# Quantifying and Reducing Kernel Attack Surface

Anil Kurmus
kur@zurich.ibm.com - ak@kernel.build
@kurmus
IBM Research – Zurich

IBM Research

“ If you don’t have a dog, your neighbor can’t poison it. ”

Sergey Nikitin, *If You Don’t Have an Aunt* (Russian song)

# The Linux kernel: All you (n)ever wanted

# The Linux kernel: All you (n)ever wanted

RDS protocol

# The Linux kernel: All you (n)ever wanted

# The Linux kernel: All you (n)ever wanted

RDS protocol

Perf events

/proc/pid/mem

# The Linux kernel: All you (n)ever wanted


RDS protocol

Perf events

/proc/pid/mem

Berkley Packet Filter (BPF)

# The Linux kernel: All you (n)ever wanted

RDS protocol
CVE-2010-3904

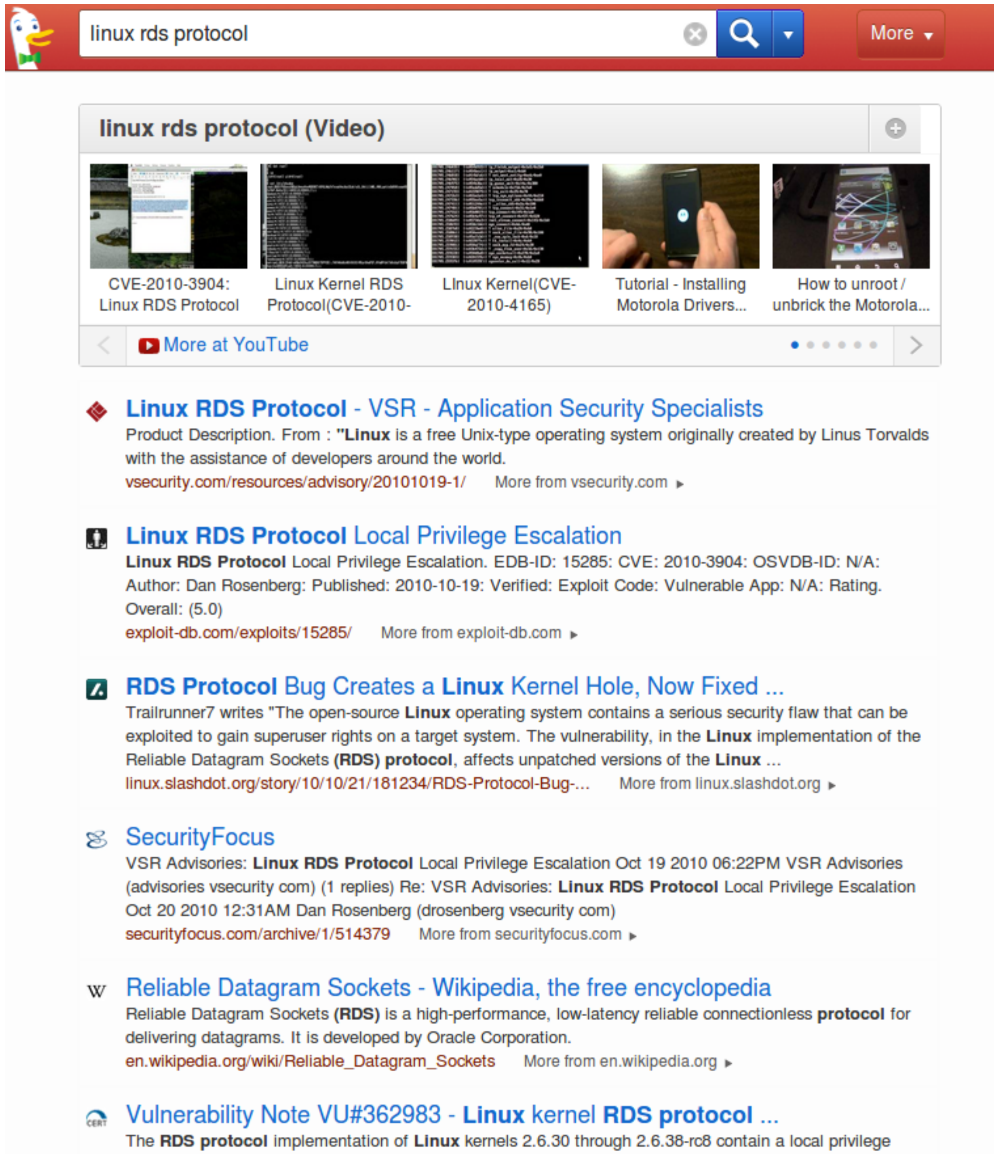
Perf events
CVE-2013-2094

/proc/pid/mem
CVE-2012-0056

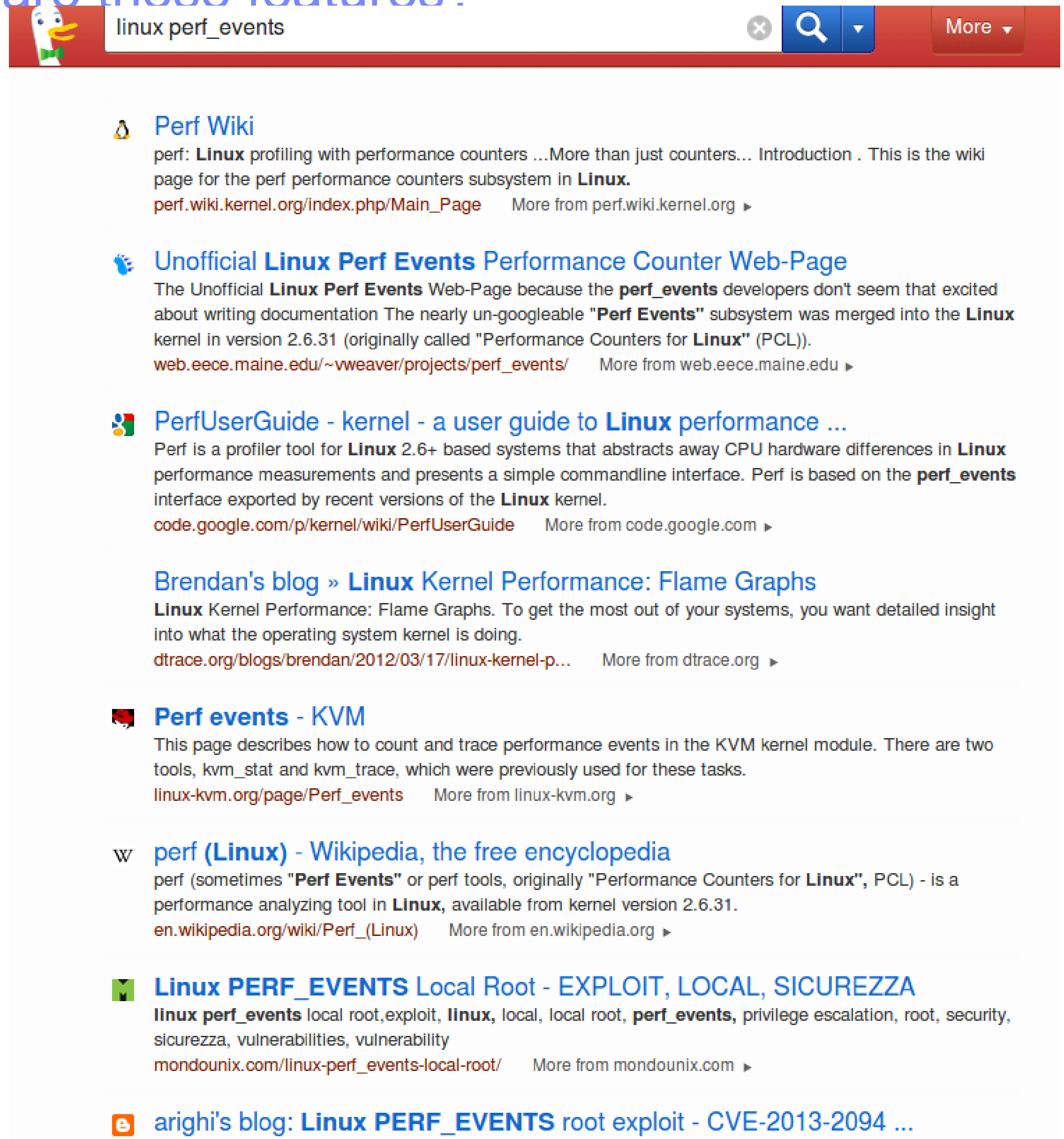
BPF
CVE-2010-4158

# How popular are those features?

# How popular are those features?



# How popular are those features?



linux perf_events

More

**Perf Wiki**
perf: **Linux** profiling with performance counters ...More than just counters... Introduction . This is the wiki page for the perf performance counters subsystem in **Linux.**
perf.wiki.kernel.org/index.php/Main_Page More from perf.wiki.kernel.org

**Unofficial Linux Perf Events Performance Counter Web-Page**
The Unofficial **Linux Perf Events** Web-Page because the **perf_events** developers don't seem that excited about writing documentation The nearly un-googleable "**Perf Events"** subsystem was merged into the **Linux** kernel in version 2.6.31 (originally called "Performance Counters for **Linux"** (PCL)).
web.eece.maine.edu/~vweaver/projects/perf_events/ More from web.eece.maine.edu

**PerfUserGuide - kernel - a user guide to Linux performance ...**
Perf is a profiler tool for **Linux** 2.6+ based systems that abstracts away CPU hardware differences in **Linux** performance measurements and presents a simple commandline interface. Perf is based on the **perf_events** interface exported by recent versions of the **Linux** kernel.
code.google.com/p/kernel/wiki/PerfUserGuide More from code.google.com

**Brendan's blog » Linux Kernel Performance: Flame Graphs**
**Linux** Kernel Performance: Flame Graphs. To get the most out of your systems, you want detailed insight into what the operating system kernel is doing.
dtrace.org/blogs/brendan/2012/03/17/linux-kernel-p... More from dtrace.org

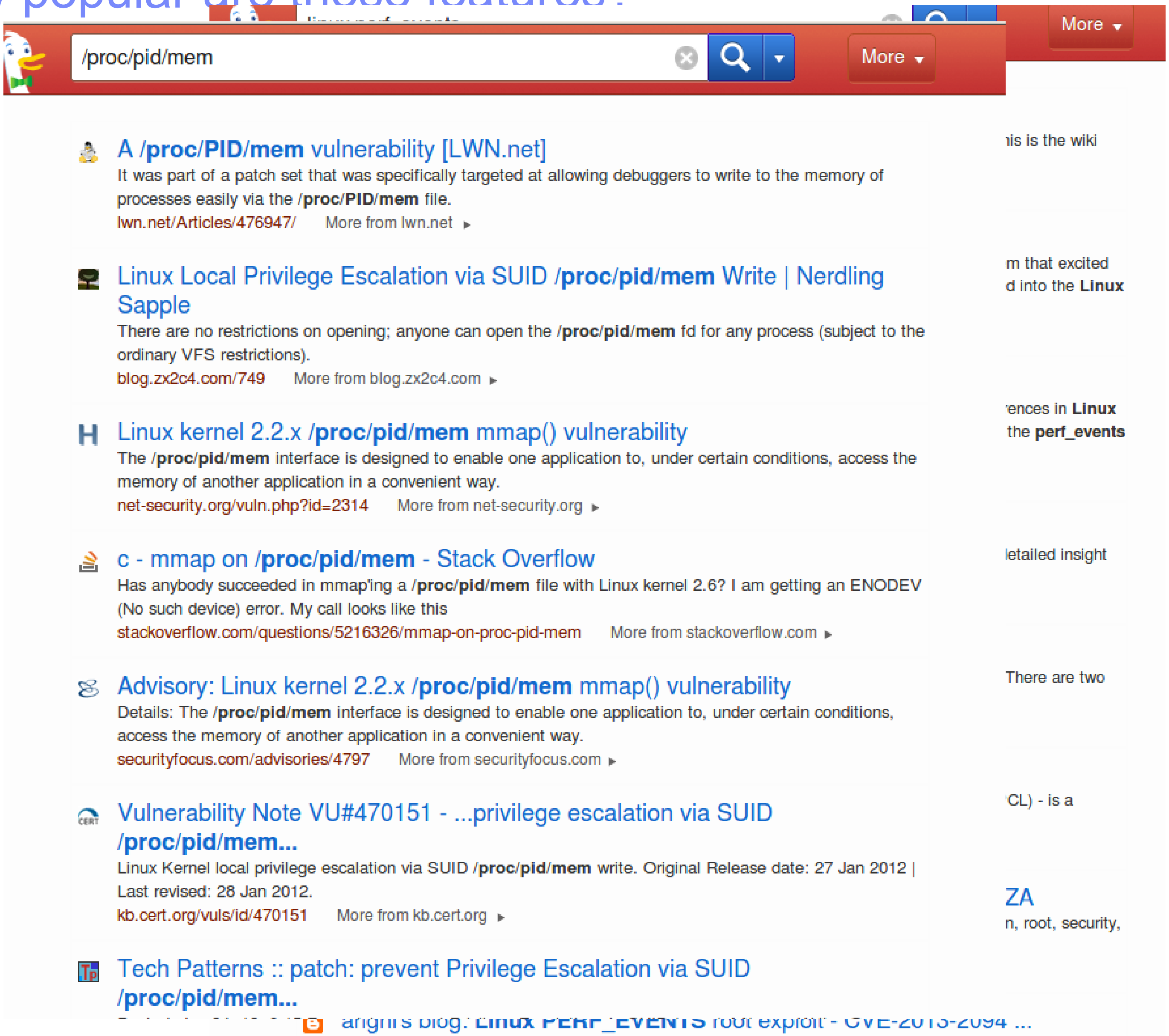
**Perf events - KVM**
This page describes how to count and trace performance events in the KVM kernel module. There are two tools, kvm_stat and kvm_trace, which were previously used for these tasks.
linux-kvm.org/page/Perf_events More from linux-kvm.org

**perf (Linux) - Wikipedia, the free encyclopedia**
perf (sometimes "**Perf Events**" or perf tools, originally "Performance Counters for **Linux",** PCL) - is a performance analyzing tool in **Linux,** available from kernel version 2.6.31.
en.wikipedia.org/wiki/Perf_(Linux) More from en.wikipedia.org

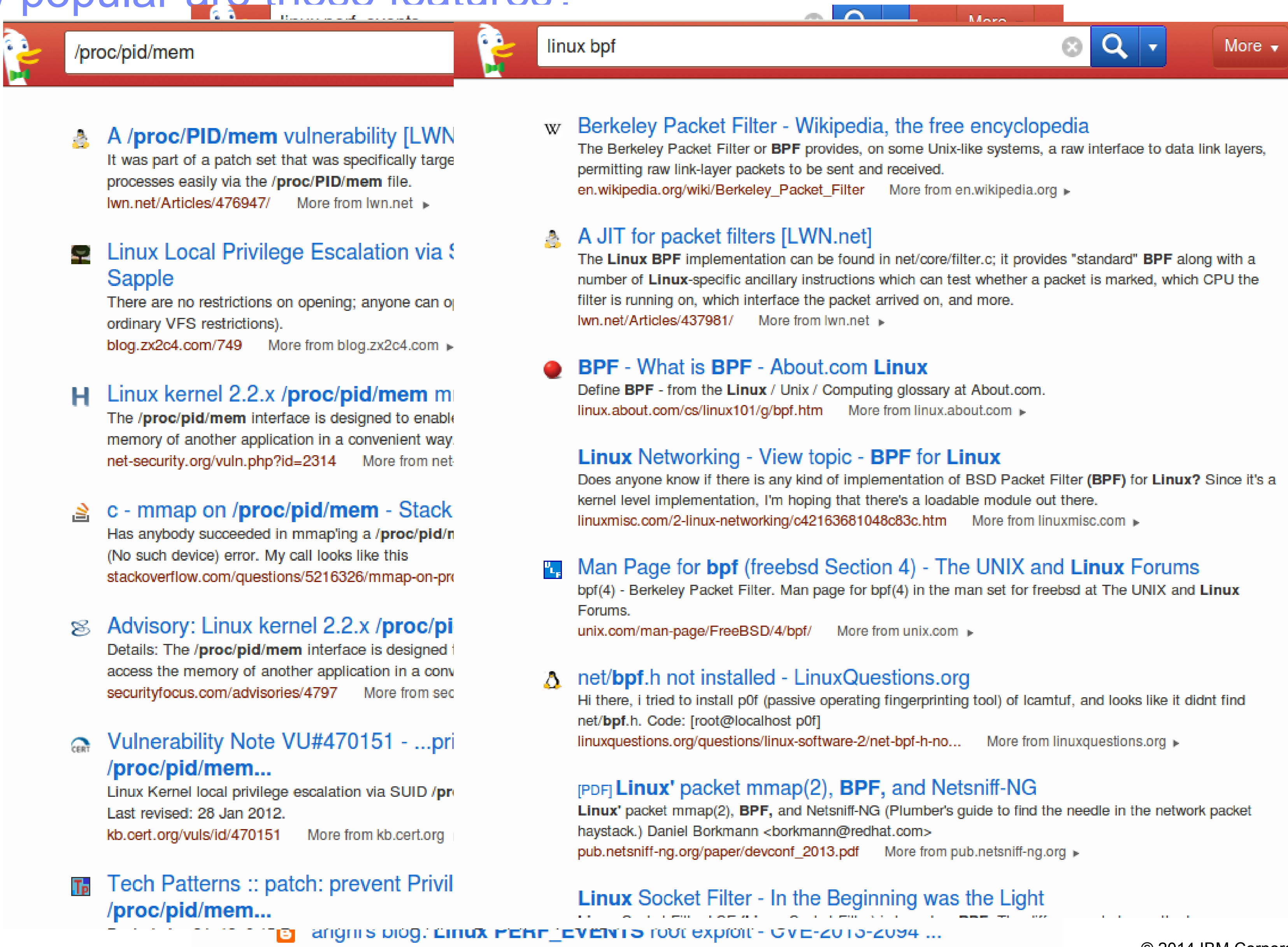
**Linux PERF_EVENTS Local Root - EXPLOIT, LOCAL, SICUREZZA**
**linux perf_events** local root,exploit, **linux,** local, local root, **perf_events,** privilege escalation, root, security, sicurezza, vulnerabilities, vulnerability
mondounix.com/linux-perf_events-local-root/ More from mondounix.com

**arighi's blog: Linux PERF_EVENTS root exploit - CVE-2013-2094 ...**
**Linux PERF_EVENTS** root exploit - CVE-2013-2094 (quick way to fix it) Recently a quite critical flaw has

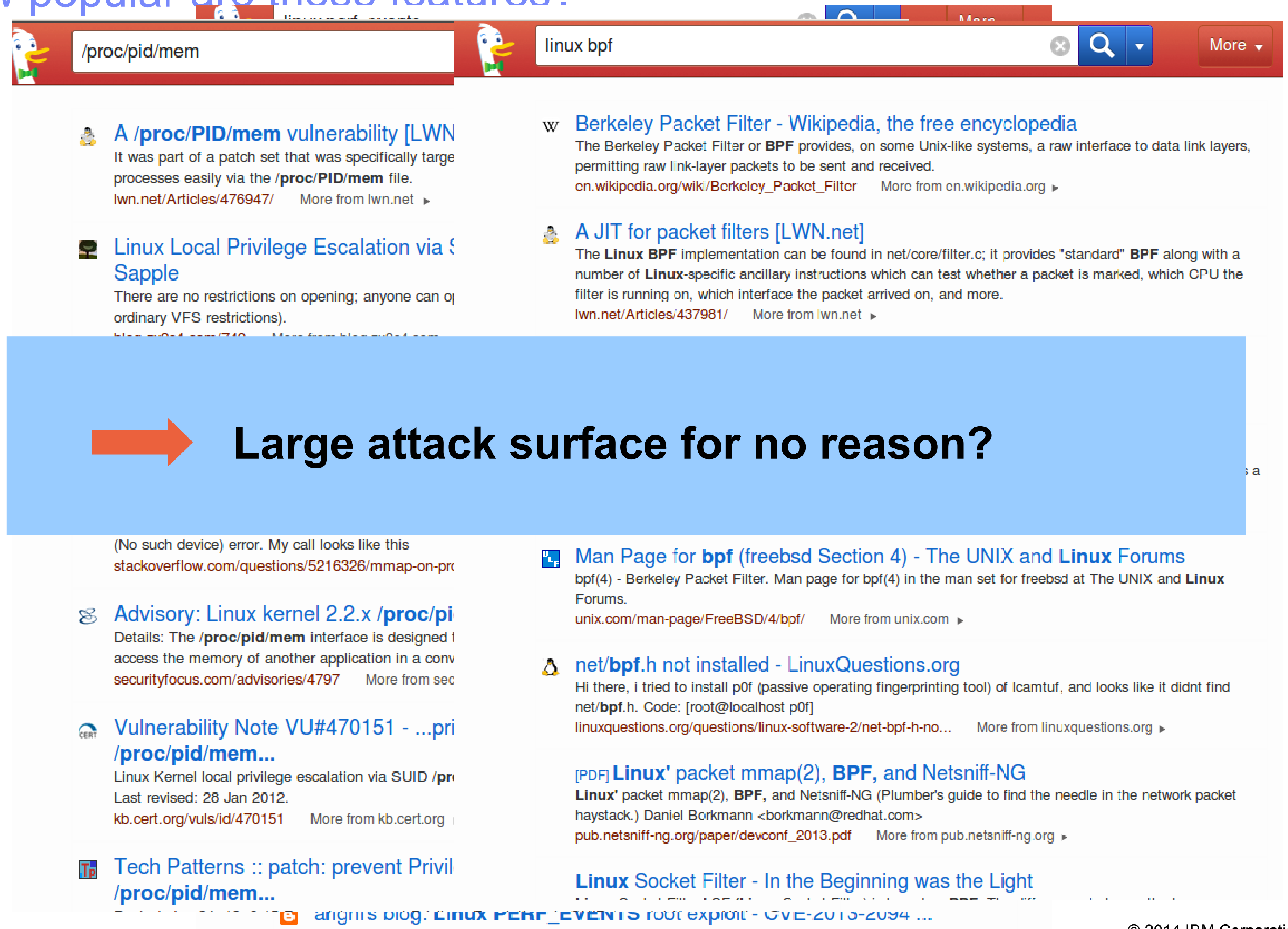
# How popular are those features?



# How popular are those features?



# How popular are those features?




**Large attack surface for no reason?**

# Research questions (1/2)

**Q1: Is it possible to precisely define the kernel attack surface? How can it be measured?**

# Research questions (2/2)

**Q2: Can we develop kernel protection mechanisms whose attack surface reduction is quantifiable? To what extent can these mechanisms be applied to commodity OSes in practice?**

# This talk

**P1: Kernel Attack Surface Quantification (NDSS'13)**

# This talk

**P1: Kernel Attack Surface Quantification (NDSS'13)**

**P2: Compile-time Kernel Tailoring (HotDep'13, NDSS'13)**

# This talk

**P1: Kernel Attack Surface Quantification (NDSS'13)**

**P2: Compile-time Kernel Tailoring (HotDep'13, NDSS'13)**

**P3: Run-time Kernel Trimming (Eurosec'11, DIMVA'14, CCS'14)**

# Measuring Kernel Attack Surface

- [NDSS'13] Anil Kurmus, Reinhard Tartler, Daniela Dorneanu, Bernhard Heinloth, Valentin Rothberg, Andreas Ruprecht, Wolfgang Schröder-Preikschat, Daniel Lohmann and Rüdiger Kapitza. "**Attack Surface Metrics and Automated Compile-Time OS Kernel Tailoring.**" In: Proceedings of the 20th Network and Distributed System Security Symposium. 2013.

  https://www.ibr.cs.tu-bs.de/users/kurmus/papers/kurmus-ndss13.pdf

- [DIMVA'14] Anil Kurmus, Sergej Dechand, and Ruediger Kapitza. "**Quantifiable Run-time Kernel Attack Surface Reduction**". In: Proceedings of the 10th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA’14). 2014.

  https://www.ibr.cs.tu-bs.de/users/kurmus/papers/kurmus-dimva14.pdf

# Existing approaches and limitations

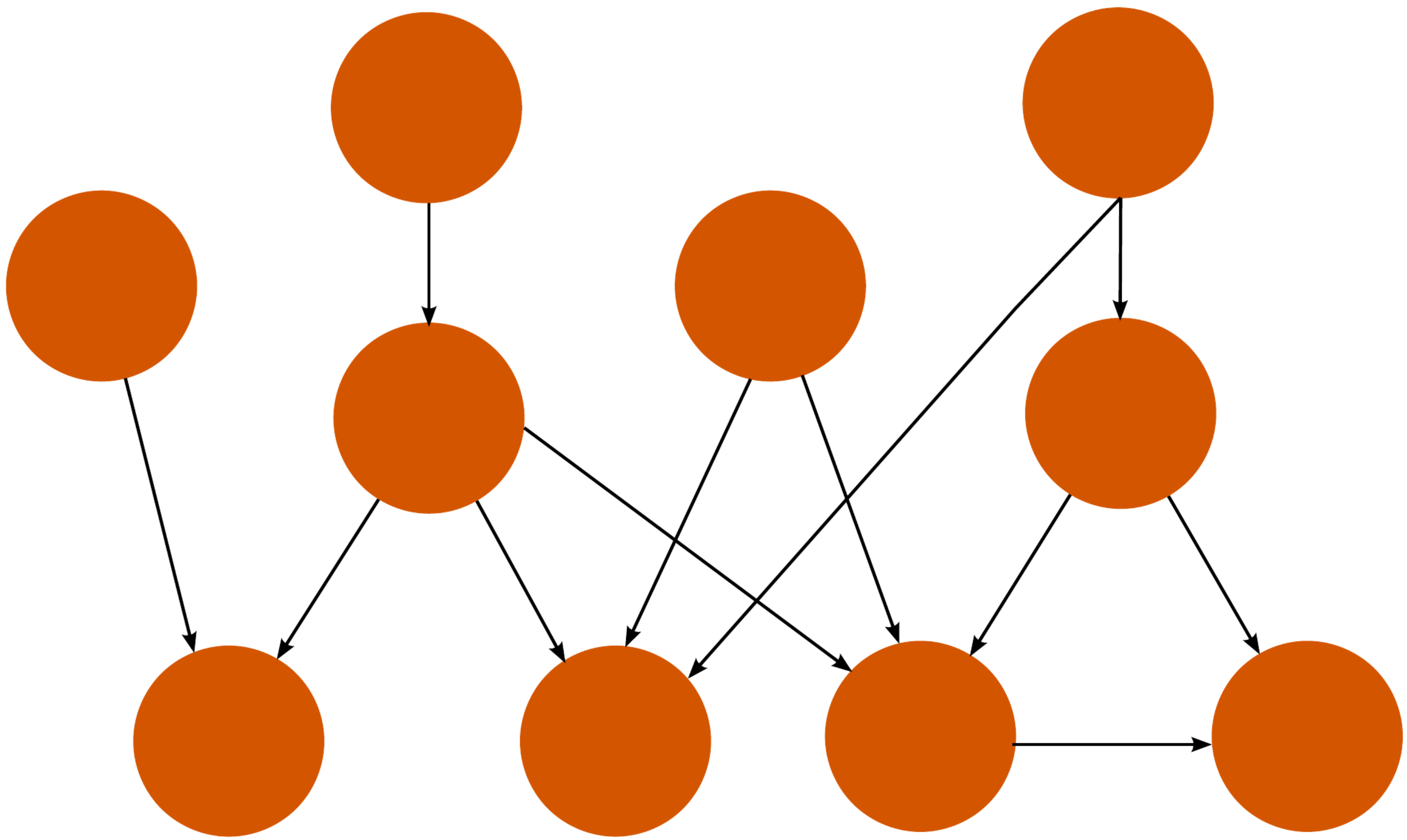
- Typically in OS research: measure TCB size in source lines of code.
  – Fiasco 15K SLOC; Minix 3 4K SLOC; Flicker 250 SLOC
  – Linux 3.0 10M SLOC;
- However:
  – Source files that are not compiled? Configuration-dependent code?
  – Loadable kernel modules (LKMs)? On-demand loadable kernel modules?
  – Code that is not reachable from the system call interface? Initialization code?
  – Code that is only reachable by privileged processes?

# General Idea

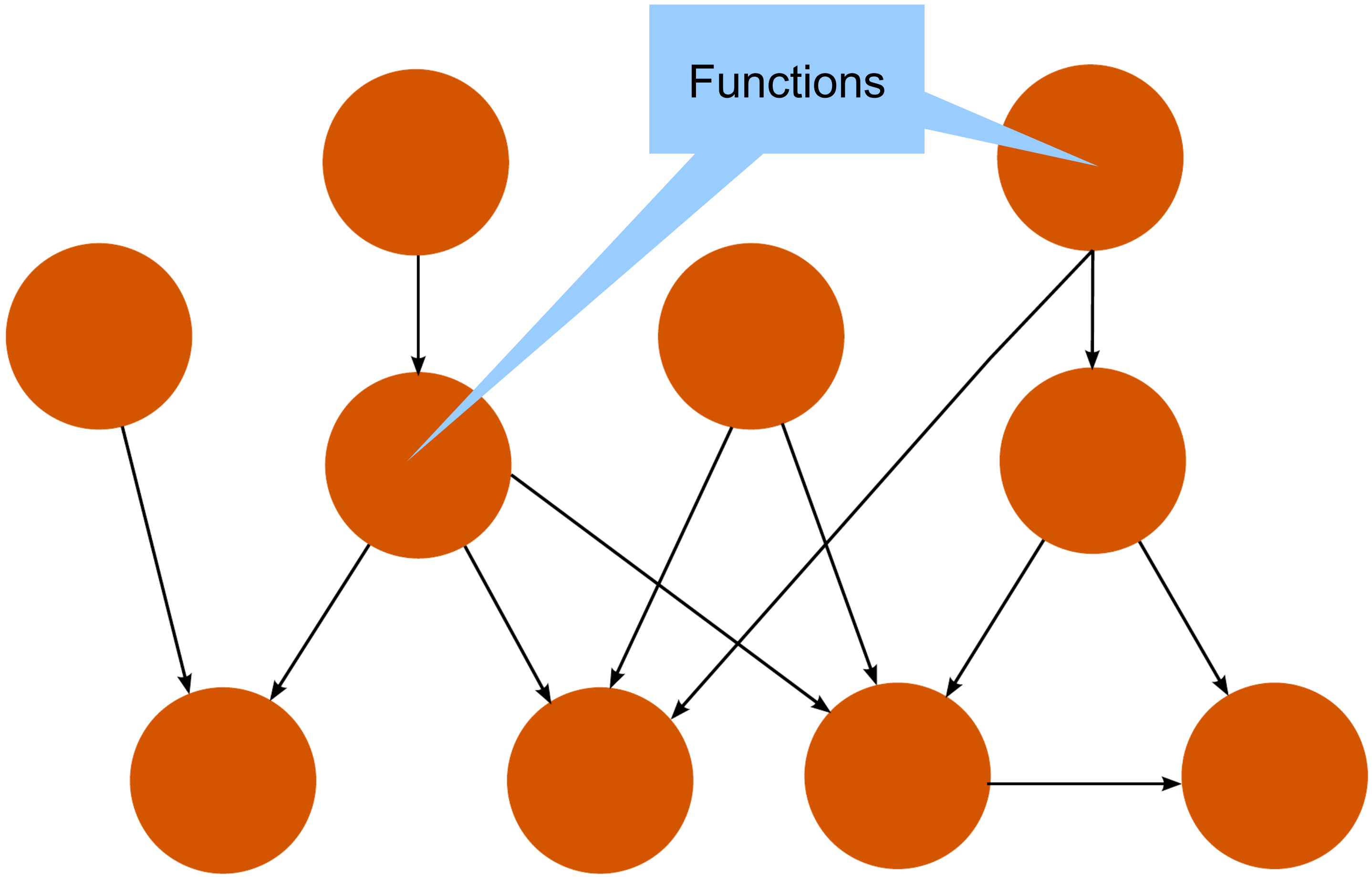
- Attack surface ~= attacker-reachable code
  - Idea: use reachability over kernel call graph
  - Assumptions on the attacker and kernel? (**security model**)

- Measurements: code quality metrics
  - SLOCs, CVEs, ...

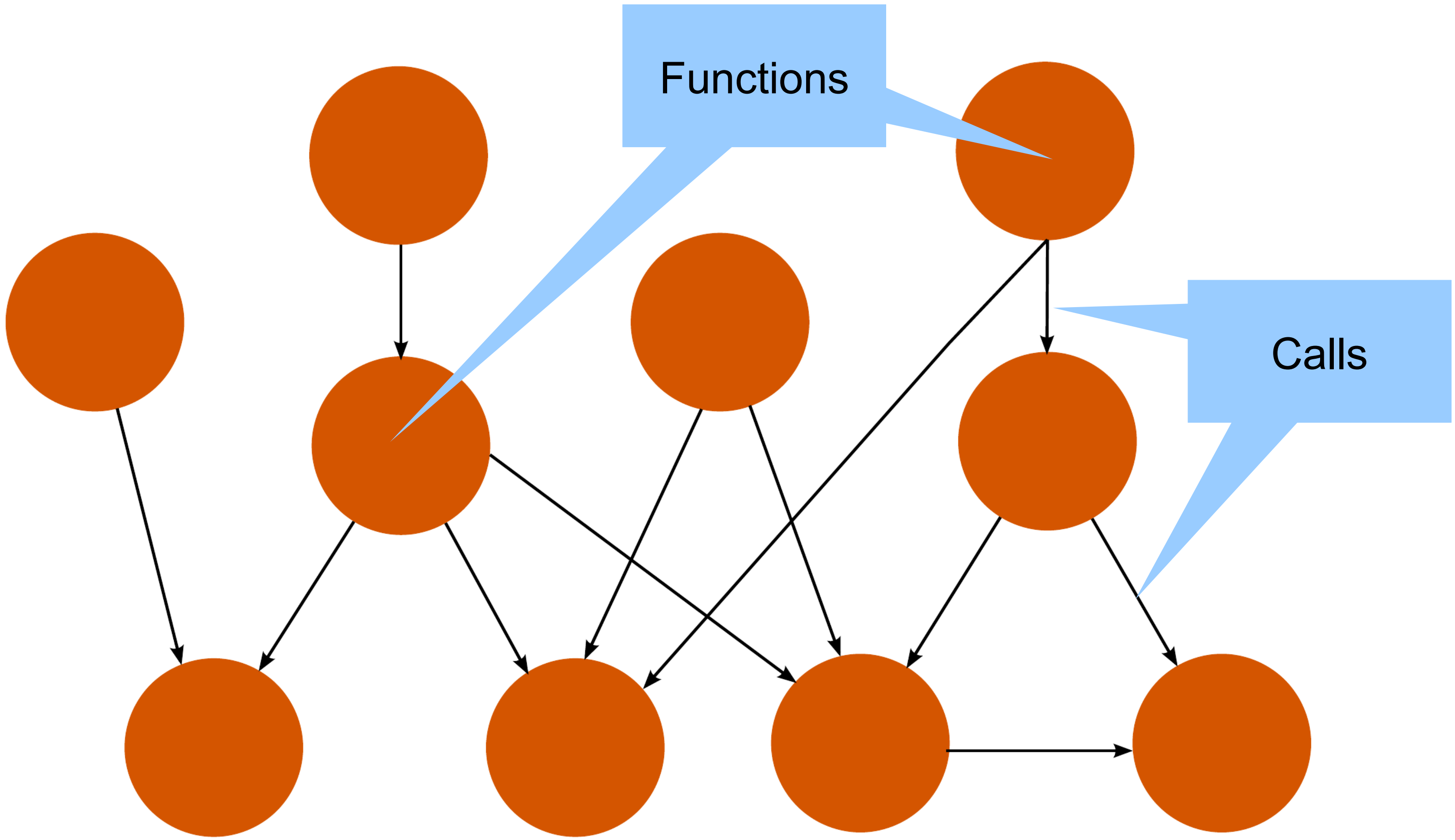
# Obtaining the attack surface: an example



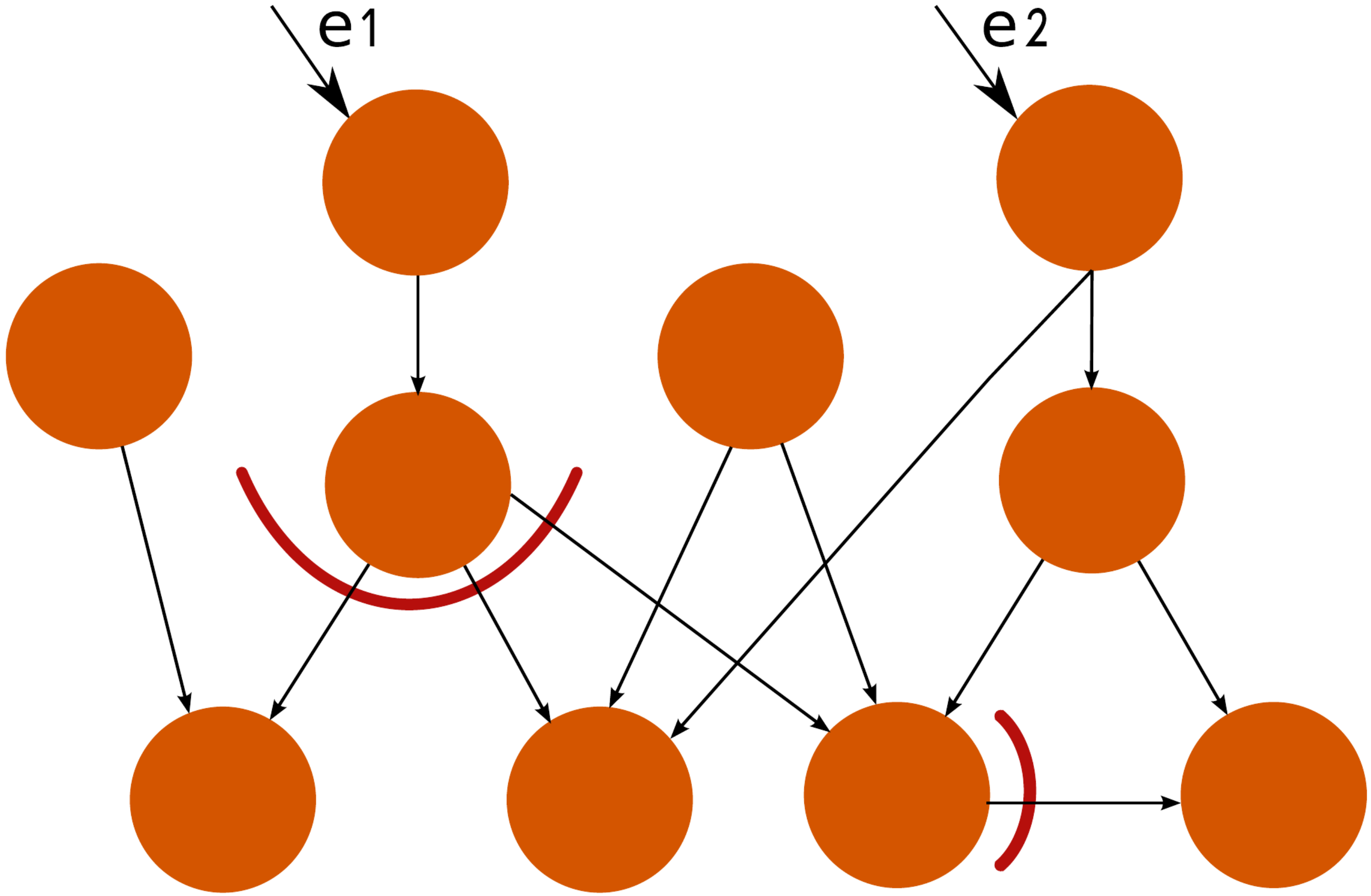
# Obtaining the attack surface: an example



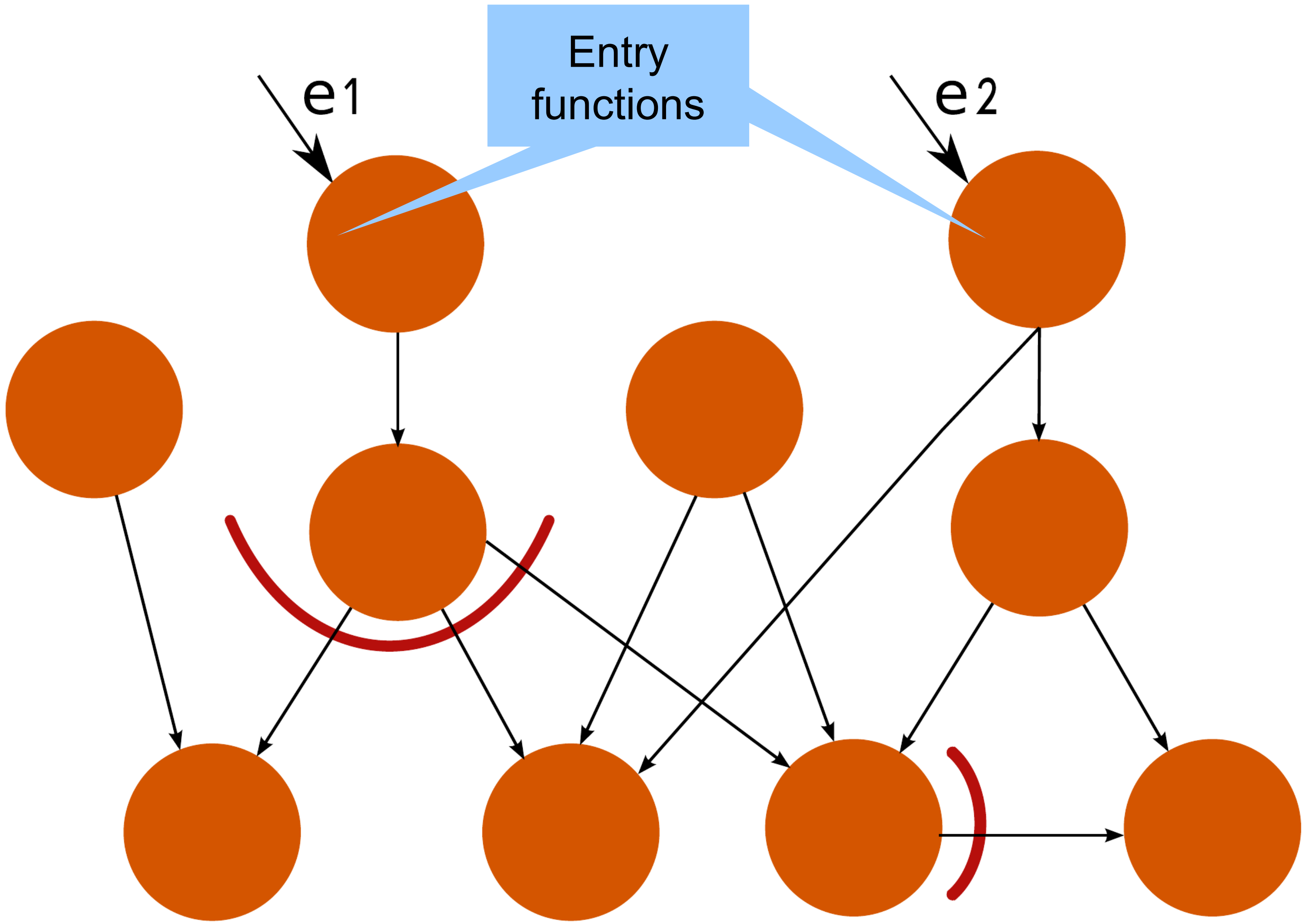
# Obtaining the attack surface: an example



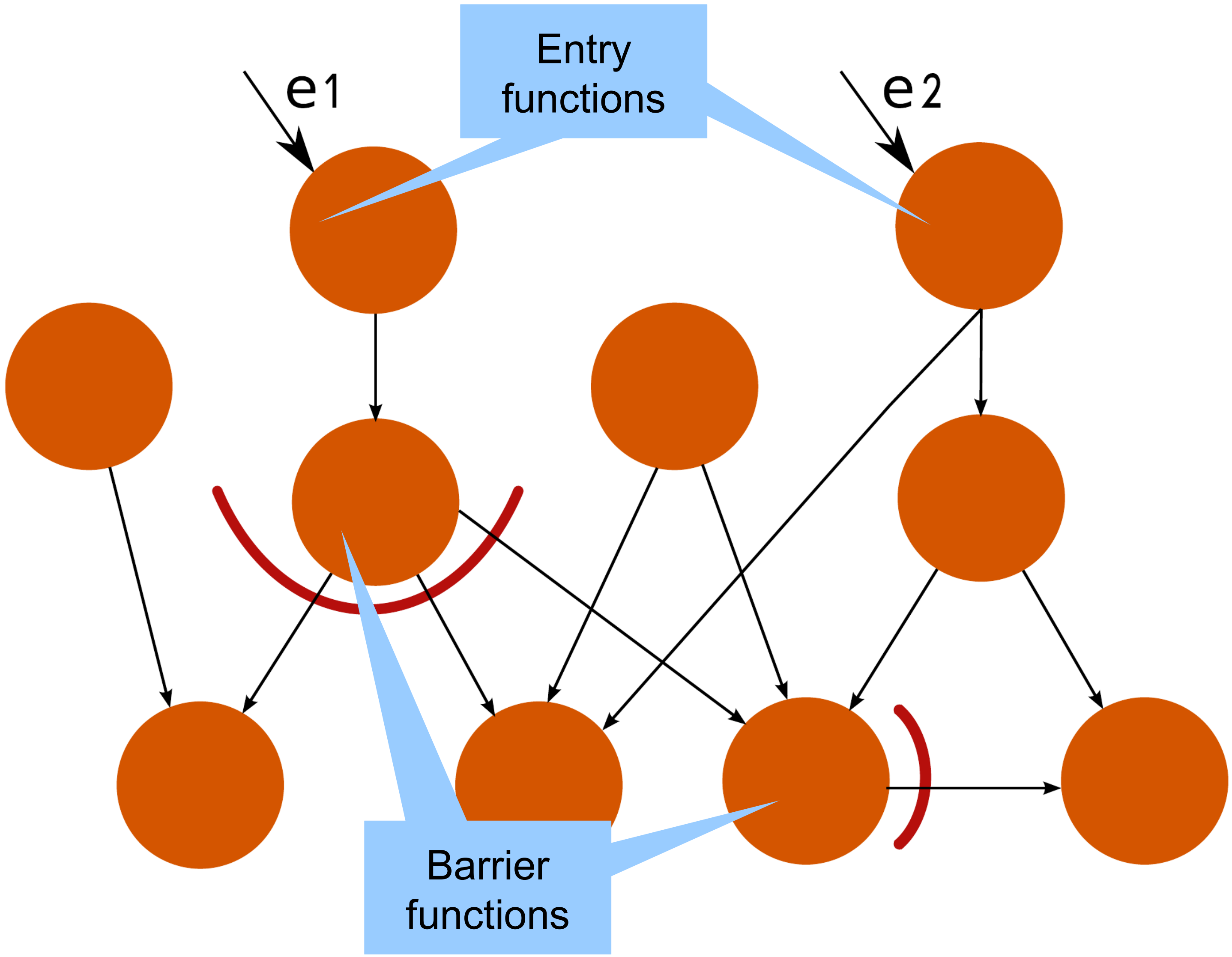
# Obtaining the attack surface: an example



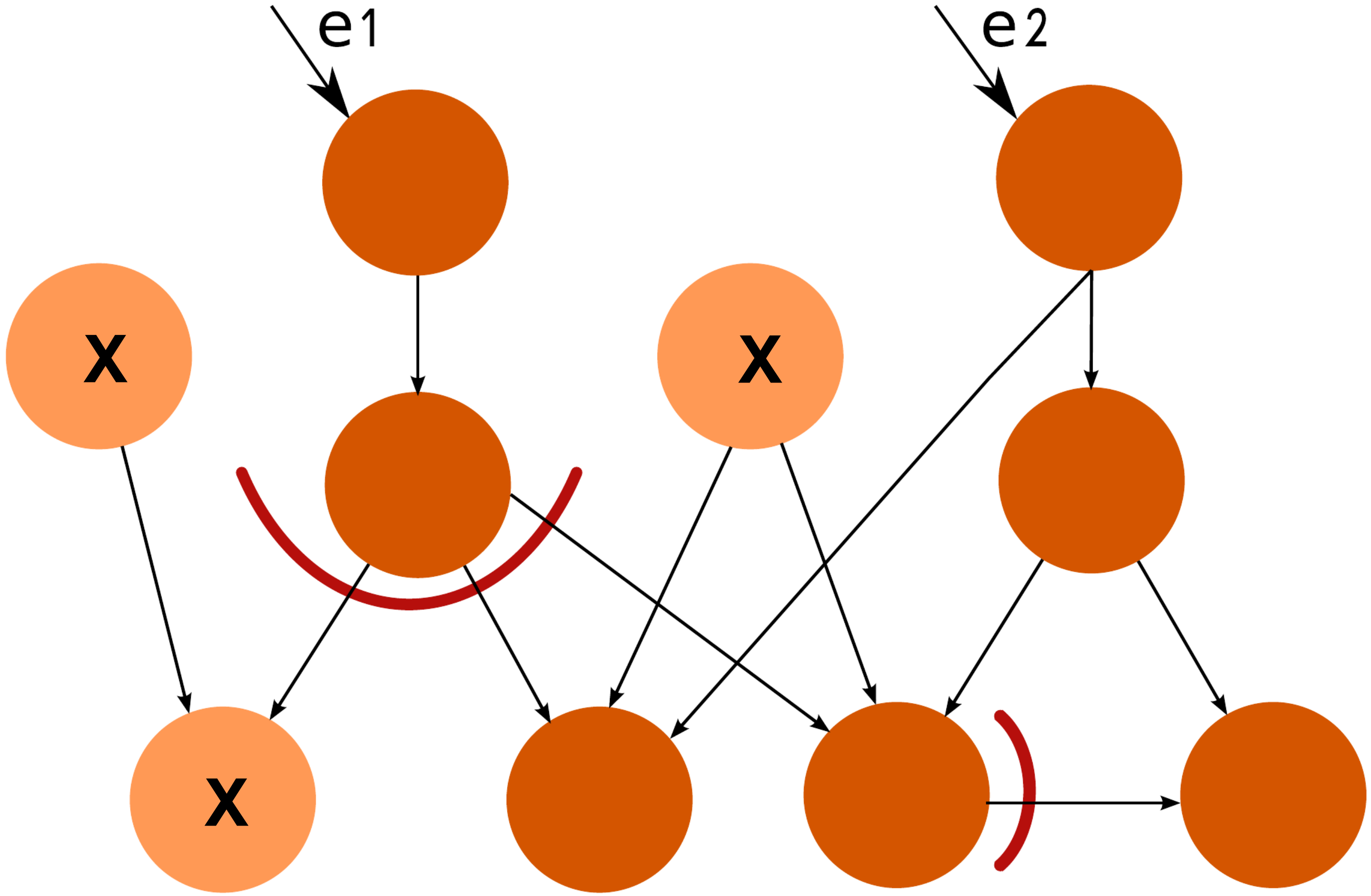
# Obtaining the attack surface: an example



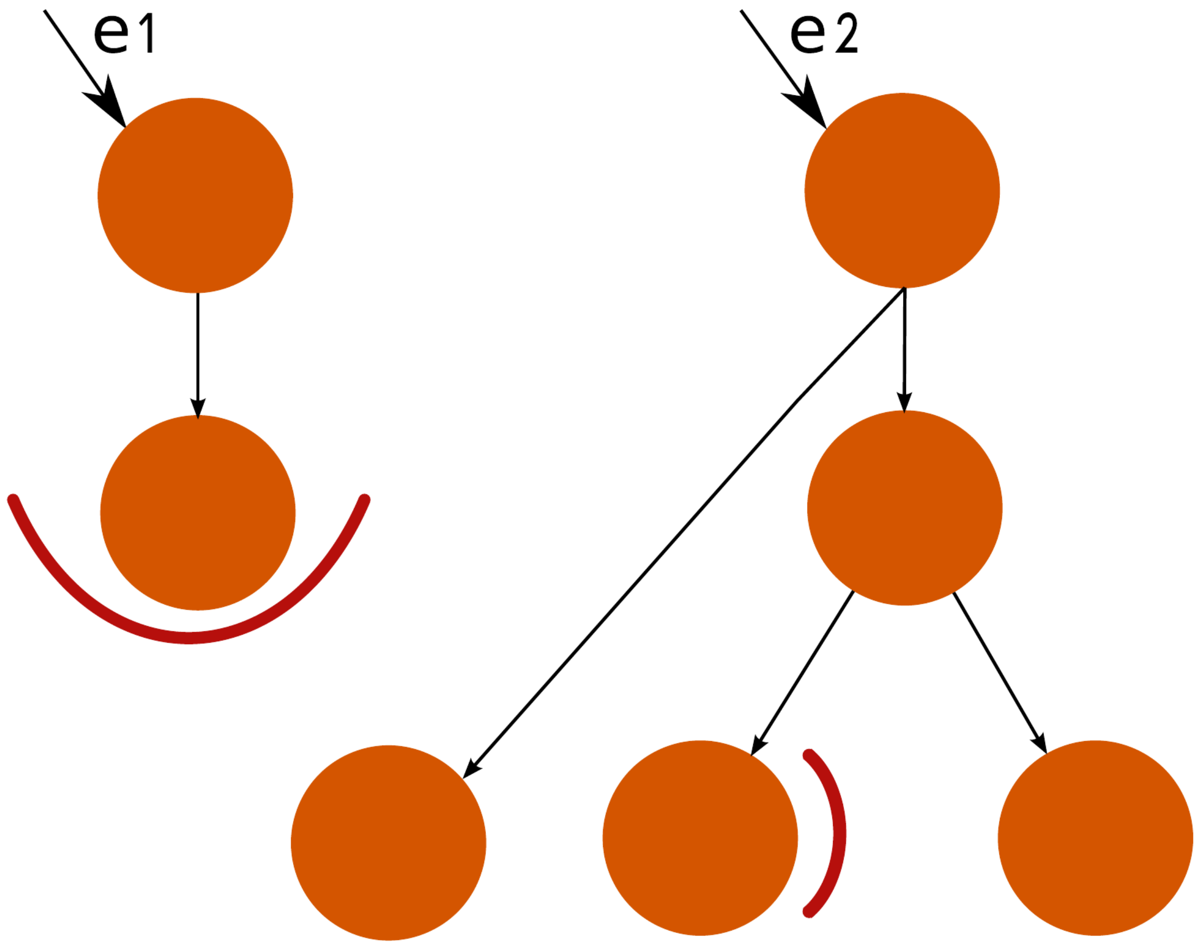
# Obtaining the attack surface: an example



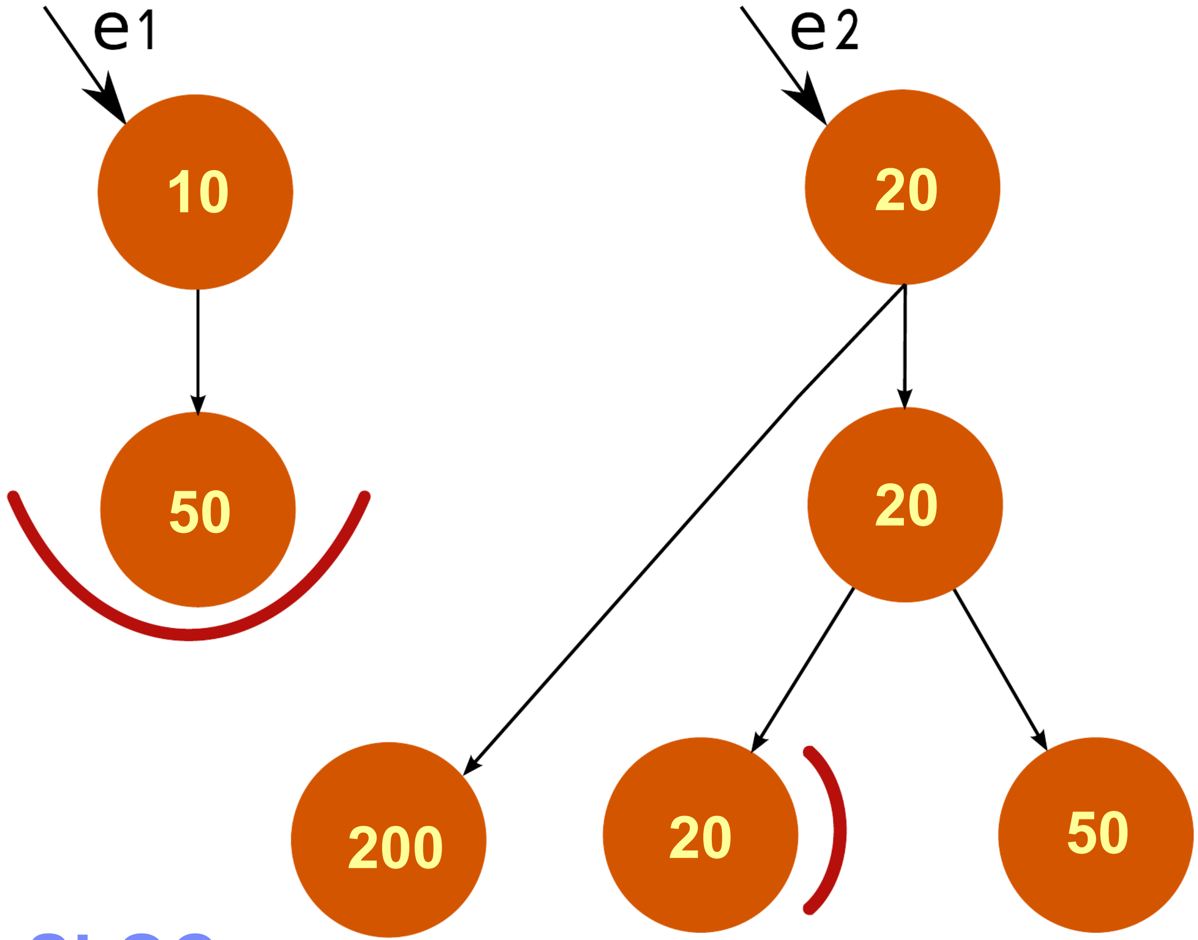
# Obtaining the attack surface: an example



# Obtaining the attack surface: an example

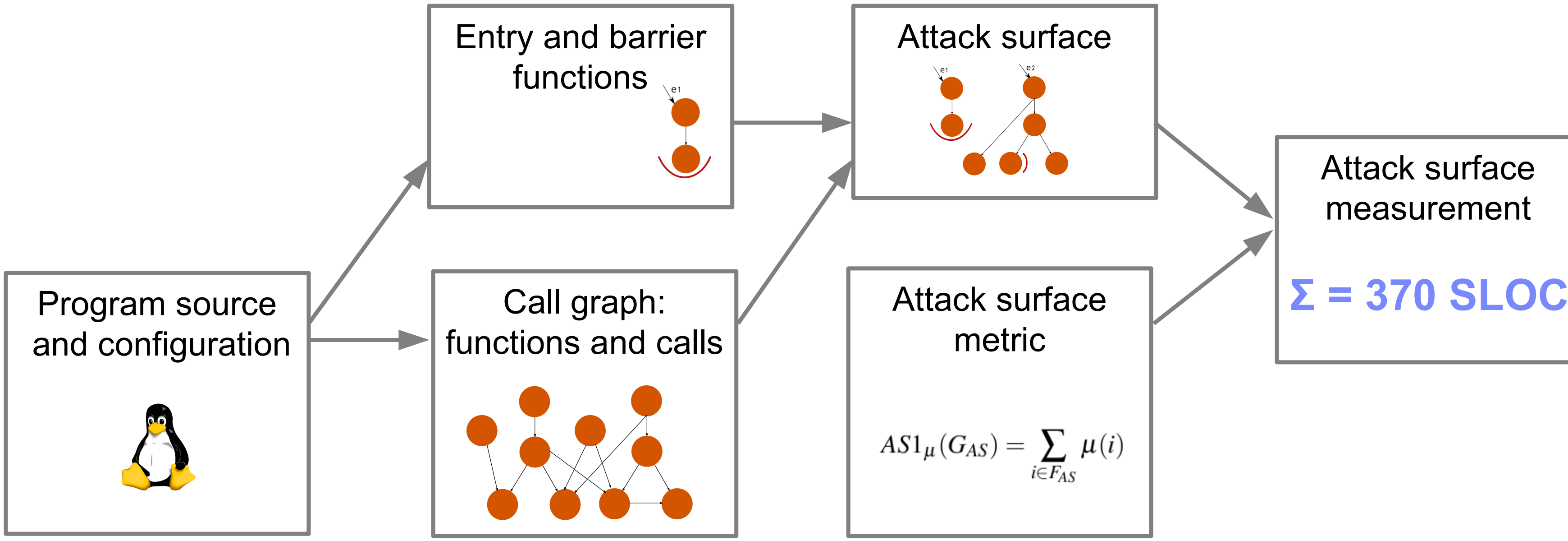


# Attack surface measurement: AS1 with SLOC metric

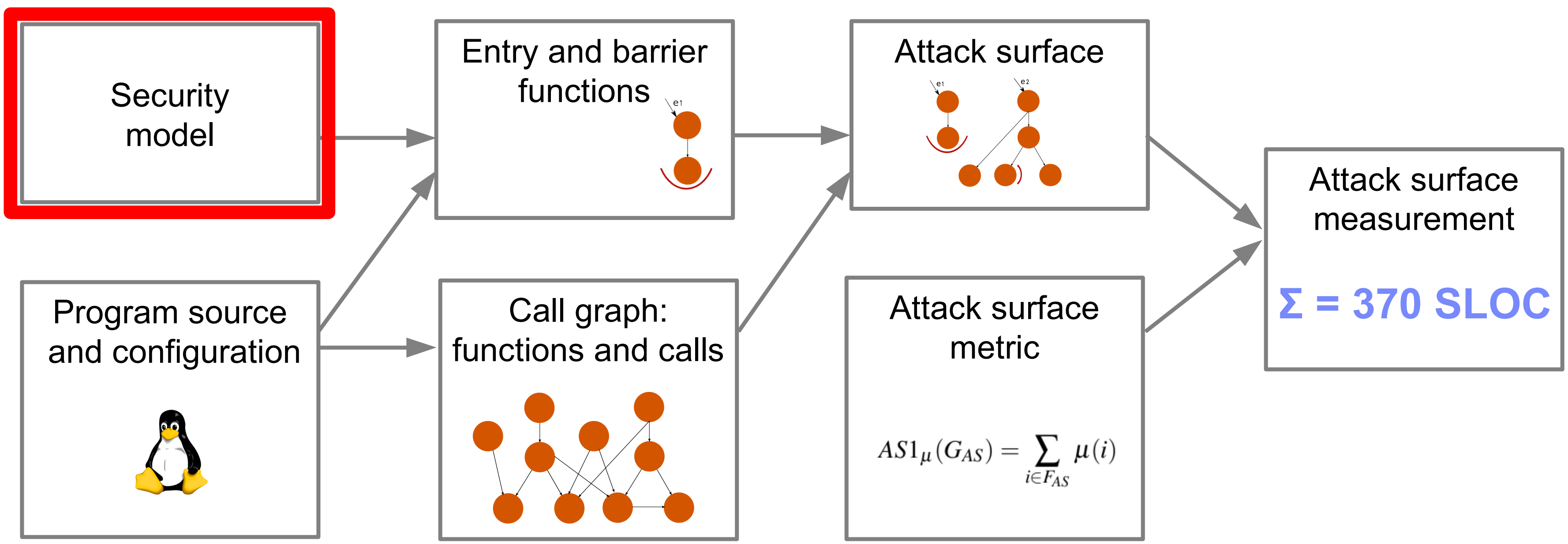



**Σ = 370 SLOC**

# Attack surface measurements: summary

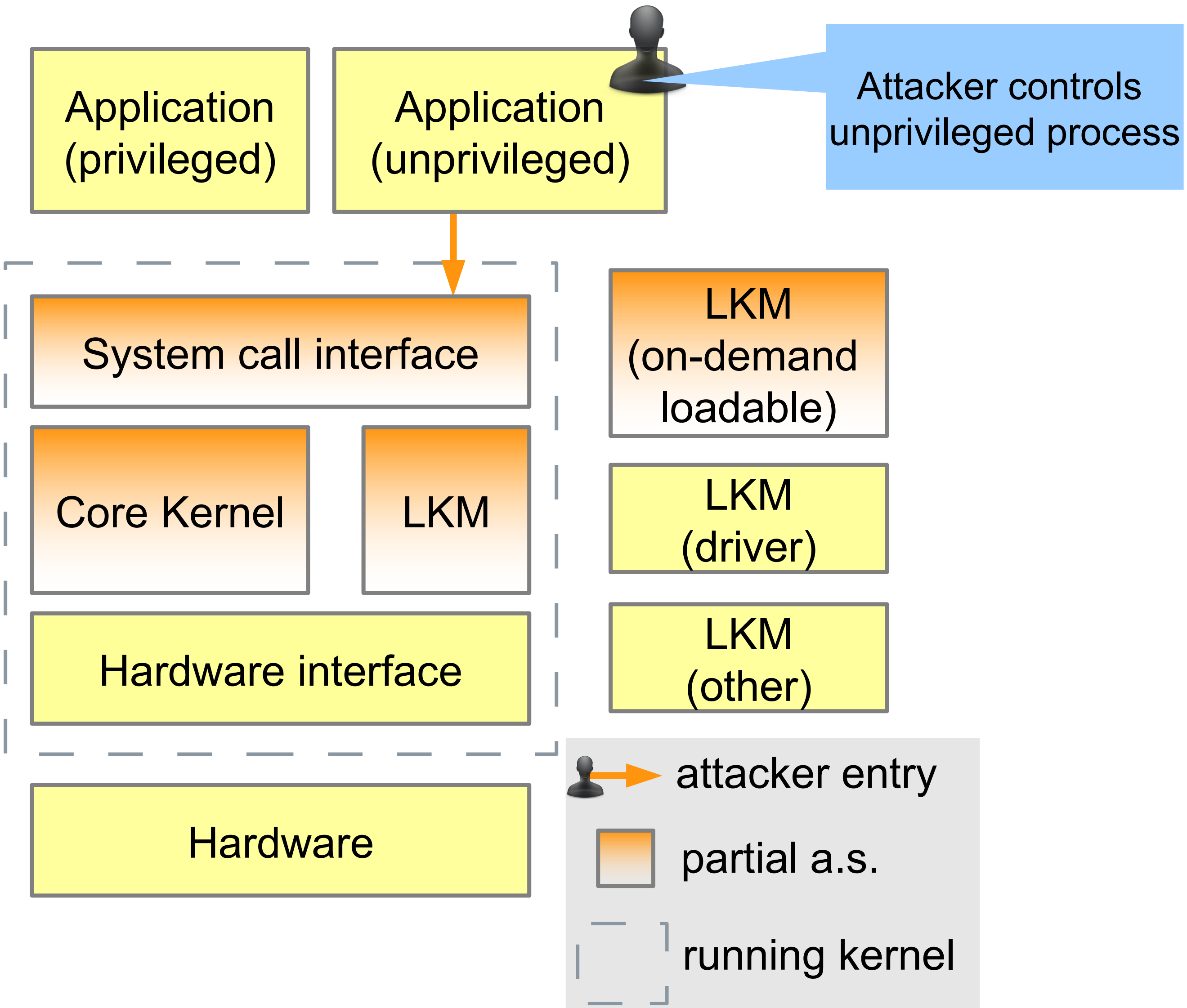


# Attack surface measurements: summary

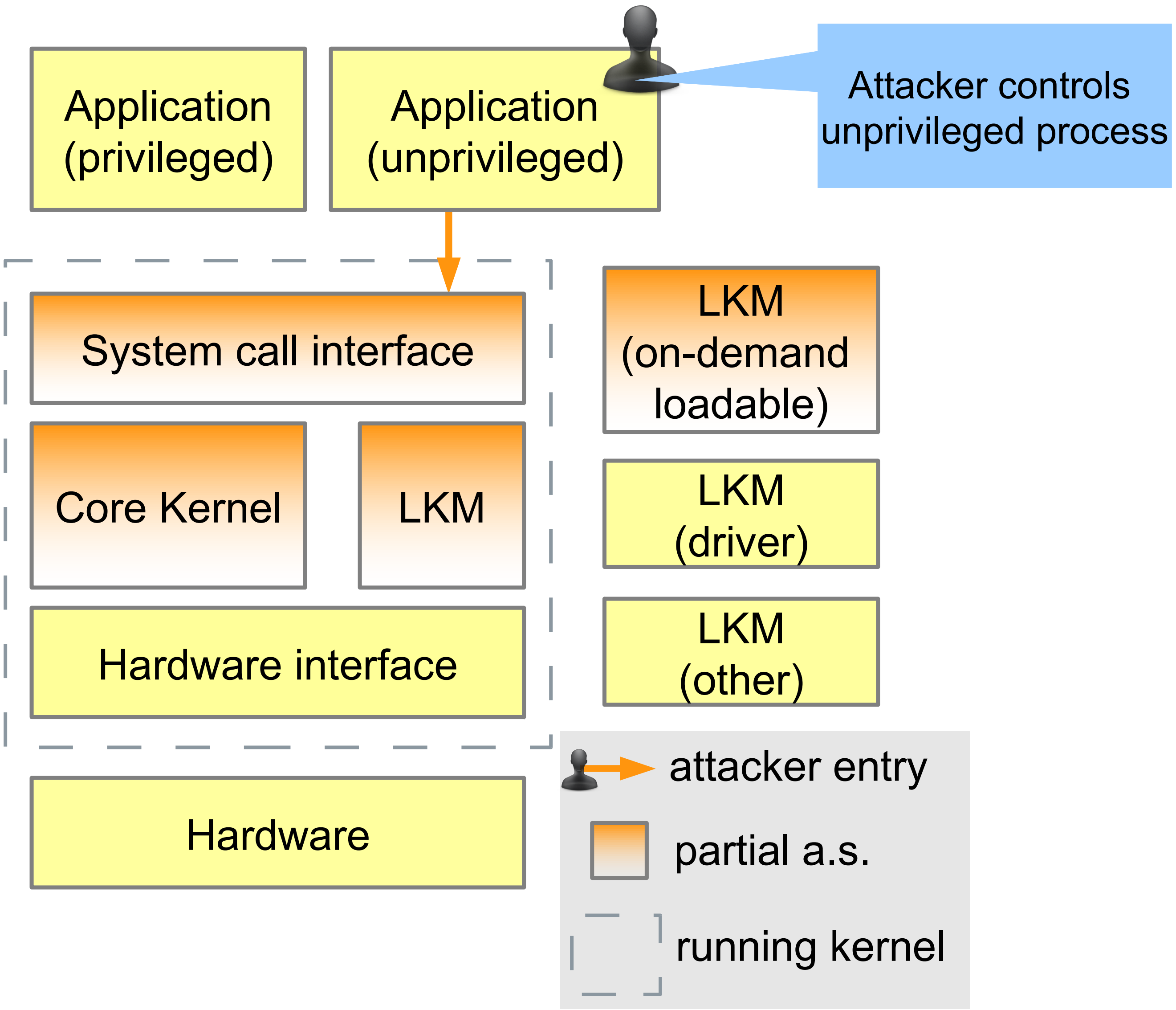



**What security model?**

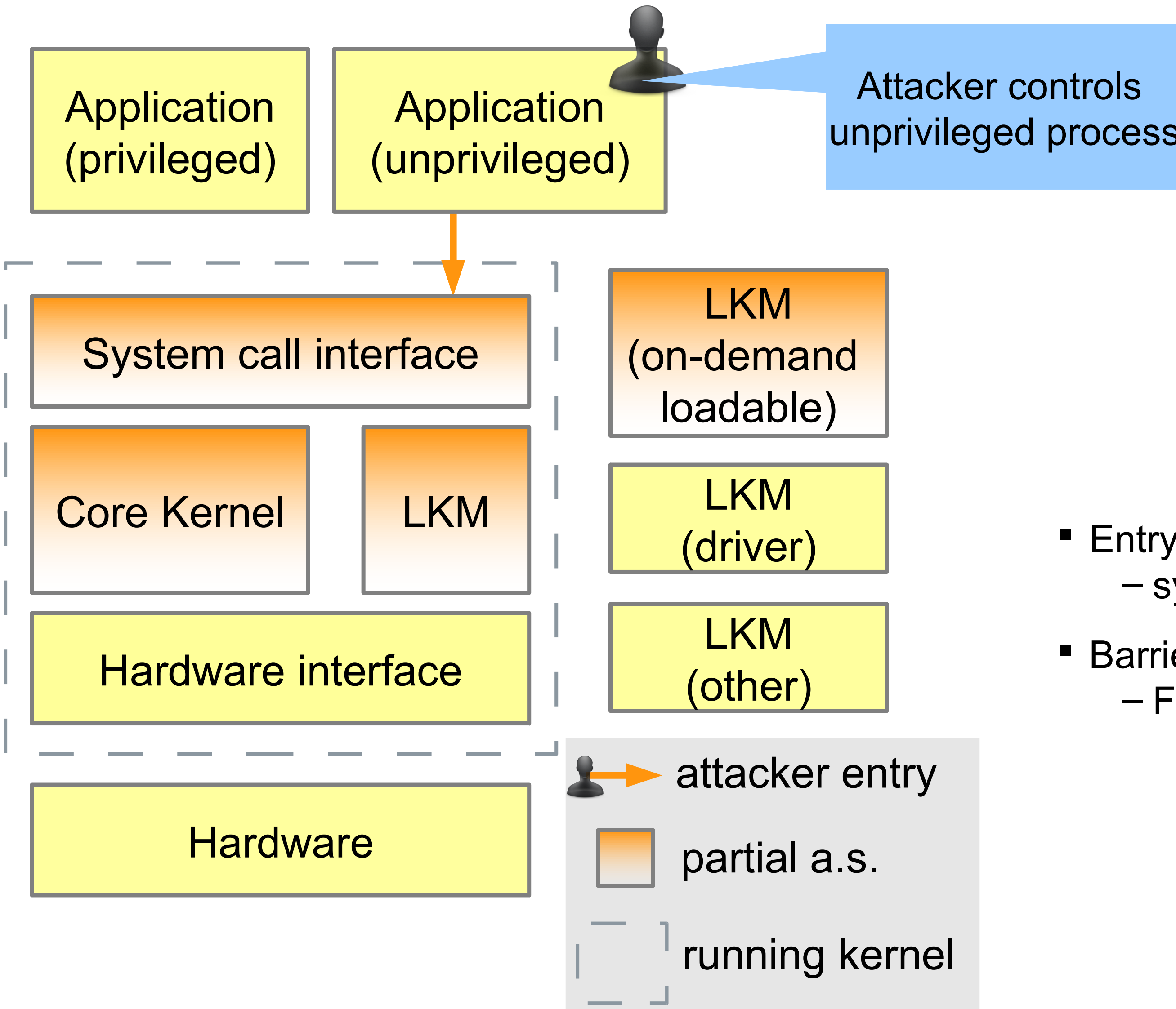
# IsolSec Linux Kernel Security Model



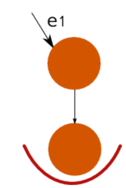
# IsolSec Linux Kernel Security Model



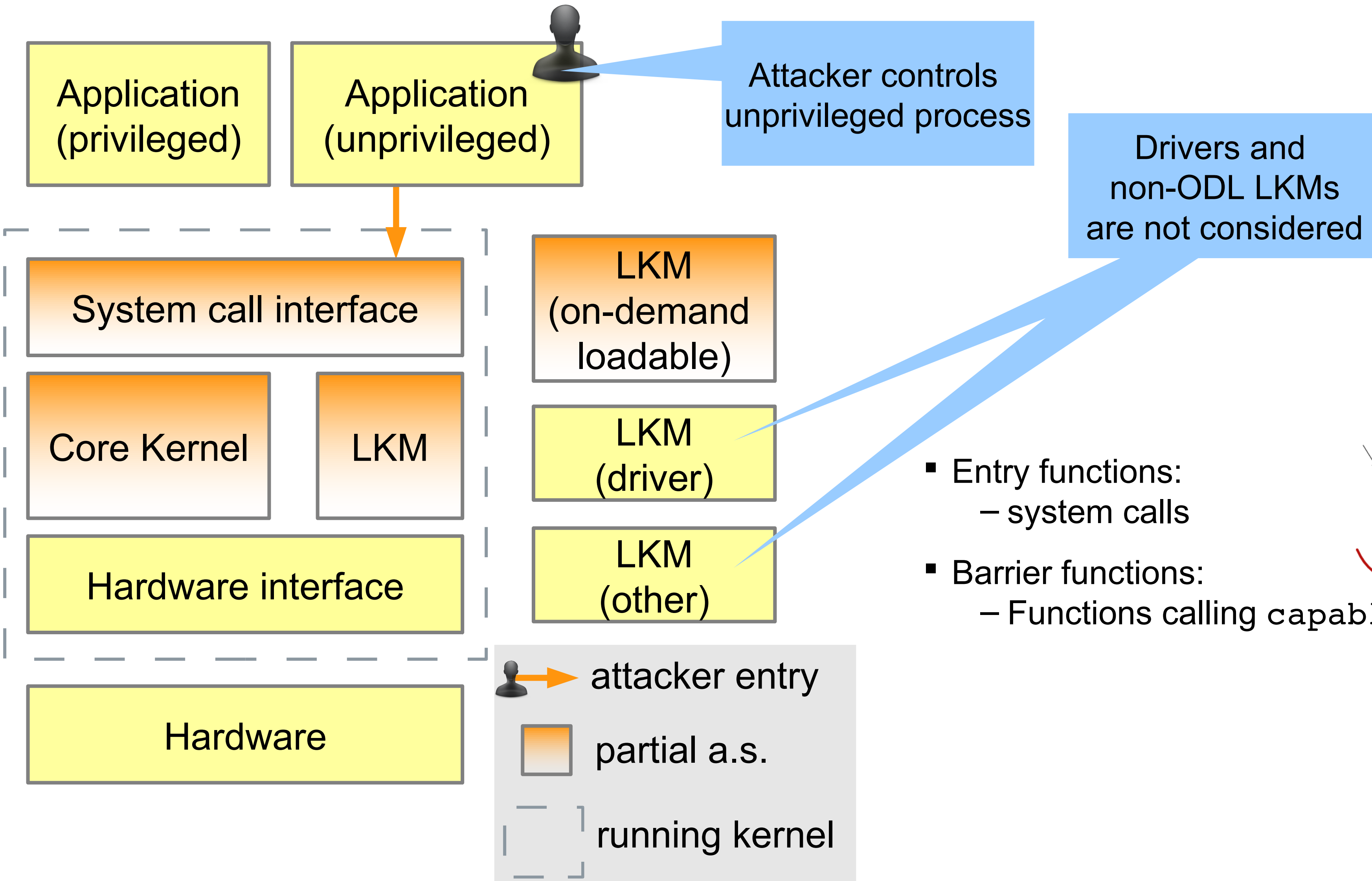
# IsolSec Linux Kernel Security Model



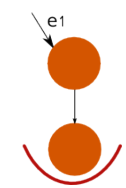

- Entry functions:
  - system calls
- Barrier functions:
  - Functions calling `capable()`



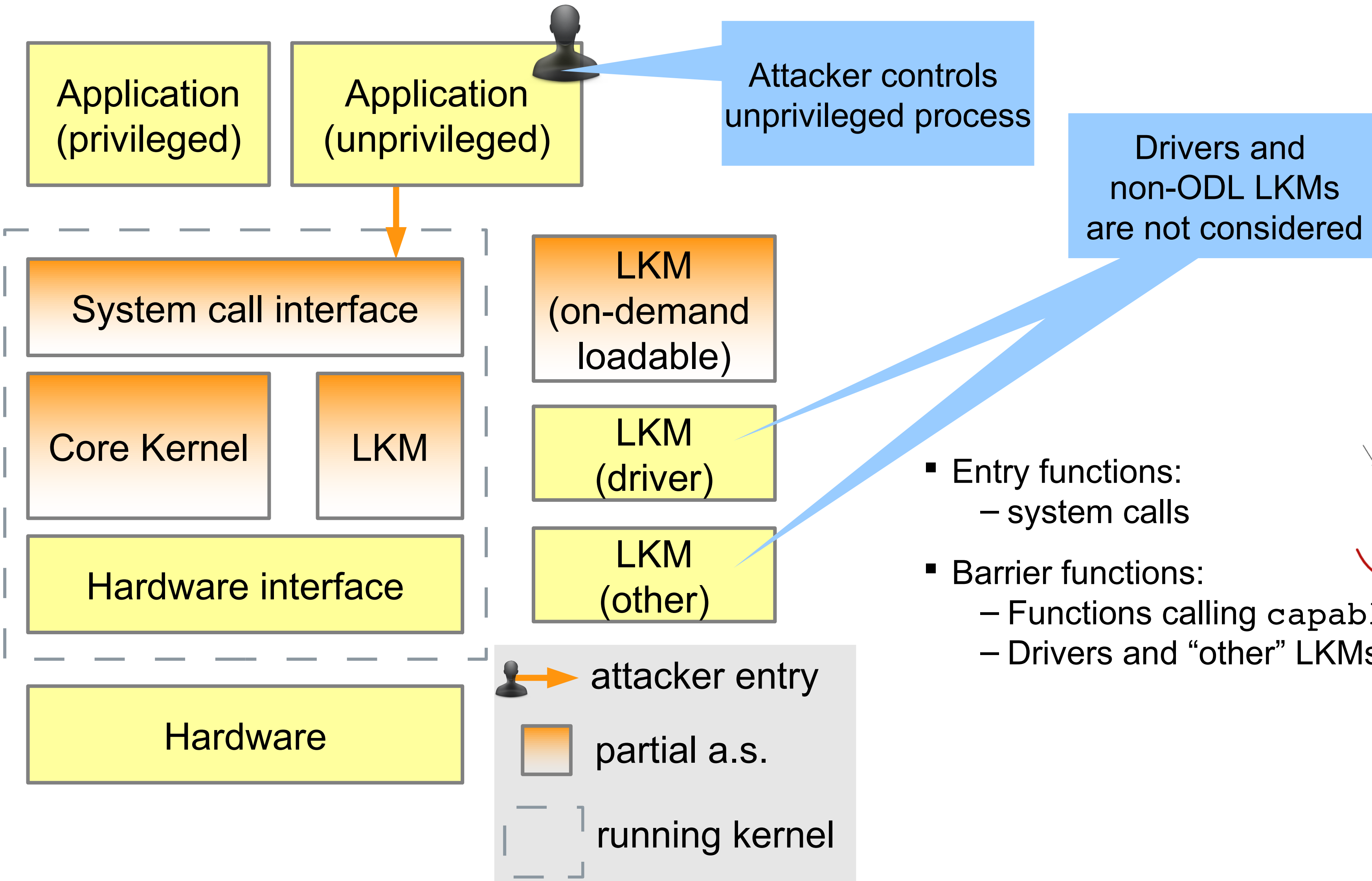
# IsolSec Linux Kernel Security Model



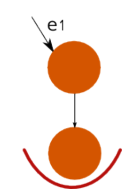

- Entry functions:
  – system calls
- Barrier functions:
  – Functions calling `capable()`



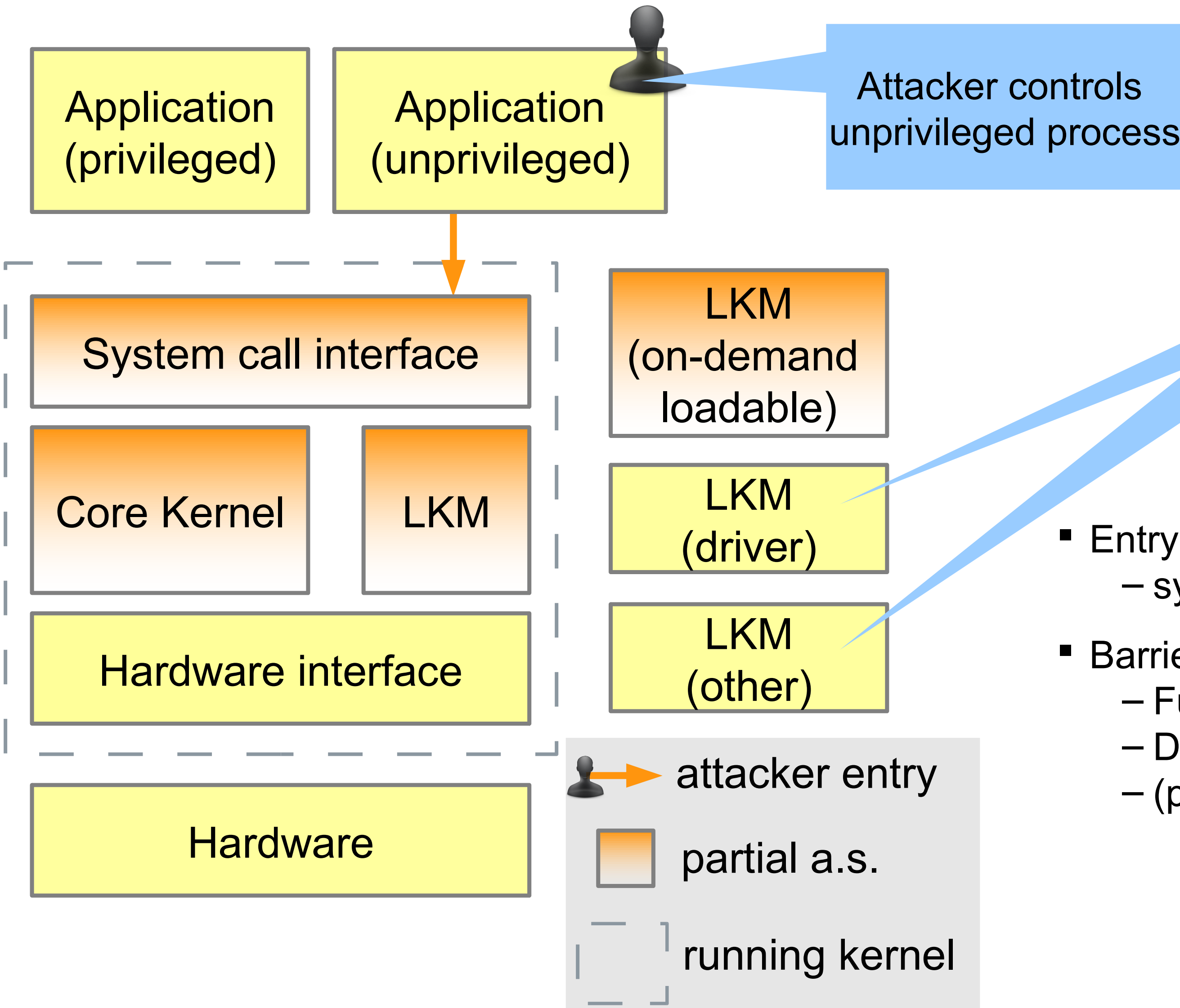
# IsolSec Linux Kernel Security Model



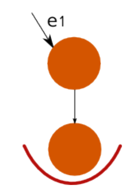

- Entry functions:
  - system calls
- Barrier functions:
  - Functions calling `capable()`
  - Drivers and “other” LKMs



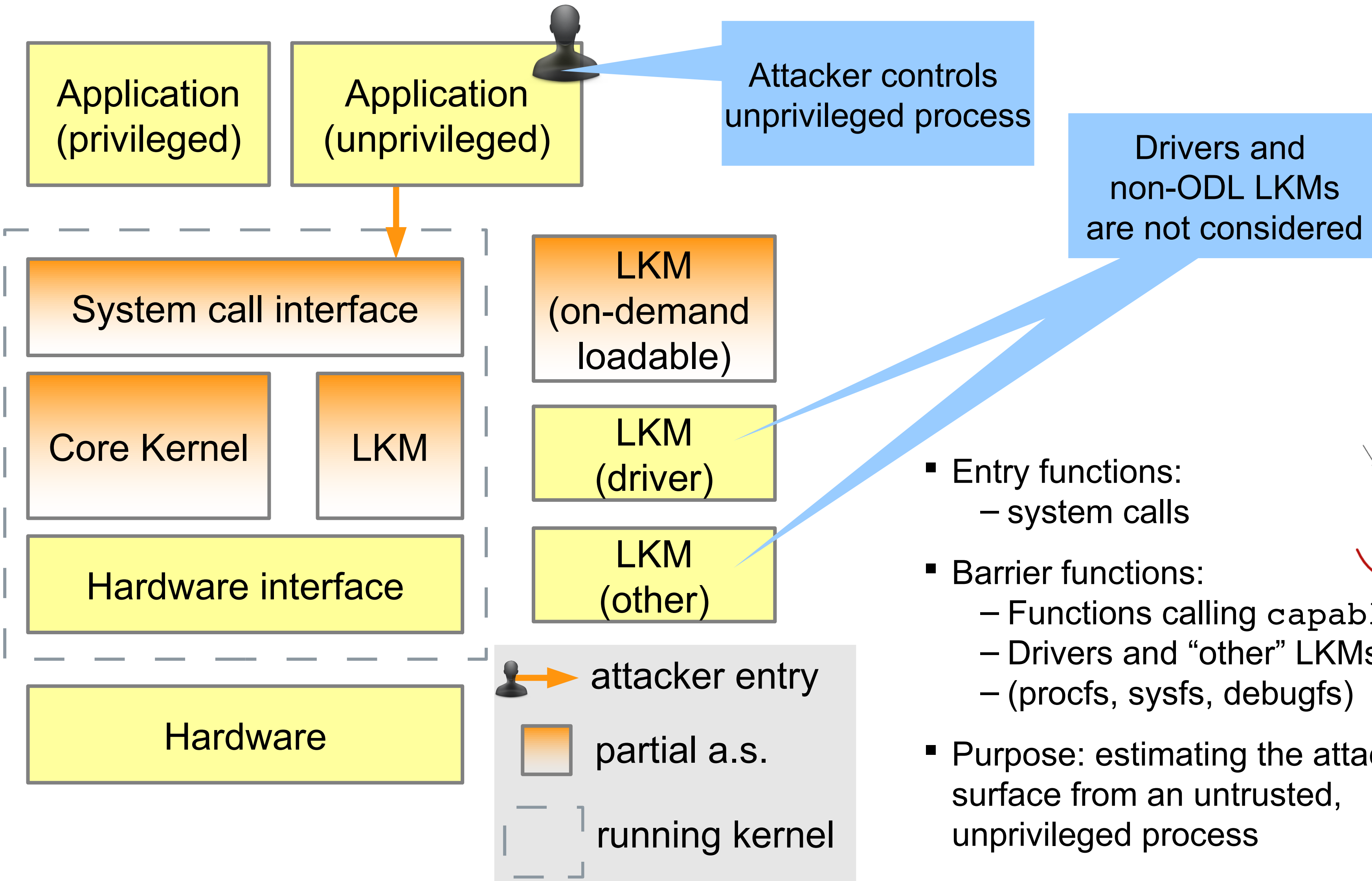
# IsolSec Linux Kernel Security Model



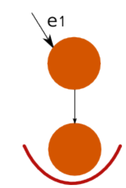

- Entry functions:
  - system calls
- Barrier functions:
  - Functions calling `capable()`
  - Drivers and "other" LKMs
  - (procfs, sysfs, debugfs)



# IsolSec Linux Kernel Security Model

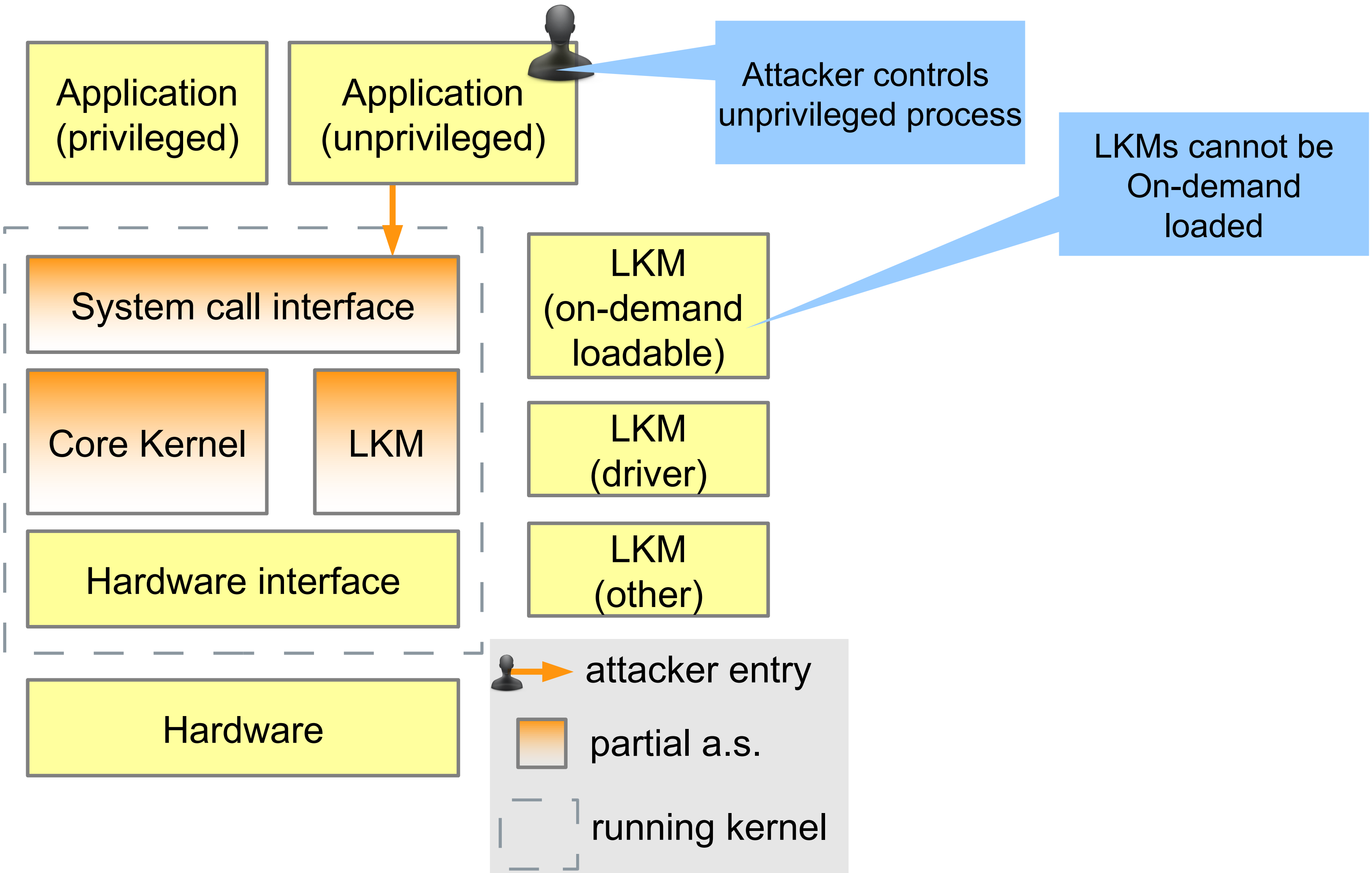



- Entry functions:
  - system calls

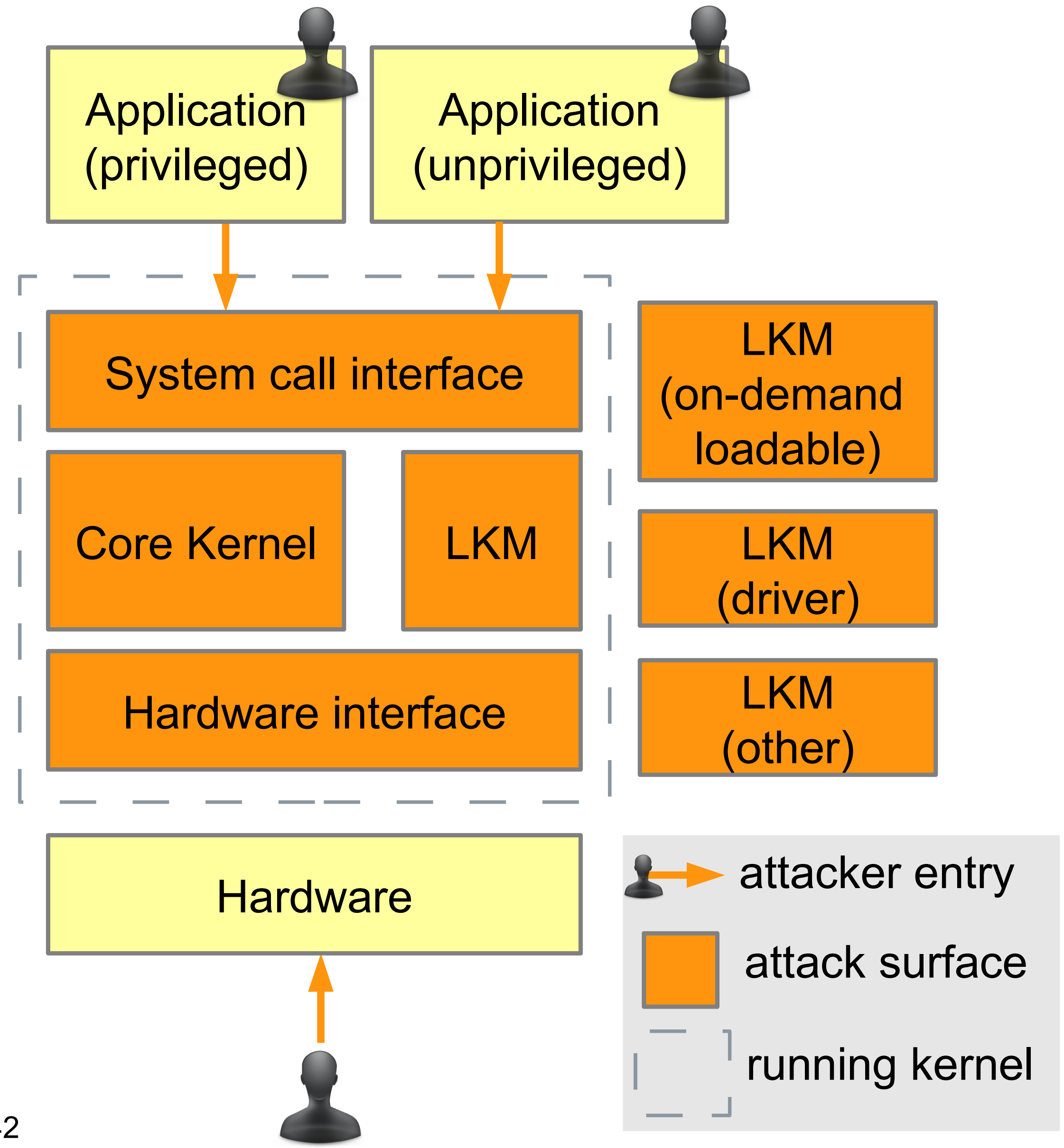



- Barrier functions:
  - Functions calling `capable()`
  - Drivers and “other” LKMs
  - (procfs, sysfs, debugfs)
- Purpose: estimating the attack surface from an untrusted, unprivileged process

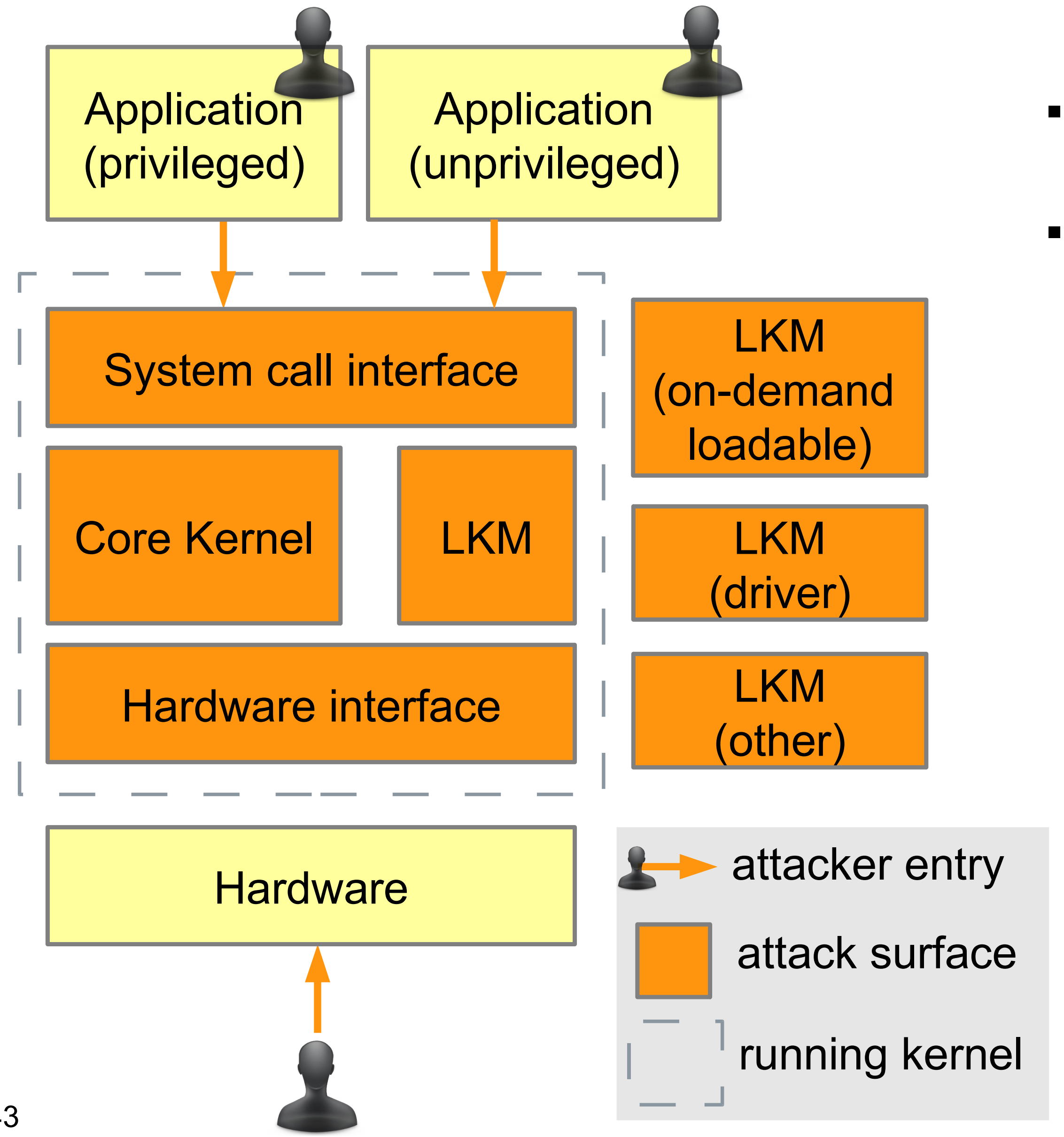
# StaticSec Linux Kernel Security Model



# GenSec Linux Kernel Security Model

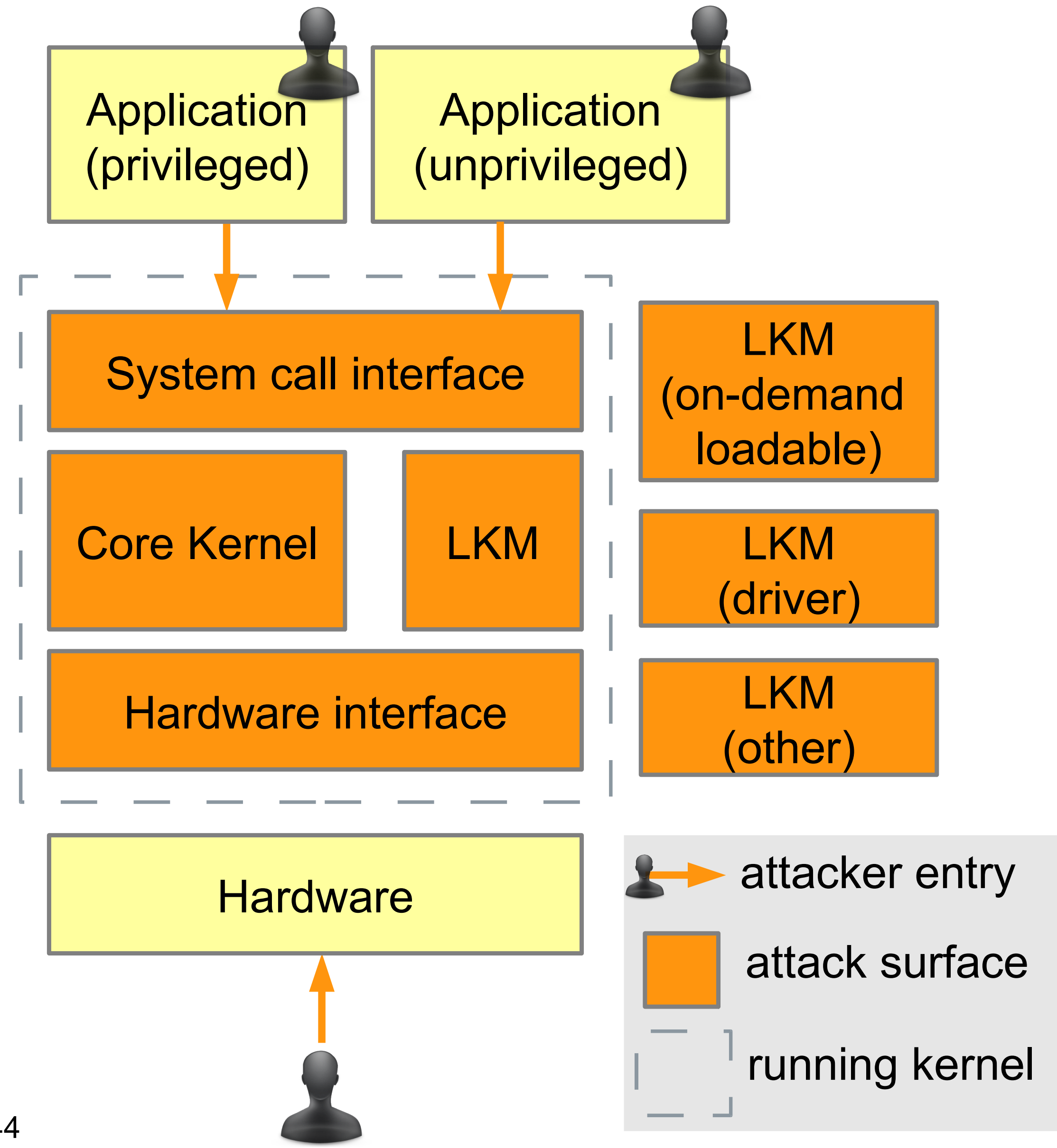


# GenSec Linux Kernel Security Model




- Entry functions:
  – all
- Barrier functions:
  – none

# GenSec Linux Kernel Security Model




- Entry functions:
  - all
- Barrier functions:
  - none
- Overestimates attack surface
  - attacker is privileged?
  - not all LKMs can be loaded
- Purpose:
  - upper bound
  - TCB point of view

# Compile-time Kernel Tailoring

- [NDSS'13] Anil Kurmus, Reinhard Tartler, Daniela Dorneanu, Bernhard Heinloth, Valentin Rothberg, Andreas Ruprecht, Wolfgang Schröder-Preikschat, Daniel Lohmann and Rüdiger Kapitza. "**Attack Surface Metrics and Automated Compile-Time OS Kernel Tailoring.**" In: Proceedings of the 20th Network and Distributed System Security Symposium. 2013.

  https://www.ibr.cs.tu-bs.de/users/kurmus/papers/kurmus-ndss13.pdf

# Making the kernel smaller

~ 5000 features
(ubuntu 12.04)

~ 500 features
(realistic use case)

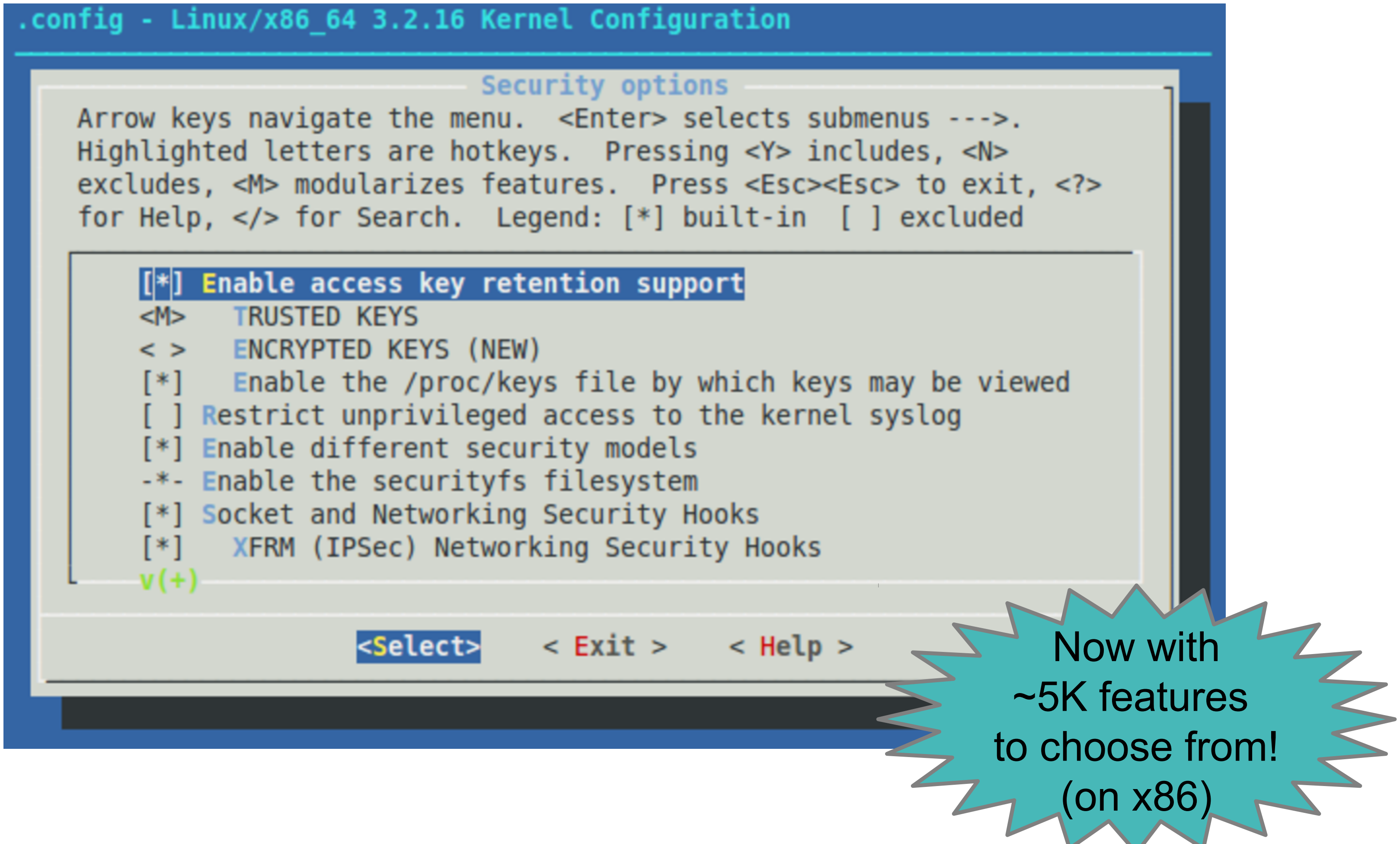
# Making the kernel smaller

~ 500 features
(realistic use case)

~ 5000 features
(ubuntu 12.04)

**→ Remove unnecessary features from the kernel by leveraging built-in configurability**

# Make (menuconfig) your way to a smaller kernel



# Don't take my word for it

## [RFC] Simplifying kernel configuration for distro issues

87 messages

**Linus Torvalds** <torvalds@linux-foundation.org> Fri, Jul 13, 2012 at 10:37 PM
To: Dave Jones <davej@redhat.com>, Greg Kroah-Hartman <greg@kroah.com>, Ubuntu Kernel Team <kernel-team@lists.ubuntu.com>, Debian Kernel Team <debian-kernel@lists.debian.org>, OpenSUSE Kernel Team <opensuse-kernel@opensuse.org>
Cc: Linux Kernel Mailing List <linux-kernel@vger.kernel.org>

So this has long been one of my pet configuration peeves: as a user I am perfectly happy answering the questions about what kinds of hardware I want the kernel to support (I kind of know that), but many of the "support infrastructure" questions are very opaque, and I have no idea which of the them any particular distribution actually depends on.

And it tends to change over time. For example, F14 (iirc) started using TMPFS and TMPFS_POSIX_ACL/XATTR for /dev. And starting in F16, the initrd setup requires DEVTMPFS and DEVTMPFS_MOUNT. There's been several times when I started with my old minimal config, and the resulting kernel would boot, but something wouldn't quite work right, and it can be very subtle indeed.

Similarly, the distro ends up having very particular requirements for exactly *which* security models it uses and needs, and they tend to change over time. And now with systemd, CGROUPS suddenly aren't just esoteric things that no normal person would want to use, but are used for basic infrastructure. And I remember being surprised by OpenSUSE suddenly needing the RAW table support for netfilter, because it had a NOTRACK rule or something.

# Don't take my word for it

**[RFC] Simplifying kernel configuration for distro issues**
87 messages

**Linus Torvalds** <torvalds@linux-foundation.org> Fri, Jul 13, 2012 at 10:37 PM
To: Dave Jones <davej@redhat.com>, Greg Kroah-Hartman <greg@kroah.com>, Ubuntu Kernel Team <kernel-team@lists.ubuntu.com>, Debian Kernel Team <debian-kernel@lists.debian.org>, OpenSUSE Kernel Team <opensuse-kernel@opensuse.org>
Cc: Linux Kernel Mailing List <linux-kernel@vger.kernel.org>

"many of the support infrastructure questions are very opaque, and I have no idea which of them any particular distribution actually depends on."

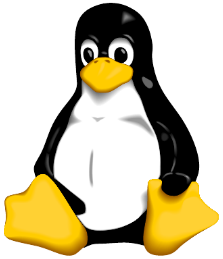
And it tends to change over time. For example, F14 (IIRC) started using TMPFS and TMPFS_POSIX_ACL/XATTR for /dev. And starting in F16, the initrd setup requires DEVTMPFS and DEVTMPFS_MOUNT. There's been several times when I started with my old minimal config, and the resulting kernel would boot, but something wouldn't quite work right, and it can be very subtle indeed.

Similarly, the distro ends up having very particular requirements for exactly *which* security models it uses and needs, and they tend to change over time. And now with systemd, CGROUPS suddenly aren't just esoteric things that no normal person would want to use, but are used for basic infrastructure. And I remember being surprised by OpenSUSE suddenly needing the RAW table support for netfilter, because it had a NOTRACK rule or something.

# Automatic Kernel-Configuration Tailoring

# Automatic Kernel-Configuration Tailoring

Distribution kernel
and use case



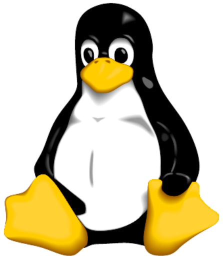
# Automatic Kernel-Configuration Tailoring

Distribution kernel
and use case

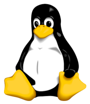
Tailored kernel

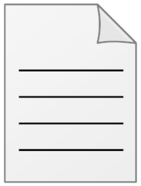
# Automatic Kernel-Configuration Tailoring

Distribution kernel
and use case



Tailored kernel





run workload
and collect **trace**

```
Makefile
arch/x86/init.c:59
arch/x86/entry32.S:14
arch/x86/...
lib/Makefile
kernel/sched.c:723
...
```

correlate to
**source line** locations
and **#ifdefs**

```
B00 <-> CONFIG_X86
&&
B1 <-> CONFIG_NUMA
&&
B2 <-> ! B1
&&
...
```

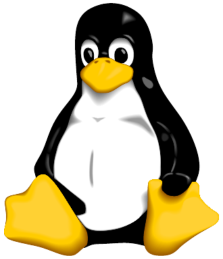
correlate to **features**
and take into account
**feature dependencies**

```
CONFIG_X86=y
CONFIG_NUMA=y
CONFIG_SCSI=m
...
...
```

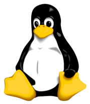
solve formula
and derive a **kernel**
**configuration**

# Automatic Kernel-Configuration Tailoring

Distribution kernel
and use case



Tailored kernel



run workload
and collect **trace**

```
Makefile
arch/x86/init.c:59
arch/x86/entry32.S:14
arch/x86/...
lib/Makefile
kernel/sched.c:723
...
```

correlate to
**source line** locations
and **#ifdefs**

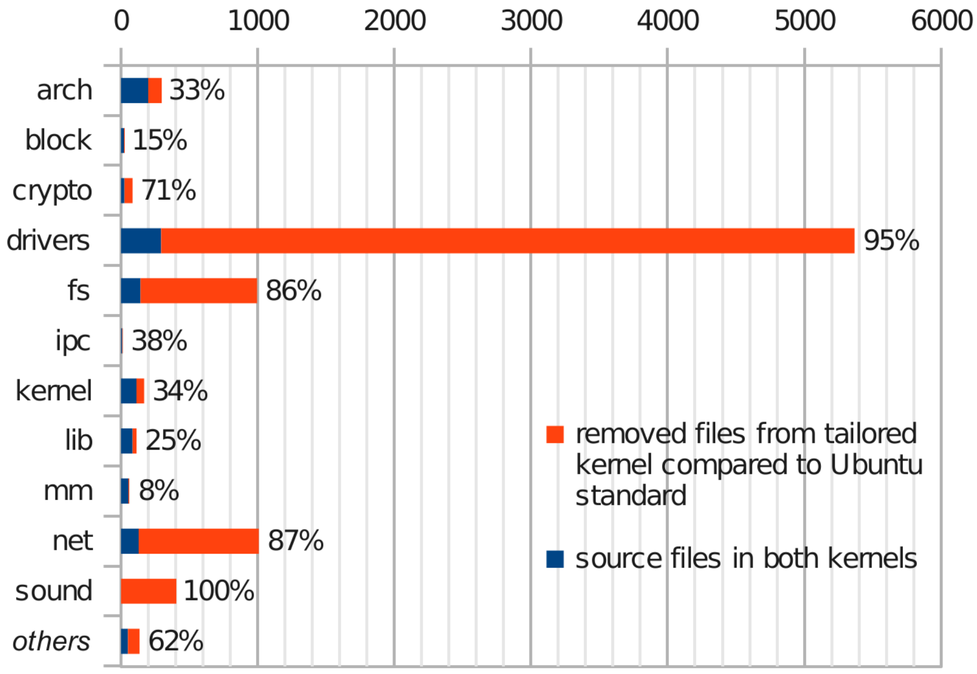
```
B00 <-> CONFIG_X86
&&
B1 <-> CONFIG_NUMA
&&
B2 <-> ! B1
&&
...
```

correlate to **features**
and take into account
**feature dependencies**

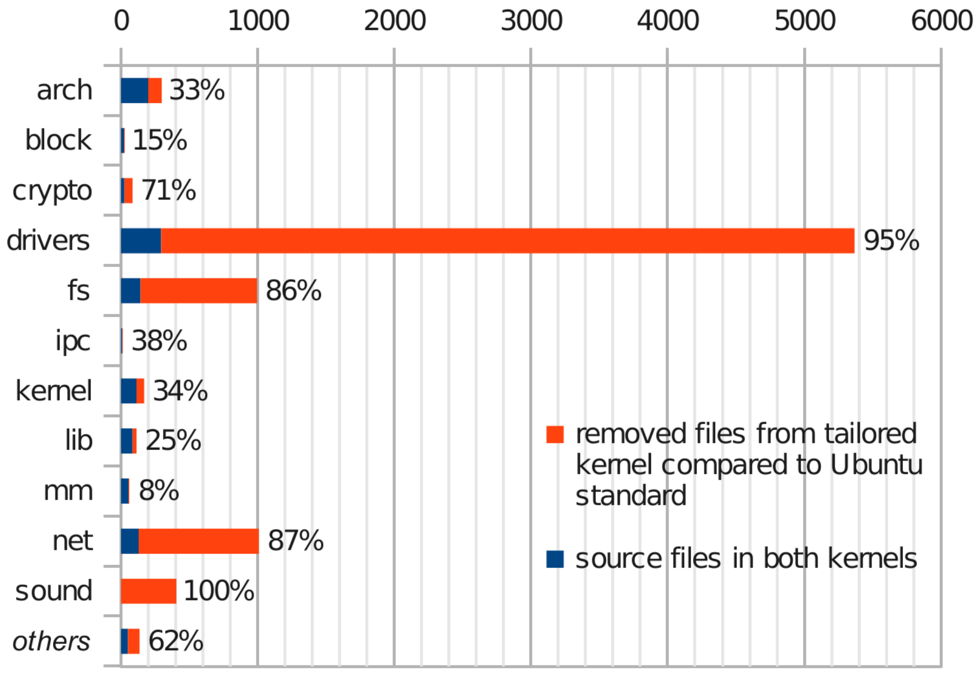
```
CONFIG_X86=y
CONFIG_NUMA=y
CONFIG_SCSI=m
...
...
```

solve formula
and derive a **kernel configuration**

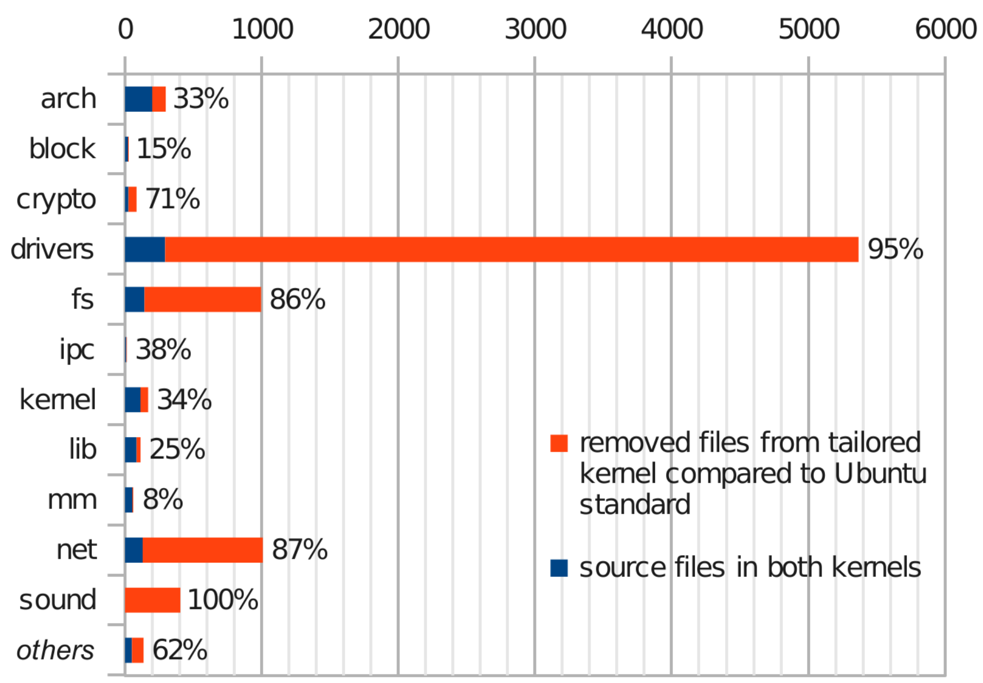
# Resulting kernel



# Resulting kernel



# Resulting kernel

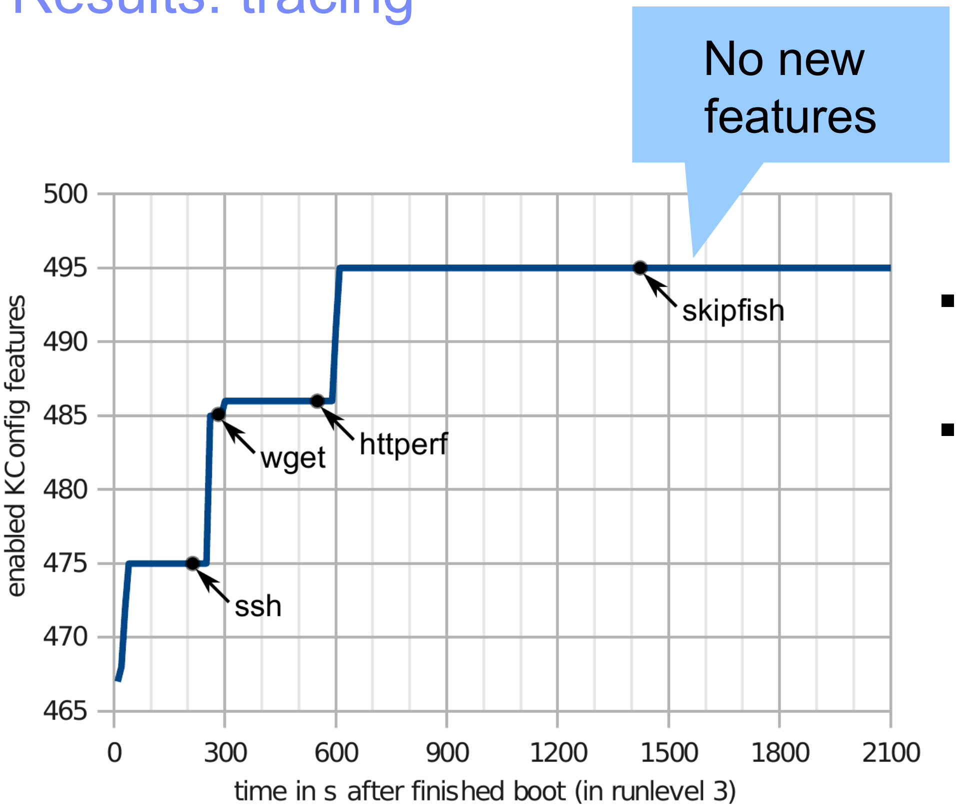



- How much **attack surface reduction**?

# Selected results of the evaluation

- Typical server use case: LAMP

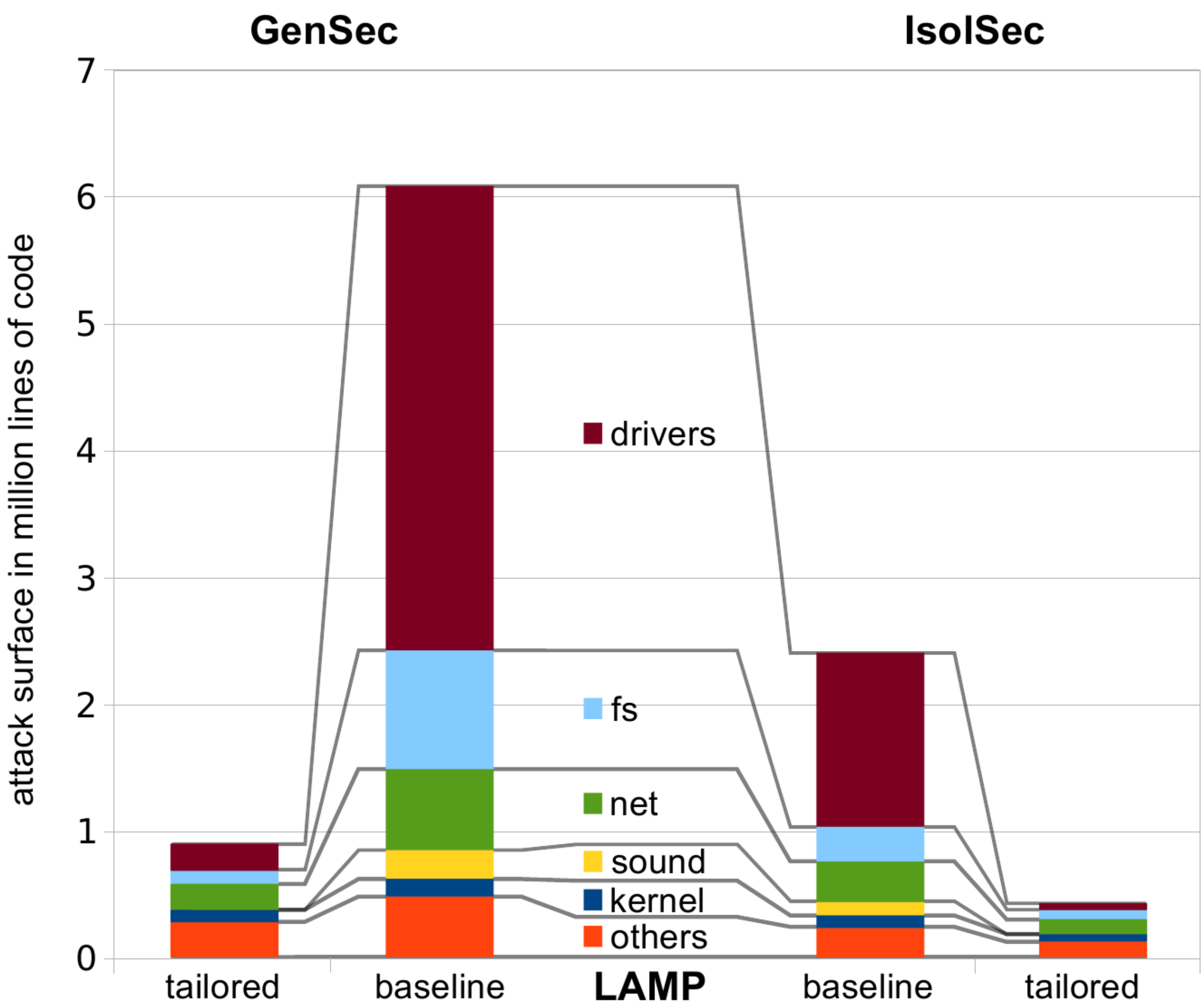
# Results: tracing



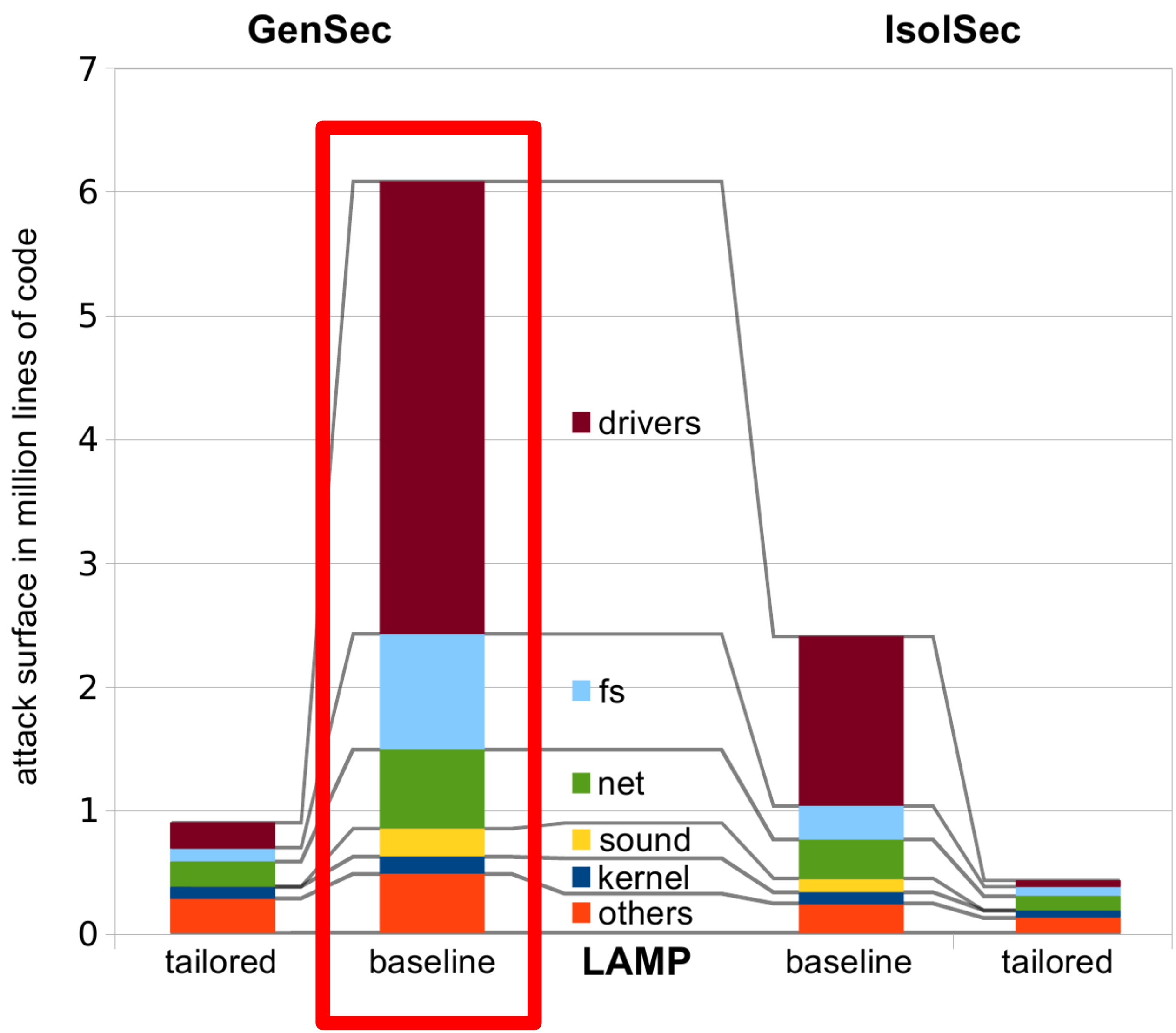

- Httperf benchmark triggers new features
  - Stabilizes at 495 features
- Skipfish: high coverage of the web application
  - Goes beyond real-world workload

➡ **Tracing at “feature-granularity” converges quickly**

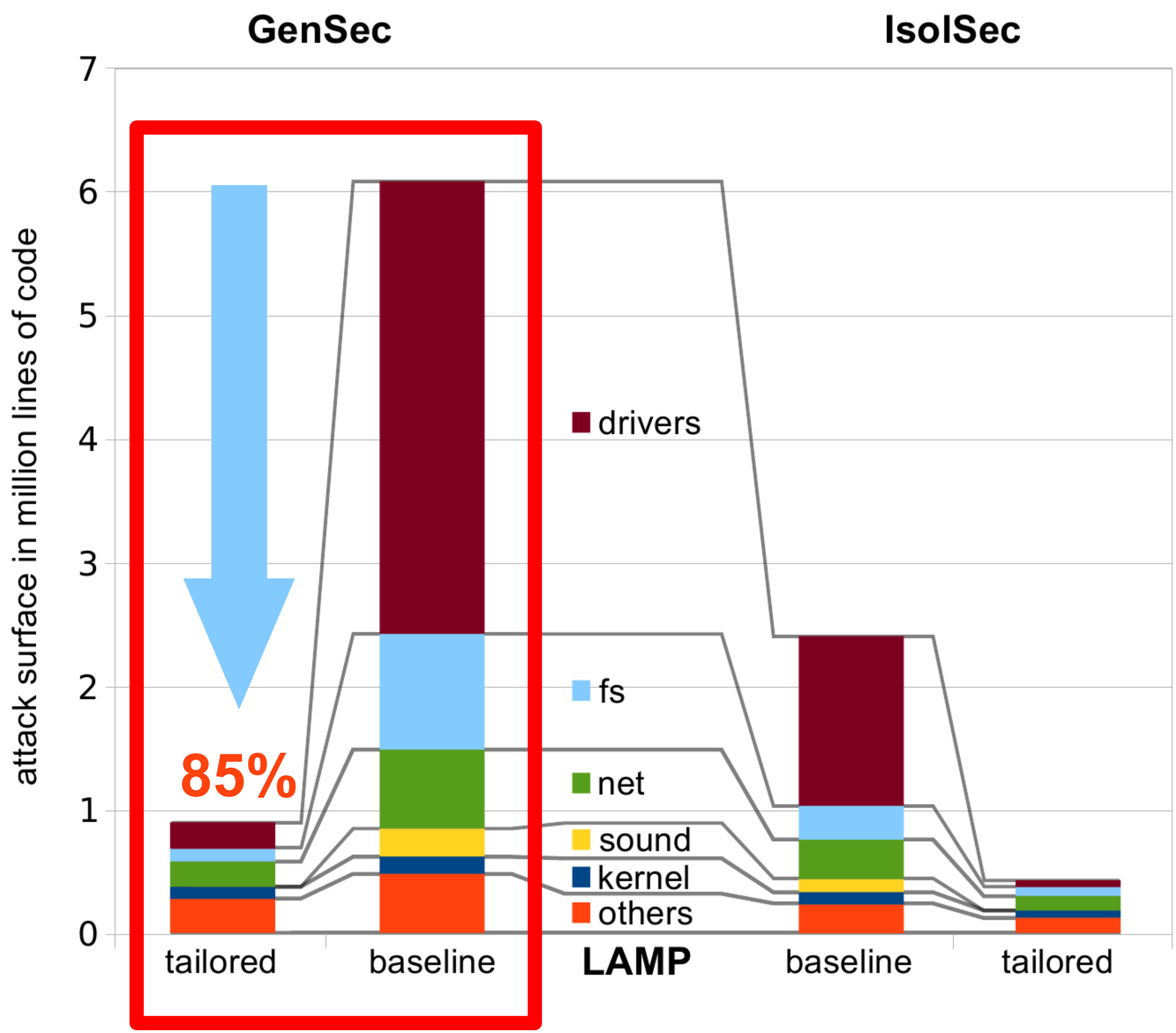
# Results: attack surface reduction



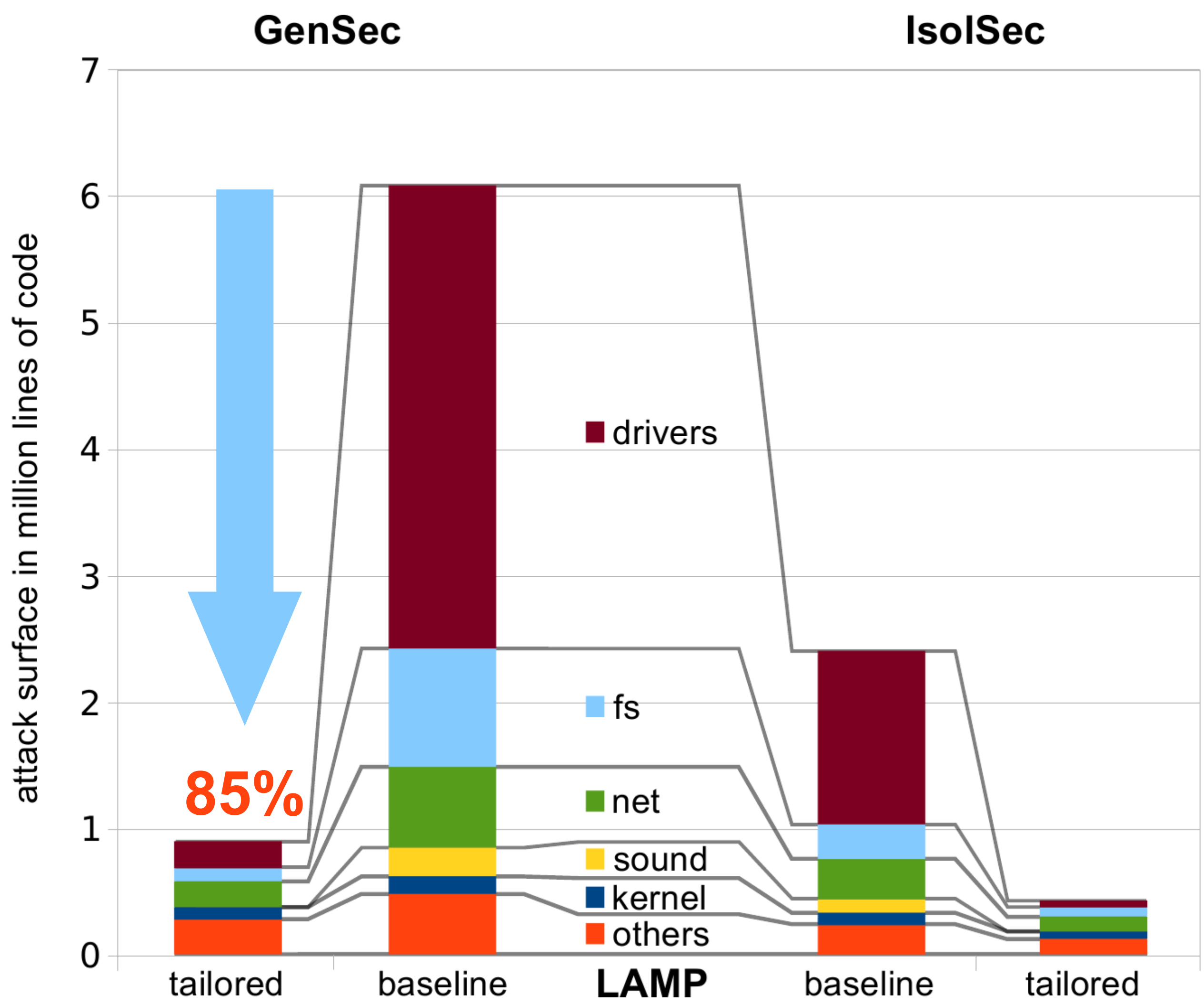
# Results: attack surface reduction



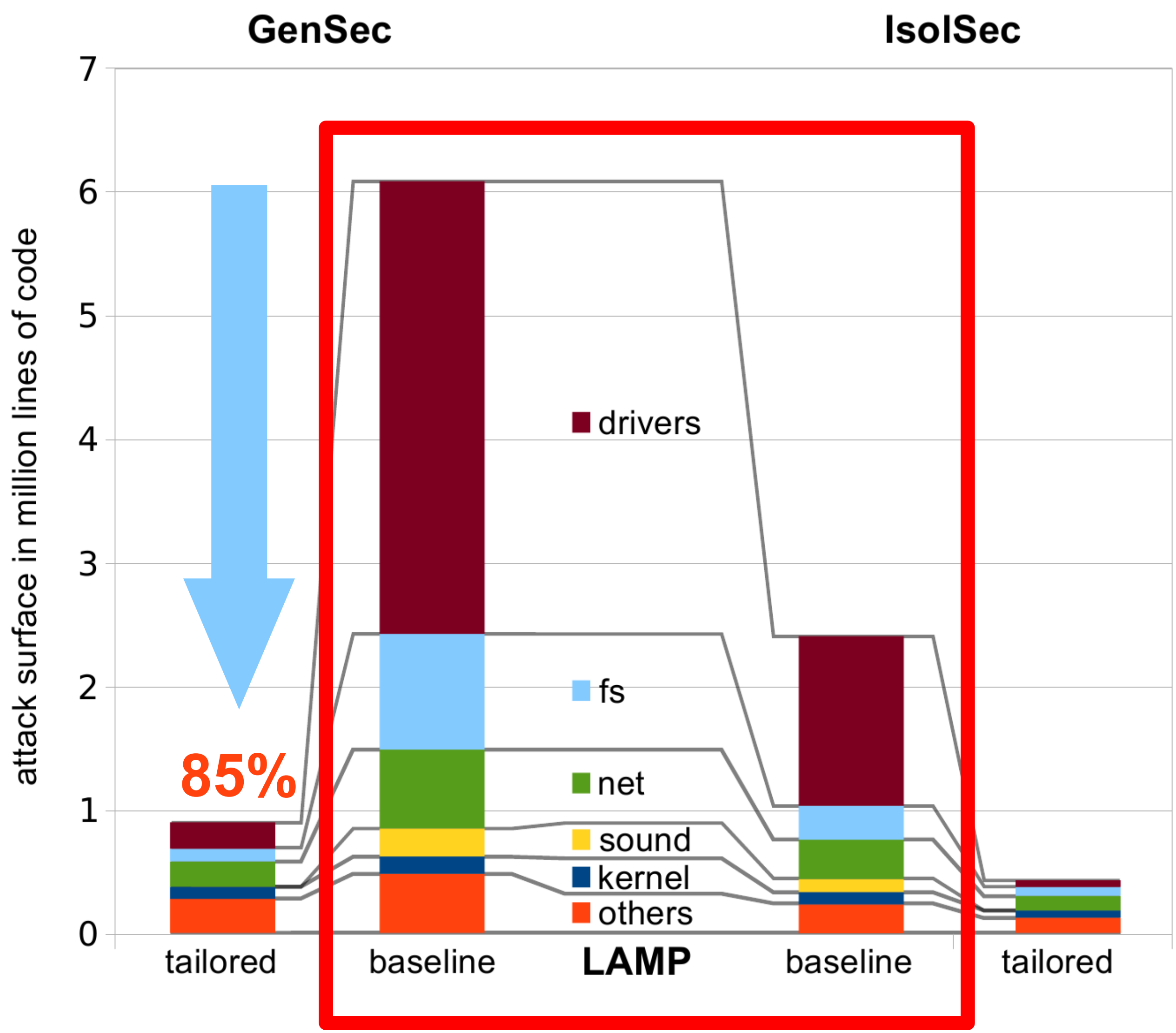
# Results: attack surface reduction



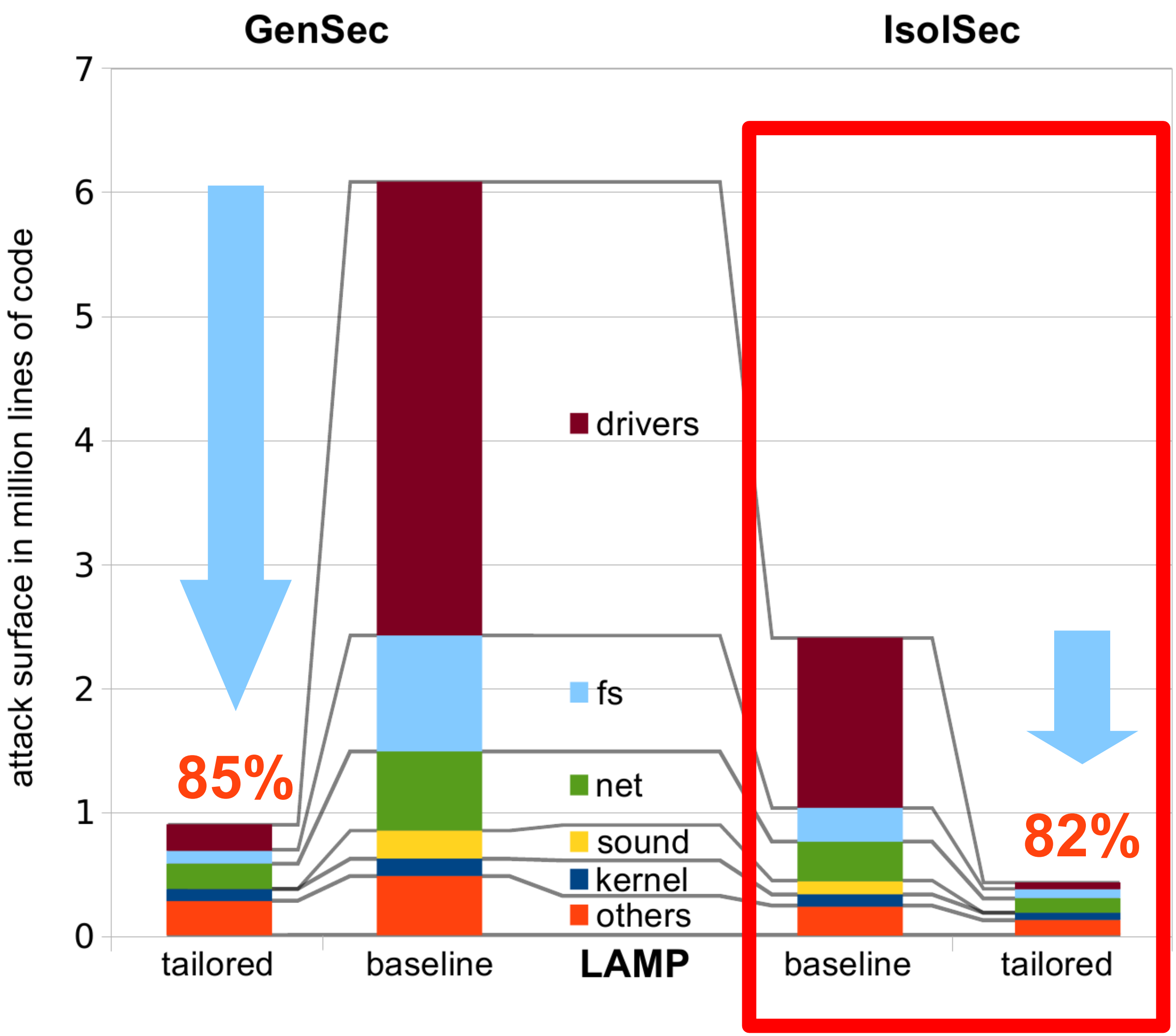
# Results: attack surface reduction



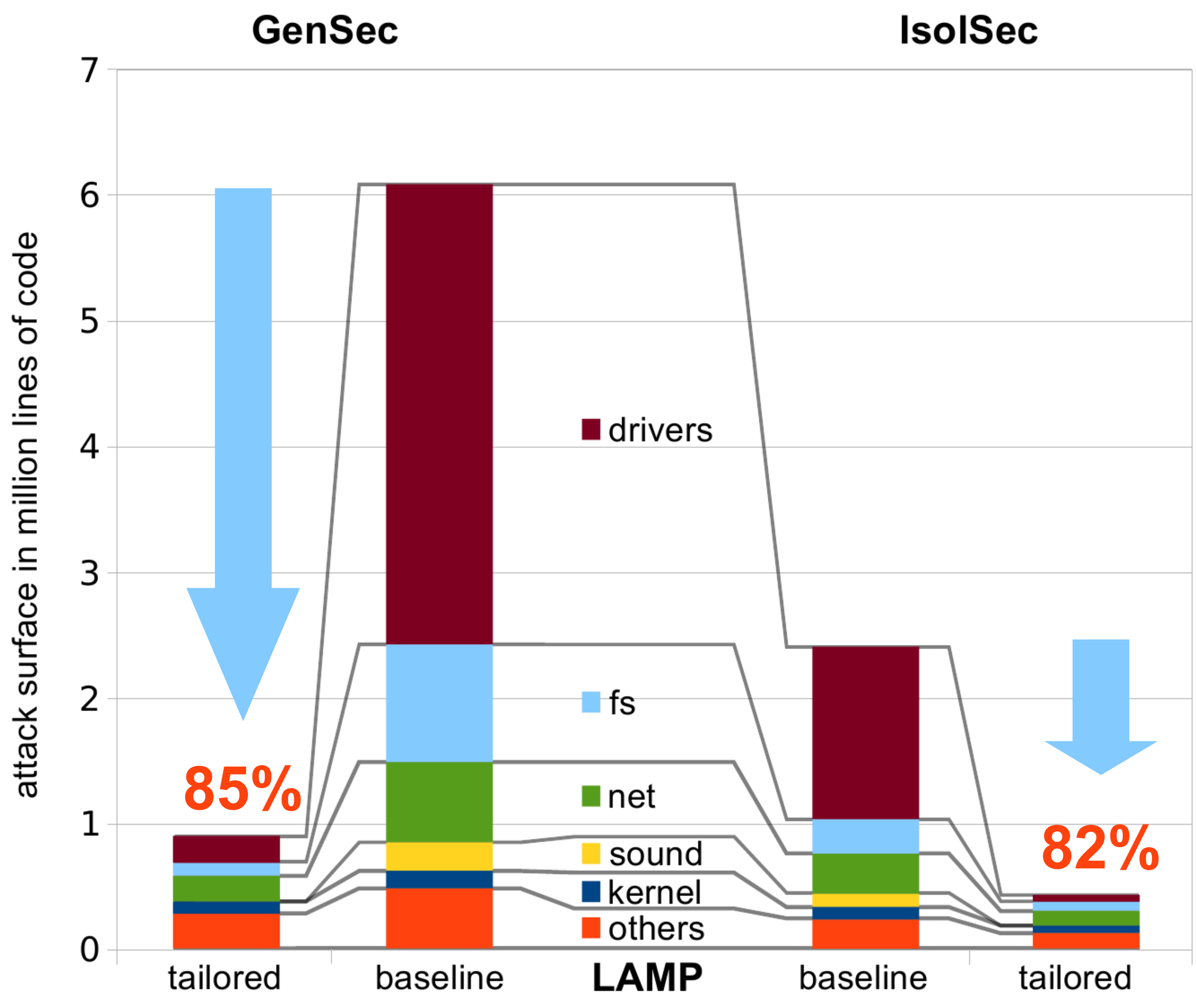
# Results: attack surface reduction



# Results: attack surface reduction



# Results: attack surface reduction



# Run-time Kernel Trimming

- [DIMVA'14] Anil Kurmus, Sergej Dechand, and Ruediger Kapitza. "**Quantifiable Run-time Kernel Attack Surface Reduction**". In: Proceedings of the 10th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA'14). 2014.

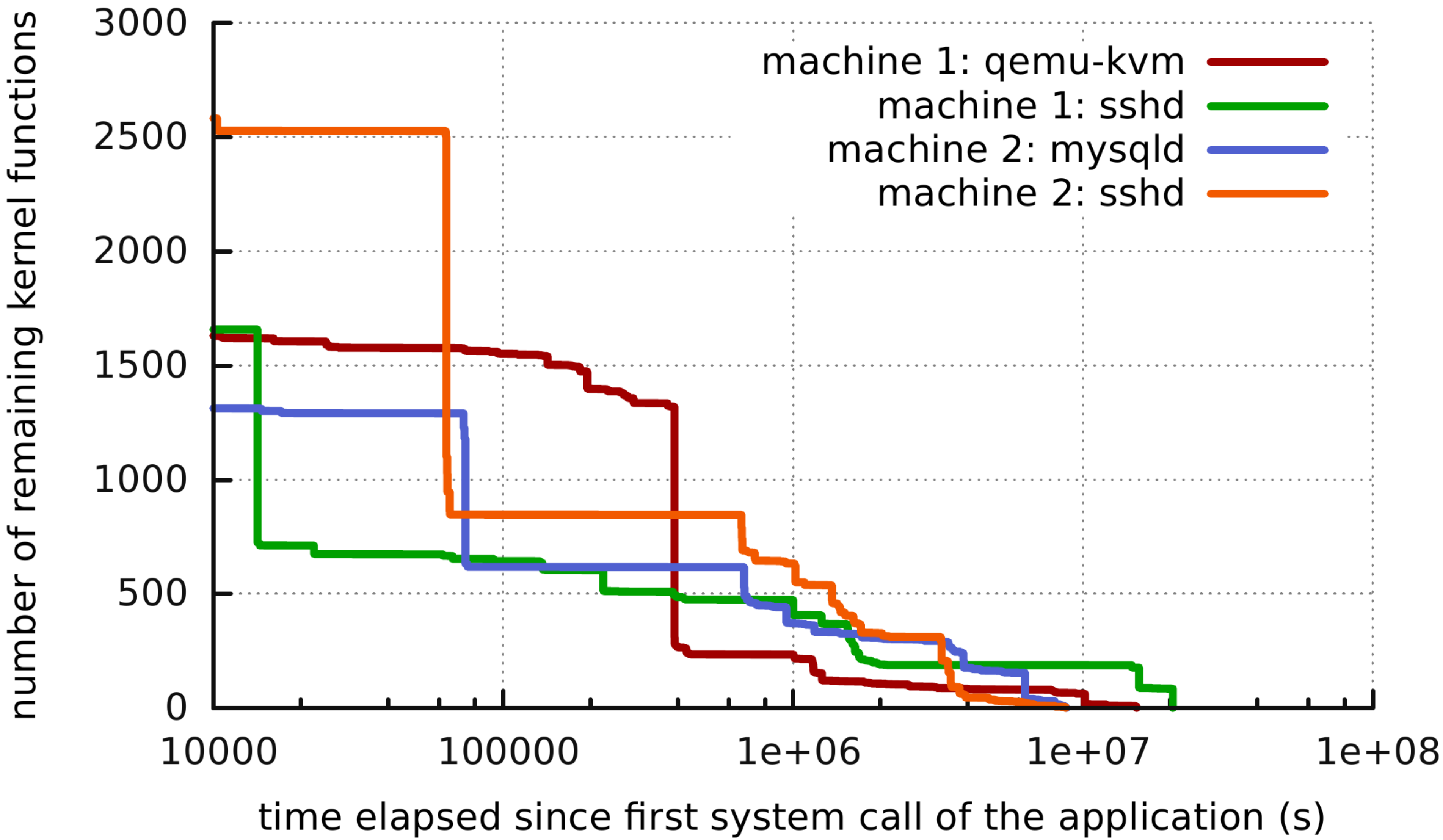
  https://www.ibr.cs.tu-bs.de/users/kurmus/papers/kurmus-dimva14.pdf

- [CCS'14] Anil Kurmus, and Robby Zippel. "**A Tale of Two Kernels: Towards Ending Kernel Hardening Wars with Split Kernel**". In: Proceedings of the 2014 ACM Conference on Computer and Communications Security (accepted for publication). 2014.

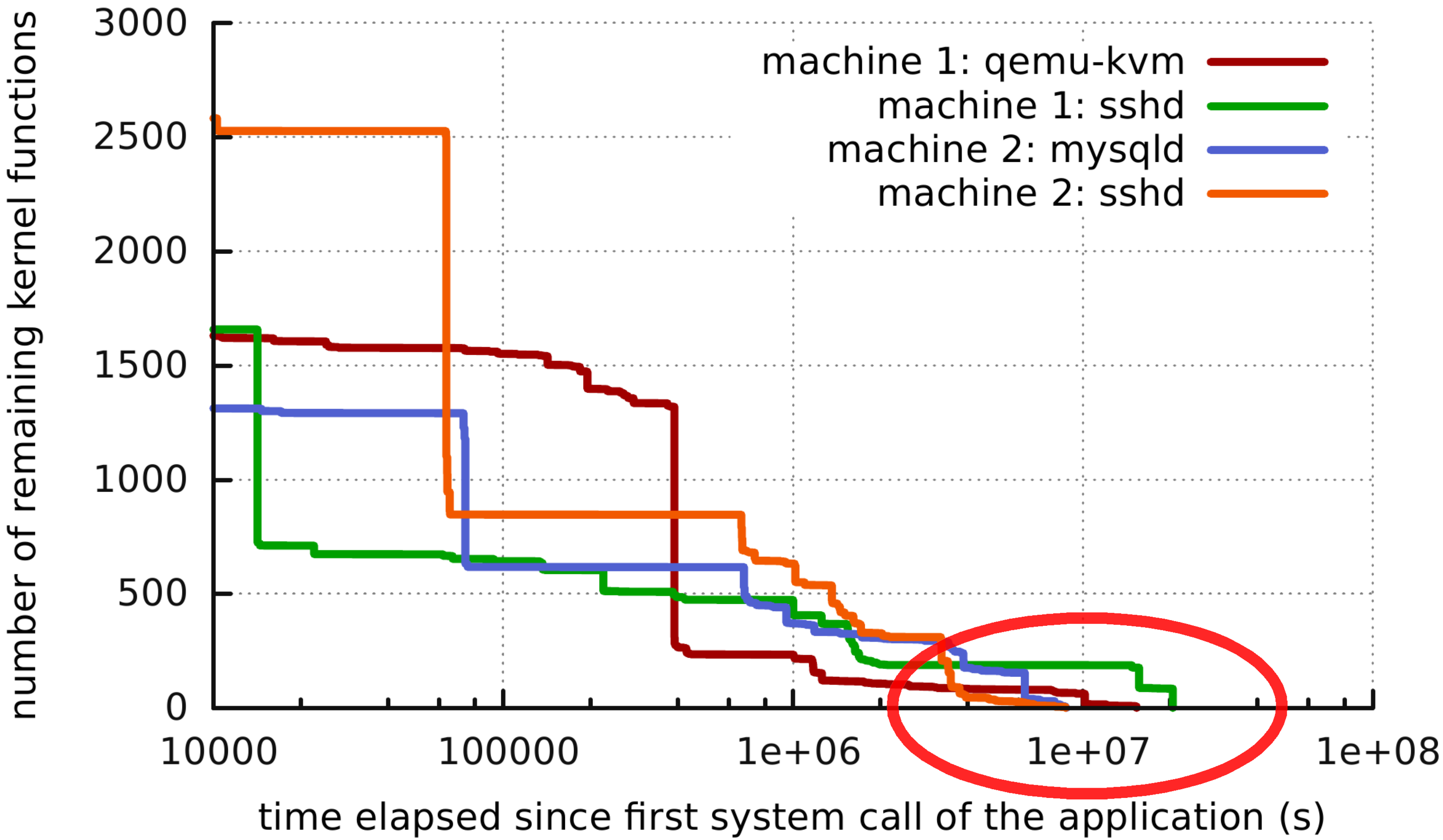
# Same idea, more attack surface reduction!

- The promises of run-time attack surface reduction:
- **More granular**
  - E.g., function-level instead of configuration-level
- **Application-specific**
  - Different application may exercise different kernel functionality
- Challenges:
  - **Performance overhead** of run-time instrumentation
  - **False positives**

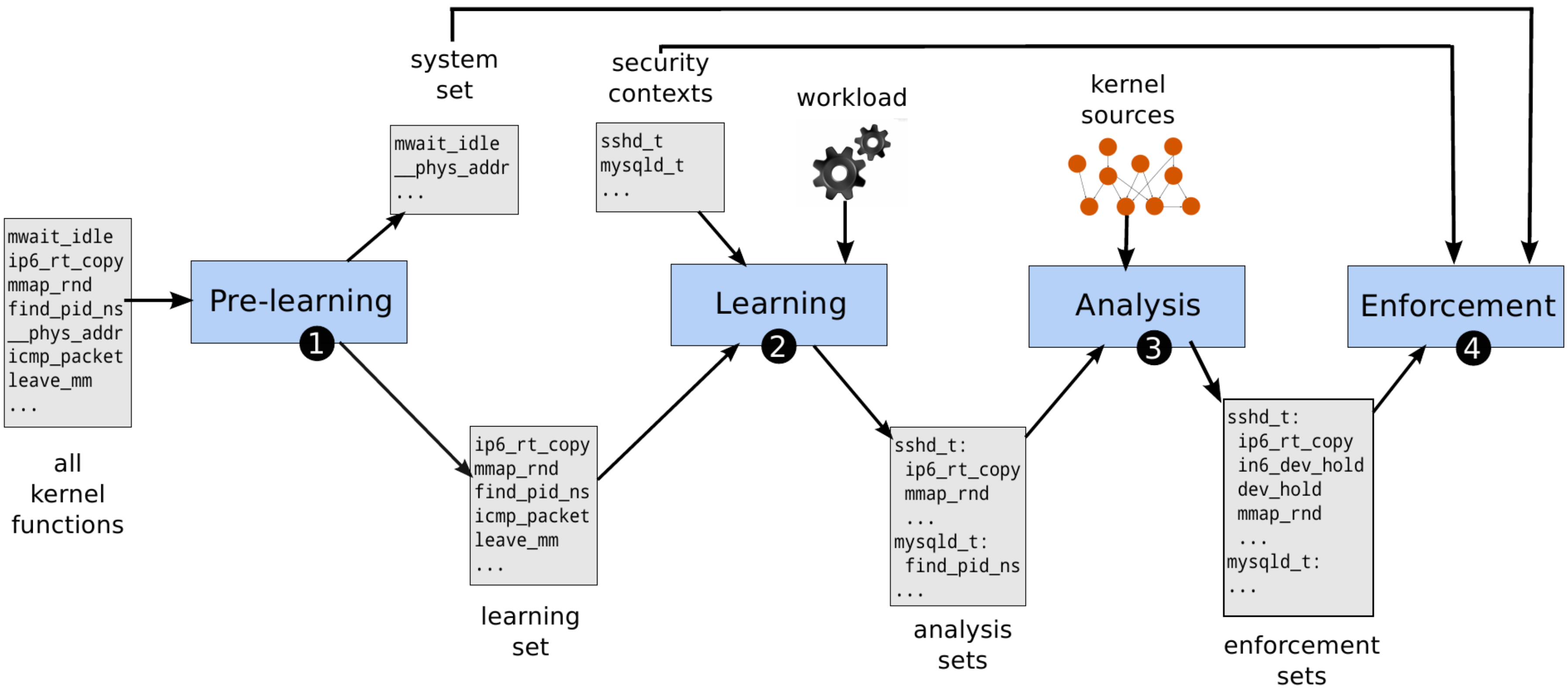
# The false positive challenge



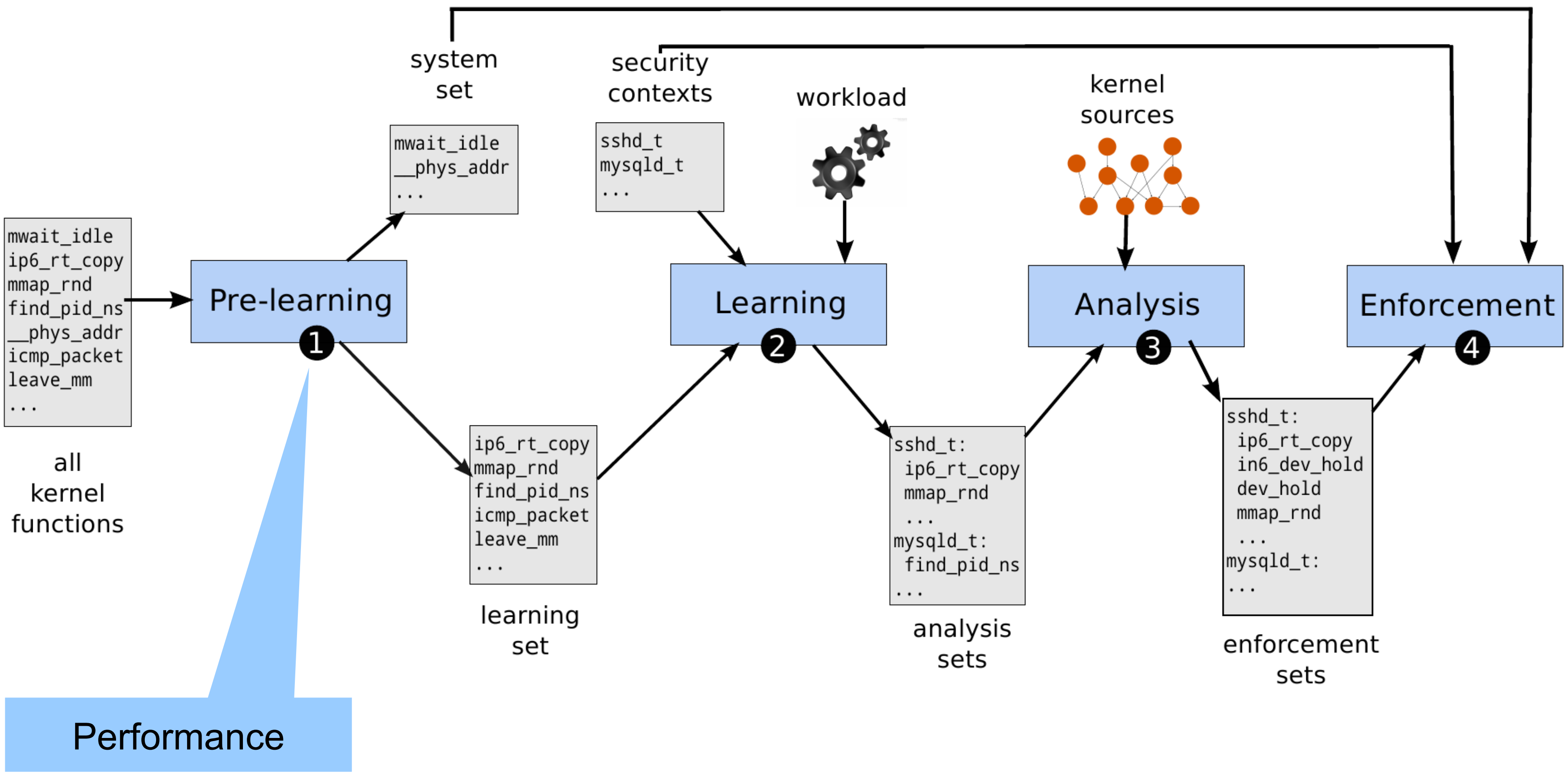
# The false positive challenge



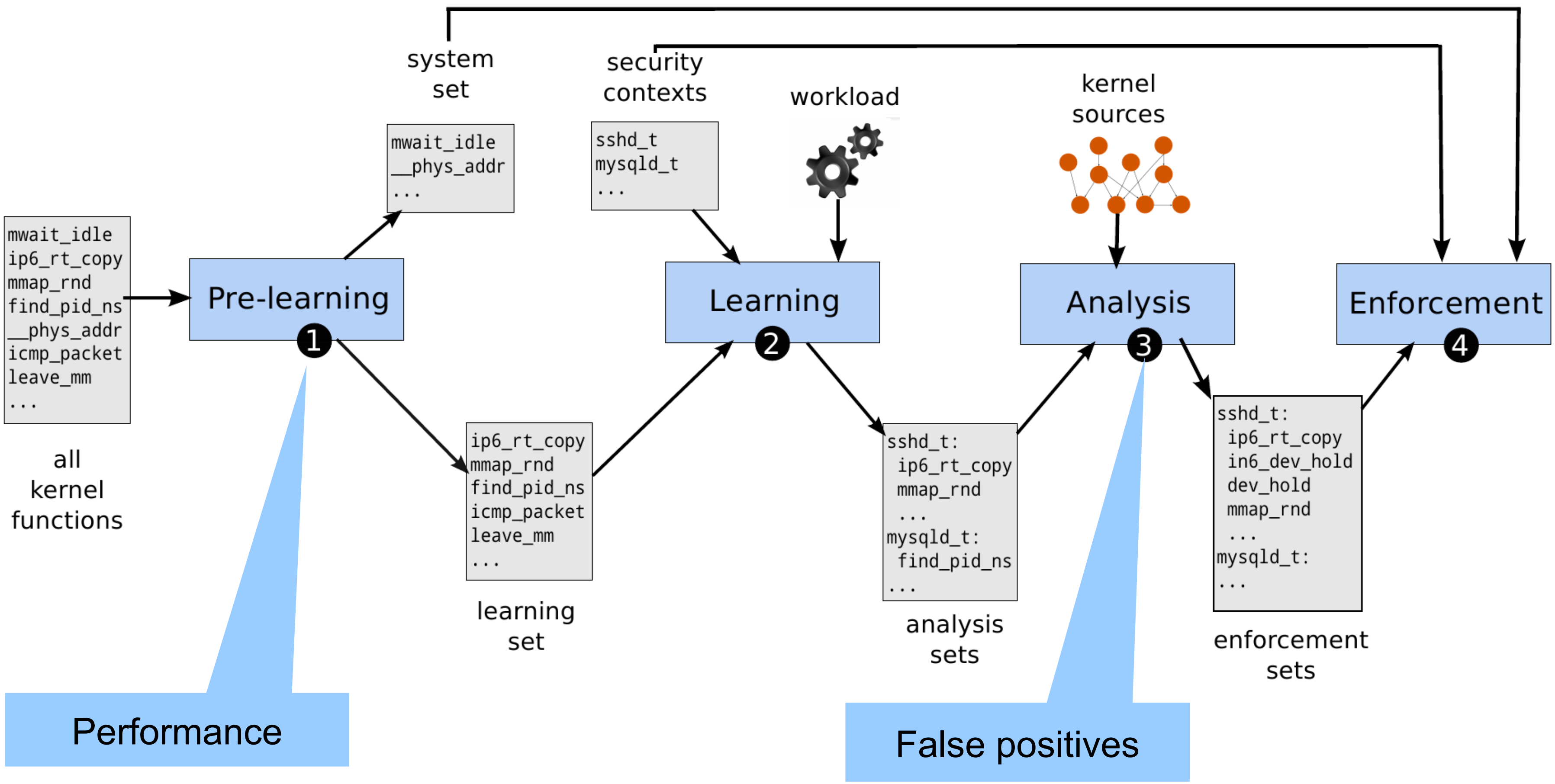
# Run-time kernel attack surface reduction



# Run-time kernel attack surface reduction



# Run-time kernel attack surface reduction



# Phase 1: Pre-learning

- Heuristic approach to improve performance
- Functions hit with frequency above a (dynamically computed) threshold are ignored
- Example:

```
ext4_fsblk_t ext4_mb_new_blocks(...)
{
  ...
    while (ar->len && ext4_claim_free_blocks(sbi,
              ar->len)) {
      /* let others to free the space */
      yield();
      ar->len = ar->len >> 1;
    }
  ...
}
```

**Pre-learning reduces performance overhead**

# Phase 3: Analysis

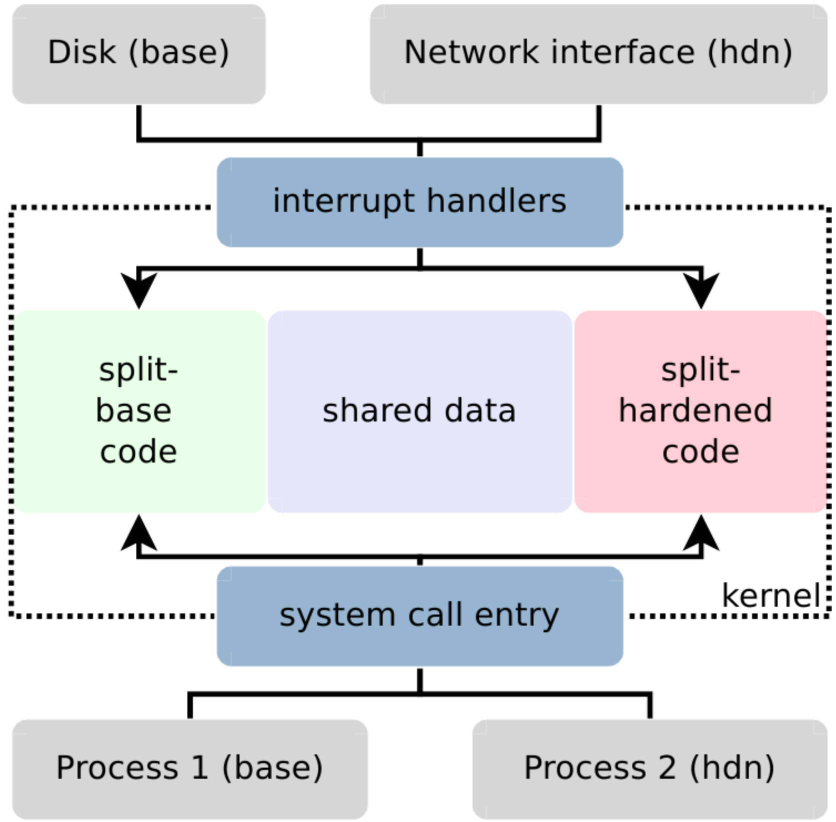
- Group functions together to reduce false positives
- 4 different modes
  - No grouping
  - File grouping
  - Directory grouping
  - Cluster grouping

# Phase 4: Enforcement

- Can't terminate process
  - False positives
  - Shared kernel state
- Two choices:
  - Logging (IDS)
  - Hardened mode enforcement via split kernel [CCS'14]

# Split Kernel overview

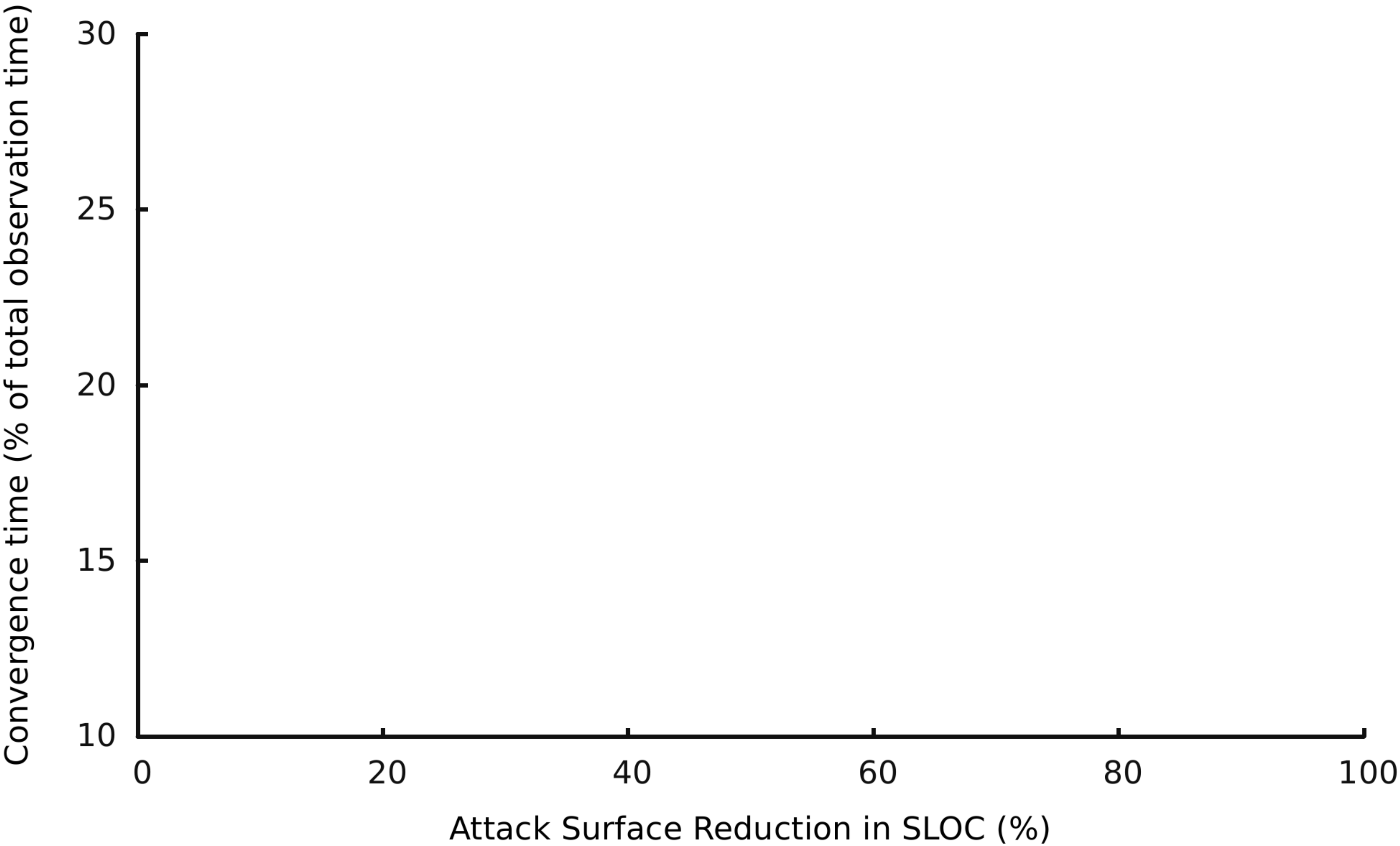
- Build kernel with and without hardening
- Chose at run-time whether to run in hardened mode
- Performance impact of hardening greatly reduced



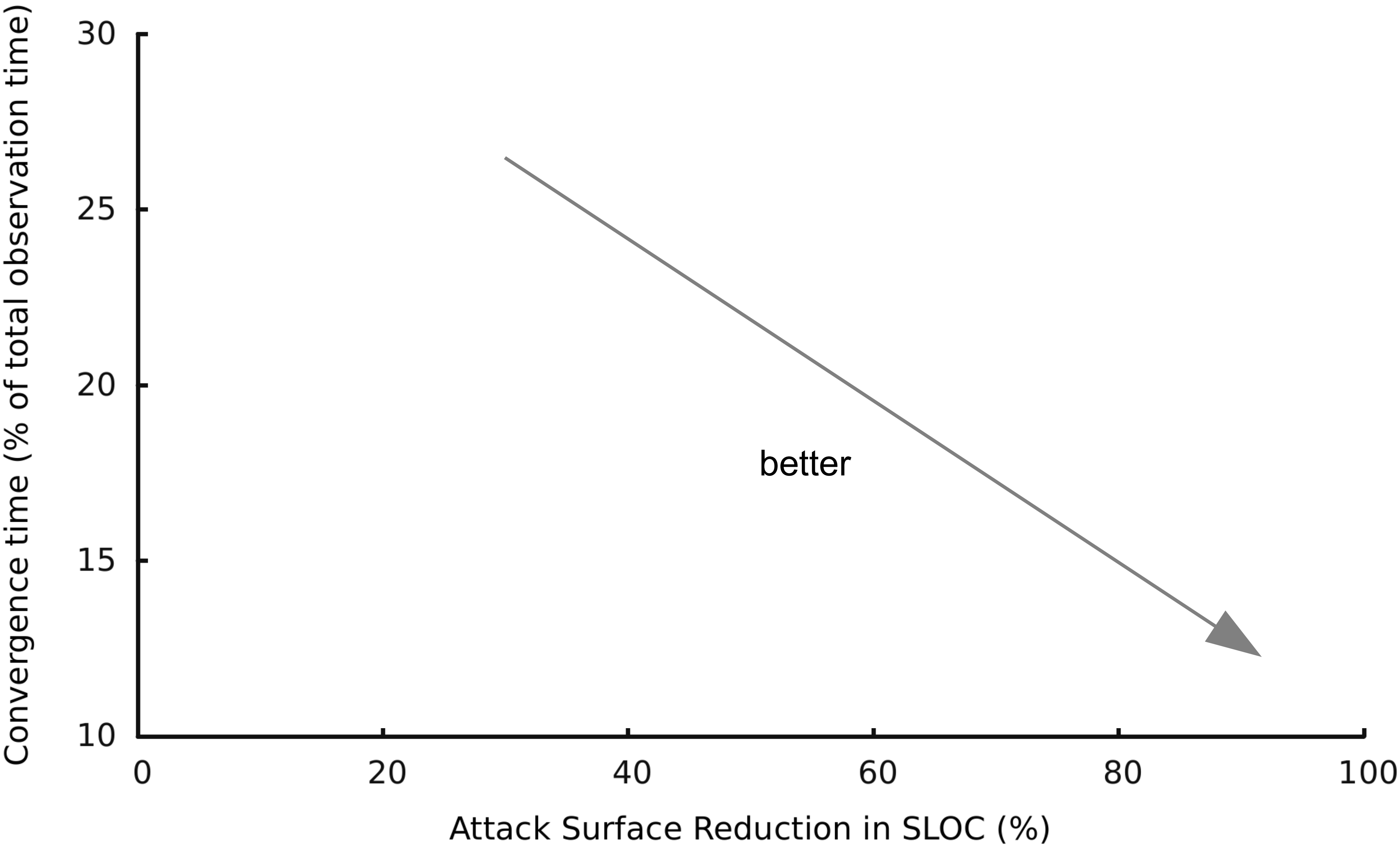
# Selected results of the evaluation

- Real-world workload on RHEL 6 development server
  - Total observation time: 403 days

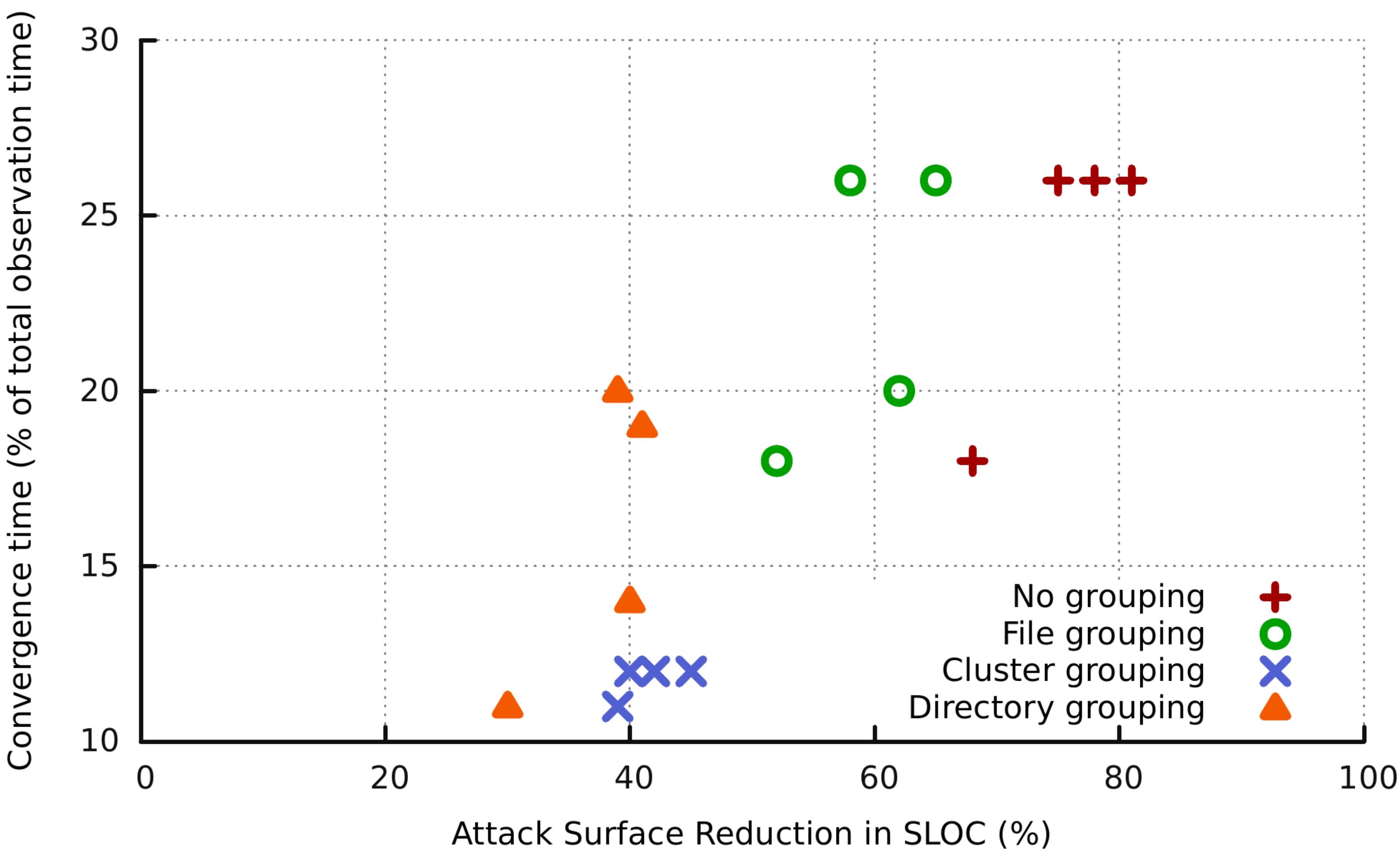
# Attack surface reduction vs. convergence rate



# Attack surface reduction vs. convergence rate



# Attack surface reduction vs. convergence rate



# Conclusion

# Conclusion

- The kernel attack surface can be quantified
- This can be used to evaluate the effectiveness of kernel attack surface reduction
- Kernel attack surface reduction is effective in preventing exploits:
  - Compile-time Tailoring
    - Prevents 285 CVEs out of 485.
  - Run-time Trimming
    - Prevents up to 184 out of 262 CVEs.
    - In general, better ASR but lower convergence rate
- Both mechanism aim to be practical
  - no significant overhead
  - non-intrusive

# References


- [Eurosec'11] Anil Kurmus, Alessandro Sorniotti, and Ruediger Kapitza. "**Attack Surface Reduction For Commodity OS Kernels**". In: Proceedings of the Fourth European Workshop on System Security. 2011.
- [NDSS'13] Anil Kurmus, Reinhard Tartler, Daniela Dorneanu, Bernhard Heinloth, Valentin Rothberg, Andreas Ruprecht, Wolfgang Schröder-Preikschat, Daniel Lohmann and Rüdiger Kapitza. "**Attack Surface Metrics and Automated Compile-Time OS Kernel Tailoring.**" In: Proceedings of the 20th Network and Distributed System Security Symposium. 2013.
- [DIMVA'14] Anil Kurmus, Sergej Dechand, and Ruediger Kapitza. "**Quantifiable Run-time Kernel Attack Surface Reduction**". In: Proceedings of the 10th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA’14). 2014.
- [CCS'14] Anil Kurmus, and Robby Zippel. "**A Tale of Two Kernels: Towards Ending Kernel Hardening Wars with Split Kernel**". In: Proceedings of the 2014 ACM Conference on Computer and Communications Security (accepted for publication). 2014.