

# *Modelling for Security Testing*

*Percy Pari-Salas, Paddy Krishnan, Kelvin Ross*

Percy supported by a KJRoss and Associates Scholarship

# *Security vulnerability testing*

- Aims to find undiscovered vulnerabilities
- Traditionally done by penetration testing
- Usually data intensive: find particular patterns
- Flaws not associated to security requirements of the application

# *Other Approaches*

- Regular Expressions for illegal behaviours
- Attack Graphs: Represent a class of attacks
- Bounded exhaustive testing
- Ideas from Fault based testing

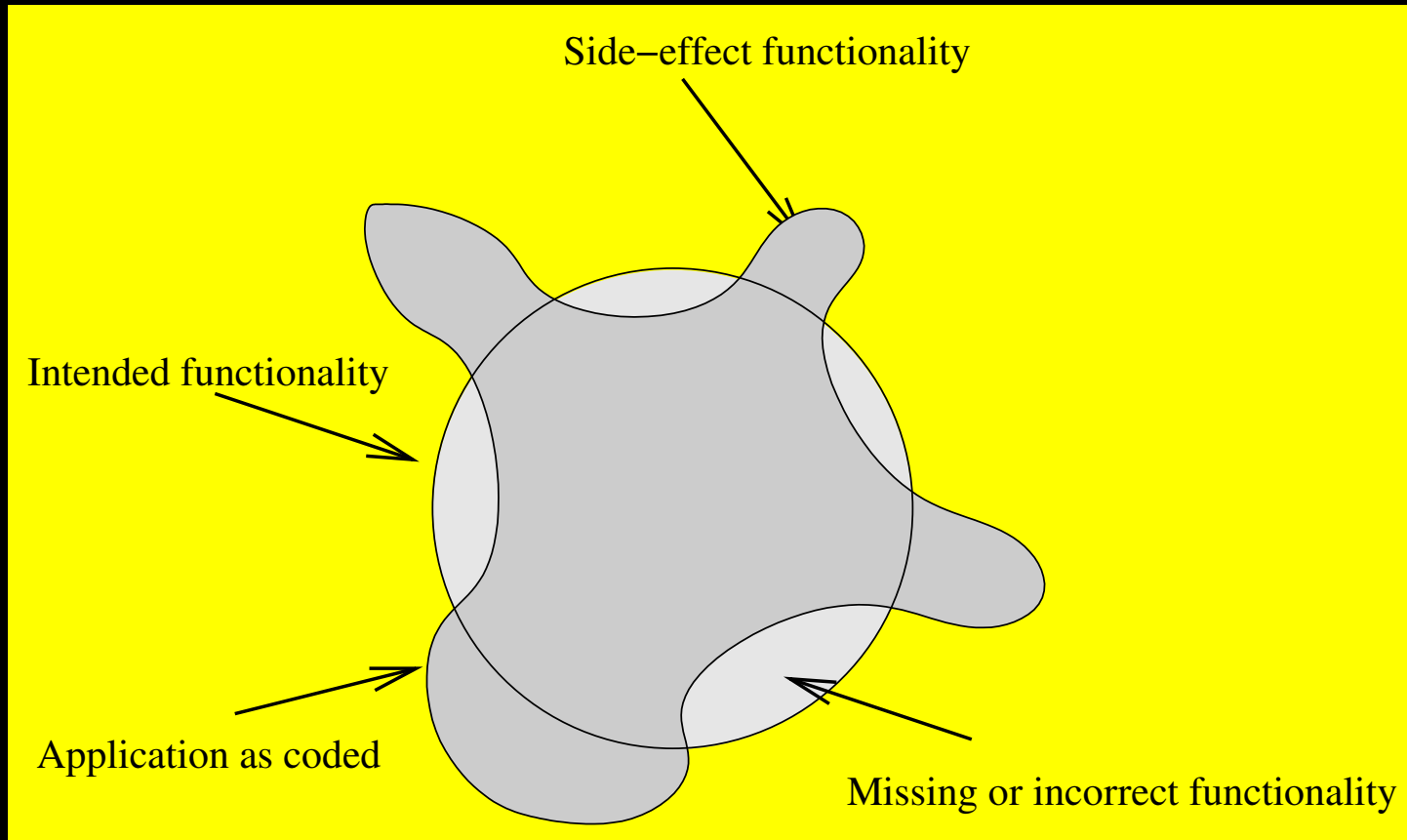
# *Model Based Testing?*

- Does not really consider implementation
- Usually driven purely from a catalogue
- Models become too large

# *Example: Buffer Overflow*

- Model: specify buffers are bounded
- Model: also specify the effect of overflow
- Is the data important?
- Are all buffer overflows equally important?
- Information on how implementation reacts.

# *Models vs Reality*



# *Our Approach*

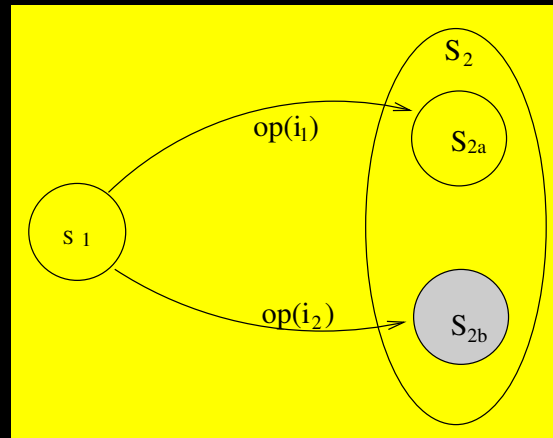
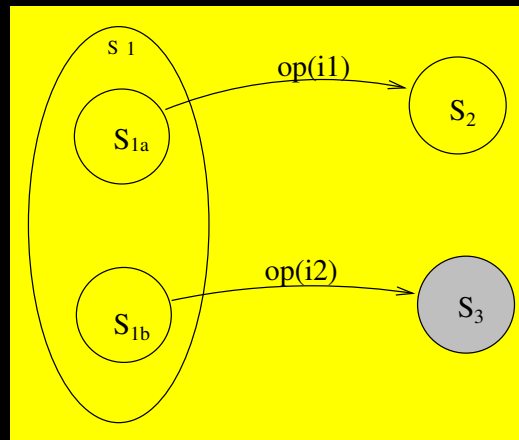
- Have different models
- Models of functional behaviour
- Models of implementation: source of vulnerabilities
- Models of attacker: exploiting the vulnerabilities

# *State transition models*

- Programs are labelled transition systems
- The labels represent the actions performed
- Transitions represent the evolution of behaviour
- The transitions have pre and post conditions

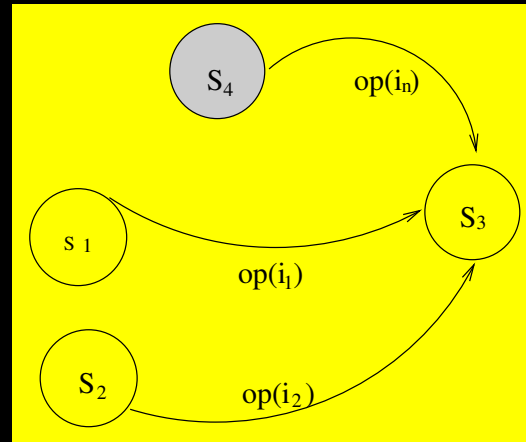
# Models: Abstractions

- Model cannot distinguish states
- Implementation: Transition does not correspond to model



# *Incomplete Information*

- Model does not have states/transitions



# *Test Generation Scenarios*

- Transition disabled in model but enabled in implementation
- Transition marked legal in model can be illegal
  - Source state is incorrect
  - Destination is incorrect

# *Example: SQL Injection Attack*

## **Model**

- Strings: user, password
- (user,password) in database
- database does not change: rarely specified

## **Implementation**

- Strings: user, password
- `sql_evaluate` calls
- If string not well formed database can return unexpected results

## Attack

- Incorrect user but returns authenticated
- Incorrect password but returns some table information
- New user hidden in user: change in database exploited later

# Conclusion

- Model based testing can be used for vulnerability testing
- Still need a library of inputs
  - `user = xxx'`
  - `user = xxx';INSERT INTO table VALUES entry`
- Input classified by type of effect
- Write data generators?
- Very preliminary: Percy's PhD