Software Security

Enforcing defensive programming through software model checking

David Wagner and Hao Chen

U.C. Berkeley

Introduction

Introduction

- Security programming is pitfall-laden
  - It’s too easy to unintentionally violate implicit usage rules of OS API’s

Introduction

- Security programming is pitfall-laden
  - It's too easy to unintentionally violate implicit usage rules of OS API's
- An approach: enforce defensive coding
  - Enumerate rules of prudent security coding
  - Use tools to automatically verify that software follows these rules

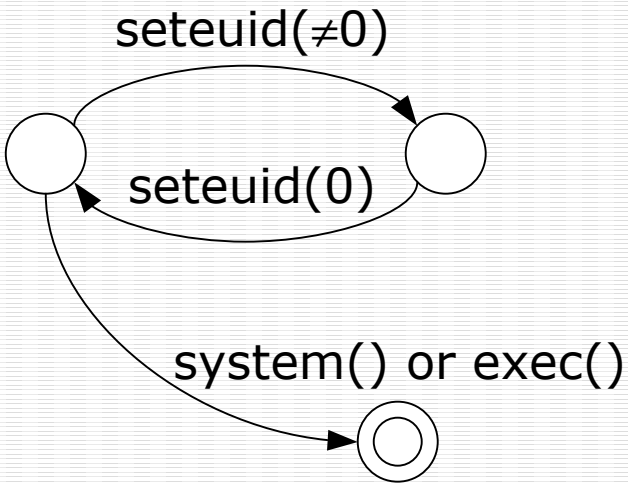
Examples of Coding Rules

Examples of Coding Rules

- Avoid calling system() or exec() with root privilege

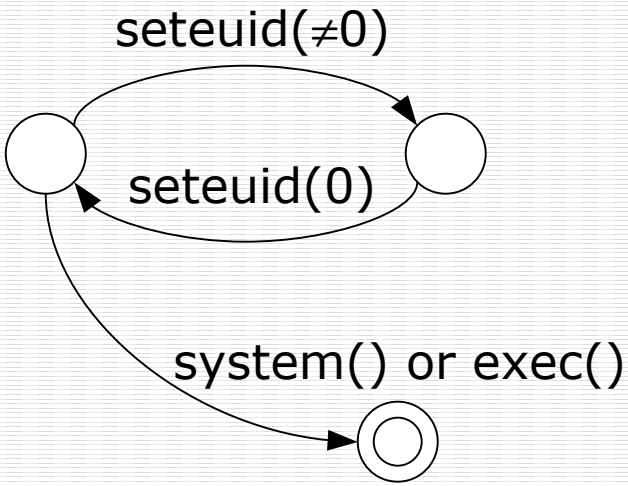
Examples of Coding Rules

- Avoid calling system() or exec() with root privilege



Examples of Coding Rules

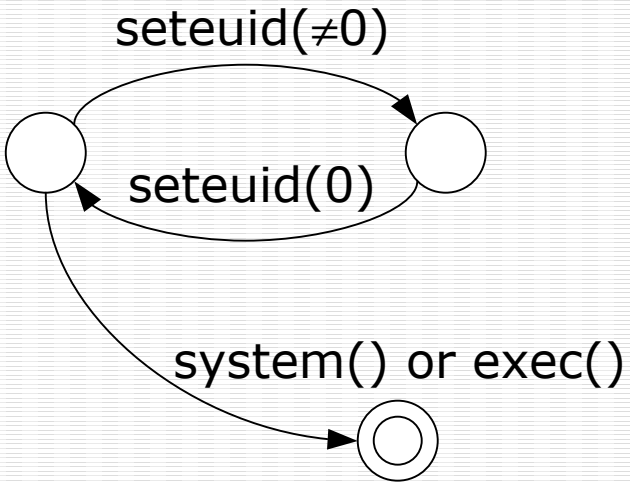
- Avoid calling system() or exec() with root privilege



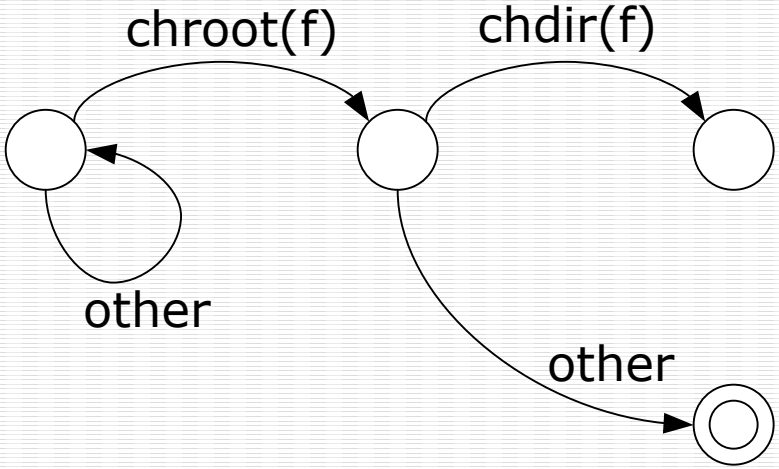
- After chroot(f), immediately call chdir(f)

Examples of Coding Rules

- Avoid calling system() or exec() with root privilege



- After chroot(f), immediately call chdir(f)



Automated Code Analysis

Automated Code Analysis

- We check whether code obeys these rules
  - Let Σ = set of security-relevant events,
    B = set of "bad" traces that violate a rule,
    T = set of feasible traces ($T, B \subseteq \Sigma^*$)
  - If $T \cap B = \emptyset$, then the rules are obeyed

Automated Code Analysis

- We check whether code obeys these rules
  - Let Σ = set of security-relevant events,
    B = set of "bad" traces that violate a rule,
    T = set of feasible traces ($T, B \subseteq \Sigma^*$)
  - If $T \cap B = \emptyset$, then the rules are obeyed

- Framework: software model checking
  - B: finite-state automaton (regular language)
  - T: pushdown automaton (context-free lang.)

Come and Get It!

- Our tool, MOPS, is available for download
  - Comes with modelchecker
  - No database of rules yet (research in progress)

http://www.cs.berkeley.edu/~daw/mops/

Conclusion

- Security requires disciplined coding
- And tools can help