Review “Hook Placement”

Name: Wenhui Zhang

Paper Name：

Muthukumaran, D., Talele, N., Jaeger, T., & Tan, G. (2015, March). Producing hook placements to enforce expected access control policies. In *International Symposium on Engineering Secure Software and Systems* (pp. 178-195). Springer, Cham.

Contribution:

In this paper proposes a method of auto-hooking placements for access control.

Motivation:

Large codebases need retroactive security features, such as authorization. We need to retrofit legacy code for security purposes. Manual hook placements for authorization hooks in Linux Security Module is tedious and incorrect. Thus there is a calling for auto-hook placements.

Related works:

There are some former works in: (1) Verifying Hook Consistency and (2) Placing Hooks Automatically. We assume code are already come with hooks for verification, and there are works on verification process: (1) For Kernels [Zhang et al.,2002, Edwards et al., 2002, Tan et al., 2008]; (2) For Web Applications [Sun et al., 2011, RoleCast 2011, FixMeUp 2012]. People also conduct auto-hooking process, with input of Sensitive data types and hook code, hooks are gemerted for Server Applications [Ganapathy et al., 2005,2006, 2007].

Methodology:

Firstly, it infers security-sensitive operations through a taint analysis and analysis on security operations Control dependence analysis. In static taint analysis, it identifies variables tainted by user request and security-sensitive objects. In control dependence analysis, it identifies security-sensitive operations, and hoist and remove redundant hooks. In this way, it achieves correct, complete mediated and non-redundant hook placement.

Results:

It performs hook placement on X Server, postgres, pennmush and memcached, and saves 90% human effort while ensuring correctness. Comparing to Manual Hooks, this approach is more fine-grained. Such as on X Server (version 1.9 with XACE hooks), manually done with 207 hooks, while automated approach has 532 hooks. Etc.

Take away:

For analyzing to get authorization constraints, we could borrow the idea of data flow analysis from compiler class. There are two approaches: (1) Top-down and (2) Bottom-up. In top-down approach, programmers propose placement and we compute authorization constraints, programmers choose authorization constraints. In bottom-up approach, we start from any placement, such as computed default, compute constraints relative to that placement, select a group of constraints that satisfy a high-level constraint automatically.