Fast, Flexible Packet Filtering

Lua Kernel Scripting in NetBSD

Andrew von Dollen (avondoll@calpoly.edu)
EuroBSDCon 2018
1. Background and Setup

2. Experiments

3. Performance Results

4. Conclusion
Co-investigator Sam Freed

Existing Lua scripting support in NetBSD, Marc Balmer [1]

*Significant* NPF Lua groundwork laid by Lourival Vieira Neto [2] [3]
Background and Setup
Lua and NetBSD

Kernel-space Lua scripting support built into NetBSD [1]
Framework to make kernel subsystems (such as NPF) scriptable
Emphasis on system integrity and performance
npf_ext_lua kernel binding [2]

- Integrates directly with NPF
- Flexibility of Lua
- Supports deep packet inspection
Packet Filtering with Lua

With the npf_ext_lua kernel binding, we can write packet filtering scripts in Lua.

**Key Question**
What is the impact on packet filtering performance when we introduce Lua?
Experiments
VirtualBox VMs running 64-bit NetBSD v7.0

4GB of RAM, 2 x Intel(R) Xeon(R) CPU D-1521 @ 2.40GHz

Virtual NIC speed: 1 Gbps
Setup: Kernel Module

Existing npf_ext_lua implementation and support code:

- [https://github.com/lneto/luadata](https://github.com/lneto/luadata)
- To avoid a stack overflow error under load, add `lua_pop(L, 1);` after the line that begins with `*decision =` in the function `npf_lua()`
procedure "lua_filter" {
  # filter_function is defined in separate .lua file
  # which must be loaded into the npf state
  lua: call filter_function
}

group default {
  pass in proto tcp to $ext_if port n apply "lua_filter"
}
local data = require("data")
local tcphdr_layout = {
    src_port = {0, 16, 'number', 'net'},
dst_port = {16, 16, 'number', 'net'},
seq_num = {32, 32, 'number', 'net'},
ack_num = {64, 32, 'number', 'net'},
data_offset = {96, 4, 'number', 'net'},
reserved = {100, 3, 'number', 'net'},
control_flags = {103, 9, 'number', 'net'},
window_size = {112, 16, 'number', 'net'},
checksum = {128, 16, 'number', 'net'},
urgent = {144, 16, 'number', 'net'}
}

# Layouts for other protocols take a similar form
--- packet_filter.lua

function filter_function(pkt)
    pkt:layout(iphdr_layout)
    tcphdr_offset = pkt.ihl * 4
    pkt = pkt:segment(tcphdr_offset)
    pkt:layout(tcphdr_layout)
    tcpdata_offset = pkt.data_offset * 4
    pkt = pkt:segment(tcpdata_offset)
    local str = tostring(pkt)

    -- packet is dropped if function returns false
    return str:find("MALICIOUS VALUE") == nil
end
# luactl load npf ./packet_filter.lua

This loads packet filtering code, defined in the file packet_filter.lua, into the npf Lua state.
Performance Results
Test Scenarios

(1) Simple pass-through
(2) Packet header inspection: length check
(3) Deep packet inspection: search packet body for malicious string
(4) ICMP flood

Where applicable, we compare with NPF-only approach
iPerf3 (https://iperf.fr/)

- `iperf3 -c host --interval 0 --bandwidth 0 --parallel N --file test.data`
- Measures average bandwidth (Mbps) and retry statistics

Nping (https://nmap.org/nping/)

- `nping --icmp --icmp-type echo-request --hide-sent --rate 100 -c 100 host`
- Measures raw packets per second and RTT
Results 1 of 4: Pass-through

Passthrough: TCP Bandwidth and Retries

Concurrent Connections

- Bandwidth, NPF
- Bandwidth, Lua Filter
- Retries per Second, NPF
- Retries per Second, Lua
# npf.conf / pcap-filter
block in pcap-filter "udp and port N and greater 128"

-- Lua

function length_check(pkt)
    pkt:layout(iphdr_layout)
    udphdr_offset = pkt.ihl * 4
    pkt = pkt:segment(udphdr_offset)
    pkt:layout(udphdr_layout)
    return pkt.length > 128
end
Deep Packet Inspection: TCP Bandwidth and Retries

Concurrent Connections

- Bandwidth, Lua Filter
- Retries per Second, Lua

Bandwidth Levels:
- 0 Mbps (0 retries/sec)
- 250 Mbps
- 500 Mbps
- 750 Mbps

Retries per Second:
- 0
- 500
- 1000
- 1500

Graph shows the relationship between concurrent connections and bandwidth levels, with retries per second increasing as bandwidth increases.
Results 4 of 4: ICMP Flood

ICMP Flood: Packets per Second and Max RTT

Attempted Packet Sends (over a span of 10 seconds)

- Max RTT, NPF
- Max RTT, Lua
- Packets/s, NPF
- Packets/s, Lua
Conclusion
Performance Summary

- Lua performs well, achieves good average bandwidth and high packets/second
- Adding Lua to NFP introduces an approximate 10-15% bandwidth reduction based on simple performance tests
Possibilities

- Advanced deep packet inspection logic using approachable Lua syntax
- Rapidly adapt to new packet filtering requirements
- Experiment with approaches to packet filtering: novel data structures and algorithms
Summary

- Lua can greatly extend NPF and pcap-filter
- Lua filtering scripts are extremely flexible and easy to maintain
- Packet filtering using the Lua scripting language, while fundamentally practicable, warrants additional investigation and profiling in areas such as:
  - CPU and memory usage
  - Impact of multi-core CPUs
  - String matching on TCP streams
Questions?
Thank you!

Further questions? avondoll@calpoly.edu
M. Balmer.  
**Lua in the netbsd kernel.**  
FOSDEM, 2012.

L. V. Neto.  
**Npf scripting with lua.**  
EuroBSDCon, 2014.

L. Vieira Neto, R. Ierusalimschy, A. L. de Moura, and M. Balmer.  
**Scriptable operating systems with lua.**  