Loren Taylor  
Using CQUAL for Static Analysis of Authorization Hook Placement

This paper discusses how CQUAL, a simple type-based static analysis tool, can be used to verify correctness of hook placement for the Linux Security Model (LSM). The LSM has been coded by manually placing hooks in the kernel, which is subject to human error. A properly placed hook ensures complete mediation, and complete authorization. The authors first considered a run-time analysis tool which required manual verification of results, so they tried a static analysis tool, CQUAL.

The paper segments objects as either being controlled or not. These include files, inodes, sockets, and a few other important types. If an object is controlled, then a security check must be performed on the object before it can be used in any controlled operation.

The authors propose a high-level seven step process for solving the complete mediation problem of a controlled variable. This process is very similar to the analyses performed by the optimizer in some compilers. Step 5 of the algorithm is along the lines of computing liveness of registers in basic/superblocks in compilers. This appealed to me since I have taken EECS 583 and can recognize that the algorithms for program optimization can be applied in other scenarios, namely LSM hook verification.

I was surprised that the simple lattice of $checked < $unchecked was all that was needed for the authors to implement their changes to the kernel. This solution is brilliant and simple at the same time. All that is needed is to change the parameters of controlled operations to take in checked objects. CQUAL would then catch any call to a checked operation with unchecked objects and report the warning/error.

The authors used PERL scripts to insert most of the un/checked qualifiers; however, they do mention that they still had to insert some by hand, which could introduce errors. Another limitation is that CQUAL cannot specify type requirements at the statement level; however, function parameters can be qualified. The fix proposed by the authors addresses the intra-procedural analysis issue by issuing warnings if controlled operations are performed on local variables. I don’t think this is a good fix since it still requires human interaction and verification.

I thought the paper’s explanation of the exploit found in Linux 2.4.9 was well done. The walkthrough of exactly how the file locking function can be exploited because a race conditions exists when retrieving a file pointer shows just how useful this tool can be.

One problem that the paper points out with CQUAL is the vast number of false positives that it generates. Their fix to this was not ideal since they had to go through and manually identify “safe” functions. They did not suggest an automated approach citing that manually identifying these functions, though painful, is manageable. I would have like to have seen a suggested automated approach. They do make up for this by suggesting an extension to CQUAL that would enable the correct verification of the type of a member of a structure.
I enjoyed this paper because of how the methods used relate to EECS 583. The detailed description of their seven step algorithm made a valuable contribution.