This paper described the use of a static analysis tool called CQUAL to verify the placement of LSM (Linux Security Modules) authorization hooks. LSM is a framework consisting of a set of hooks for implementing flexible access control policies in the Linux kernel. This allows writers of kernel modules to define and choose the access control policy that is best for them. To use LSM, the programmer inserts a call to the appropriate LSM hook before calling a function that operates on security-relevant kernel objects. Since the placement of LSM hooks is done manually, it is vulnerable to programmer error, and improper hook placement could lead to security vulnerabilities. The authors first used runtime analysis for verifying LSM hook placement, but then turned to static analysis because it could provide better test coverage and is more automatic. Their aims are to verify “complete mediation,” which means that an LSM authorization occurs before any controlled operation is executed, and “complete authorization,” which means that each controlled operation is completely mediated by hooks that enforce the required authorizations.

The authors selected CQUAL as their best option from a number of existing static analysis tools. To use CQUAL, the programmer specifies a lattice of type qualifiers, and CQUAL then uses a qualifier inference algorithm to detect violations of the qualifier relations in the lattice. For LSM verification of complete mediation, the type qualifiers are “checked” and “unchecked.” A variable’s qualifier starts as unchecked when it is initialized, and must be checked by the time a controlling function is called. A variable changes from unchecked to checked whenever a security check (authorization) is performed on it. CQUAL’s task is to ensure that this happens for all necessary variables on every program path from an initializing function to a controlling function. For complete authorization, the qualifier lattice need only be extended to include unique qualifiers for each necessary security check.

In order for this to work, the original program must be annotated with the appropriate qualifiers. This is too large and tedious a task to be done manually, so the authors used a modified version of GCC to find where annotations needed to be added, and then input this information to a Perl script that actually added the annotations to the source code.

The paper includes an evaluation in which four subsystems of Linux are checked: file system, virtual memory, networking, and inter-process communication. The analysis found 524 conflicts, which could be divided into three categories: inconsistent checking and usage of controlled object variables, controlled objects modified without security checks, and kernel initiated operations bypassing security checks. At least one of the errors caused a race condition that the authors have demonstrated to be exploitable. They also provided a fix. The problem with the analysis is that many of the errors found are false positives. Fortunately, the authors could remove many of these by manually identifying “safe” functions that do not lead to vulnerabilities, and then eliminating the errors that involve these safe functions. There are also other possible ways to reduce false positives, but these would involve some modifications to CQUAL.

This paper was well written and presented some interesting work. The authors stated that they were surprised that the analysis caught so few real errors, and I also find this surprising. Perhaps since their use of this technique is new and probably not very sophisticated yet, the analysis is just not capable of catching all errors. The rate of false positives is also somewhat alarming, though their initial method of removing false positives seems quite effective, and the additional filtering that they propose would also probably help. I liked how they were able to do the code annotations largely automatically, as this relieves programmers of a big burden. Overall, this was a neat application of static analysis techniques, and with further refinement, I think it could be quite valuable in catching bugs.