NPF Scripting with Lua

Scripting the NetBSD Packet Filter with Lua

Lourival Vieira Neto
lneto@NetBSD.org
“Any sufficiently complicated C or Fortran program contains an ad hoc, informally-specified, bug-ridden, slow implementation of half of Common Lisp.”

Greenspun’s tenth rule
Topics

- Introduction
  - Scriptable Operating System
  - Scripting a Packet Filter
- Example
  - SSH version
- Issues
- Why Lua?
- Kernel-scripting Environment
  - Lua(4)
- Luadata
- NPFLua
- Conclusion
Introduction
The combination of extensible operating systems with extension scripting languages.
Scriptable Operating System

- Motivation
  - Flexibility
    - Meet new user requirements
    - Configuration of kernel subsystems
  - Easy development
    - Allow application developers to customize the kernel
  - Prototyping
    - Add new features
Scriptable Operating System

- Key idea
  - OS kernel scripting

- Halfway between..
  - Kernel parameters and kernel modules
  - Domain-specific and system languages
Scriptable Operating System

- Two ways of scripting
  - Extending (a scripting language)
    - treats kernel as a library
  - Embedding (a scripting language)
    - treats kernel as a framework
Use Cases

- Embedding
  - Packet filtering
  - Device drivers
  - Process scheduling

- Extending
  - Web servers
  - File systems
  - Network protocols
Packet Filter Scripting

- Motivation
  - Deep packet inspection
  - Traffic shaping
  - Intrusion detection/prevention
- New features
  - Port knocking
  - Protocols
  - Port stealthing
local data = require'data'

function filter(pkt)
    -- convert packet data to string
    local str = tostring(pkt)

    -- pattern to capture the software version
    local pattern = 'SSH%-[^-G]+-%-\([^-G]+\)'

    -- get the software version
    local software_version = str:match(pattern)

    if software_version == 'OpenSSH_6.4' then
        -- reject the packet
        return false
    end

    -- accept the packet
    return true
end
Issues
Issues

- System integrity
  - Correctness
  - Isolation
  - Liveliness

- Ease of development

- Effectiveness and efficiency
System Integrity

- **Correctness**
  - Sandboxing
  - Automatic memory management
  - “Single” thread
  - Protected call / fail-safe
  - Privileged only

- **Isolation**
  - Fully isolated execution states

- **Liveliness**
  - Cap the number of executed instructions
Ease of Development

- High-level language
- Dynamically typed
- Domain-specific API
Effectiveness and Efficiency

Proper bindings
- Interface between scripts and kernel
- Suited for addressing SOS issues
- Most difficult task
Why Lua?
Why Lua?

- Extensible extension language
  - Embeddable and extensible
  - C library
- Almost freestanding
- Small footprint
  - has 240 KB on -current (amd64)
- Fast
- MIT license
Why Lua?

- Safety features
  - Automatic memory management
  - Protected call
  - Fully isolated states
  - Cap the number of executed instructions
Why not .... ?

- **Python**
  - has **2.21 MB** on Ubuntu 10.10 (amd64)

- **Perl**
  - has **1.17 MB** on Ubuntu 10.10 (amd64)

- **Also..**
  - OS-dependent code
  - Hard to embed\(^1\)

\(^1\) twistedmatrix.com/users/glyph/rant/extendit.html
Kernel-scripting Environment: Lua(4)
Brief History

- 2008 – Lunatik/Linux
- 2010 – Lunatik/NetBSD
  - Google Summer of Code
  - Kernel-embedded Lua (mainly)
- 2013 – Lua(4)
  - New infrastructure (Marc Balmer)
- 2014 – NPFLua
Kernel-embedded Lua
- has no floating-point numbers

User Interface
- luactl

Kernel Programming Interface
- sys/lua.h
System Memory Binding: Luadata
- Regular Lua library
  - Kernel and user space

- Binds system memory
  - Memory block (pointer + size)
  - mbuf

- Safe
  - Boundary verification

- Packed data
  - Declarative layouts
Other features
- Bit fields
- String fields and conversion
- Segments (data decomposition)
- Endianness conversion
local data = require('data')

function filter(pkt)
    -- convert packet data to string
    local str = tostring(pkt)

    -- pattern to capture the software version
    local pattern = 'SSH%-[^-%G]+-%-([^-%G]+)'

    -- get the software version
    local software_version = str:match(pattern)

    if software_version == 'OpenSSH_6.4' then
        -- reject the packet
        return false
    end

    -- accept the packet
    return true
end
```lua
local rtp = {
    version    = {0, 2},
    extension  = {3, 1},
    csrc_count = {4, 4},
    marker     = {8, 1},
    type       = {9, 7},
}

-- apply RTP header layout in the payload
pld:layout(rtp)

-- if packet is encoded using H.263
if pld.type == 34 then
    -- reject the packet
    return false
end
```
Packet Filter Binding: NPFLua
The **NetBSD** Packet Filter

- Layers 3 and 4
- Stateful
- IPv4 and IPv6
- Extensible
  - Rule procedures
Binds **NPF to Lua**
- Kernel module + parser module
- Rule procedure
  ```
  #npf.conf
  procedure "lua_filter" {
    lua: call filter
  }
  ```
  ```
  group default {
      pass in all apply "lua_filter"
  }
  ```
- Script loading
  ```
  luactl load npf ./filter.lua
  ```
Conclusion
- **Actual Lua rules**, e.g.:
  block out final `lua-filter "filter.lua"`
- **mbuf** handling
  - `m_pulldown` fail-safe
  - non-contiguous strings using `luaL_Buffer()`
  - packet mangling
- **Multiple Lua states**
- **Automatic script loading**
  - `filter.lua`
- **Pre-defined layouts**
  - IP, TCP, UDP
- **Lua network library**
- **Rule editing**
- **Lua user-space** configuration
  - `/etc/npf.lua`
General-purpose and full-fledged programming language for packet filtering
  - e.g., pattern matching, hash table

“SSH Version” example
  - no measurable overhead (on 100 Mpbs virtual NIC)
    - 96 Mbps with or without scripting
    - 20 lines of Lua code

Luadata is a generic binding for memory
  - can be used for other kernel extensions
    - e.g., device drivers, network protocols

M. Rasiukevicius. NPF—Progress and Perspective. AsiaBSDCon 2014.

M. Rasiukevicius. NPF documentation. URL netbsd.org/~rmind/npf/.

Source code:
- netbsd.org/~lneto/pending/
- github.com/lneto/luadata
“Got questions?”
Lourival Vieira Neto
lneto@NetBSD.org