Kernel side channel security project

Kristen Accardi

https://www.flickr.com/photos/thomasleuthard/31155135274/in/photostream/

Mission Possible

Help prevent future side channel attacks on the kernel, and harden the kernel against other potential exploits.

★ *not addressing specific CVEs or security gaps*

A few of our projects

- Make existing kernel address space randomization finer grained.
- Implement module address space randomization
- Allow security modules to selectively apply security mitigations when switching tasks
- Allow applications to protect memory areas containing secrets
- Remove cache breadcrumbs when returning error from system calls

Kernel boot address randomization

Objective:

Increase kernel address space randomization by relinking the kernel object files at every boot.

Description:

- Leverage existing module loading code as in kernel linker
- Break the kernel up effectively into many modules and possibly leverage our randomized module text/vmalloc allocation algorithm.
- Create new section for modules to load at boot time pre-fs.
- Modify startup to link in modules at boot

Status:

Still in the Research phase, not committed yet to a design.

Possible Benefits

- KASLR increases the difficulty of side channel attacks
- KASLR is already merged, and this strengthens what we've already adopted

Possible Challenges

- Even fine grained KASLR can be worked around
- A single vulnerability in a module might be exploited to find the kernel
- Increases complexity and reduces reproducibility of bugs.

Module text randomization

Objective:

Improve kernel address space randomization for module text sections

Description:

- Split the 1GB module text range into 2 - a randomized(fragmented) area, and a linearly allocated area.
- Randomize each loaded module with respect to each other
- If we fail to find adequate space in the randomized area, fall back to original algorithm in the linearly allocated space.

Status:

Under review upstream

Possible Benefits

- Better load time performance in randomized memory locations
- Increased randomness (17 bits)
- If one module address leaks, the others cannot be immediately inferred

Possible Challenges

- One address leak in a single module might be sufficient for an exploit
- Increased memory usage
- Increases complexity and reduces reproducibility of bugs.

Protect pages with secrets

Objective:

Allow user space applications to indicate that a page contains a secret that needs special protection

Description:

- Add a new flag to the mlock2() syscalls: MLOCK_SECRET
- Apply mitigations to both the memory area, and the process that mapped it
- Mitigations may include: make not dumpable, do not copy or fork, disable caching

Status:

PoC under development

Remove cache breadcrumbs

Objective:

Make it harder to perform cache timing attacks on data left behind by system calls

Description:

- When a system call would return an error, randomly perturb the cache before returning back to userspace
- Only apply to specific set of errors (for example, -EPERM)

Status:

PoC under development

Possible Benefits

- Cache contents will not be as easy to guess
- Pretty simple implementation

Possible Challenges

- Performance will be impacted in the case of an error - assumption is that errors are not the fast path.
- Increased memory consumption with current PoC

LSM interface for side channel mitigations

Objective:

Make it possible for an LSM module to determine if a particular security mitigation can be applied

Description:

- Create a LSM Hook which will allow a module to recommend whether to apply IBPB when switching tasks.
- Make a side channel LSM
    - Check effective UID, capability sets, or namespaces
    - Option to just always apply

Status:

PoC under development

INTEL OPEN SOURCE TECHNOLOGY CENTER | 01.org

@kcaccardi