Enterprise-grade security for your cloud infrastructure with FOSS

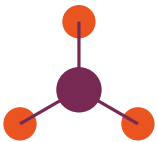
A layered, cost-effective approach built entirely on FOSS

Balachandran Sivakumar

OOSC ’26 — IIT Allahabad

whoami

- **Balachandran Sivakumar** — Chief Architect, CISOGenie
- Worked on cloud-native security & detection engineering
- Long-time FOSS user, evangelist and tinkerer
- Today: the security stack that I wish people took seriously



Built by the community,
for the community

What we'll leave with

A design that we can use with any kind of infrastructure — docker, k8s, etc. and no commercial licences required.

Where we're going

1. The problem: The attack surface
2. FOSS is enterprise-grade
3. Defense-in-depth: the design
4. The stack, layer by layer
5. Wiring it together
6. Live demo on 5 cloud nodes

Outcomes we're aiming for

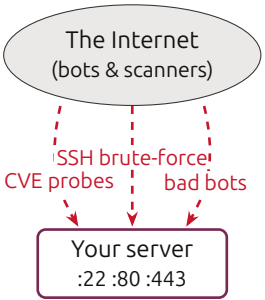
- **Detect** threats
- **Reduce** attack surface
- **Automate** incident response
- **See** what's happening (visibility)

PART 1

The problem, and why FOSS

Your workload is a target the moment it boots

- You don't have to be a target *by name or brand* — the whole internet is swept continuously
- A fresh public IP sees *attacks* within minutes
- Automated bots hunt for: open SSH, weak creds, unpatched CMS, exposed admin panels, known CVEs



The uncomfortable maths

Attackers automate. If your defence is manual, you lose even before you start.

”Enterprise-grade” is not a synonym for ”expensive”

What enterprises look for:

- Layered controls (defence in depth)
- Detection + prevention, not just walls
- Automated response
- Central visibility & audit trail
- Threat intelligence

FOSS gives you every one:

- nftables, NGINX
- Suricata (IDS *and* IPS)
- fail2ban, CrowdSec
- Wazuh (SIEM/XDR)
- CrowdSec community signals

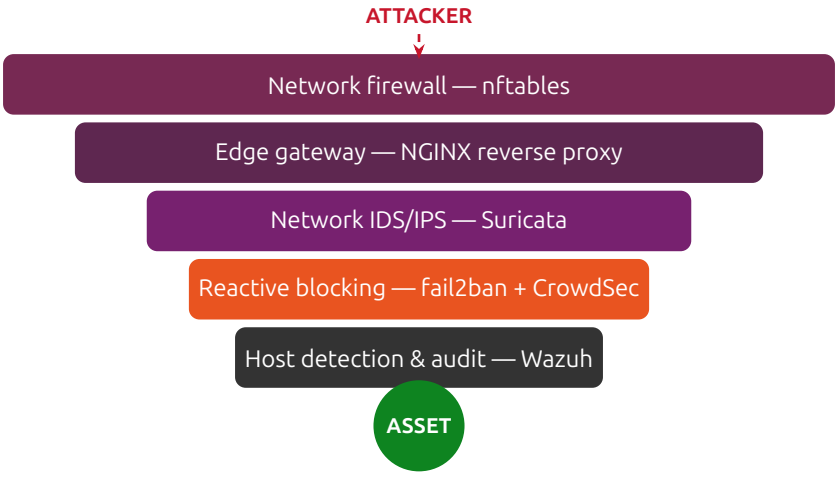
The real benefit

Not just cost — combines specialized security, cost efficiency, and absolute sovereignty Pick the perfect tool for every defense layer ***and*** claim total ownership of your security infrastructure.

PART 2

Defense in depth: the design

One wall fails. Layers don’t (all at once)



The stack at a glance

Tool	Role	Buys us
nftables	Stateful packet firewall (Netfilter)	Reduce attack surface
NGINX	Reverse proxy / TLS / web gateway	Reduce surface, control
fail2ban	Log-driven auto-ban of abusers	Automate response
CrowdSec	Behavioural detection + shared IP intel	Detect + community intel
Suricata	Network IDS/IPS (signatures + flows)	Detect + prevent
Wazuh	Host IDS, FIM, log SIEM/XDR	Detect + visibility

Six tools → four outcomes: detect • reduce • automate • see

PART 3

The stack, layer by layer

Layer 0 — nftables: close the doors first

- Modern replacement for iptables (Netfilter)
- Default-deny inbound; allow only what you serve
- One clean ruleset, simple syntax, atomic reloads
- Cheapest security win there is: surface you don't expose can't be attacked

Rule of thumb

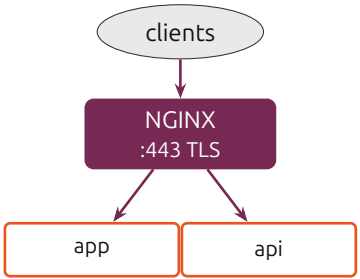
If you can't explain why a port is open, close it.

/etc/nftables.conf (essence)

```
table inet filter {
  chain input {
    type filter hook input
      priority 0; policy drop;
    ct state established,related accept
    iif "lo" accept
    tcp dport { 22, 80, 443 } accept
    ip protocol icmp accept
  }
}
```

Layer 1 — NGINX: one guarded front door

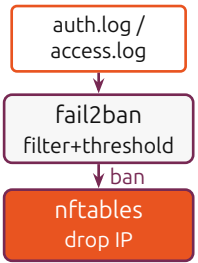
- All web traffic enters through one reverse proxy
- Terminates TLS; backends never face the internet
- Natural choke point for: rate limits, request size caps, header hardening, allow/deny lists
- Optional **ModSecurity** / Coraza WAF for app-layer rules
- Its access log becomes the fuel for fail2ban & CrowdSec



backends: private only

Layer 2 — fail2ban: the reflex

- Watches log files for patterns of abuse (failed SSH logins, 401/403 storms, probing 404s)
- Matches → bans the source IP via nftables for a set time
- Zero intelligence, total reliability: the security *reflex arc*
- Jails are just: a filter (regex) + an action (ban) + a threshold

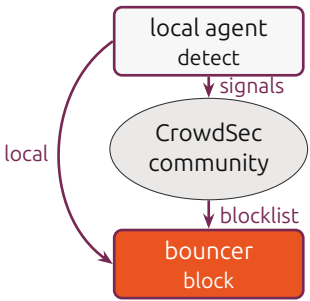


Rationale

”Test me 5 times in 10 minutes, you’re out for an hour.”

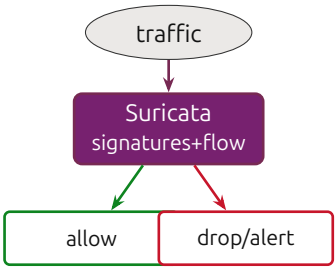
Layer 3 — CrowdSec: the neighbourhood watch

- Behavioural engine: parses logs, detects scenarios (brute force, scanning, credential stuffing)
- Decisions are enforced by bouncers (nftables, NGINX, cloud LB)
- The twist: community blocklist — an IP that attacked *someone else* is blocked *before* it reaches you
- fail2ban reacts to your box; CrowdSec reacts to the *whole community*



Layer 4 — Suricata: the network's own eyes

- High-performance network IDS/IPS
- Reads traffic against signatures (ET Open ruleset) + protocol & flow analysis
- **IDS** mode: watch & alert (span/mirror)
- **IPS** mode: inline, drop malicious packets
- Rich metadata: TLS SNI, HTTP, DNS, JA3, file hashes — gold for detection



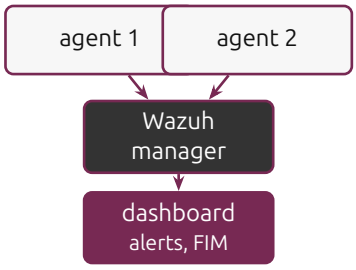
eve.json → Wazuh

Where it sits

On the wire, seeing what host tools can't — lateral movement, C2 beacons, exploit payloads.

Layer 5 — Wazuh: the memory and the dashboard

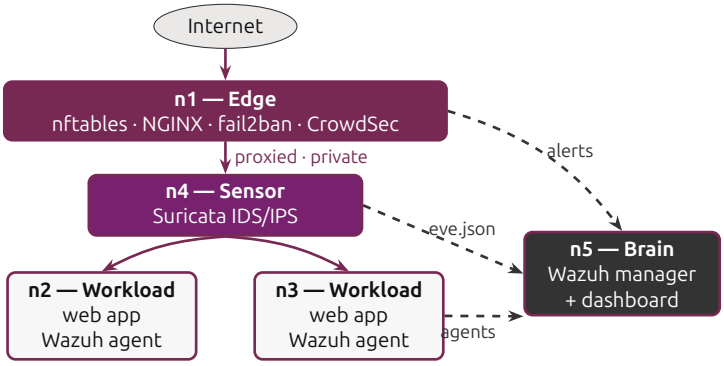
- Host-based detection + log analysis SIEM/XDR
- Agents on every node report to one manager
- Does: file-integrity monitoring, log rules, rootcheck, CIS/compliance, vulnerability signals
- Ingests Suricata & CrowdSec events too — becomes the single pane of glass
- This is your visibility and audit trail



PART 4

Wiring it together

The blueprint — one edge, one sensor, one brain



Five cloud nodes — two of them the workload we're protecting. Everything above is FOSS.

How the four outcomes map onto the stack

Outcome	Delivered by
Detect threats	Suricata (network) · CrowdSec (behaviour) · Wazuh (host, FIM)
Reduce surface	nftables default-deny · NGINX single front door · private backends
Automate response	fail2ban bans · CrowdSec decisions/bouncers · Suricata IPS drop
Improve visibility	Wazuh dashboard aggregating all of the above

Every promise in the abstract, mapped to a box you can `apt install`.

Cloud-native & Kubernetes: same blueprint, different knobs

Public / private cloud

- Security groups *and* nftables (belt + braces)
- CrowdSec bouncer at the cloud LB
- Wazuh agents bake into your golden image

Kubernetes

- Ingress-NGINX = your reverse-proxy layer
- Suricata as a DaemonSet / node sensor
- Wazuh agent as DaemonSet; watch the API & audit logs
- CrowdSec ingress bouncer for L7 bans

The point

The layers don't change — only *where* each control attaches. Learn the blueprint once; re-target it anywhere.

Honest trade-offs (so you deploy this well)

Watch out for

- IPS inline = you own the failure mode (fail-open vs closed)
- Auto-bans can lock *you* out — allowlist your admin IPs
- Suricata + Wazuh are memory heavy; size those nodes appropriately
- Tuning noise is ongoing work, not one-time

Good habits

- Start in *detect* mode, promote to *prevent*
- Version-control every config
- Alert on the manager being *silent*, too
- Test bans from a throwaway IP

PART 5

Live demo

Live demo — five real nodes, right now

Attack the apps on n2/n3 — watch nothing happen to them:

1. Meet the crown-jewel app; reach it *only* via n1
2. Brute-force SSH off vs on → fail2ban bans it
3. HTTP scanner → CrowdSec + community blocklist
4. Malicious request → Suricata alert, then IPS drop
5. Tamper a file → live Wazuh FIM alert; one dashboard
6. The real auth.log: strangers who found us today

Topology

n1 shield: nftables, NGINX, fail2ban, CrowdSec
n2, n3 the **workloads** (crown jewels) + Wazuh agents
n4 Suricata IDS/IPS sensor, inline
n5 Wazuh manager + dashboard

Full runbook in the speaker notes & `hands-on-demo.md`

If you take one slide home

The blueprint

1. **nftables** — close the doors
2. **NGINX** — one guarded front door
3. **fail2ban** — reflexive banning
4. **CrowdSec** — community threat intel
5. **Suricata** — see inside the traffic
6. **Wazuh** — remember & show everything

Start with 1–3 this week. Add 4–6 as you grow.

Enterprise-grade.
Zero licences.

Thank you

Questions?

Balachandran Sivakumar • OOSC ’26 — IIT Allahabad