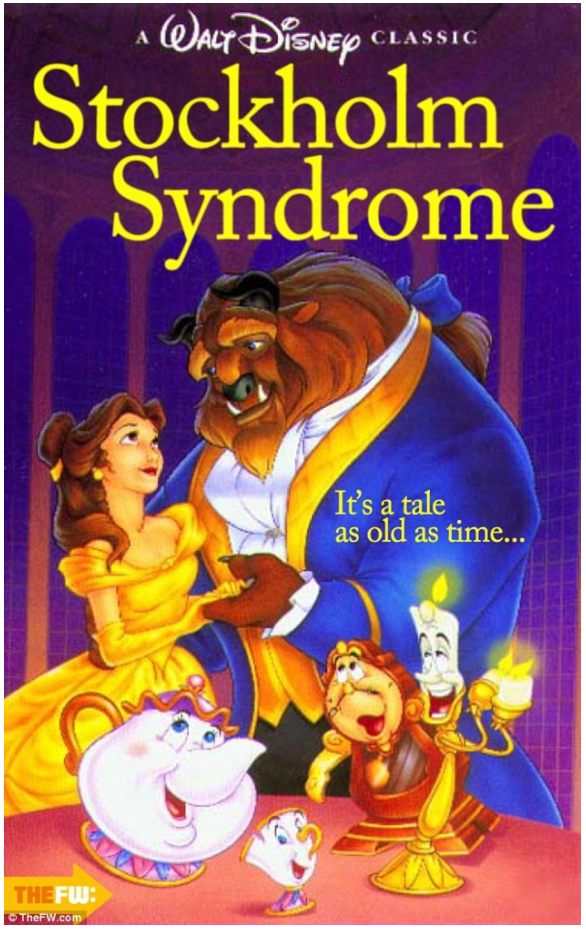
- Apache Module
- Unique DSL
- Request/response headers and body
- Extensible via custom C modules, and Lua scripts

```
SecRule REQUEST_COOKIES|!REQUEST_COOKIES:/__utm/|REQUEST_COOKIES_NAMES|ARGS_NAMES|ARGS|XML:/* "(?i:(j|(&#x?
0*((74)|(4A)|(106)|(6A));?))([\t]|(&((#x?0*(9|(13)|(10)|A|D);?)|(tab;)|(newline;))))*(a|(&#x?0*((65)|(41)|(97)|
(61));?))([\t]|(&((#x?0*(9|(13)|(10)|A|D);?)|(tab;)|(newline;))))*(v|(&#x?0*((86)|(56)|(118)|(76));?))([\t]|
(&((#x?0*(9|(13)|(10)|A|D);?)|(tab;)|(newline;))))*(a|(&#x?0*((65)|(41)|(97)|(61));?))([\t]|(&((#x?0*(9|(13)|
(10)|A|D);?)|(tab;)|(newline;))))*(s|(&#x?0*((83)|(53)|(115)|(73));?))([\t]|(&((#x?0*(9|(13)|(10)|A|D);?)|
(tab;)|(newline;))))*(c|(&#x?0*((67)|(43)|(99)|(63));?))([\t]|(&((#x?0*(9|(13)|(10)|A|D);?)|(tab;)|
(newline;))))*(r|(&#x?0*((82)|(52)|(114)|(72));?))([\t]|(&((#x?0*(9|(13)|(10)|A|D);?)|(tab;)|(newline;))))*(i|
(&#x?0*((73)|(49)|(105)|(69));?))([\t]|(&((#x?0*(9|(13)|(10)|A|D);?)|(tab;)|(newline;))))*(p|(&#x?0*((80)|(50)|
(112)|(70));?))([\t]|(&((#x?0*(9|(13)|(10)|A|D);?)|(tab;)|(newline;))))*(t|(&#x?0*((84)|(54)|(116)|(74));?))
([\t]|(&((#x?0*(9|(13)|(10)|A|D);?)|(tab;)|(newline;))))*(:|(&((#x?0*((58)|(3A));?)|(colon;))))." \
"phase:request,\
rev:'3',\
ver:'OWASP_CRS/3.0.0',\
maturity:'8',\
accuracy:'8',\
id:941210,\
capture,\
logdata:'Matched Data: %{TX.0} found within %{MATCHED_VAR_NAME}: %{MATCHED_VAR}',\
t:none,t:removeNulls,t:utf8toUnicode,t:urlDecodeUni,t:htmlEntityDecode,t:jsDecode,t:cssDecode,\
block,\
ctl:auditLogParts=+E,\
msg:'IE XSS Filters - Attack Detected.',\
tag:'application-multi',\
tag:'language-multi',\
tag:'platform-multi',\
tag:'attack-xss',\
tag:'OWASP_CRS/WEB_ATTACK/XSS',\
tag:'WASCTC/WASC-8',\
tag:'WASCTC/WASC-22',\
tag:'OWASP_TOP_10/A3',\
tag:'OWASP_AppSensor/IE1',\
tag:'CAPEC-242',\
setvar:'tx.msg=%{rule.msg}',\
setvar:tx.xss_score=+%{tx.critical_anomaly_score},\
setvar:tx.anomaly_score=+%{tx.critical_anomaly_score},\
setvar:tx.%{rule.id}-OWASP_CRS/WEB_ATTACK/XSS-%{matched_var_name}=%{tx.0}"
```

ModSecurity DSL

ModSecurity's DSL makes my baby cry

ModSecurity DSL



"Give it 2 or 3 years for the Stockholm Syndrome to kick in"

ModSecurity Rule Definitions

- Directive
  - 'SecRule', 'SecAction'...
  - Originally (still) available as Apache module directive
- Collection
  - What in the transaction to look for
  - Headers, query string, source/dest addr,

    file upload metadata
  - Single element, or string- or integer-indexed array

Rule Defs

- Operator
  - How to examine the collection
  - Regex, string equality, pattern-matching, numeric comparison, external script execution
- Action
  - Disruptive
    - Allow, deny, redirect
  - Nondisruptive
    - Chain, setvar, sleep, exec
  - Metadata
    - Version, maturity, severity

Example Rule

```
GET /index.php?foo=bar HTTP/1.0


SecRule                 \
   ARGS_GET:foo         \
   “@streq bar”         \
   “id:12345,           \
   phase:2,             \
   deny                 \”
```

ModSecurity Variables

- Key-value store based on type
- Store data per-transaction and per-client/session/user
- Per-transaction
  - TX
- Persistent
  - IP, session, etc
  - Stored on disk

ModSecurity Variables

```
SecRule ARGS foo
“id:12345,setvar:TX.matched=12345,pass”

[…snip…]

SecRule TX:matched “@eq 12345” “deny”
```

ModSecurity Variables

```
SecAction “initcol:IP=%{REMOTE_ADDR}

SecRule URI “@streq /login.php” \
   “id:12345,setvar:IP.login_attempt=+1”

SecRule IP:login_attempt “@ge 5” \
   “id:12346,deny,msg:'Too many logins'”
```

Extending via Lua (1/3)

- Operator @inspectFile
  - Executes the given script for every collection element
  - Designed for file upload inspection
  - Any language
  - .lua extensions are executed internally

```
SecRule FILES_TMPNAMES
“@inspectFile /path/to/inspect.lua”
```

Lua @inspectFile

```
function main(filename)
  local f = io.open(filename, 'rb')
  local chunk = f:read(1024)

  local ret = string.match(chunk, '<script')

  return ret
end
```

Extending via Lua (2/3)

- Non-disruptive `exec`: action
  - Runs on rule match
  - No provided parameters
  - Some environmental variables
  - Must write something (anything) to stdout
  - Lua detection

```
SecRule ARGS “@streq foo”
“id:1234,exec:/path/to/script.lua”
```

Extending via Lua (3/3)

- Directive SecRuleScript
  - Executes Lua script with no target or operators
  - Optional actions

```
SecRuleScript “/path/to/script.lua” \
    “id:12345,deny,...”
```

Extending via Lua

Simple Lua API

- log()
- getvar()
- getvars()
- setvar()

- Exposed via module 'm'
  - Expects a function `main`

SecRuleScript

```
function main()
  m.log(9, “debug message”)

  local user-agent = m.getvar(“REQUEST_HEADERS.User-
Agent”)

  local ret = string.match(user-agent,
'base64_decode')

  if ret then m.setvar(“TX.match”, user-agent) end

  return ret
end
```

Implementation

- apache2/msc_lua.c
  - 518 loc
  - First commit 2007
  - 3 globals
    - transaction and rule struct userdata
    - m for function register

```
> $ git log -c msc_lua.c | egrep -c 'commit [[:alnum:]]{40}'
> 34
```

Implementation – ModSec Vars

- Variables set on rule match
  - Unconditional set
  - Additional logic requires complex chaining
- Lua API
  - Set variables on a whim
  - Get/transform arbitrary variables

```
224     msre_var *vx = NULL;
225     msre_var *var;
226
227     /* Retrieve parameters. */
228     p1 = (char *)luaL_checkstring(L, 1);
229
230     /* Retrieve msr. */
231     lua_getglobal(L, "__msr");
232     msr = (modsec_rec *)lua_topointer(L, -1);
233
234     /* Retrieve rule. */
235     lua_getglobal(L, "__rule");
236     rule = (msre_rule *)lua_topointer(L, -1);
237
238     /* Extract the variable name and its parameter from the script. */
239     varname = apr_pstrdup(msr->msc_rule_mptmp, p1);
240     param = strchr(varname, '.');
241     if (param != NULL) {
242         *param = '\0';
243         param++;
244     }
245
246     /* Resolve variable. */
247     var = msre_create_var_ex(msr->msc_rule_mptmp, msr->modsecurity->msre,
248         varname, param, msr, &my_error_msg);
249
250     if (var == NULL) {
251         msr_log(msr, 1, "%s", my_error_msg);
252
253         lua_pushnil(L);
254
255         return 0;
256     }
257
258     /* Resolve transformation functions. */
259     tfn_arr = resolve_tfns(L, 2, msr, msr->msc_rule_mptmp);
260
261     /* Generate variable. */
262     vx = generate_single_var(msr, var, tfn_arr, rule, msr->msc_rule_mptmp);
263     if (vx == NULL) {
264         lua_pushnil(L);
265
266         return 0;
267     }
268
269     /* Return variable value. */
270     lua_pushlstring(L, vx->value, vx->value_len);
271
272     return 1;
```

```c
/*
* \brief New setvar function for Lua API. Users can put back
* data in modsecurity core via new variables
*
* \param L Pointer to Lua state
*
* \retval -1 On failure
* \retval 0 On Collection failure
* \retval 1 On Success
*/
static int l_setvar(lua_State *L) {
    modsec_rec *msr = NULL;
    msre_rule *rule = NULL;
    const char *var_value = NULL;
    const char *var_name = NULL;
    int nargs = lua_gettop(L);
    char *chr = NULL;

    lua_getglobal(L, "__msr");
    msr = (modsec_rec *)lua_topointer(L, -1);

    lua_getglobal(L, "__rule");
    rule = (msre_rule *)lua_topointer(L, -1);

    if(nargs != 2)  {
        msr_log(msr, 8, "m.setvar: Failed m.setvar funtion must has 2 arguments");
        return -1;
    }
    var_value = luaL_checkstring (L, 2);
    var_name = luaL_checkstring (L, 1);

    lua_pop(L,2);

    if(var_value == NULL || var_name == NULL)
        return -1;

    chr = strchr((char *)var_name,0x2e);

    if(chr == NULL) {
        msr_log(msr, 8, "m.setvar: Must specify a collection using dot character - ie m.setvar(tx.myvar,mydata)");
        return -1;
    }

    return msre_action_setvar_execute(msr,msr->msc_rule_mptmp,rule,(char *)var_name,(char *)var_value);
}
```

Use Case

- Shared hosting provider
  - Existing ModSecurity infrastructure
  - Zero-day patching, brute-force prevention
  - No current request upload inspection

- Requirements
  - Extended functionality
  - Performance
  - Fail-open

Use Case - Functionality

- Proactive backdoor/shell/malware prevention
  - Proactive security posture
- Pair with existing on-disk malware scanning functionality
- POSIX regex searching
  - lrexlib: binding for GNU, POSIX, PCRE regex libs

```
{
"decoded" : "^\s*send\(\s*flood,\s*pack\(.{1000}",
"target" : "PERL",
"action" : "disable",
"type" : "flooder",
"id" : "12345",
}
```

Use Case – Fail Forward

- Don't allow a bad expression, or large file, to hamper user experience
  - Short circuit expensive processing
  - Log failures

- lalarm – Lua binding for system alarm()

```
-- throw an exception after one second
alarm(1, function() error("timeout") end)

-- do our search
local res, err = pcall(search_sig)
if not res then m.log(ERR, err) end

-- cancel the pending alarm
alarm()
```

Use Case - Performance

- Goal: < 250 ms induced latency
  - 100 POSIX expressions, 125 fixed string searches
  - Per uploaded file
  - Apache's prefork model
  - Shared hosting environment = minimal additional CPU footprint
- Result = ~60 ms induced latency

ModSecurity – Moving Forward

- libmodsecurity
  - v3
  - Platform-agnostic
  - No Lua support... yet
    - Soon ^TM

WAFs in Nginx

- ModSecurity
  - Unstable
- Libmodsecurity
  - Nginx Plus
  - Feature-incomplete
- Naxsi
  - Lightweight, but feature lacking
    - No anti-evasion, persistent storage, response analysis
  - New DSL

OpenResty

- Nginx + Lua
- High-performance proxy + app server
- Extended integration
  - Third-party modules
  - Shared libs via LuaJIT FFI

OpenResty WAF Implementations

- High-level DSL
  - Compile down to Lua
  - Sup Cloudflare
- quickdefence
  - First appeared Jan. 2014
- lua-resty-waf

lua-resty-waf - Overview

- FreeWAF
  - Free, open, scalable cloud reverse proxy
  - Did I mention free?
- FOSS development
  - Two years of on-off development
  - 6 contributors

lua-resty-waf – Project Goals

- ModSecurity compatibility
  - No new DSL
  - No new design paradigm
  - Functional/architectural limitations
- Performance
  - Minimize engine footprint
  - ~ 300 us latency
  - ~ 15000 req/s

lua-resty-waf – Project Goals

- Stability / Compatibility
  - Novelty is unnecessary
  - Quick feature request, bugfix turnaround
- Extensibility
  - Configurable function hooks
  - Back to flexible DSL
  - OPM integration

lua-resty-waf - Implementation

- Configuration inheritance
- Precalculation and memoization
  - Rule jumps
  - Collection transformation caching
- Extensive use of lookup tables

```
match, value = operators.lookup[operator](
    self,
    collection,
    pattern,
    ctx
)
```

lua-resty-waf – ModSec Translation

- ModSecurity DSL → JSON
  - lua-resty-waf reads JSON directly
  - Possible to implement our own DSL for additional features

```
./modsec2lua-resty-waf.pl < \
/path/to/modsec.conf      > \
/path/to/rules/ruleset.json
```

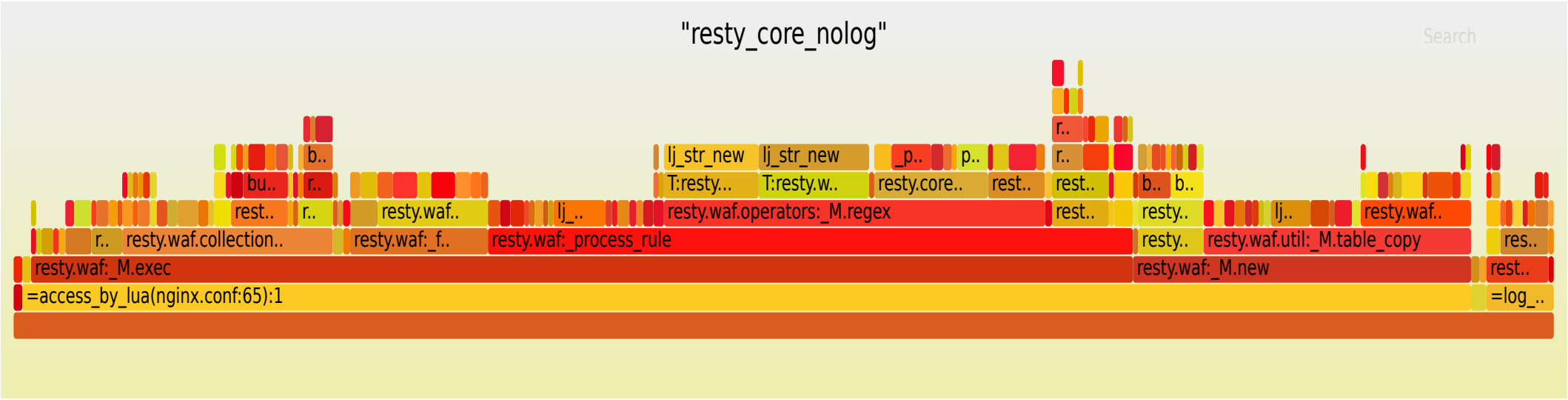
lua-resty-waf – Persistent Storage

- ModSecurity
  - SDBM
  - Slow, double-disk read, subject to race conditions

- lua-resty-waf
  - Per-worker shdict, memcached, redis
  - Inheritance architecture allows for rapid development of new backend engine

lua-resty-waf – resty.core

- FFI API for ngx_lua
- Compatible with lua-resty-core >= v0.1.5
  - Bugfix merged based on lua-resty-waf use case
  - FOSS ftw!
- Significant performance increase
  - ~ 450 us → ~ 300 us
  - Still more to do


"resty_core_nolog"
Search
r..
b..
lj_str_new
lj_str_new
_p..
p..
r..
bu..
r..
T:resty...
T:resty.w..
resty.core..
rest..
rest..
b..
b..
rest..
r..
resty.waf..
lj_..
resty.waf.operators:_M.regex
rest..
resty..
lj..
resty.waf..
r..
resty.waf.collection..
resty.waf:_f..
resty.waf:_process_rule
resty..
resty.waf.util:_M.table_copy
res..
resty.waf:_M.exec
resty.waf:_M.new
rest..
=access_by_lua(nginx.conf:65):1
=log_..

```
Distribution of Lua code pure execution time (accumulated in
each request, in microseconds) for 10000 samples:
(min/avg/max: 279/334/1219)
value |-------------------------------------------------- count
   64 |                                                      0
  128 |                                                      0
  256 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@  9523
  512 |@@                                                     472
 1024 |                                                      5
 2048 |                                                      0
 4096 |                                                      0
```

lua-resty-waf – Third Party Libs

- libinjection
  - Small C lib
  - 608 → 544 bytes per instance (PR pending ;))
- lua-aho-corasick
  - @pm operator
- Community lua-resty-* libs
  - lua-resty-iputils, lua-resty-cookie

lua-resty-waf - Limitations

- Some rule limitations
  - Disruptive actions in phase 4
  - Same limitations in libmodsecurity
- No rule sanity checking on load
  - Yet!
- Not every collection/operator translated
  - Some architectural limitations, some not worthwhile

Resources

- https://github.com/SpiderLabs/ModSecurity
- http://openresty.org/
- https://github.com/p0pr0ck5/lua-resty-waf

Questions?