Mining Security-Sensitive Operations in Legacy Code

Vinod Ganapathy, Dave King, Trent Jaeger, and Somesh Jha

The classical argument is that systems can be built securely from the ground-up. However, for many systems, security becomes a priority after its design. For example, it took over two years to retrofit the Linux kernel with the Linux Security Module to enforce mandatory access control. If done manually, this process can be buggy, error-prone, and ad-hoc. We are interested in ways to automatically retrofit security as a design concern in legacy systems.

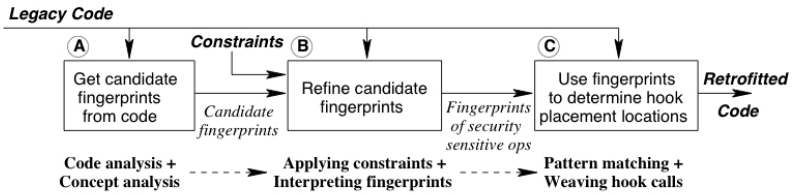
Automatic Refactoring of Legacy Code for Security

A *security sensitive operation* is a distinct policy operation that is performed within code. Our hypothesis is that security sensitive operations leave fingerprints on protected data structures by reading or writing to them.

We model *fingerprints* as sets of *structure member accesses* -- READs or WRITEs of protected data structures. Using static analysis on the legacy codebase, we determine what possible structure member accesses each API function.

Next, we use *concept analysis*, a hierarchical clustering techniq organize these structure member accesses into distinct secur sensitive operations, resulting in a set of *candidate fingerprir* which can be refined into fingerprints by domain-specific constraints.

Finally, we *place hooks* into the old code, ensuring that every structure member access is mediated by the appropriate policy operation.



Experimental Results

We ran our static analysis on three distinct servers.

- The ext2 filesystem, included in the Linux Kernel
- The main dispatch loop of the X Windows server
- PennMUSH, the server for a multi-user online game

Source code analysis was done with a module written in CIL (C Intermediate Language), which uses plugins written in Objective Caml to perform source-code analysis on C programs.

It took about about a half hour of manual work to check whether or not each of the candidate fingerprints was security-interesting.

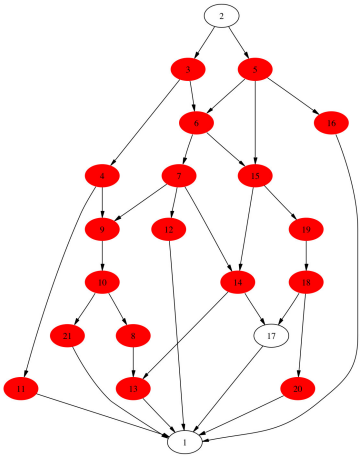
Server	Lines of Code	Fingerprints	Avg. Fingerprint Size
ext2	4,476	18	3.36
X Server	30,096	115	3.76
PennMUSH	94,014	38	1.42

(1) *Read* `address_space->host`
(2) *Read* `ext2_dir_entry_2->rec_len`
(3) *Write* `0` *To* `ext2_dir_entry_2->inode`
(4) *Read* `inode->i_mtime`
(5) *Read* `inode->u->ext2_inode_info->i_dir_start_lookup`
(6) *Write* ⊥ *To* `inode->u->ext2_inode_info->i_dir_start_lookup`

Future Work

A large amount of work remains in the area of retrofitting legacy systems for security. For example, it is necessary to improve our static analysis such that our results can scale better to even larger servers such as the Linux Server. What role can domain-specific and domain-independent constraints play in improving these results?

The security model that we use is heavily based on the model of structure member accesses: can we automatically mine richer policy from code and gain better guarantees?



A lattice generated by Concept Analysis for ext2. Security-interesting nodes are marked in red.

Sponsored by IBM Research