Security Testing beyond Functional Tests

David Basin
ETH Zurich
June, 2017

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

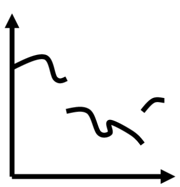
Associated Publication: Mohammad Torabi Dashti, David Basin: ***Security Testing Beyond Functional Tests.*** ESSoS 2016.

Security Testing

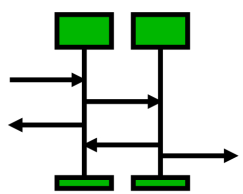
Testing is king

- Widely used and accepted QA measure
- Ca. 50% project time and costs

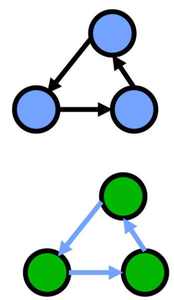
Testing methods well established, also for security



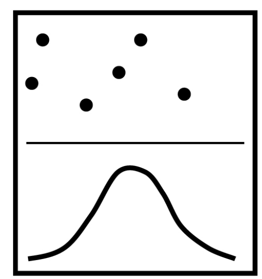
ad hoc



functional



structural



stochastic

failure based

risk based

vulnerability driven

fuzzing

But what is security testing?

What is Security Testing?

Which statements do you agree with?

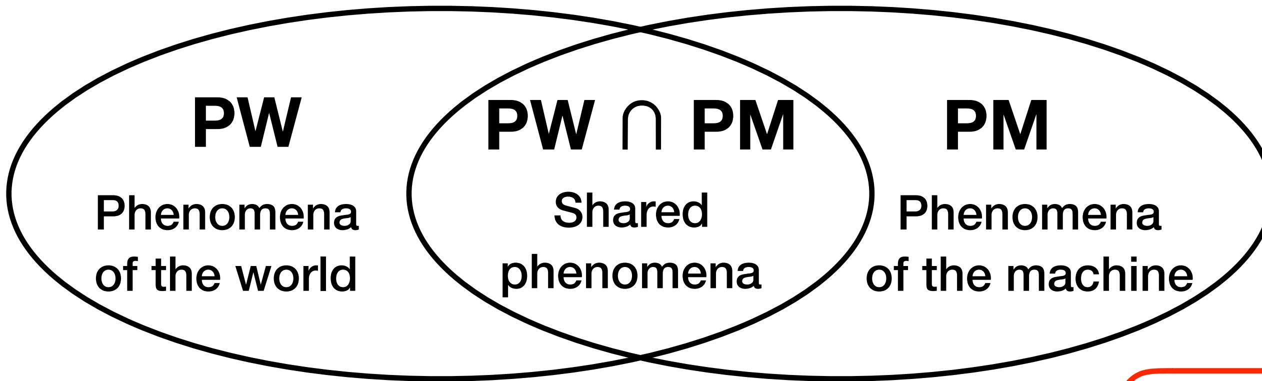
- Security testing is more difficult than functional testing
- One cannot measure the adequacy of security tests
- Some aspects of security testing defy automation

Objectives of talk

- Provide an **elementary theory** of security testing
- Use it to explain current practice and highlight limitations

Some Inspiration

Michael Jackson, The World and The Machine, ICSE 1995



Machines serve a purpose in the world

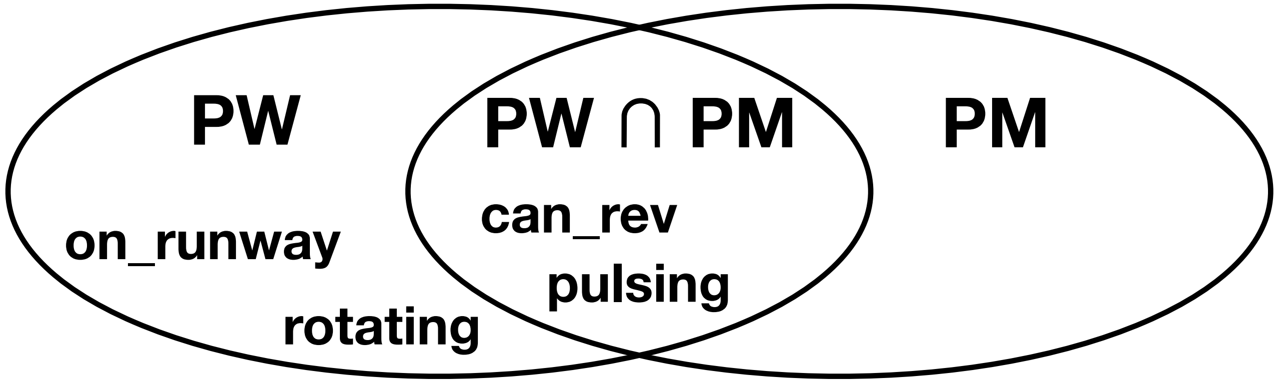
- **Machine**: software + hardware system
- **Purpose**: control an airplane, edit a document, …

Different terms describe aspects of machine and world

- **Requirements**: address phenomena of the world
- **Specifications:** address behavior of machine
- **Programs** (or **systems**): executable and comply to specification

Requirements are what ultimately matters!

World and Machine — An Example



Avionics: reverse thrust engaged iff plane on runway

Req: $can\_rev \leftrightarrow on\_runway$

Sensors on landing wheels generate pulses when wheels rotate

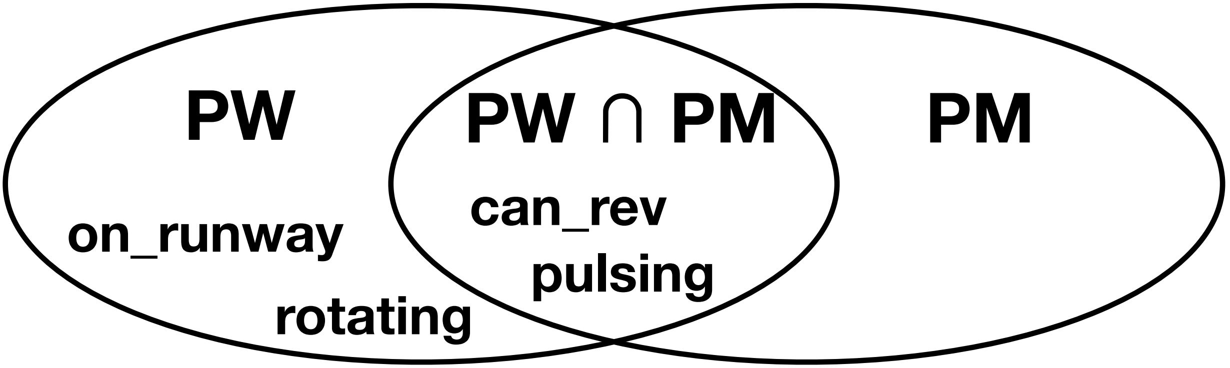
World1: $pulsing \leftrightarrow rotating$

World2: $rotating \leftrightarrow on\_runway$

Can derive specification

Spec: $can\_rev \leftrightarrow pulsing$

Development Explains Requirement's Satisfaction



World1: pulsing $\leftrightarrow$ rotating

World2: rotating $\leftrightarrow$ on_runway

Spec: can_rev $\leftrightarrow$ pulsing

Req: can_rev $\leftrightarrow$ on_runway

But **after rainfall:**

aquaplaning may occur, whereby World2 fails

⇒ reverse thrusters fail to fire and plane slides off runway

Road Map

I. Motivation and Context

II. Specifications and Requirements

III. Security Rationales and Security Cases

IV. Security Testing

Requirements and Specifications

Starting point: valuable resources

Security requirements express constraints on resource usage.

- Should hold in presence of an adversary.
- **Example**: valid library card required to borrow books.

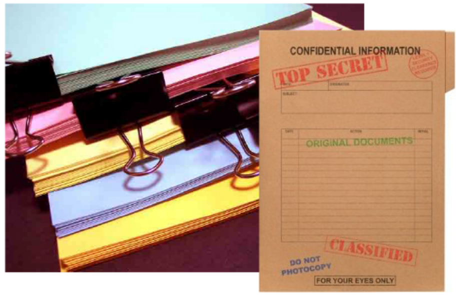
System (aka machine): artifact whose behaviors can be regulated and controlled

Specification: describes desired system behaviors (over interface)

Thought experiment:

- Specify an IT System for authorizing book loans
- How might an unauthorized user take books from the library?

Example: R&D Lab



Sensitive documents in lab

- Access limited by an electronic lock system at door

Security requirement

- Only staff members working in lab may read document
- Does not prohibit/oblige any behavior for lock

Specification for lock: $\Phi(\mathbf{key}, \mathbf{open}) = \mathbf{open} \Leftrightarrow (\mathbf{key} \in \mathbf{validKeys})$

Output signal **open** (which triggers cylinder's actuator) is produced only upon receiving an input **key** belonging to the set **validKeys**

If lock works correctly, is the security requirement satisfied?

- **No: room may have windows**
- Excluding this requires **environmental assumptions**

Example: Parking Lot

Work out Specification, Requirements, and Missing Assumptions

Example: Publisher (and Interfaces)

Integrity requirements for publisher's database

- Only **copy editor**s may **delete** data

Violated by dynamite exploding in vicinity

- Input that deletes data

World has no definite interface for requirements

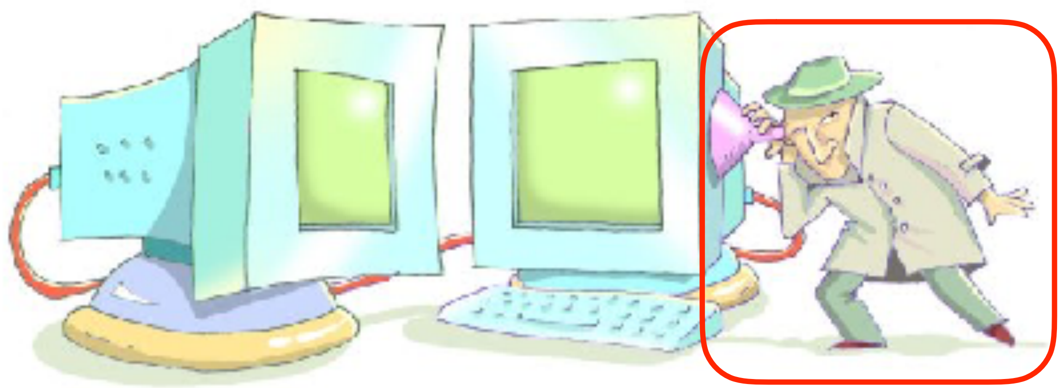
- **DB System**: interface realized through APIs
- **World/Environment**: Dynamite, axe, degausser, server's format command, …

Specifications are over definite interfaces.

- E.g., only users of role **copy editor** may execute the API's **delete** command

Nominal versus Side Channels

System's nominal channels are anticipated and constrained by *Spec*



A side channel is an unanticipated communication channel between system and its adversarial environment. E.g.

- Reading secret data through timing or power analysis
- Writing data by row-hammer attacks

Side channel's exploitability depends on adversary

Road Map

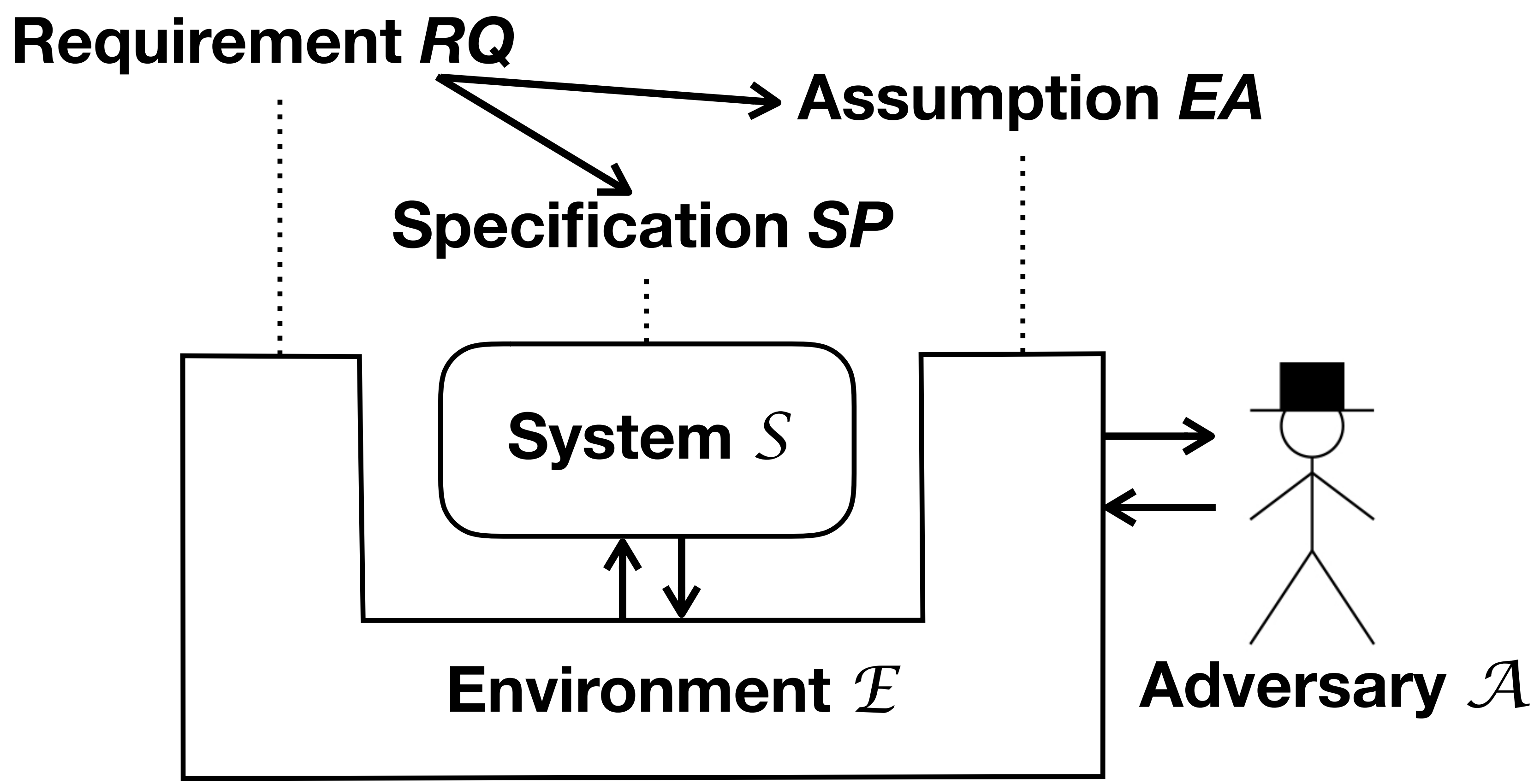
I. Motivation and Context

II. Specifications and Requirements

III. Security Rationales and Security Cases

IV. Security Testing

Relating World and Machine



Environmental assumptions link system to behavior in the world

System (machine): symbol processing entity governed by *SP*

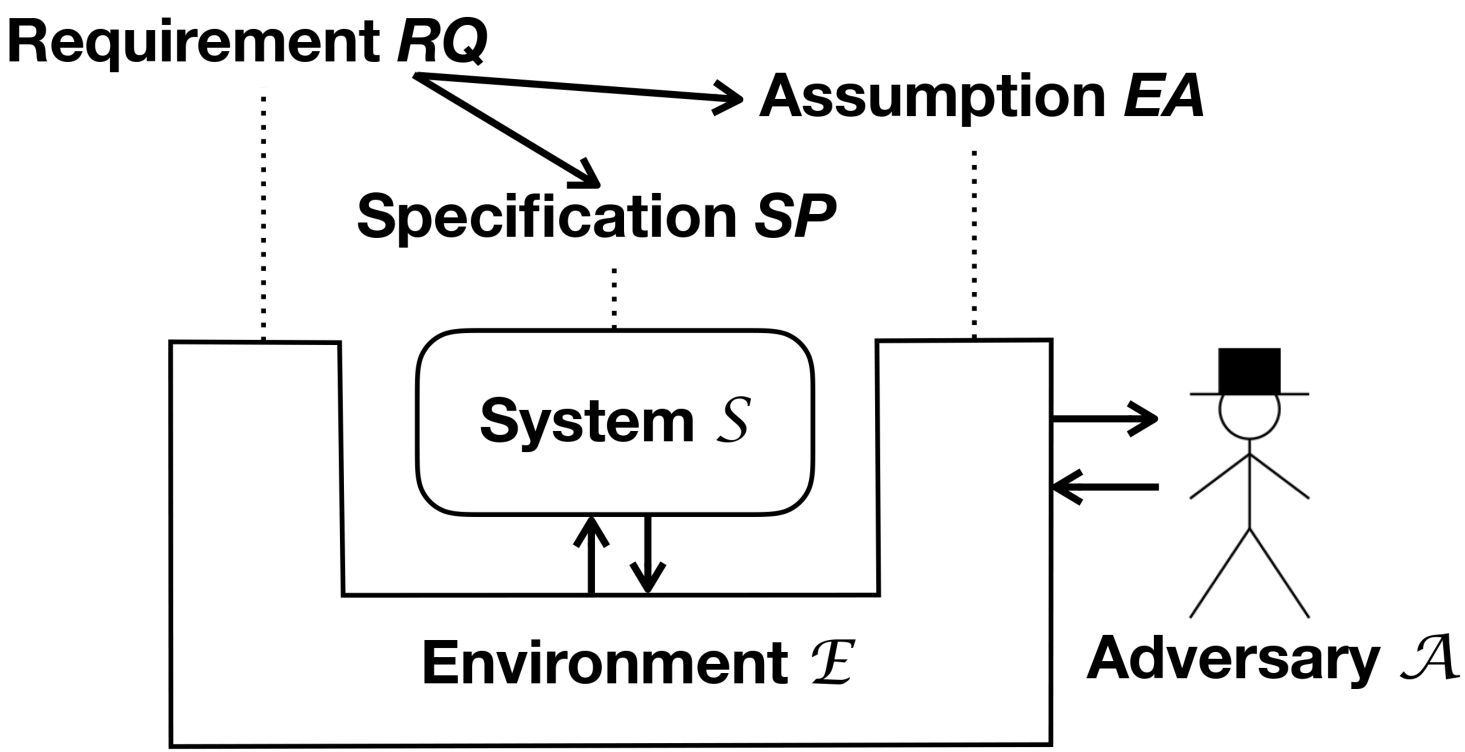
Behavior independent of deployment context

Requirements: Stakeholder desired usage of resources in the world

Security Rationale is argument for reduction

- Only copy editors have role **copy editor**
- No way to delete data except by executing API **delete** command

Security Rationale



Security rationale for **<RQ, SP, $\mathcal{E}$, EA>** justifies condition:

For all System ***S*** and Adversary ***A*:**

$$S \models SP \wedge S \parallel \mathcal{E} \parallel A \models EA \Rightarrow S \parallel \mathcal{E} \parallel A \models RQ \qquad (\dagger)$$

Comments on Rationale

For all System *S* and Adversary *A*:

$$S \vDash SP \wedge S \parallel \mathcal{E} \parallel A \vDash EA \Rightarrow S \parallel \mathcal{E} \parallel A \vDash RQ \qquad (\dagger)$$

1. **SP** regulates **S** behavior over nominal channels (1st conjunct)
   Adversary may abuse system over side channels (2nd conjunct)

2. $S \vDash SP$ is formal. Remaining two satisfactions are informal

- $\mathcal{E}$ and ***A*** have no clear boundaries
- So (†) is an **informal guideline** to clarify verification/refutation objectives

3. If ***EA*** is ***RQ***, **(†)** is trivially satisfied

- Whether statement is **requirement** or **assumption** depends on context
- **Example:** no building entry through window is a requirement if we are designing the building.

Comments on Rationale (cont.)

For all System *S* and Adversary *A*:

$$S \models SP \wedge S \parallel \mathcal{E} \parallel A \models EA \Rightarrow S \parallel \mathcal{E} \parallel A \models RQ \qquad (\dagger)$$

4. Rationale can only account for small set of entities and interaction

- Cannot reason about entire world!
- Need assumption that excluded entities and interactions are unimportant for requirement's satisfaction

Example: system ***S*** has no side channels to communicate with the adversary (Note also role of ***S*** in 2nd conjunct!)

5. Simplification: conflate $\mathcal{E}^* = \mathcal{E} \parallel A$ in **(†)**

$$S \models SP \wedge S \parallel \mathcal{E}^* \models EA \Rightarrow S \parallel \mathcal{E}^* \models RQ$$

R&D Lab Example

Constructing a Security Rationale

RQ = only staff members may enter lab

Reduce ***RQ*** to following **requirement**

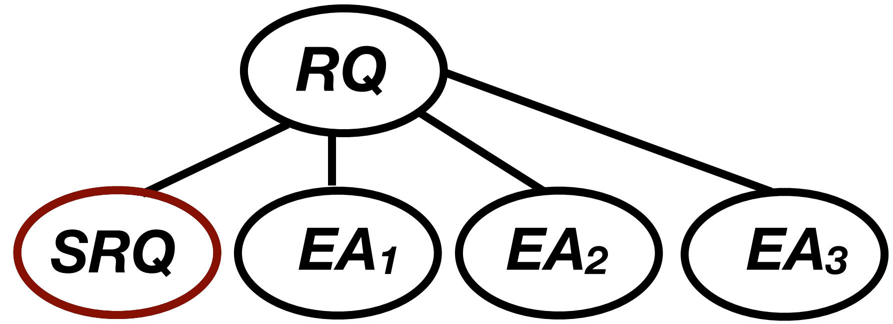
SRQ: lock only opens after valid key presented.

Relies on 3 environmental assumptions:

- EA_1: Only staff members have valid key
- EA_2: Door opens only after receiving lock's signal
- EA_3: Only entry into lab is through door

Logical reasoning justifies reduction

Rationale can be further elaborated



(*SRQ*) signalFor(X) → hasValidKey(X)

(EA_1) hasValidKey(X) → isStaff(X)

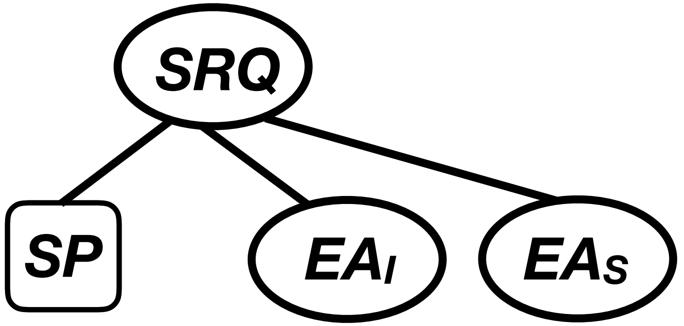
(EA_2) doorOpensFor(X) → signalFor(X)

(EA_3) enterLab(X) → doorOpensFor(X)

(*RQ*) enterLab(X) → isStaff(X)

R&D Lab Example (cont.)

Constructing a Security Rationale



Reduce *SRQ* to specification on nominal channel

SP: output signal **open** produced only after receiving a **key** belonging to set **validKeys**

Requires two more assumptions

1) *EA_I*: **open**, **key**, and **validKeys** interpreted as expected and entity cannot send key to lock system without possessing key

2) *EA_S*: all communication between system ***S*** and ***A*** are regulated by ***SP*** *(excludes, e.g., hidden backdoor in **S**, or power cutoff opens door)*

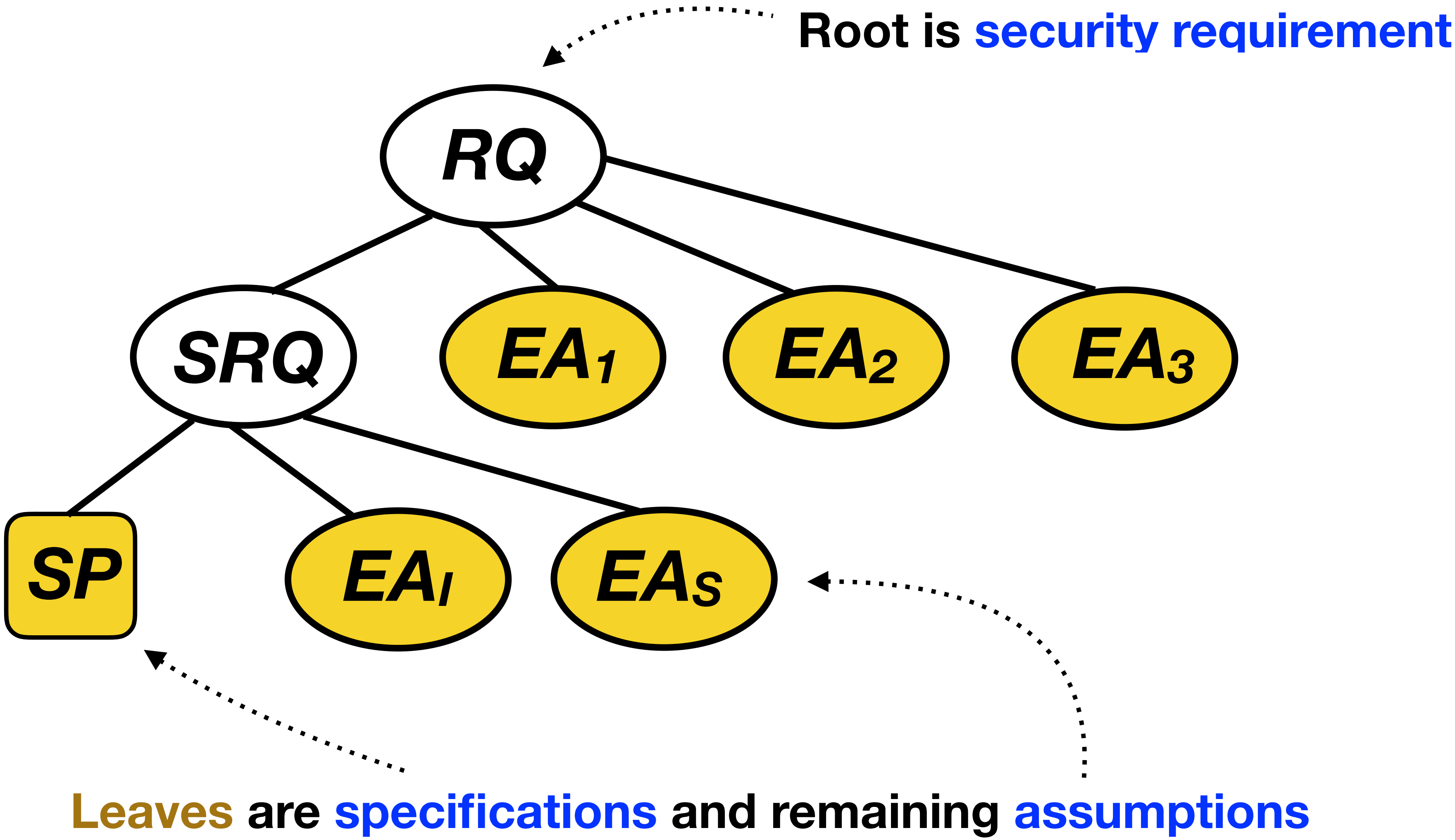
This constitutes a security rationale for <RQ, SP, $\mathcal{E}$, EA> where:

— $\mathcal{E}$ is lab's environment

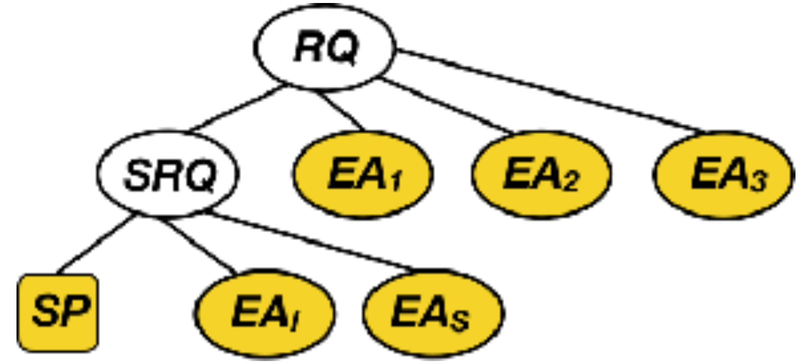
— ***RQ*** and ***SP*** are defined above

— ***EA*** is conjunction of ***E1, E2, E3, EA_I, EA_I***

Visualization as Reduction Tree



Security Cases



When we deploy system *S* in environment *E*, with adversary *A* reduction yields:

$$S \vDash SP \wedge S \parallel E \parallel A \vDash EA$$

Security case is argument for truth of these conjuncts

- Justifies **leaves** of reduction tree

Analogous to safety cases, provided by designers

- Verification may be used to establish $S \vDash SP$
  + analysis how system used in adversarial environment, $S \parallel E \parallel A \vDash EA$

Role of adversary

- Irrelevant for security rationale & system analysis $S \vDash SP$
- Highly relevant for $S \parallel E \parallel A \vDash EA$

Example

Security Rational for Lab

Rationale holds by logical argument, independent of adversary

(*SRQ*) signalFor(X) → hasValidKey(X)
(*EA*$_1$) hasValidKey(X) → isStaff(X)
(*EA*$_2$) doorOpensFor(X) → signalFor(X)
(*EA*$_3$) enterLab(X) → doorOpensFor(X)

(*RQ*) enterLab(X) → isStaff(X)

But, assumptions express constraints on adversary's capabilities

Example: *EA*$_1$ is violated if adversary can threaten or bribe a staff member and thereby obtain a valid key

- Security case must argue why an **anticipated adversary** cannot violate this assumption
- E.g., threat agent = a curious visitor

Security Cases and Closed-World Assumption

***EA_S*: all communication between system *S* and *A* are regulated by *SP* (excludes, e.g., hidden backdoor in *S*, or power cutoff opens door)**

Closed-world assumption: excludes various adversarial actions

- That which has not been considered in ***SP*** plays no role
- Completes security case in “formal sense”

Example: lock system has no side channels.

- Suppose lock leaves door open if power cut off
- Assumption fails for an adversary who can disrupt power
- Might be valid for weaker adversary.

Since all possible channels cannot be enumerated, closed-world assumption must invariably be invoked.

Road map

I. Motivation and Context

II. Specifications and Requirements

III. Security Rationales and Security Cases

IV. Security Testing

Kinds of Testing

Functional tests (S-Tests): aim at refuting that system *S* meets its (functional, security, …) specifications SP

- Specification not just the “functional” ones, derived from use cases (Call these: “**restricted functional tests**”)
- **Examples**: bound on delay in producing output, or threshold in electromagnetic radiation levels

Environment assumption tests (E-Tests)

Aim at refuting environmental assumptions *EA*, for some system *S* environment *E* and adversary *A*

Security Testing: both types of tests

Security Tests and Falsification

Recall security rationale (for given *S* and *A*)

$$S \models SP \wedge S \parallel \mathcal{E} \parallel A \models EA \Rightarrow S \parallel \mathcal{E} \parallel A \models RQ \qquad (\dagger)$$

Refuting either conjunct does not refute conclusion

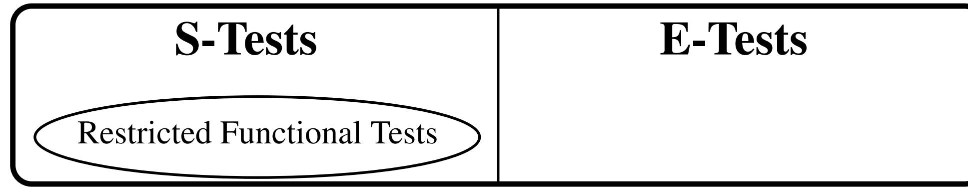
- But it does indicate something wrong with system or design!

Refutation of a conjunct *suggests* *RQ* violated as it is unlikely satisfied due to unintended causes

Call converse of (†) the Intentional Security Hypothesis (H)

- Says system satisfies requirement by design, not by chance!
- **(H)** will be implicitly used on all remaining slides

S-Test Examples



S-tests (restricted functional tests)

System	(Security) SPEC
Gate Controller	Alarm goes off if the bar is forced open
ATM	After three consecutive wrong PINs, card is blocked inside
Phone	All communications are encrypted using AES
Web Server	Only users with the role `auditor` can read the log file

S-Tests (general)

- Electromagnetic radiation levels do not exceed some threshold
- S-Test over anticipated (nominal) channel

Most security tests are S-Tests, e.g., buffer overflow,

- Feed the lock system a very large key
- Might produce **open** signal without inputting a **key** in **validKey**

E-Tests Examples

EA_2: Door opens only after receiving lock's signal
EA_3: Only way to enter lab is through door
EA_I: open, key, and validKeys interpreted as expected

EA_2: try to intercept communication between lock and door and inject an open signal

EA_3: try climbing through window

EA_I: test if lock's variables are misinterpreted

- E.g., **validKeys** contains invalid key of a former staff member
- Alternatively a replay attack would allow a non-staff member to present a ***key*** in **validKeys**

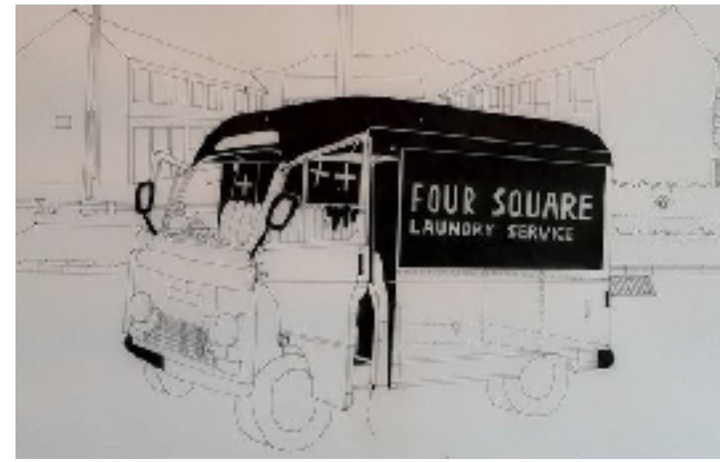
Feasibility depends on environment and adversary

- Can adversary climb in through window, squeeze between window bars, unhinge the door, remove the lab's roof with a large can-opener?
- Checklists and brainstorming help. ***But are never complete!***

Inherent Incompleteness of E-Tests

Fundamental distinction with S-Tests: domain has no boundaries

- Not merely the problem of infinite cardinality



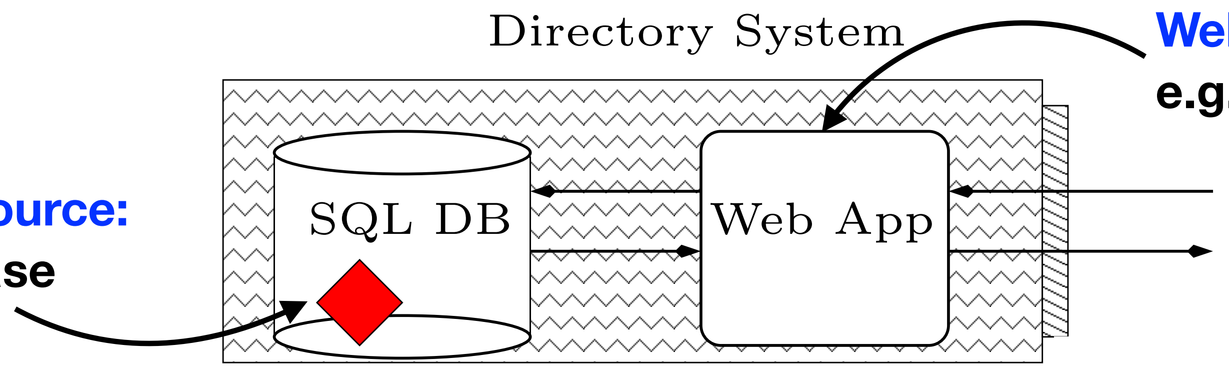
Essentially unlimited experience and creativity required of tester

Example: Four-Square Laundry

A British secret operation, the *"Four Square Laundry Affair"* was carried out in Northern Ireland to collect information about the residents of a troubled neighborhood. A rogue laundry service van visited the neighborhood regularly, and sent the collect laundry for various tests and inspections before washing it. The tests included checking for traces of explosive material or blood. The service also noted changes in the amount or kinds of clothing sent by each household for washing, which could indicate the presence of guests, and so forth.

Incompleteness/Challenges (cont.)

Another example: sensitive data on the web



Web App specification: e.g., sanitize inputs

Protected resource: data in database

Integrity requirement: only authorized modifications of data

Closed-world assumption (CWA): data can only be modified over web app

<u>Examples of events that can violate CWA:</u>

- Remotely degaussing the storage device
- Reformatting system storage
- Exploiting BoF in FTP server running on Web App platform
- Bribing system administrators

So Security Testing *is* Harder!

System specification describes behavior over interfaces

- Basis for constructing S-Tests, independent of adversary and environment
- **Example:** PDP should function consistently independent of environment

Security testing hinges on assumptions validity in adversarial env.

- Environments and adversaries are nebulous entities with no clear interface
- No domain boundaries to limit search for test cases
- E-Tests only as thorough as attack scenarios that tester anticipates

Vulnerability Remediation also Differs

What do we do when security case fails?

$$S \models SP \land S \parallel E \parallel A \models EA$$

System fails to meet *SP*, revealed through S-Tests.

- Debug and fix the system!

***EA* violated, revealed through E-Tests**

- Fixing the system is not enough. Fix design and update security rationale

Update ***SP***	Change **Environment**
Account for revoked keys ⟹ change system	Add window bars ⟹ May need to update ***SP***

Security Testing in Practice

Security case typically not available

- Tester must reconstruct it: adversary capabilities, specs, assumptions
- Or tester is reduced to “playing around” with the system (typical case)

Even when security case is available…

- Tester must anticipate how adversary can violate assumptions
- Relies on experience and creativity

⇒ Manual task outside of formal methods or informal guidelines

- Not surprising that existing methods fall short!

Almost all Tests are S-Tests

Risk-based security testing

- Work out specification from (mis)use-cases, risk analysis, documents
- Convert risks into security requirements demanding risk's mitigation
- Countermeasure is system spec. defining mechanism to meet the req.
- Test the mechanism. **This is an S-Test.**

Fuzz testing and fault injection

- Refute generic system specifications, e.g., concerning memory access
- Generate tests guided by relevant fault model,
  - e.g., failure to check input's length or format
- Resulting tests focus on system's nominal channel. **They are S-Tests**

Vulnerability-based testing

- Try to identify common vulnerabilities in system. **Again S-tests**

Methodologies with E-Test Flavor

BSI Baseline Protection

T 0.10 Failure or Disruption of Mains Supply

In a building, many networks are used for basic services that support an institution's business processes, including IT. Examples include:

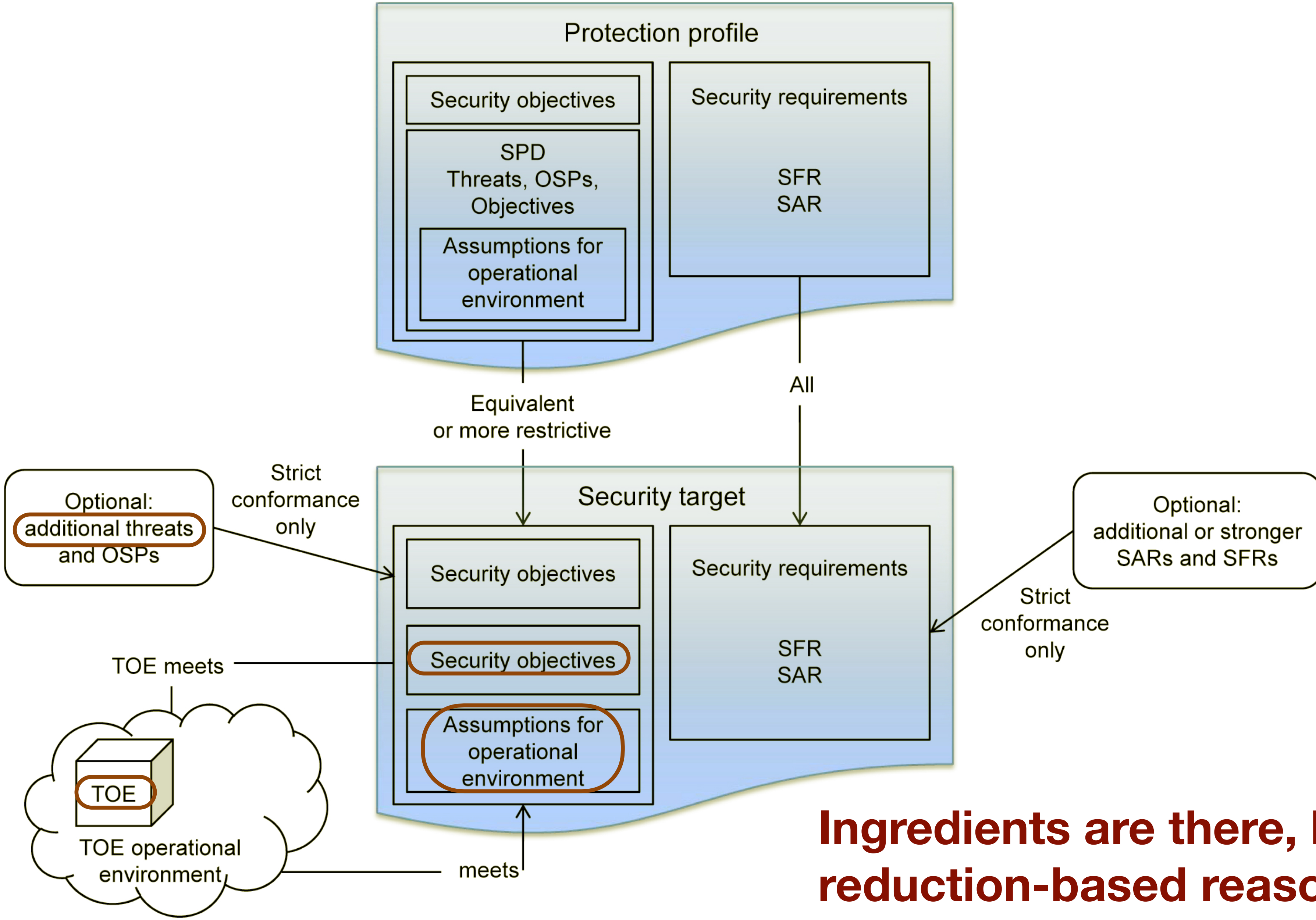
- power,
- telephone,
- cooling,
- heating or ventilation,
- water and sewage,
- supply of fire fighting water,
- gas,
- alarm and control systems (e. g. for burglary, fire, etc.)

A disruption of a supply network can lead to a situation where employees cannot work in the building and hence information processing is impaired.

Provides a starting point for developing E-Tests

Methodologies with E-Test Flavor

Common Criteria



Ingredients are there, but reduction-based reasoning methodology is missing

Summary

Distinction between specification and requirements fundamental

- **Ingredients for theory of security testing**: security rationales, security cases, requirement decomposition, intentional security hypothesis, S-tests, E-Tests

Theory answers questions initially posed:

1. Security testing ***is*** more difficult.
2. Adequacy can***not*** be measured.
   Environment without boundaries; domain of E-Tests undefined.
3. Testing can***not*** be automated.
   Code analysis and other formal methods are useless.

Starting point for documenting, classifying, and reusing experience

- Explicating violated assumptions
- Associating common assumptions with attacks
- Classifying threats on different systems/environment with countermeasures

Final Thoughts for Practitioners

Go beyond the well-chartered world of functional tests.

Lift your sights beyond machine and target world as well!