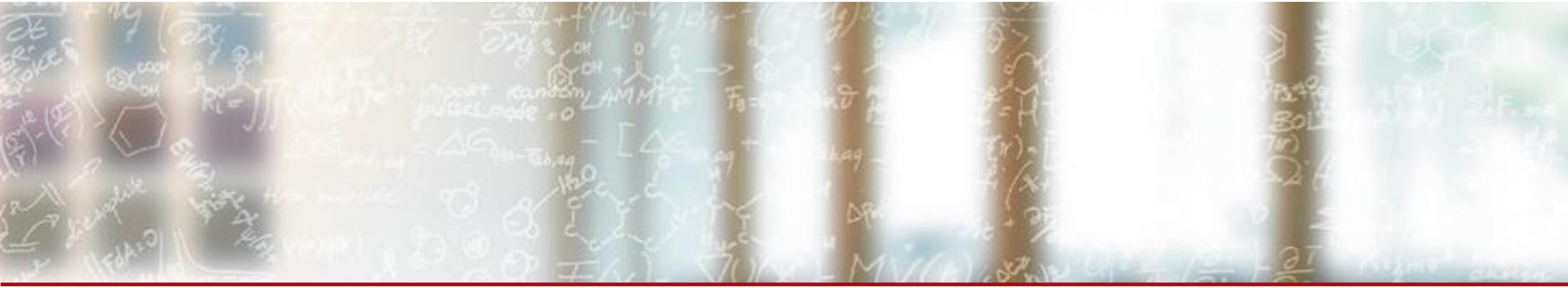
CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich



DenyRoot: An eBPF-based Confinement Mechanism for HPC Environment

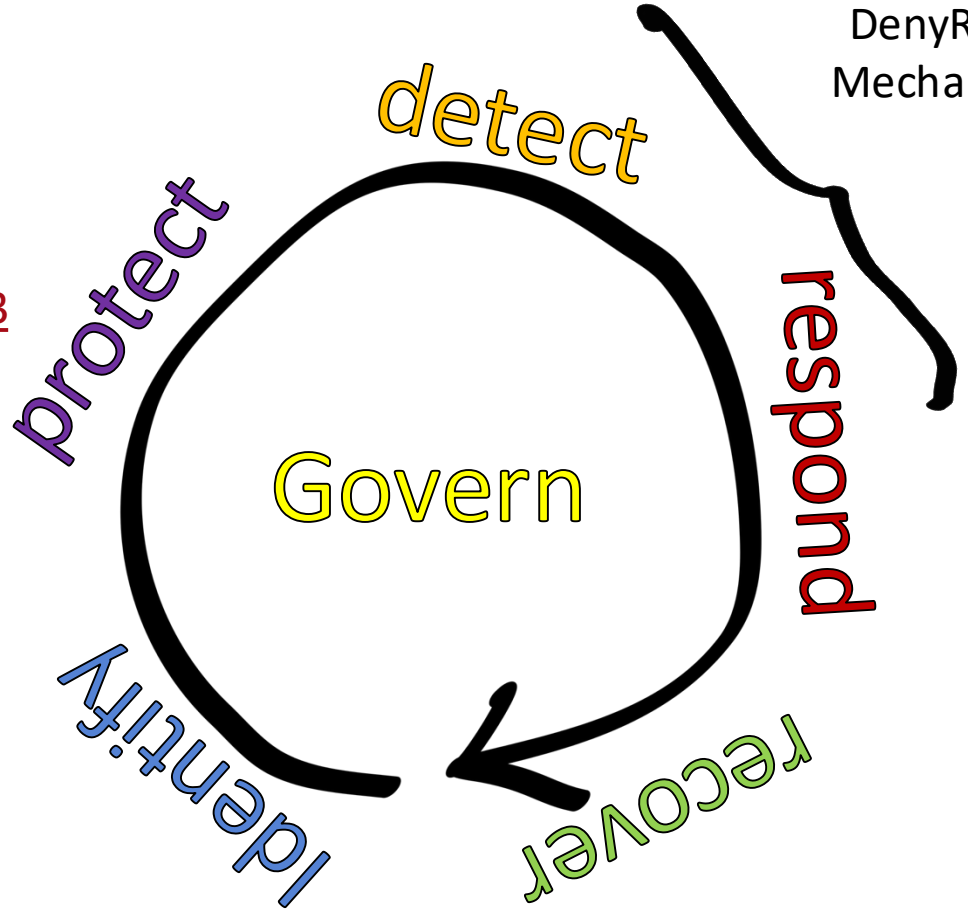
Victor Holanda Rusu, CSCS

April 23rd, 2026

What is the context of this work?

Because the “S” in HPC stands for security

Experimenting With Security Compliance Checking Using Reframe – Victor Holanda https://doi.org/10.1145/3757348.3757351



DenyRoot: An eBPF-based Confinement Mechanism for HPC Environment – Victor Holanda (this presentation)

Operationalizing Security in Dynamic HPC Infrastructures (Lessons from CSCS Alps) – Matteo Basso

Enhancing Infrastructure Management and Datacenter Visibility and Security at CSCS with MarmotGraph – Massimo Benini

DenyRoot: Has a very small performance impact

Exec, and fork syscall stress tests

vCluster	Mode	Benchmark	Baseline (bogo ops/s)	DenyRoot (bogo ops/s)	OH (%)	p	Equivalence (TOST)
Scopi	Enforcing and Monitoring	exec	41.39 ± 1.74	41.39 ± 1.74	+0.0061	0.266	±0.5%, $p<10^{-300}$
Daint	Monitoring only	exec	41.37 ± 1.75	41.37 ± 1.75	-0.0024	0.757	±0.5%, $p<10^{-300}$
Scopi	Enforcing and Monitoring	fork	111114.78 ± 61.60	111114.79 ± 61.61	+0.0001	0.651	±0.5%, $p<10^{-300}$
Daint	Monitoring only	fork	24954.21 ± 5175.38	24954.21 ± 5175.41	+0.00006	0.944	±0.5%, $p<10^{-300}$

$$OH = 100 \times \frac{T_{with} - T_{without}}{T_{without}}$$

DenyRoot: Has a very small performance impact

GROMACS strong-scaling results

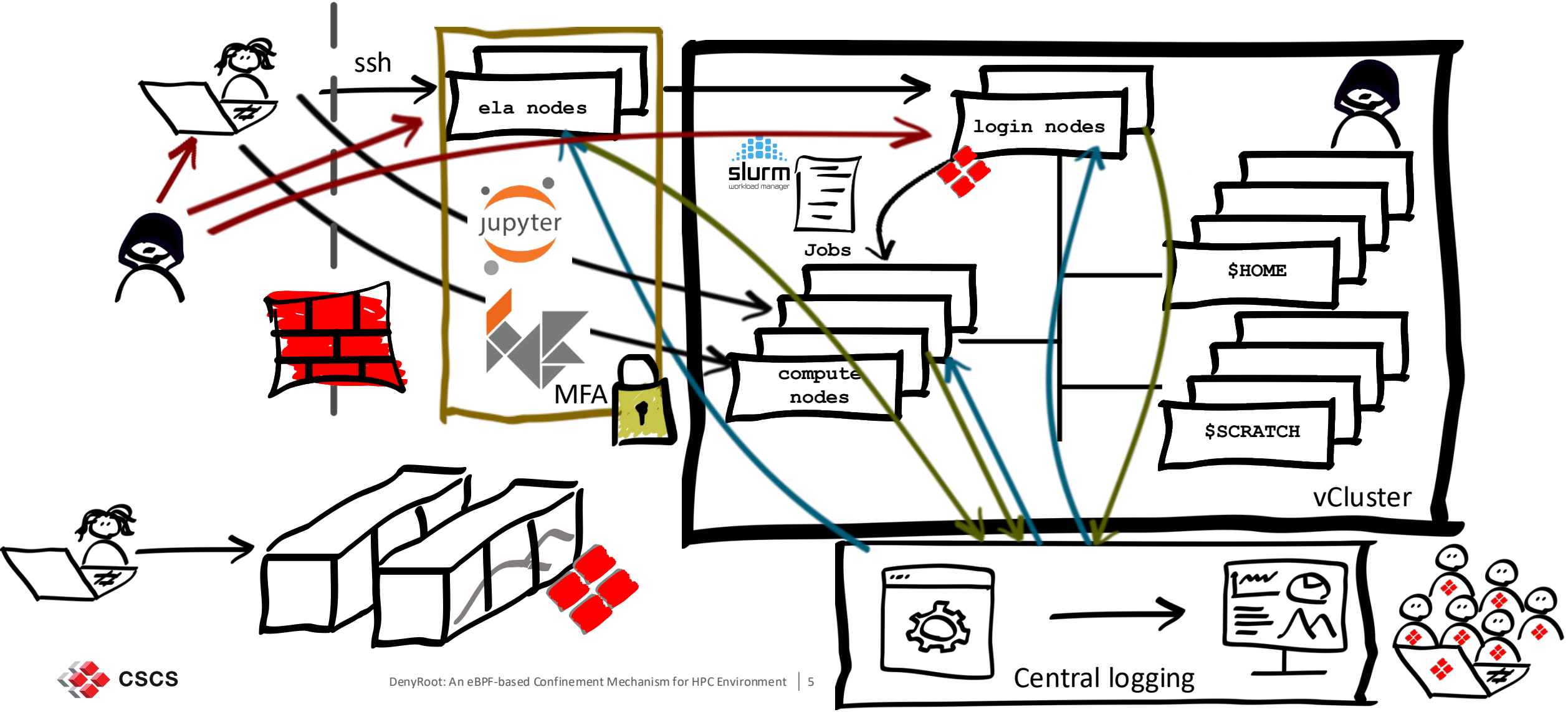
Benchmark	Accelerator	Nodes	n	Baseline (ns/day)	DenyRoot (ns/day)	OH (%)	p_{Holm}
hEGFRDimer	CPU	1	12	19.8250	19.8282	-0.0156	1.000
hEGFRDimerPair	GPU	8	18	29.0617	29.0673	-0.0192	0.3896
hEGFRtetramerPair	CPU	2	12	6.7317	6.7343	-0.0396	1.000
hEGFRtetramerPair	GPU	2	15	16.5300	16.5341	-0.0260	1.000

$$OH = 100 \times \frac{T_{with} - T_{without}}{T_{without}}$$

These are the worst performance results!

A Bird's-Eye View

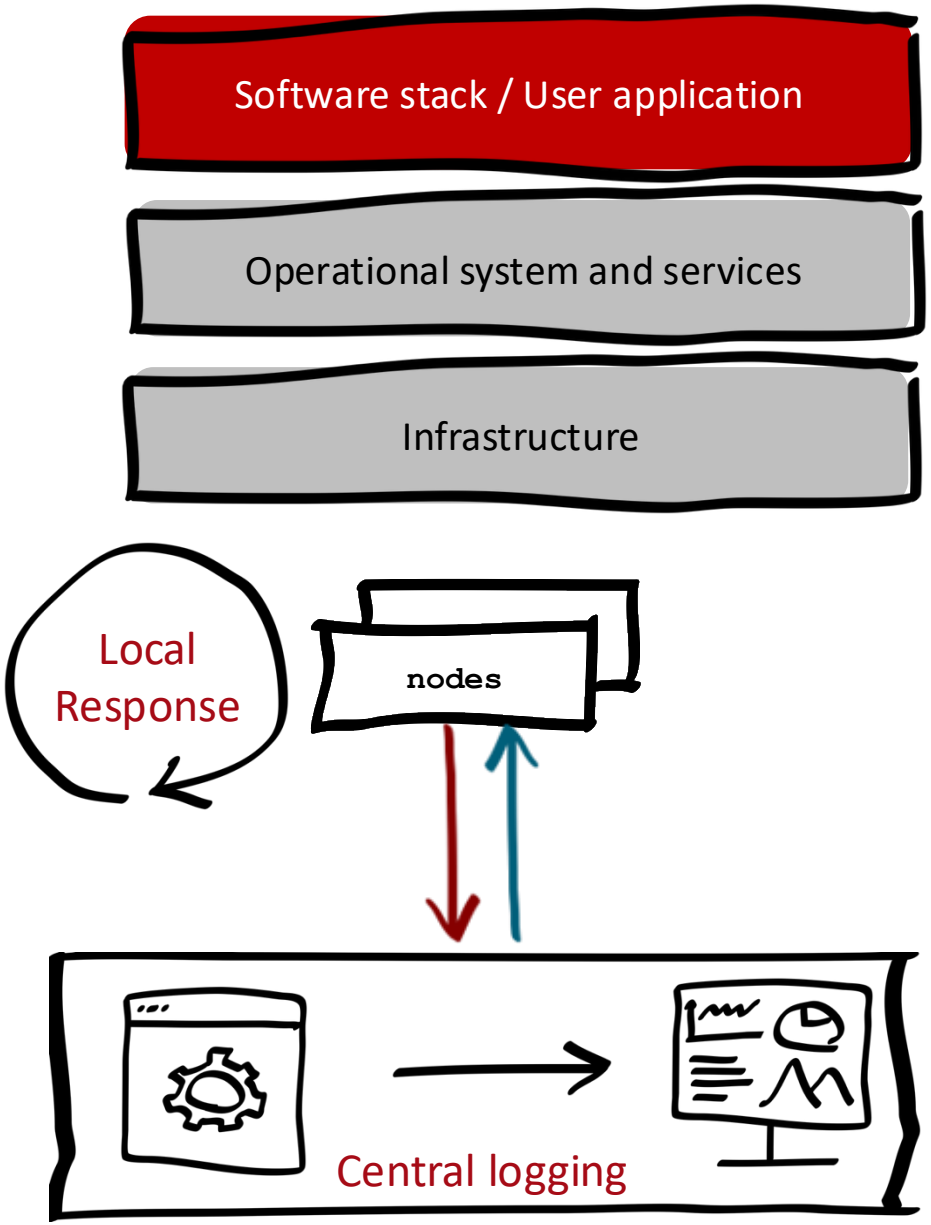
A Visual Blueprint of HPC Access



How does one detects and responds to threats?

The 50’000 feet view of the process

- Centralised response
- Local response
- Infrastructure monitoring
  - Several tool and standards
  - Depend on the vendor or OEM
- Operational system and services
  - System logs
  - Services logs
- Software stack and user application
  - Auditd
  - eBPF



What are the challenges of enabling auditd?

Because we still care about the P in HPC

- Auditd slows down the scientific software performance, especially at scale
- Per-CPU buffer contention & lock spinning
  - The kernel needs to acquire the per-CPU spinlock to push an entry
- Disk I/O overhead
  - Audit logs are written to a file in the system
- Audit buffer overflow / backlog limit
  - When the per-CPU audit ring buffer is filled, the kernel blocks the originating syscall until auditd drains the buffer
- Network forwarding overhead
  - Every audit event is transmitted over the network, adding latency and consuming bandwidth that can interfere with MPI traffic
- Rule-based filter
  - E.g. auditd forces the kernel to evaluate every syscall against a given rule filter, adding per-syscall overhead.

What's ideal?

Has no locks or fewer locks

Does not write events to disk or at least avoids as much as possible

Decrease the number of events

Increase the signal-to-noise ratio

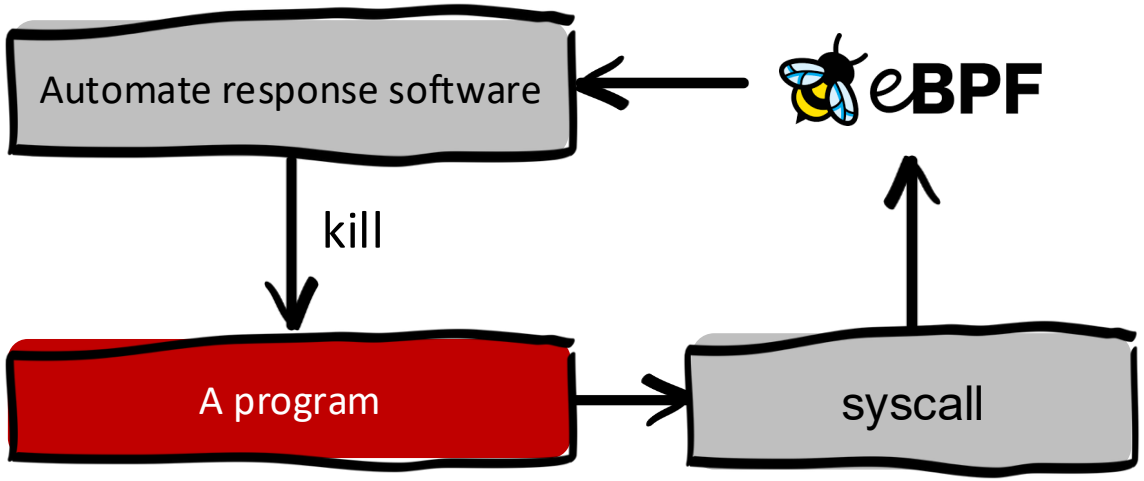
Relies on a different mechanism (callback based?)

We chose to use eBPF

Is this a new idea?

The challenges of integrating eBPF in HPC

- HPC compute nodes process thousands of syscalls per second
- The noise-to-signal ratio is low
- Our preliminary results point to larger performance impact (up to 11%)



Falco

sysdig

We need to monitor something other than syscalls

tetragon

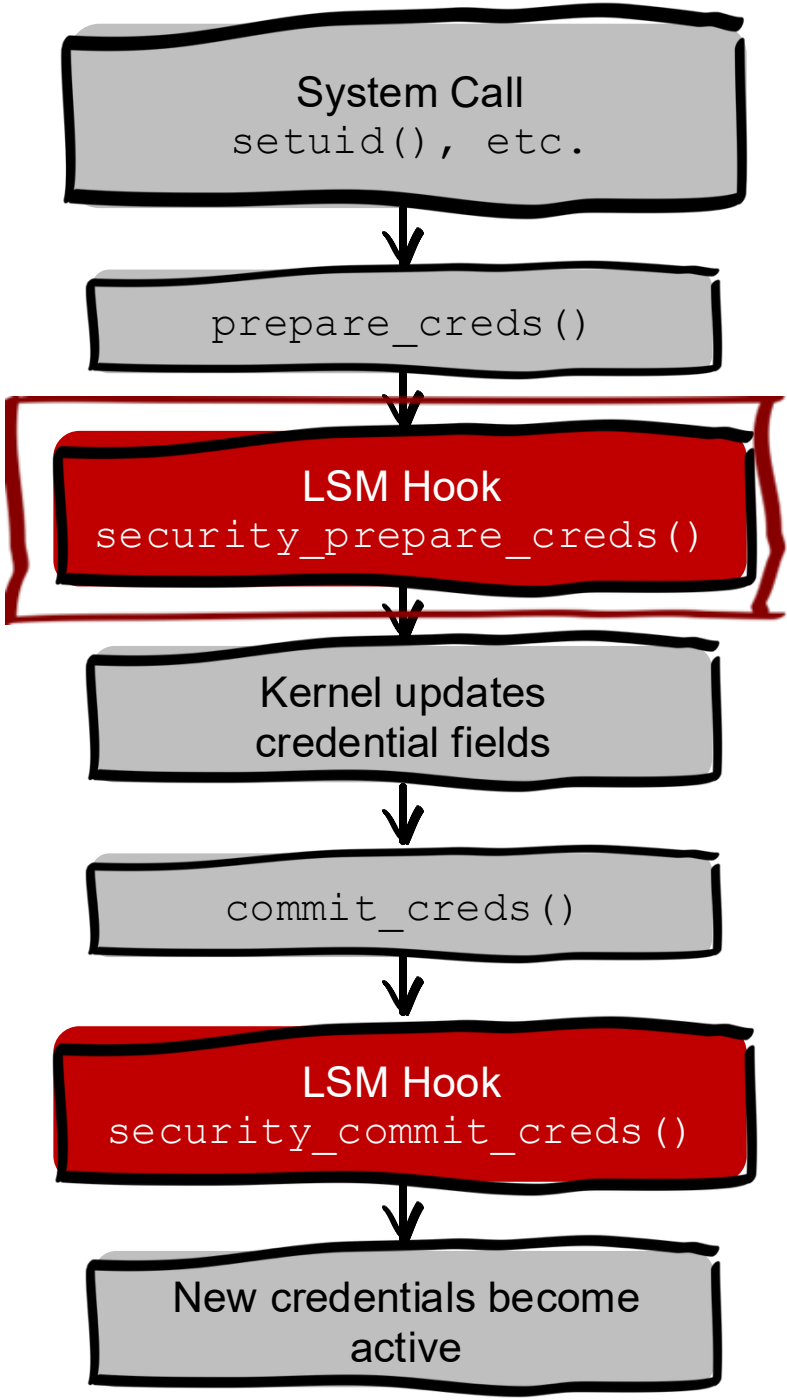
aqua tracee

How does the Linux kernel transitions credentials?

A call tree view

- One requests a new process
  - Every process requires credentials
- Linux Security Module hook at time of request
  - E.g. used by SELinux
- Kernel updates the process credential with proper credentials
- Commits the credentials to the process
- Linux Security Module hook at time of commit
- Process has new credentials

We hook here!



What does it do?

The detection and response machine

DenyRoot enabled and SELinux disabled

```
[myuser@daint ~]$ getenforce
Permissive

[myuser@daint ~]$ sudo -i
sudo: PERM_ROOT: setresuid(0, -1, -1): Cannot allocate memory
sudo: error initializing audit plugin sudoers_audit
```

Protect against user impersonation

```
[myuser@daint ~]$ sudo -i -u anotheruser
sudo: PERM_ROOT: setresuid(0, -1, -1): Cannot allocate memory
sudo: error initializing audit plugin sudoers_audit
```

An additional security control

Incremental security

DenyRoot and SELinux enabled

```
[myuser@daint ~]$ getenforce
Enforcing

[myuser@daint ~]$ sudo -i
sudo: PERM_ROOT: setresuid(0, -1, -1): Cannot allocate memory
sudo: error initializing audit plugin sudoers_audit
```

DenyRoot disabled and SELinux enabled

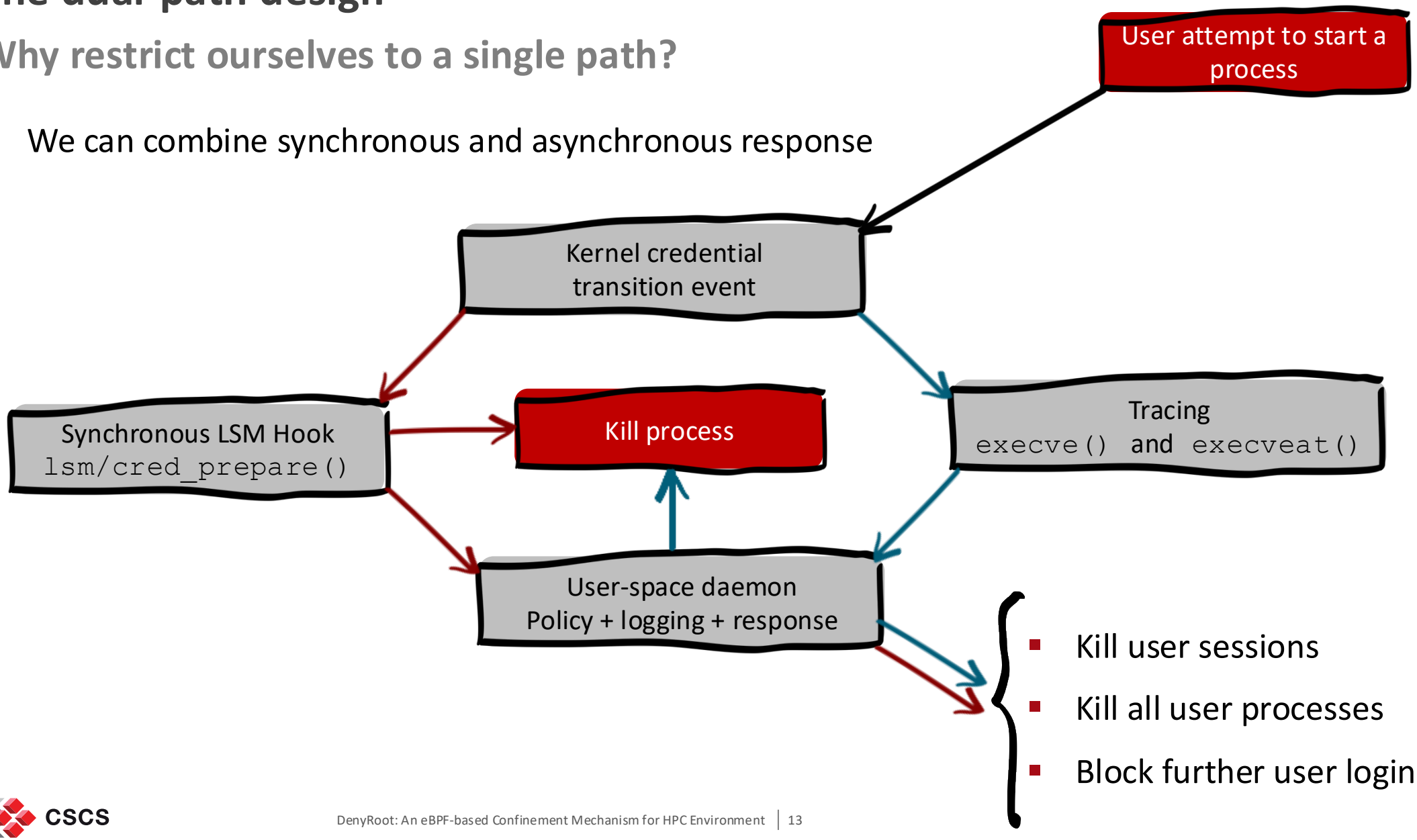
```
[myuser@daint ~]$ getenforce
Enforcing

[myuser@daint ~]$ sudo -i
sudo: PERM_SUDOERS: setresuid(-1, 1, -1): Operation not permitted
sudo: no valid sudoers sources found, quitting
sudo: setresuid() [0, 0, 0] -> [1000, -1, -1]: Operation not permitted
sudo: error initializing audit plugin sudoers_audit
```

The dual-path design

Why restrict ourselves to a single path?

- We can combine synchronous and asynchronous response



What are the limitations?

This is not a silver bullet

- It does not prevent privilege escalation via vulnerabilities unrelated to credential transitions
  - E.g. logic errors in privileged kernel subsystems or bugs that directly manipulate kernel memory
- It does not defend against kernel-level exploits that allow execution of arbitrary code in kernel space
  - The attacker could modify kernel data structures, disable security mechanisms, or bypass DenyRoot entirely
- It does not prevent exploit of binaries already authorized to execute with elevated privileges

What's next?

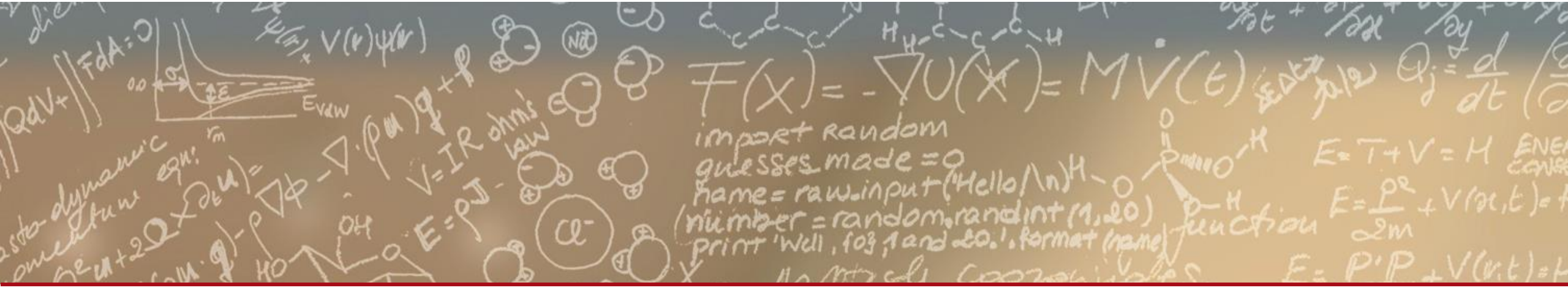
There are a lot more to do

- Capabilities-transition control
- User namespace-transition control
- Expand our comparative investigation against general-purpose eBPF runtime security tools under standardized policies, event filters, and output pipelines

DenyRoot: What is it?

The Take Home Message

- DenyRoot prevents unauthorized UID/GID transitions *at the kernel credential boundary* with negligible overhead (< 0.25%)
- Can be deployed standalone or as an incremental/additional security control together with other Linux Security Modules, e.g. SELinux and AppArmor
- It is based on eBPF, a Linux Kernel technology
- It can be deployed in two modes:
  - Enforcing
  - Monitoring



Thank you for your attention.

Questions?

What's eBPF?

A brief intro? Nope! It is more like an ultra fast, supersonic one!

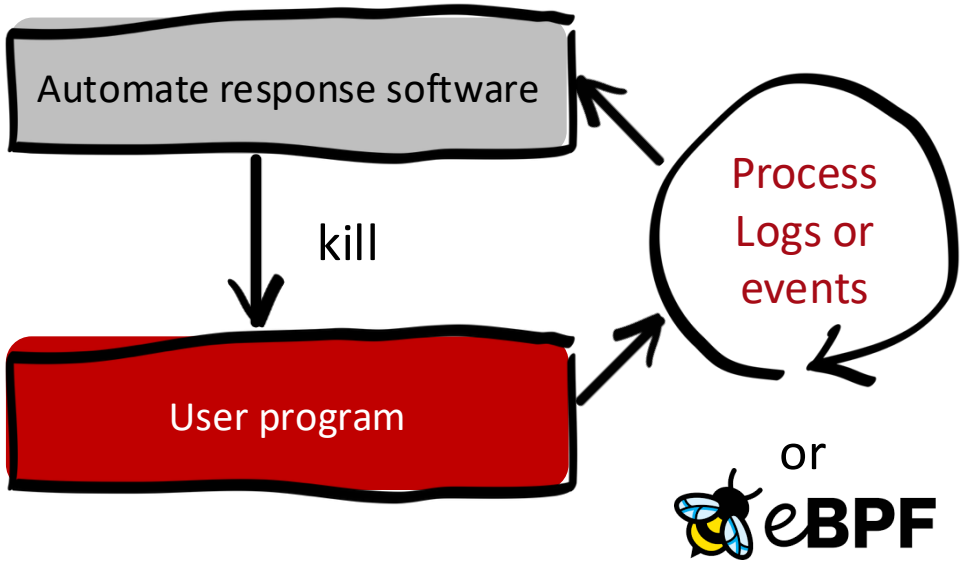
- eBPF is a Linux kernel technology
- eBPF allows to run **sandboxed programs in the Linux kernel context**
  - Programs are executed within the Kernel's Virtual Machine
- It aims to **safely and efficiently extend the kernel capabilities** without loading kernel modules or kernel source code changes
  - The programs are **restricted to a subset of operations** (e.g. no memory allocation, no sleeping)
- **It is event-driven** and works when the kernel passes certain hook points
- It can be used to **trace applications** from the kernel point of view
  - Perfect for tracing **containerised applications** (share the kernel with the host system)
- There are several development toolchains
  - libbpf (C/C++) – CORE (**Compile Once Run Everywhere**) approach
  - bcc (python)
  - Bpftrace (standalone language similar to awk)
  - ebpf Go lib

How do automated response work?

The two types of control

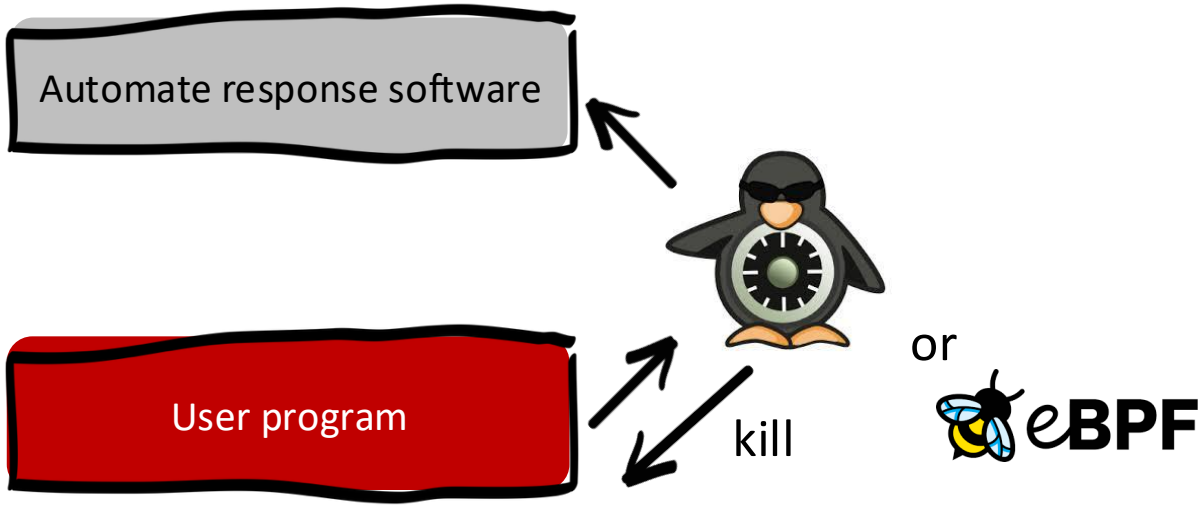
Asynchronous control

- Attach tracepoints, probes, or hooks to kernel functions
- Process logs or system events



Synchronous control

- Use the LSM hooks
- Same behavior as SELinux and AppArmor



DenyRoot algorithm

The logic behind the machine

Require: Current task identity (`uid` and `gid`), executable path `p`, policy sets A_u (allow uids), A_g (allow gids), D_u (block uids), and authorized service paths `S`
Ensure: `DENY` or `ALLOW`

```
if uid eq 0 then
  return ALLOW
else if uid ∈ D_u then
  return DENY
end if
if (uid ∉ A_u) and (gid ∉ A_g) then
  return DENY
end if
if p ∉ S then
  return DENY
end if
return ALLOW
```

Require: Event `e`, configuration: `ipsmode`, operation list `Ops`, notification sinks `Sinks`

```
if IsPolicyViolation(e) then
  EmitEvent(e, Sinks)
  if ipsmode is true then
    for all operation op ∈ Ops do
      Execute(op, e)
    end for
  end if
end if
```