LINUX
PLUMBERS
CONFERENCE
August 24-28, 2020

Linux Kernel dependability - Proactive & reactive thinking

Shuah Khan
Linux Kernel Fellow
The Linux Foundation

@ShuahKhan

skhan@linuxfoundation.og

LINUX
PLUMBERS
CONFERENCE
August 24-28, 2020

We would like our

Systems

- Available
- Deterministic
- Reliable
- Responsive
- Resilient to remote and local attacks
- Safe & Secure

Data on these systems

- Easily accessible
- Easily shareable with trusted entities
- Safe from corruption
- Secure from unwanted intrusions

Bottom line, when we pick up our phones we want to be able to make calls, read news, take pictures, record video/audio and keep all of that data safe. In short – Dependable.

@ShuahKhan

skhan@linuxfoundation.og

LINUX
PLUMBERS
CONFERENCE
August 24-28, 2020

Obstacles

- Overflows
  - Heap
  - Integer overflows
  - Stack overflows
- Privileged information leak
  - kernel addresses in messages & API - sysfs etc.
- Insufficient error and boundary checking
- Out of bounds access

We are worried about being vulnerable to intentional and unintentional, remote and local user actions.

@ShuahKhan
skhan@linuxfoundation.og

LINUX PLUMBERS CONFERENCE
August 24-28, 2020

Obstacles

- Memory leaks
- Use-after-frees
- Uninitialized variable use
- Unsafe data from userspace
  - Input arguments – e.g ioctls, system calls etc.
  - In network & usb etc. packets

We don’t want kernel panics leading to out of service systems & unauthorized access leading to data leaks and losses.

Obstacles stand in the way of having highly available and dependable infrastructure & systems.

@ShuahKhan
skhan@linuxfoundation.og

Reactive thinking

- Find and fix regressions
  - Fuzzers
  - Regression tests
- Use dynamic and static analysis tools
- Scan and identify vulnerabilities
- Harden kernel code paths

Focus is on finding and fixing problems in the released code.

@ShuahKhan
skhan@linuxfoundation.og

LINUX
PLUMBERS
CONFERENCE
August 24-28, 2020

Proactive thinking

- Invest time in defensive designs
- Understand common design & coding mistakes
- Focus on detection, mitigation, testing before code release
- Use Static analysis
  - coccicheck, Sparse, Smatch etc.
  - Found gaps in tools – enhance/write new
- Use Dynamic analysis & Regression testing
  - Syzkaller, Trinity fuzzer, scripts: e.g leaking_memory.pl
  - No existing test? Write one to go with your patch.
  - Use error injection tests

Focus is on finding and fixing problems before releasing the code.

@ShuahKhan
skhan@linuxfoundation.og

LINUX PLUMBERS CONFERENCE
August 24-28, 2020

Proactive designs

- Avoid leaking kernel addresses in kernel messages
- Avoid exposing kernel addresses in user API
- Error check input arguments from user-space
- Boundary (range) check input arguments from user-space
- Sanitize input arguments from user-space before use
- Pay attention to error and cleanup paths
- Avoid repeating mistakes with the use of common helpers
  - When a helper doesn’t exist write one
- Kernel wide scope – is this a common problem across subsystems?

@ShuahKhan
skhan@linuxfoundation.og

Be mindful of error and cleanup paths

LINUX PLUMBERS CONFERENCE
August 24-28, 2020

- Init and run-time paths can be easier to verify
- Error and cleanup paths are prone to
  - Memory leaks due to not releasing resources
  - Unbalanced lock acquire/release leading to potential deadlocks
- Enable debug config options to verify prove locks, locking.
  - e.g: CONFIG_DEBUG_SPINLOCK, CONFIG_PROVE_LOCKING
- Enable debug options to check for use-after frees and memory leaks
  - CONFIG_KASAN, CONFIG_KCSAN, CONFIG_KMSAN, CONFIG_UBSAN

@ShuahKhan

skhan@linuxfoundation.og

LINUX
PLUMBERS
CONFERENCE
August 24-28, 2020

Connect the dots for effective testing

- Adding kcov hooks for collect coverage & facilitate coverage-guided fuzzing with syzkaller.
  - Reference: Linux 5.8
    kcov: collect coverage from usb soft interrupts
    work by Andrey Konovalov – extends kcov to allow collecting coverage from soft interrupts and then uses the new functionality to collect coverage from USB code.

@ShuahKhan
skhan@linuxfoundation.og

Regression test

- Regression test - Kernel Selftests and other tests for regression
- Run fuzz tests - syzbot reproducers
  - Linux Arts (Linux Auto-generated Regressions Tests) Repo
- Scan for vulnerabilities

@ShuahKhan
skhan@linuxfoundation.og

Concurrency

- Race Condition Enabling Link Following: Race condition between file/dir status check and access. Related to TOCTOU and DAC
  - Detection – KCSAN (?)
  - Mitigation
    - Time-of-check Time-of-use (TOCTOU) Race Condition (seccomp)
    - Discretionary Access Controls (YAMA)

@ShuahKhan
skhan@linuxfoundation.og

Concurrency

- Signal handler race conditions
  - Detection – KCSAN (?)
  - Mitigation
    - CONFIG_SIGNALFD: Allow receiving signals on file descriptor
    - pidfd_send_signal(): enables signaling a process through a pidfd to eliminate the PID wrap resulting in sending signals to a wrong process.

Memory Buffer Errors

- Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')
- Write-what-where Condition
- Access of Memory Location After End of Buffer
- Buffer Access with Incorrect Length Value
  - Detection – coccinelle, sparse, smatch, gcc W=1
  - Mitigation – replace unbounded copy functions with safer API - e.g scnprintf() instead of snprintf()/strncpy()

@ShuahKhan

skhan@linuxfoundation.og

Memory Buffer Errors

- Buffer Underwrite ('Buffer Underflow')
- Access of Memory Location Before Start of Buffer
- Incorrect Calculation of Buffer Size
  - sparse, smatch
- Out-of-bounds Read/Write
  - Detection: Static checkers, Dynamic syzkaller tests with CONFIG_KASAN

@ShuahKhan
skhan@linuxfoundation.og

Resource Locking Problems

- Improper Resource Locking
- Missing Lock Check
- Double-Checked Locking
- Multiple Locks of a Critical Resource
- Multiple Unlocks of a Critical Resource
- Unlock of a Resource that is not Locked
- Deadlock

@ShuahKhan
skhan@linuxfoundation.og

LINUX PLUMBERS CONFERENCE
August 24-28, 2020

Resource Locking Problems

- Coccinelle: missing unlocks, double locks, find improper lock API usages) e.g: holding locks in paths that require no lock holds.
- Kernel Lock Torture Test Operation locktorture test
- Locking API boot-time self-tests
- Syzkaller

@ShuahKhan
skhan@linuxfoundation.og

References

- CWE CATEGORY: Concurrency Issues
- CWE CATEGORY: Memory Buffer Errors
- CATEGORY: Resource Locking Problems

@ShuahKhan

skhan@linuxfoundation.og

Bringing it all together

- Promoting & incorporating proactive thinking
- Identify detection/mitigation
  - Static analysis (static checkers + compilers)
  - Dynamic analysis (test tools + config + features)
- Extend and write new detection tools as needed
- Harden framework/code for identified gaps
  - e.g work: pidfd_send_signal(), seccomp(), %n, scnprintf() use etc.
- Others

@ShuahKhan

skhan@linuxfoundation.og