Understanding Linux Lists

The tool for looking at kernel lists: https://gitlab.inria.fr/lawall/liliput

TCB safety
Thread Control Block. struct that holds thread data; safety critical applications of accidental tampering by priv. Or non-priv. users.

https://linuxplumbersconf.org/event/7/contributions/699/attachments/539/1089/TCB_Safety_v1.0_final.pdf

CONFIG_HARDENED_USERCOPY

Memprotect()

Use hypervisor to trap: prevents wild pointer failure mode

Use CRC to detect change between switching out and before switching back in

Safety in processes CPU execution state

Task isolation
Concern is that some of the task sanitizers are rather heavy.

Codethink mitigation
wraps system calls in safe layer - different from seccomp?
Doesn't cover 100% of the code - entry_64.s not covered
Tested with getpid() call
Significant overhead in system calls
qemu is used as a test tool (virtual vs. real hardware differences issue)
Upstream acceptability is a concern

Going forward: More user-space mitigations, partial task isolation

Testing is time consuming

Assessing kernel system call correctness by testing

Linux running on several systems and we should be harvest the assurance data ... This is very difficult to do.
Systems are different and failure modes are different

Testing is the second issue. Can we achieve completeness of testing. With Linux it is hard to answer.. Linux is complex and non-deterministic.
It is hard to put a multi-core system in a deterministic state. Async events from outside and inside the system at play.

Because of the non-deterministic nature of the kernel, it is hard to repeat an operation in a deterministic way.

A simple system call could take detours depending on the global kernel state - possibly slow paths, memory reclaim could happen in between for example. The path is not the same from one execution of the call to another.

The test case with monitoring the system state. Using a random background load to monitor "hidden state".

The key question is "independence"
Is the test methodology valid? The test strategy depends on the assumption that the system calls can be independently analyzed.

Will 2 reads in succession influence each other.

syzkaller generates random sequence of system calls. What this test is doing is opposite of what syzkaller does. The system calls are called with correct parameters and hence work.

This isn't a kernel development tool. It is an assurance tool.

Background load has intersections with the system call.

http://www.osadl.org/SIL2LinuxMP.sil2-linux-project.0.html

**Maintaining results from static analysis collaboratively?**

Sharing info to help with identification of false positives.

Open question: after running sparse and cocciinelle, there are 1000's of warning. Most which can't be silenced.

**KCIDB -**

Kernel CI - is aggregating. How to collect results in common way is already in progress. Run time from static analysis differing.

kunit & static analysis. Talk about this tomorrow. BOF later in the week.

Common shared format - needs to be agreed on.

Heridoto tool during 5.6 tools, using cocciinelle was done http://coccinelle.lip6.fr/herodotos/herodotos.php

Pick this up in KernelCI talk tomorrow. Community has started to gather there.
Following the Linux Kernel Defence Map

stackleak is part of the kernel since 4.20

https://github.com/a13xp0p0v/linux-kernel-defence-map

Stack protector mitigates some return address overwrite attacks, but not in all cases.

The last slide has links to references

https://github.com/a13xp0p0v/kconfig-hardened-check

Recommendations for configs are provided by different groups, and they are listed in kconfig-hardened-check

https://github.com/clipos

Some recommendations for kernel harden check come from kspp, clipos, defconfig, lockdown and reducing attack surface discussions, userspace hardening work/discussions, direct feedback from maintainers

One company used it with security tool without giving credit.  Not good.

see also https://www.kernel.org/doc/html/latest/arm64/pointer-authentication.html

Linux Kernel Defence Map
Kernel hardening check
Don't change config without understanding attack surface for your system

https://github.com/KSPP/linux/issues/14 ?

https://github.com/a13xp0p0v/kconfig-hardened-check/issues/44

Linux Kernel dependability - Proactive & reactive thinking

Challenge is how to shift focus to more proactive designs.

4000 contributors per year into the kernel, how can we get this resolved better - getting good commit logs are hard.

Avoiding Security Flaws
Security Flaws are just Bugs.