CYBERAID

AI-Driven Cybersecurity for Financial Service Providers

Learning Center

Tutorial 4 | eBPF-Enhanced Network Monitoring

Seeing Inside the Kernel: Real-Time Visibility and Incident Identification for Financial Infrastructure

CyberAID has received funding from the DIGITAL EUROPE Programme under grant agreement No. 101249596

What is eBPF?

A technology that runs small, sandboxed programs inside the Linux kernel — triggered by events — without changing kernel source code or loading kernel modules.

Safe
A verifier checks every program before loading: bounded execution, valid memory access, no kernel crashes.

Fast
JIT-compiled to native machine code; data is filtered in the kernel, so only relevant events reach user space.

Dynamic
Programs are loaded and removed at runtime — no reboot, no kernel rebuild, no application restart.

Programmable
Maps and ring buffers share state and events between kernel programs and user-space collectors.

Evolution

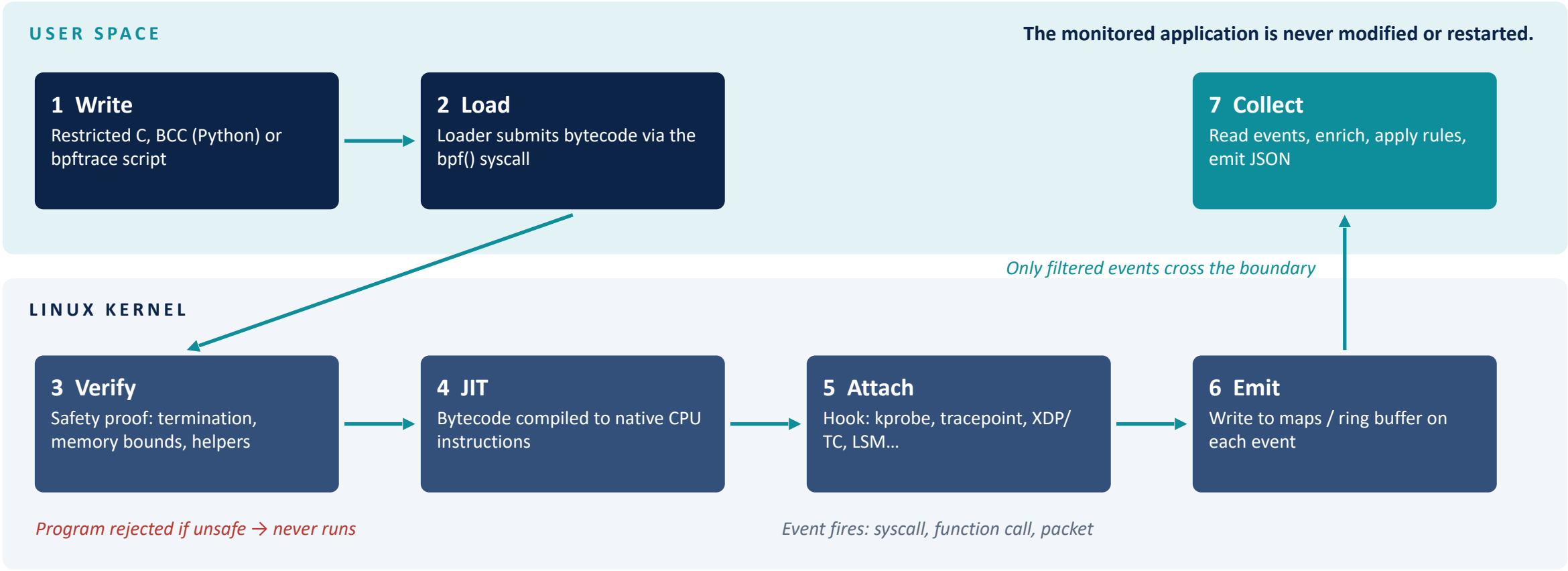
1992
Classic BPF: in-kernel packet filtering (e.g. tcpdump)

2014
Extended BPF (eBPF) merged in Linux 3.18

2015–19
Tooling matures: BCC, bpftrace, Cilium, libbpf

Today
Mainstream for cloud-native networking, observability and runtime security

How an eBPF program runs



Hook points: where eBPF can observe

kprobes / kretprobes

Any kernel function entry/return. Flexible, but tied to kernel internals.

Finance example: tcp_v4_connect → every outbound connection from a payment API

Tracepoints

Stable, documented kernel events. Prefer when available.

Finance example: sched_process_exec → a shell spawned by the banking app

uprobes

User-space functions in binaries and libraries.

Finance example: TLS library calls → visibility before encryption (strict privacy controls)

XDP / TC

Packet path at the network driver or traffic-control layer.

Finance example: Drop scanning or flood traffic before it reaches the stack

LSM (BPF-LSM)

Linux Security Module hooks: observe and enforce.

Finance example: Block unexpected reads of key material or DB files

cgroup / socket

Scope programs to containers, pods or sockets.

Finance example: Identity-aware, per-pod network policy (Cilium)

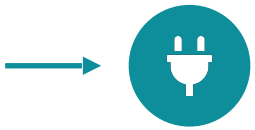
CYBERAID

The eBPF toolchain landscape

Tool	What it is	Best for	Role in CyberAId
BCC	Python/Lua front-end + C kernel code	Rapid prototyping of custom tracers	Used in the CyberAId PoC (TCP tracer, process monitor)
bpftrace	High-level one-liner language	Ad-hoc investigation, incident triage	Analyst toolbox during investigations
libbpf + CO-RE	C / Rust / Go libraries	Portable production agents across kernels	Path to hardened, production-grade tools
Cilium + Hubble	Kubernetes CNI and flow observability	Identity-aware network policy and visibility	Planned for Kubernetes-based pilots
Tetragon / Falco	Runtime security engines with policies	Process, file and network threat detection	Reference designs for rule libraries



Prototype
BCC tracers in a simulated financial stack



Integrate
Wazuh decoders, rules, OpenSearch dashboards



Scale
CO-RE agents and Cilium for Kubernetes pilots

CYBERAID

eBPF compared with traditional monitoring

Approach	Network visibility	Low overhead	Process context	Host stability
Application logs	Weak	Strong	Weak	Strong
auditd (syscall auditing)	Partial	Partial	Strong	Strong
Packet capture / network IDS	Strong	Partial	Weak	Strong
Kernel modules / legacy agents	Strong	Partial	Strong	Weak
eBPF	Strong	Strong	Strong	Strong

Qualitative, indicative comparison for typical deployments.

Rule of thumb

Logs tell you what the application **says**. eBPF tells you what the system actually **did**.

Use eBPF to complement — not replace — existing SIEM, log and network sources.

Note: eBPF sees connections and processes even when traffic is encrypted, because it observes the host rather than the wire.

eBPF monitoring architecture in CyberAId

5 **Wazuh / OpenSearch integration**
Custom decoders, financial rules, graded events, indexing and dashboards

4 **Structured output**
JSON events: timestamp, process, PID/UID, src/dst IP and port, rule, severity

3 **Detection rules**
Suspicious ports, connection bursts, suspicious process execution → INFO…CRITICAL

2 **Kernel hooks (BCC/eBPF)**
TCP connection tracer · process monitor · file-access monitor

1 **Simulated financial environment**
Payment, core banking and auth services (Docker) + normal and attack traffic generators

Design principles

- No changes to monitored applications
- Metadata, not payloads: no transaction content processed
- Minimal performance impact, benchmarked under load
- Open-source stack: BCC, Wazuh, OpenSearch, Cilium

From kernel event to SOC dashboard

eBPF capture → JSON event → Forward to Wazuh → Decoder → Detection rules → OpenSearch → Dashboard

Capture & structure

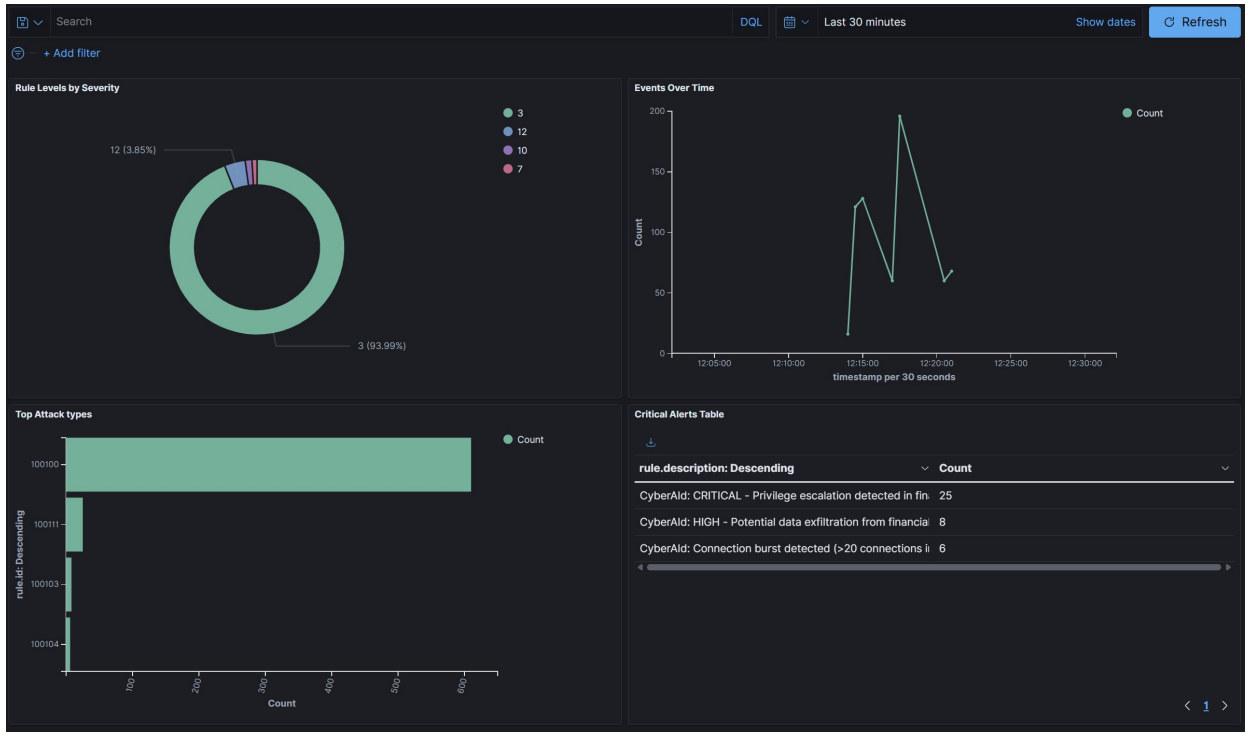
Kernel-level events are turned into structured records with process, user and connection context.

Decode & match

The SIEM parses each record and applies financial-domain detection rules that grade severity.

Index & visualise

Detections are indexed and shown to analysts, and shared with correlation, alerting and forensics.



CyberAId PoC dashboard on simulated traffic: severity split, events over time, top attack types, critical detections.

CYBERAID

Mapping eBPF signals to financial threats

Threat scenario	eBPF signal	Example detection logic	MITRE ATT&CK
Brute force on authentication service	Bursts of TCP accept/connect to the auth port	More than N attempts from one source in T seconds	T1110
Internal reconnaissance / port scanning	Many short-lived connections to distinct ports	Distinct-port count per source above threshold	T1046
Data exfiltration from payment or banking hosts	Outbound connect to a destination not in the profile	Per-service destination allow-list	T1041
Reverse shell / C2 from a compromised service	execve of shell or nc + outbound connect, same PID tree	Combined process + network rule	T1059, T1071
Privilege escalation	Child process with UID 0 from non-root parent	Parent/child UID mismatch	T1068
Access to keys, DB files or credentials	File-open hooks (file-access monitor)	Path watch-list per monitoring profile	T1552

Monitoring profiles: Payment processors · Core banking · Authentication · Trading systems

Deploying in regulated environments & key takeaways

Deployment checklist

- **Kernel readiness:** Check kernel version and BTF support for portable (CO-RE) programs
- **Least privilege:** Use CAP_BPF / CAP_PERFMON where supported instead of full root
- **Performance:** Benchmark CPU and memory overhead under peak transaction load
- **Privacy by design:** Collect metadata only; never transaction payloads (GDPR)
- **Kubernetes:** Complement host probes with Cilium identity-aware policies
- **Governance:** Change control for rules; retain events for DORA reporting and forensics

Key takeaways

1. eBPF gives safe, low-overhead kernel visibility without touching the monitored systems.
2. Value comes from context: process + network + identity, mapped to financial threat profiles.
3. In CyberAId, eBPF monitoring turns kernel events into structured detection outputs for correlation, alerting and forensics.

Further resources

- ebpf.io — introduction and project landscape
- github.com/iovisor/bcc — BCC tools and tutorial
- documentation.wazuh.com — decoders and rules
- docs.cilium.io · attack.mitre.org



CYBERAID

Thank you!

Connect With CyberAID:

LinkedIn: CyberAID

CyberAID has received funding from the DIGITAL EUROPE Programme under grant agreement No. 101249596