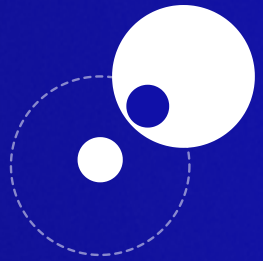


Lunatik

LUA IN THE KERNEL

Scripting Netfilter with Lua

A Cooperative Kernel-Userspace Pipeline

Lourival Vieira Neto[1] · Md. Shehar Yaar Tausif[2] · Firas Shaari[3] · Marcel S. A. de Moura[1] · Arif Alam[4]

[1] Ring Zero Networks [2] Independent [3] 802.11 Networks Corp [4] NetExperience

NETDEV 0X1A

Why Lua?

~250 KB

small and fast: the whole interpreter

freestanding C89

the core avoids OS-dependent libc; it embeds in any host and extends with its APIs

- Already scripts your network tools: **Wireshark** dissectors, **Nmap** NSE, **Snort** app detectors.
- **FreeBSD**'s boot loader runs Lua; **OpenBSD**'s httpd uses Lua's string library; **NetBSD** includes Lua in the kernel (`lua(4)`).
- Independent states: own heap, own garbage collector, and only the libraries you load.
- **The allocator comes from the host:** every allocation goes through a function it supplies (`lua_Alloc`).

BUT... REALLY? WHY LUA?

It’s easy.

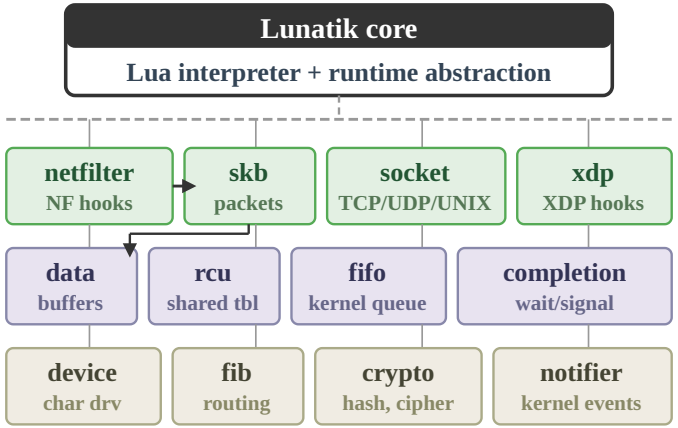
No complex toolchain. No static verifier: sandboxing instead. **Edit, reload, done.**

```
local device = require("device")
local linux  = require("linux")

local driver = {name = "passwd", mode = linux.stat.IRUGO}

function driver:read()
  return string.char(linux.random(32, 126))
end

device.new(driver)
:w /lib/modules/lua/passwd.lua
```

```
$ sudo lunatik run passwd
$ ls /dev/passwd
/dev/passwd
$ head -c 16 /dev/passwd
i9$Kv?xQ2pL&wZ7c
$
```

Lunatik: scripting the kernel in Lua

- A core: the Lua interpreter, running inside the kernel.
- Each **runtime** selects allocation and locking primitives per context; objects are refcounted and synchronized.
- An **open set of bindings** exposes kernel subsystems as Lua objects.
- Netfilter, socket buffers, RCU, routing table, notifier chains, XDP, more.



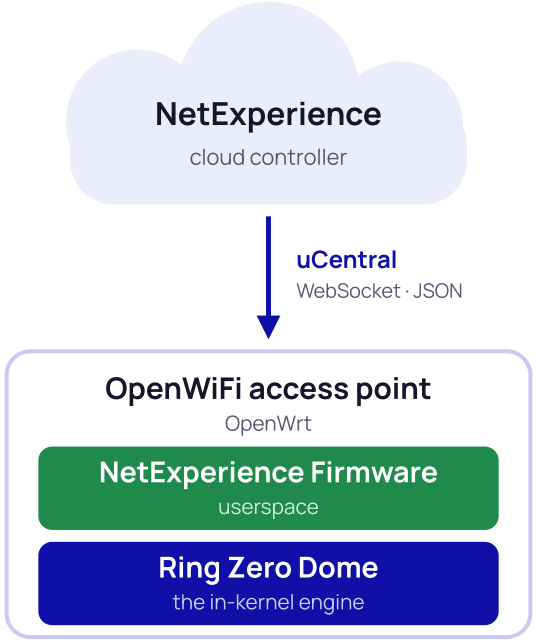
20+ independent kernel libraries, loaded on demand at runtime

What's a Secure Web Gateway?

- A web filter for outbound HTTPS: policy-based access control, anti-malware, ad blocking, parental controls.
- The standard play: **intercept TLS**, with a CA certificate on every client device.
- Per-session cryptographic work, per device.
- Cloud backhaul instead? Latency, uplink, jurisdiction.

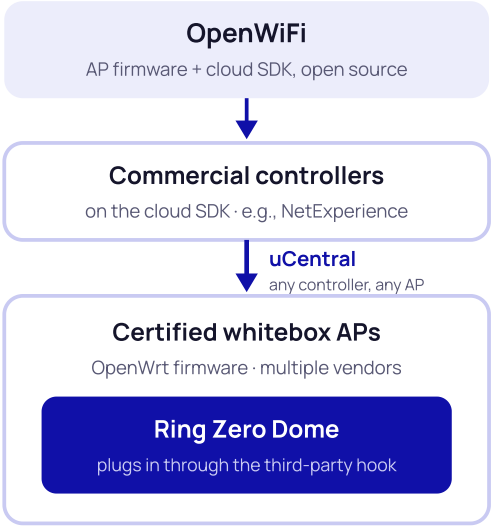
Dome: a Secure *Wireless* Gateway

- **OpenWiFi**: the Telecom Infra Project's open AP stack.
- **NetExperience**: a cloud controller for OpenWiFi, offering a Secure Wireless Gateway.
- **Ring Zero Dome**: the gateway's in-kernel engine, on the AP.
- No interception: it reads cleartext metadata, the TLS **SNI** (Server Name Indication), DNS names, HTTP Host.



The OpenWiFi ecosystem

- **OpenWiFi**: an open, disaggregated ecosystem for enterprise Wi-Fi.
- **Open source, both halves:** an OpenWrt-based AP firmware and a cloud controller SDK.
- **Certified whitebox APs** from multiple vendors; any controller manages any AP.
- **NetExperience** ships both halves on OpenWiFi: the controller and the AP firmware.
- **Services** plug in via a third-party hook: Dome.

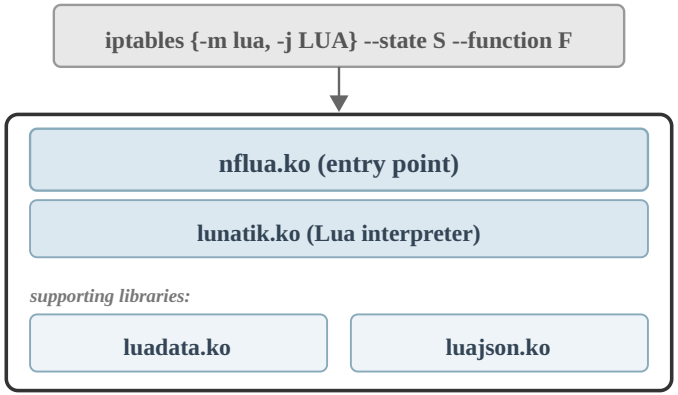


NFLua: where it started

20M routers

a safe-browsing service on NFLua, protecting 500M+ devices (Netdev 0x14)

- **Netdev 0x14 (2020):** NFLua filters via iptables (`-m lua` / `-j LUA`).
- Monolithic: fixed libs, iptables only, rebuild to extend.
- **Netdev 0x17 (2023):** rebuilt on the modular Lunatik.



all bindings statically linked at build time, iptables only

luanetfilter: direct Netfilter hooks

	NFLUA · NETDEV OX14	LUANETFILTER
Dispatch	`xtables` (-m lua / -j LUA)	**`nf_register_net_hook`**
Families	iptables only	bridge, inet, ipv4/6, arp, netdev
Userspace plugin	required	none
Bindings	static; rebuild to extend	loadable at runtime

A **Google Summer of Code 2024** project on the modular framework. The `BRIDGE` family filters bridged AP traffic with no IP routing; a hook fires only on its mark.

One hook composes the bindings

```
local netfilter = require("netfilter")
local nf      = require("linux.nf")
local eth     = require("linux.eth")
local pack    = require("pack")
local config  = require("config")
local ipv4    = require("ipv4")
local ipv6    = require("ipv6")
local tls     = require("tls")

local action      = nf.action
local unpacker    = pack.unpacker
local blocklist   = config.blocklist
local extract_sni = tls.extract_sni

local ETH_TYPE = 12
local parsers = {
  [eth.IP]   = ipv4.parse,
  [eth.IPV6] = ipv6.parse,
}
```

```
local function classify(skb)
  local frame = skb:data("mac")
  local _, _, be16 = unpacker(frame)
  local parse = parsers[be16(ETH_TYPE)]
  local _, off = parse(frame)
  local sni = extract_sni(frame, off)

  if blocklist[sni] then
    return action.ACCEPT, 0xd2   -- deny
  end
  return action.ACCEPT, 0xd1   -- allow
end

netfilter.register{
  hook     = classify,
  pf       = nf.proto.BRIDGE,
  hooknum  = nf.br.PRE_ROUTING,
  mark     = 0xd0,
}
```

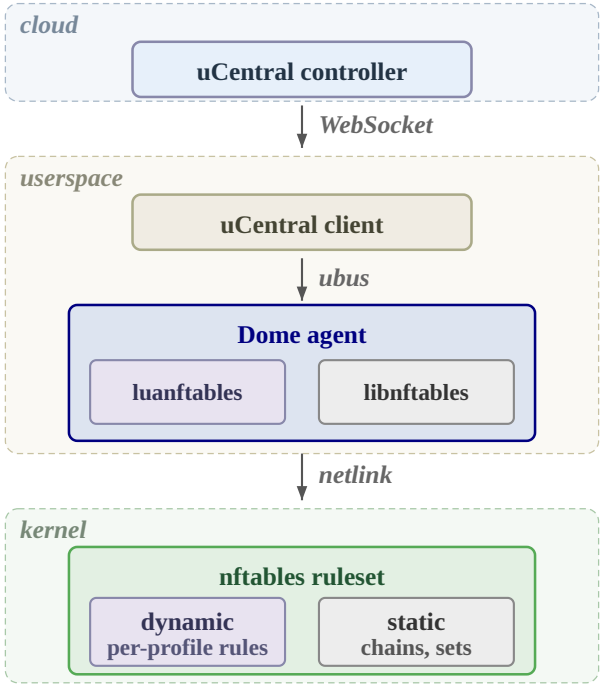
Walk the ClientHello to the SNI

```
function tls.extract_sni(packet, offset)
  local str, u8, be16 = unpacker(packet, offset)
  -- skip the variable-length fields
  local cipher = SESSION_ID_OFF + 1 + u8(SESSION_ID_OFF)
  local comp   = cipher + 2 + be16(cipher)
  local ext    = comp + 3 + u8(comp)
  for _ = 1, MAX_EXTENSIONS do          -- walk exts
    if be16(ext) == SERVER_NAME then
      return str(ext + 9, be16(ext + 7))
    end
    ext = ext + 4 + be16(ext + 2)
  end
end
```

- The SNI is nested in length-prefixed fields.
- Each offset is a length read from the field before it.
- Bounds-checked reads, so a malformed length can't overread kernel memory.

OpenWiFi pushes config over uCentral

- **uCentral:** a WebSocket from the AP to the controller; JSON config, schema-validated.
- A **third-party** extension point, where Dome plugs in.
- Each push hands Dome its policy blob, and the agent turns it into rules.



Policy is named profiles

```
"firetv": {
  "devices": ["f0:81:73:8a:1f:42"],
  "policy": "block",
  "allowlists": ["amazon-devices",
    "amazon-prime", "netflix"]
},
"household": {
  "ssids": [{ "name": "Home" }],
  "blocklists": ["threats", "trackers", "ads"]
}
```

- Matched by **MAC** (device) or **SSID**.
- Default allow or block; allow/blocklists per profile.
- Pushed as the uCentral `third-party.dome` blob.

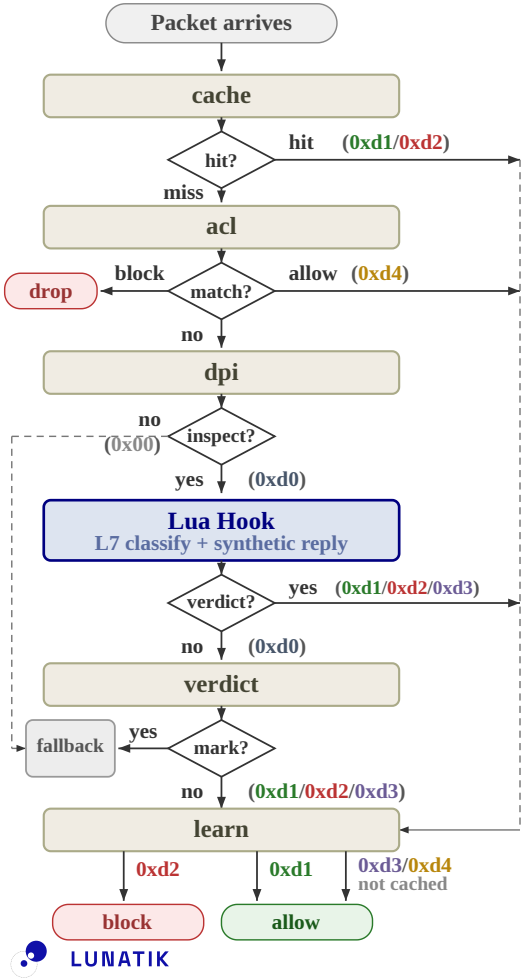
The agent generates the ruleset

```lua
local nft = require("nftables")
local ctx <close> = nft.new()

for name, p in pairs(profiles) do
  if p.devices then
    local set = "profile_" .. name
    ctx:cmd(fmt(new_set, set))

    for _, mac in ipairs(p.devices) do
      ctx:cmd(fmt(add_mac, set, mac))
    end

    if p.policy == "block" then
      ctx:cmd(fmt(drop, set))
    end
  end
end
```

- The agent is Lua too: it reads the profile blob and emits rules.
- **luanftables** drives `libnftables` in-process: no `nft` subprocess.
- A static template defines the pipeline; profiles fill the dynamic rules.

What lands in the kernel

```
$ nft list ruleset
table bridge dome {
  set skip {                                  # the verdict cache
    type ipv4_addr . ipv4_addr . inet_service . inet_service
  }
  set profile_firetv {                        # from the agent's loop
    type ether_addr
    elements = { f0:81:73:8a:1f:42 }
  }
  chain cache {
    type filter hook prerouting priority -310;
    ip saddr . ip daddr . tcp sport . tcp dport @skip meta mark set 0xd1 accept
    ...
  }
  chain acl {
    type filter hook prerouting priority -300;
    ether saddr @profile_firetv counter drop   # default-block profile
    ...
  }
  ...
}
```



THE DATAPATH PIPELINE

Six chains, one byte of state

Every chain reads and writes the packet's mark, threading classification state through the pipeline.

0x00	NONE	not inspected
0xd0	INSPECT	sent to Lua
0xd1	SKIP	allow; cache
0xd2	DENY	block; cache
0xd3	PENDING	reassembly
0xd4	PASS	acl allow

Forge a reply, in the kernel

```
-- flip DNS flags in place → NXDOMAIN
local function dns_nxdomain(frame, off)
  local _, _, be16  = unpacker(frame)
  local _, _, pbe16 = packer(frame)
  local flags = be16(off + 2)
  pbe16(off + 2, (flags & 0x0100) | 0x8083)
end

local function dns_reply(skb, off)
  local r = skb:copy()
  local frame = r:data("mac")
  dns_nxdomain(frame, off)
  swap_headers(frame); r:checksum(); r:forward()
end
```

- REJECT gives only RST / ICMP.
- luaskb: clone, rewrite, and **forward** an NXDOMAIN or HTTP 302.
- Blocked flows stay uncached → block page every time.

Track interfaces with one callback

```
local notifier = require("notifier")
local common   = require("common")
local box, lookup = common.box, common.lookup

-- vlans: the shared luarcu table, from the daemon
local function attacher(vlans)
  notifier.netdevice(function(event, ifname)
    if event == netdev.REGISTER and lookup(ifname) then
      local idx = linux.ifindex(ifname)
      vlans[tostring(idx)] = box(ifname)    -- writer
    end
  end)
end
```

- VLAN sub-interfaces appear at runtime (802.1X, MPSK).
- The notifier writes them into a **luarcu** table, in process context.
- The packet hook reads that same table **lock-free**, in softirq. No locks.
  - **luanotifier** (Netdev 0x17) and **luarcu**: pre-existing bindings, reused here.

METHODOLOGY

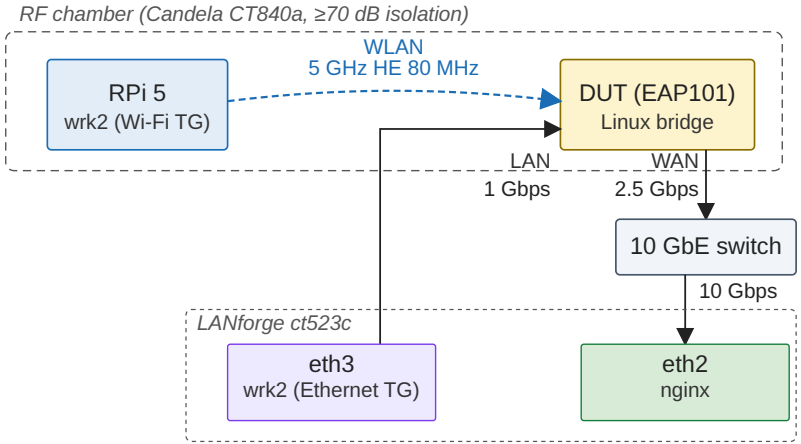
Response sizes, grounded in the Web Almanac

The HTTP Archive **Web Almanac** reports per-resource size distributions across the real web. Each swept size maps to a percentile or workload class.

1 KB	below p25: IoT / API heartbeats, short-lived connections
10 KB	median individual image; median HTML document
100 KB	between p75 and p90 of individual images
300 KB	above p90 of individual images
1024 KB	well above p90: bulk transfers (video streaming)

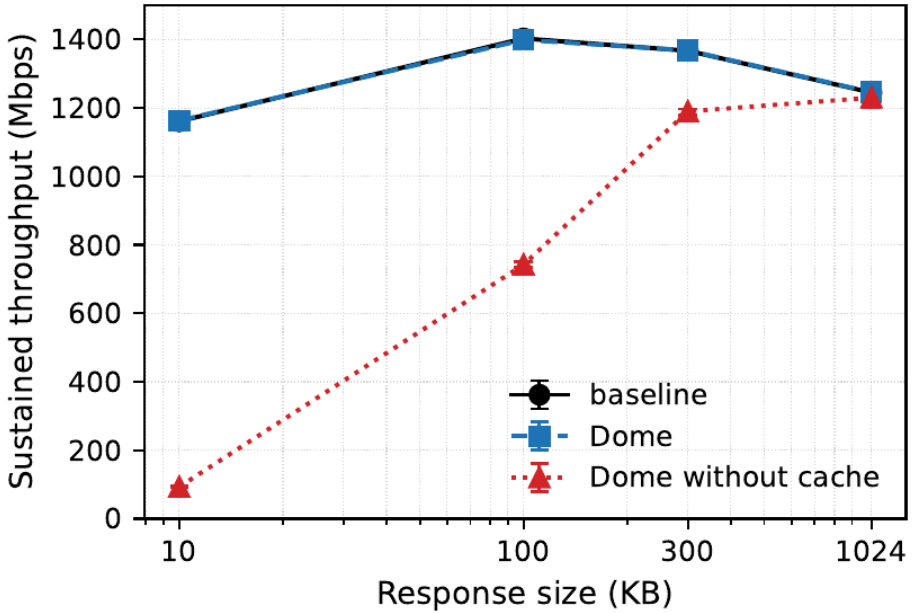
Testbed

- **Conditions:** plain bridge (baseline) · Dome · Dome without the cache.
- **Load:** `wrk2` from Wi-Fi and Ethernet at once, into `nginx` on the wired side (open loop, 4 threads × 100 TLS connections per ingress).
- **Sampling:** 10 reps of 30 s per cell; median with 95% CI; p50 and p99.
- **DUT:** EdgeCore EAP101, a Wi-Fi 6 AP, four Cortex-A53 cores, OpenWiFi.
- **RF chamber:** Raspberry Pi 5 station sealed with the AP; Candela CT840a, ≥70 dB isolation.



RESULT

Parity with a plain bridge



1.4 Gbps

aggregate throughput, cache active

latency: parity

p50 and p99 indistinguishable from run-to-run variability, every size

+14 to 19 pp

sys%, the whole cost

RESULT

The numbers

Table 2 of the paper: medians over 10 repetitions of 30 s per cell

CONDITION	REQ/S	MBPS	P50	P99	SYS%
10 KB					
baseline	12 758	1 161	176 ms	1.64 s	31.1
Dome	**12 774**	**1 163**	**163 ms**	**1.61 s**	**48.1**
without cache	1 017	93	18.3 s	27.8 s	99.5
100 KB					
baseline	1 607	1 403	12 ms	239 ms	31.2
Dome	**1 599**	**1 399**	**15 ms**	**245 ms**	**45.7**
without cache	845	742	6.3 s	21.6 s	94.8

CONDITION	REQ/S	MBPS	P50	P99	SYS%
300 KB					
baseline	521	1 367	145 ms	315 ms	31.7
Dome	**521**	**1 368**	**146 ms**	**350 ms**	**45.7**
without cache	454	1 190	539 ms	19.0 s	79.3
1024 KB					
baseline	137	1 245	503 ms	716 ms	34.5
Dome	**137**	**1 246**	**555 ms**	**786 ms**	**53.1**
without cache	135	1 230	710 ms	1.45 s	60.8

CROSS-CHECK

A second opinion: LANforge

CONDITION	URLS/S	L4 MBPS	BG MBPS
baseline	6 180	506	940
Dome	**5 994**	**491**	**940**
without cache	924	76	739

100 L4 HTTPS endpoints (10 KB) over Wi-Fi, plus a 1 Gbps non-inspected wired TCP background.

- A different generator model: 100 independent endpoints, not a persistent pool.
- Dome tracks the baseline within **3%**; without the cache, the 10 KB foreground falls to 924 URLs/s.
- The non-inspected background holds **940 Mbps** under Dome, same as baseline: no trace on flows the hook never sees.

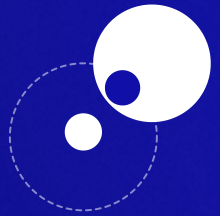
CLOSING

Final remarks

- **A Secure Web Gateway inside OpenWiFi APs:** 1.4 Gbps, bridge parity in throughput and latency.
- **Validated** on real-web sizes (Web Almanac), in an RF chamber with a LANforge appliance, cross-checked across tools.
- **Evolved from NFLua to luanetfilter:** up to 12× the throughput, 90× lower latency.
- **Kernel scripting:** powerful components in C (Netfilter), the glue in Lua (Dome).
- **In production** on the NetExperience OpenWiFi controller.

WHAT'S NEXT?

- **conntrack + act_ctinfo:** per-application QoS.
- **netlink:** rtnetlink, generic netlink, `nl80211`.
- **GSoC 2026:** eBPF Abstraction Layer: tc, `sched_ext`, and maps.



Lunatik

LUA IN THE KERNEL

Don’t compile.
Script the kernel.

Lunatik github.com/luainkernel/lunatik

luanftables github.com/ring0networks/luanftables

```
local netfilter = require"netfilter"
local netlink   = require"netlink"
local nf        = require"linux.nf"

local ANSWER <const> = 42
local audience = netlink.channel"Q&A"

local function talk(skb)
  print"Thank you!"

  local question = tostring(skb:data())
  if question:find"?$" then
    audience:multicast(ANSWER)
    skb:connmark(0xC105E)
  end
  return nf.action.ACCEPT
end

netfilter.register{
  hook    = talk,
  pf      = nf.proto.NETDEV,
  hooknum = nf.netdev.EGRESS,
  mark    = 0xC00010FF,
}
```