



# Systems and Internet Infrastructure Security

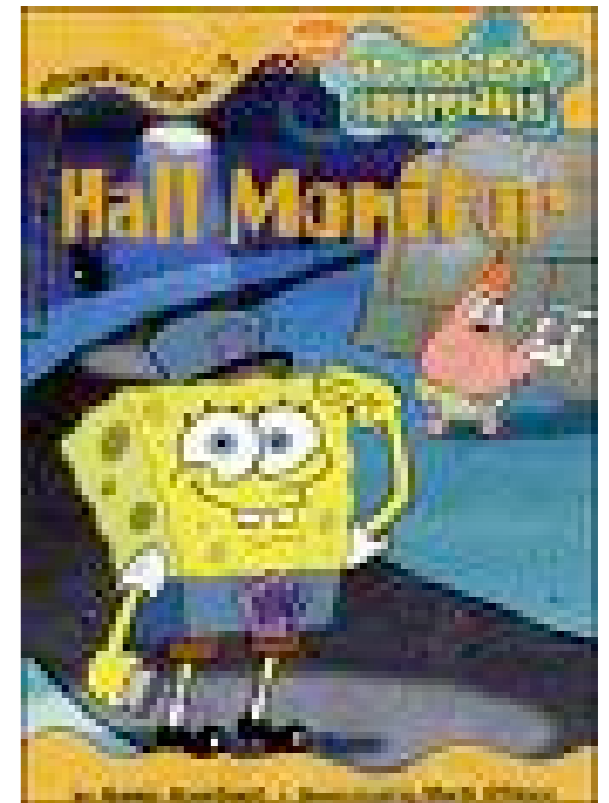
Network and Security Research Center  
Department of Computer Science and Engineering  
Pennsylvania State University, University Park PA

# Retrofitting Legacy Code for Authorization Policy Enforcement

Vinod Ganapathy, Trent Jaeger, and Somesh Jha  
Presented by Kevin Butler  
CSE 544 - 13 February 2007

# Reference Monitors

- What is a reference monitor?
- What properties does it enforce?
- Does it have to be designed into a system or can it be added afterward?



- Why retrofit?
  - ▶ LSM: retrofitting kernel to enforce MAC
  - ▶ Sometimes you want security but it's not there
- Is it easy to do?
  - ▶ CQUAL analysis showed over 500 type errors in the LSMs (of course, many were false positives)
  - ▶ Answer, though, is No

# Steps for Automated Retrofitting

- Determine what are security-sensitive operations
  - ▶ How do you know?
- Find fingerprints of these operations
  - ▶ Based on code patterns... is this accurate?
- Find all security-sensitive locations
  - ▶ How do you know when you have them all?
- Server instrumentation, generate ref. mon. queries, link the two
  - ▶ What property does this enforce?

- What is a code pattern?
  - ▶ Function call, read/write to a particular data structure, data comparison
- How do we know which ones are security-sensitive?
  - ▶ tangible side-effects: something is bound to happen
  - ▶ problem: lots of stuff happens
  - ▶ what next? Compare traces associated with a security-sensitive operation with those not associated (manual labeling required)
  - ▶ What about multiple security side-effects?



- Given set  $S$  of all security-sensitive operations,  $B \subseteq S$  is a particular operation,  $C_i$  is a label of trace  $T_i$  from a collection of labels  $C = \{C_1, C_2, \dots, C_n\}$  representing sets of performed security-sensitive operations, a set-equation can be formed
- $B = C_{j1} * C_{j2} * \dots * C_{jk}, * = \cap \mid \cup$
- Finding a CNF set equation is NP-complete
  - ▶ However, solution space is small enough that this is OK... or is it?

- Almost 54,000 code patterns per trace
- Tools like *AID* pared this down by order of magnitude
- Does this analysis catch everything?
  - ▶ If not, what does it miss?
- Manual labeling gives about a 15% error rate
  - ▶ Does this matter?

- Static analysis used to determine all locations in code where a fingerprint occurs
- Identify code patterns in function body, check if fingerprint occurs in this set
- If operation contains only function calls, mark each function as performing operation
- Match code patterns in function bodies using abstract syntax trees
  - ▶ Where else do we see these?
- Does this catch everything?

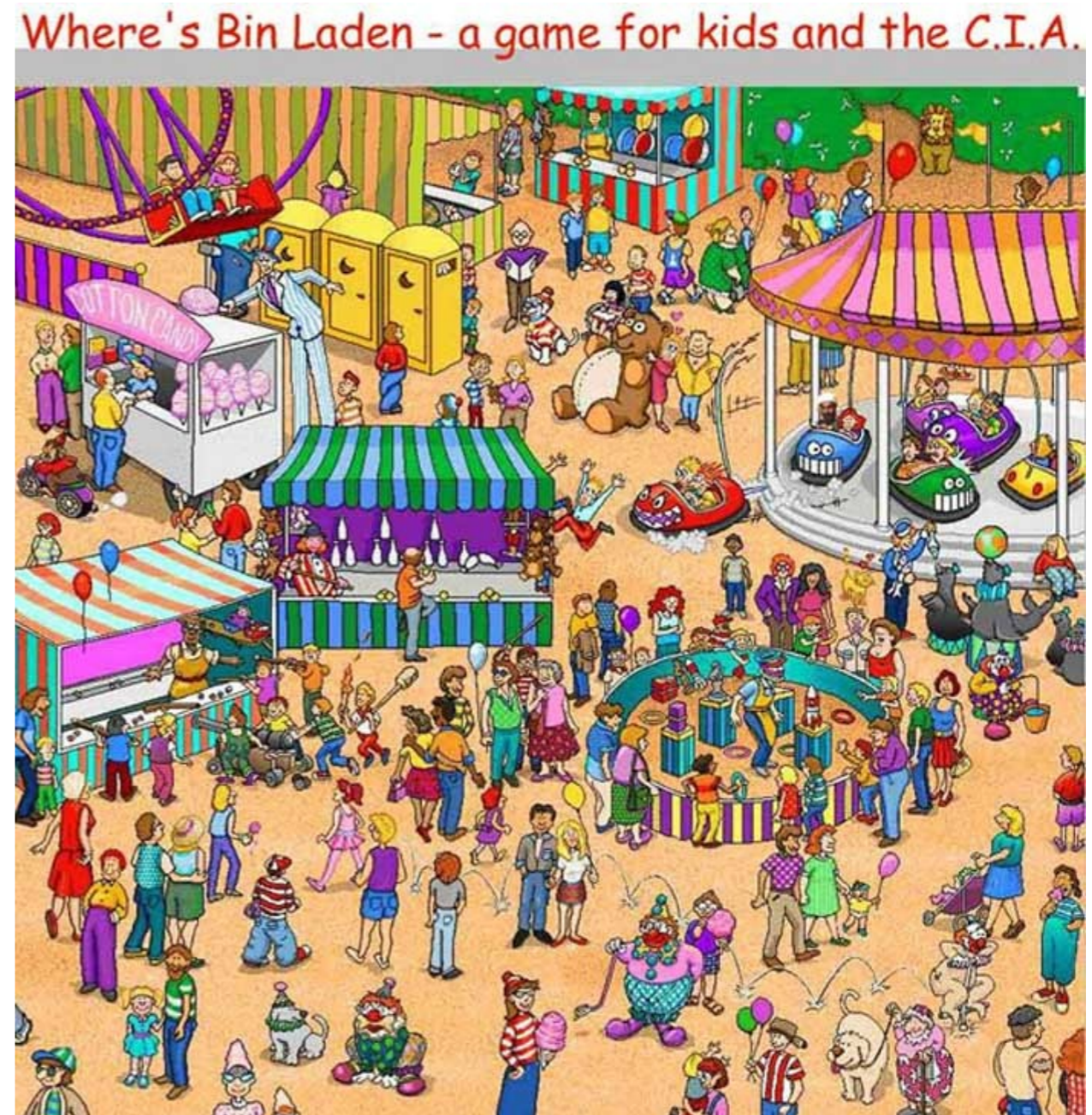


- Once security-sensitive locations determined, insert policy in those locations
- Policy consuler: template to be implemented from policy management API (not specified)
- State of reference monitor is subsequently updated
- How to bootstrap security labeling?
  - ▶ SELinux or similar OS support required

- Disallow clients from changing properties of unowned windows: easy-peasy with this system
- Enforcing MLS constraints on same X server
  - ▶ able to enforce \*-property between two xterms from different security levels
  - ▶ Any difficulties with this scheme?

# Finding everything

- How do you know you've found all code paths?
- Is this possible to do?
- Determining code coverage is an issue
- Other limitations?
- How would you solve them?



- *Aspect-oriented programming* is another way to attack this problem
- *Join points*: defined points in a program flow (these can be dynamic if using an aspect-based language)
- *Pointcuts*: A set of join points on which a predicate may or may not match
- *Advice*: A specified piece of code run when a pointcut is reached
- Crosscut across multiple concerns using a *weaver*
  - ▶ “Walk up to a system, drop pointcuts in, and program against even if it doesn’t hve the structure the programmer is expecting”

- What does this do to our model of security?
- Is security a bunch of aspects or features that can be arbitrarily dropped in at a later date?
- How far can we go with this model?
- What do we lose with this sort of retroactive designing?
- What do we gain?