SecArch: Architecture-level Evaluation and Testing for Security

Sarah Al-Azzani
Rami Bahsoon

University of Birmingham, UK
August 2012

Motivation

- Calls for further security architecture testing that is design-specific (McGraw, 2004)
- Supporting testers to locate problematic areas in large specifications
- Lack of research in analysing dependability with respect to interactions across multiple views (France, 2007)

  =

- Proposing a systematic architecture evaluation method to guide testers to vulnerable interactions
  - Merges the concept of implied scenarios and race condition detection

Background

- Previous Research: “Using Implied Scenarios for Security Testing” (SESS ’10)

- Implied Scenarios (Alur, 2000): hidden behaviour that arrises due to mismatch between the specified behaviour and the architecture.

  - Scenario-based languages models behaviour as partial views
  - A result of components having only local views of the execution in concurrent systems (Uchitel, 2003)
  - Lack of synchronisation between components
  - Attackers often intentionaly probe unspecified behaviour

Outline

- Overview of Implied Scenarios
- Overview of the problem
- Proposed solution applied to an industrial case study
  - Using syntactics analysis and sematics analysis
- Results

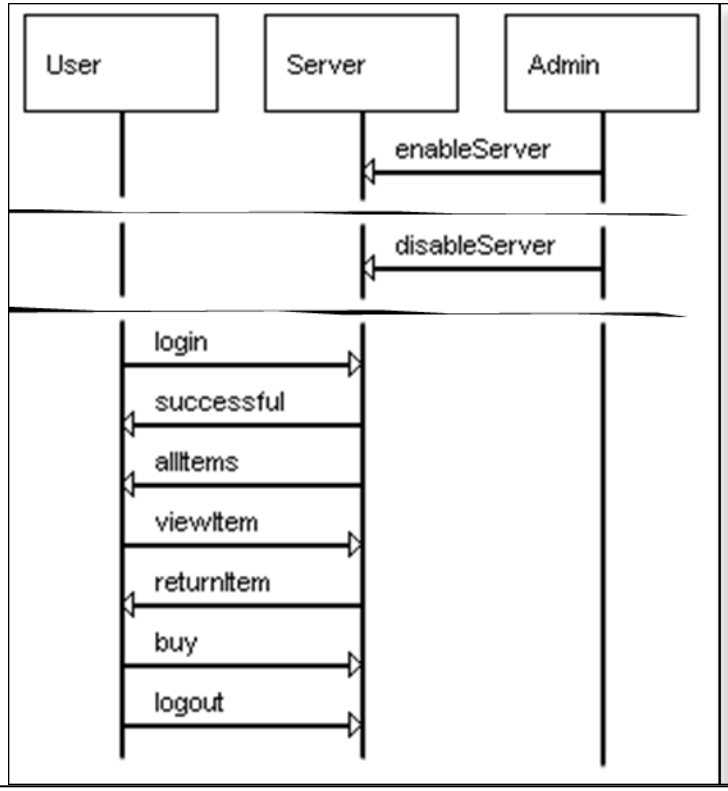
Contribution

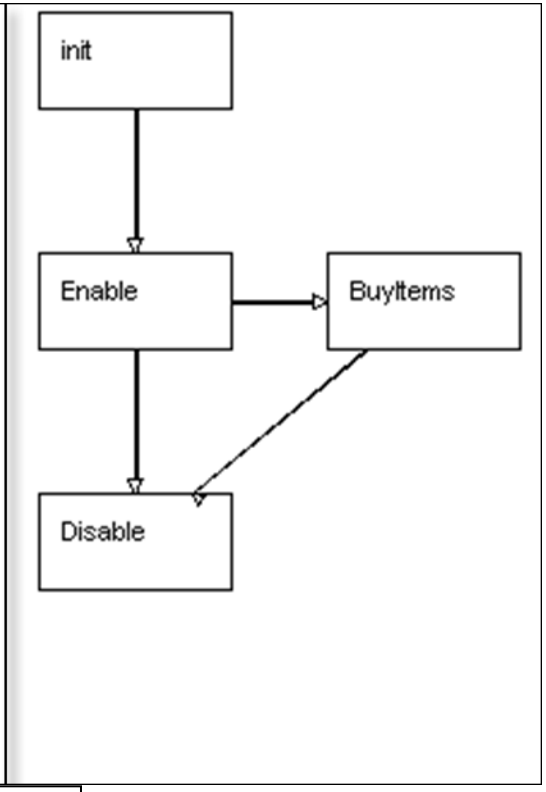
- Extending on the foundation of Implied Scenario detection (Sebastian Uchitel 2003) to search for hidden race conditions, while allowing for evaluation of security with the presence of negative behaviour

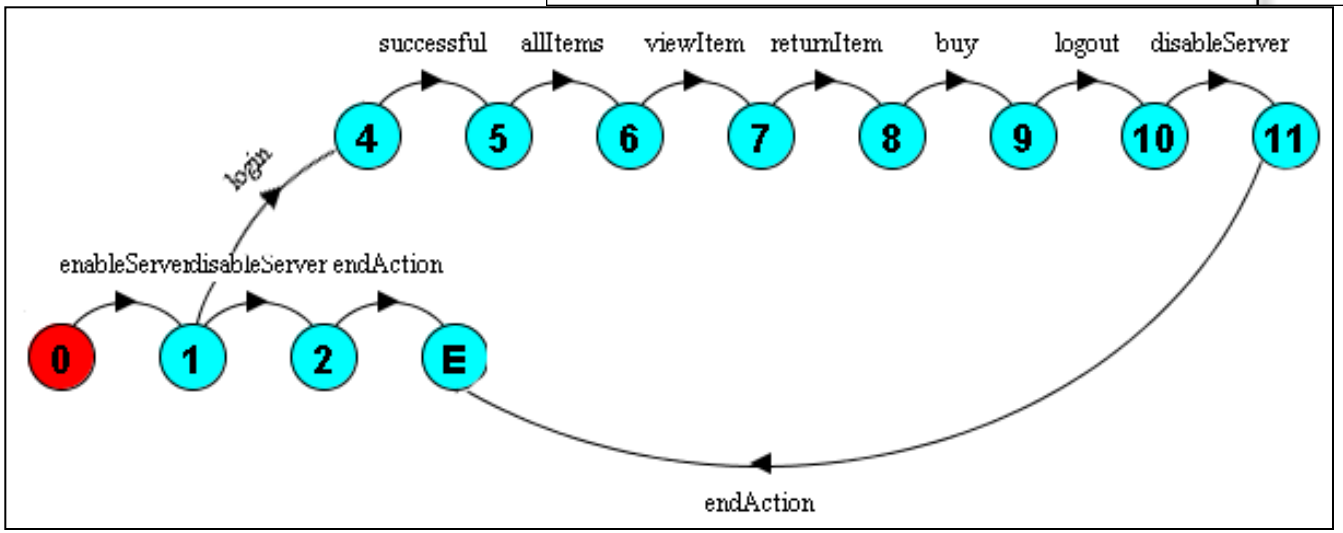
Implied Scenarios Detection

- Implied Scenario detection algorithm introduced by Uchitel 2003 in LTSA-MSC tool
  - Incremental elaboration algorithm using behaviour models for detecting implied scenarios from incomplete scenario-based models
  - Dynamically combining different scenarios together to provide an architectural view of system behaviour.
  - Architecture model is the parallel composition of a collection of LTSs, where each LTS model represents the local knowledge of each component from all scenarios

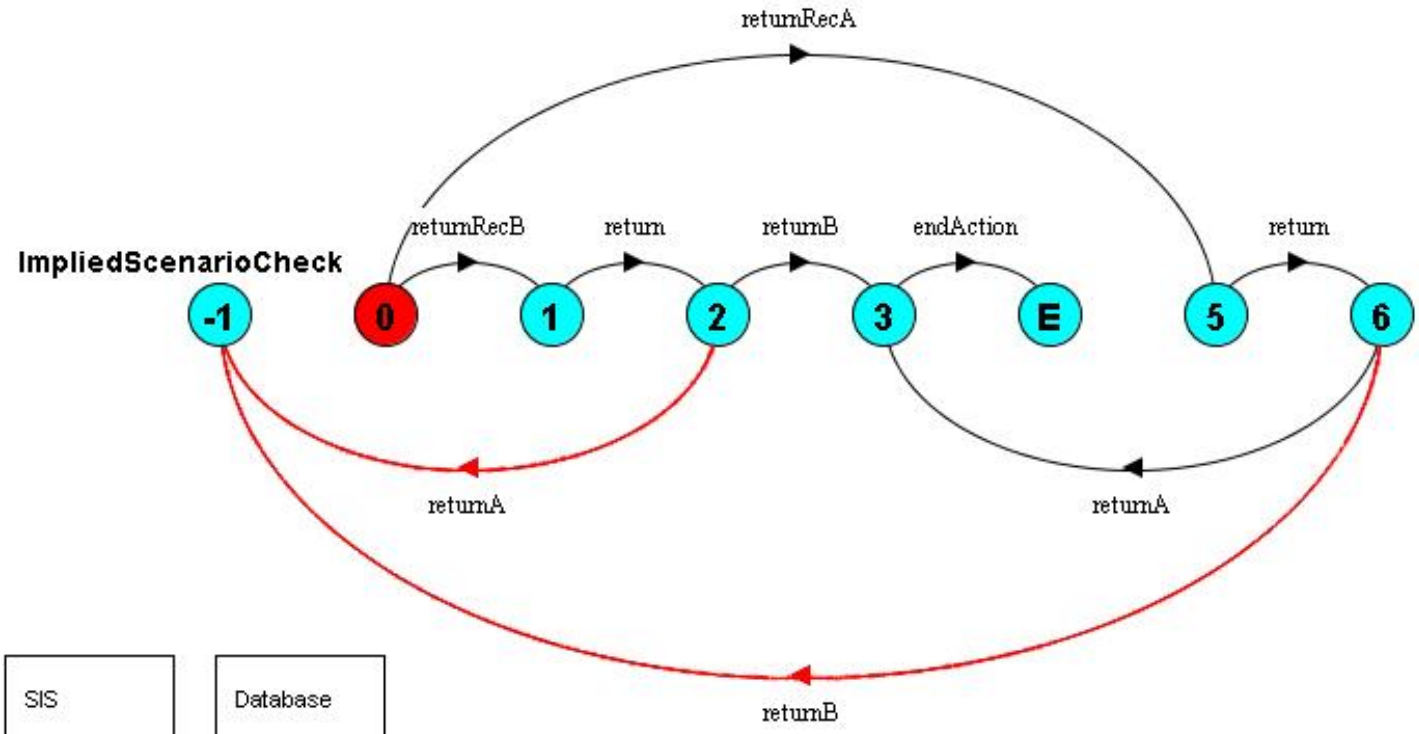
Simplified Example

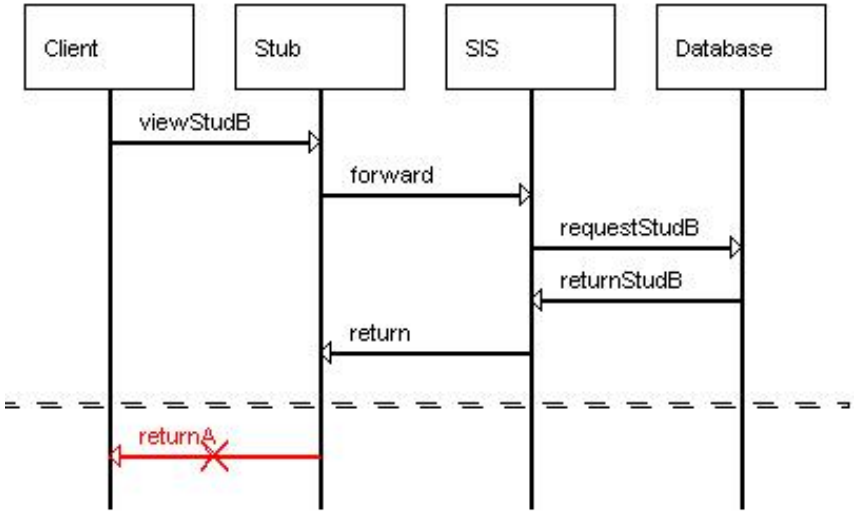






Example of detected implied scenarios





Limitations in the Implied Scenario algorithm

- **Does not provide complete coverage of possible traces**
  - supporting FIFO queues only, with strong assumptions about message orderings
  - Scenarios are only composed in parallel when there is a shared message between the involved scenarios
  - From previous initial example, each individual scenario yields the traces:

    1: enableServer

    2: disableServer

    3: login > successful > AllItems > selectItems > returnItems> buy > logout
  - The behaviour model on the other hand, gives the following traces

    1: enableServer > disableServer

    2: enableServer > login > successful > AllItems > selectItems > returnItems> buy > logout > disableServer

- ***Is this the total number of possible traces? No***

Trace difference between Behaviour and Scenario models

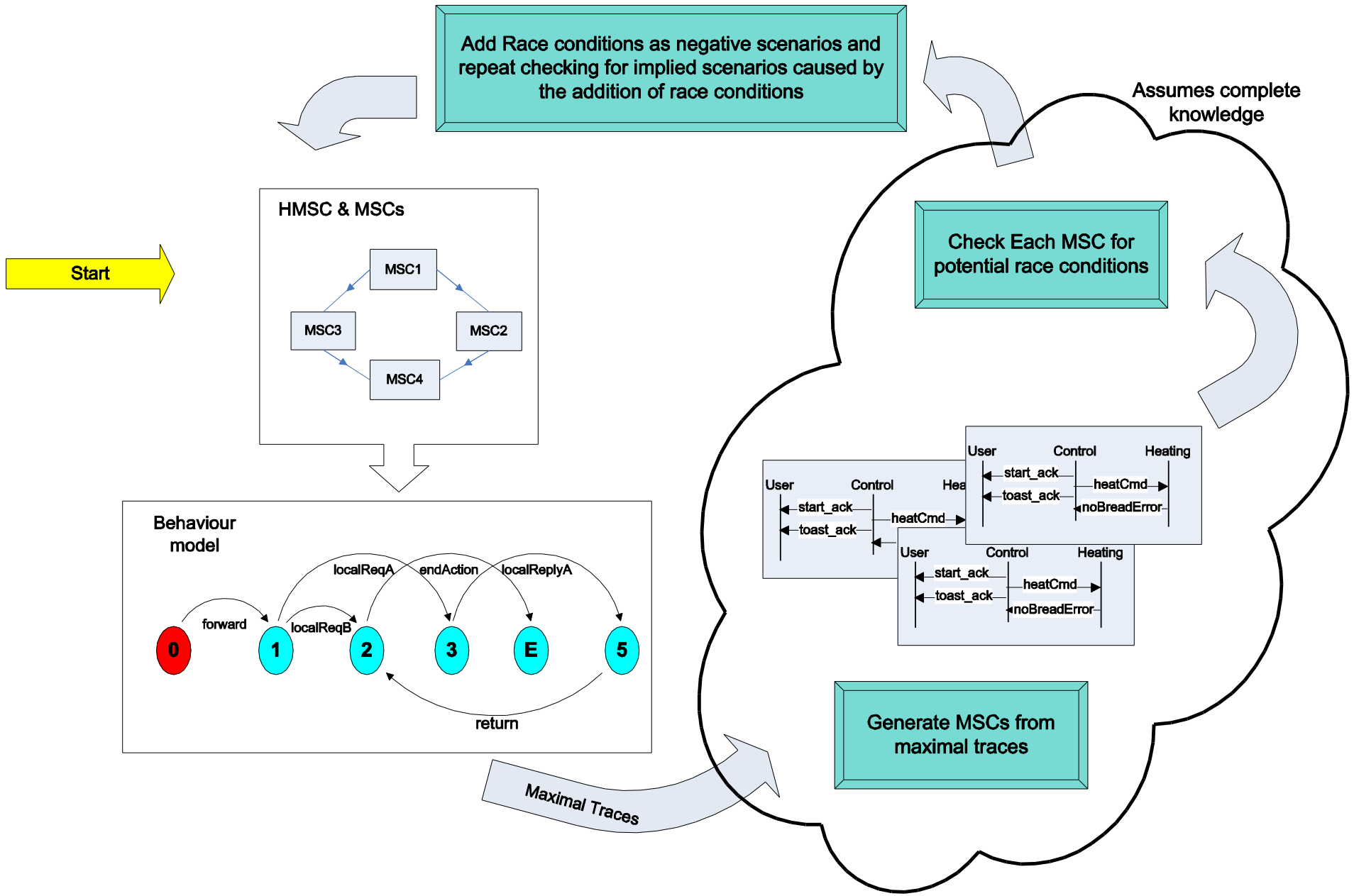
- Behaviour model traces do not model the scenarios individually, but instead they model the
  1. Composition of scenarios from *multiple component views*
  2. Possible continuations of scenario
  3. Hidden implied scenarios.

- Analysing each sequence diagram may result in:
  1. Only sub traces being addressed rather than overall maximal execution
  2. What might be reported as a race condition might be acceptable in another scenario in the specification.

- We can obtain a more holistic view of concurrent behaviour by merging the behaviour model and interaction models.

Proposed Approach

- Use LTSA-MSC tool to search for implied scenarios (Uchitel 2003)
- Use UBET tool to search for Race conditions (Alur 2000)
- Proposal: Searching for race conditions in behaviour model traces
  - STEP1: Take specified scenarios + HMSC into the LTSA-MSC tool.
  - STEP2: LTSA-MSC tool generates the architecture model and reports detected Implied scenarios.
  - STEP3: Transform all maximal traces of the architecture model into an MSC form; these traces can be generated using a built in simulator in the LTSA-MSC.
  - STEP4: Feed the new MSCs into the UBET tool to search for race conditions.
  - STEP5: If negative race conditions are found, feed back into the LTSA-MSC tool and update the HMSC, then repeat step 1.
  - STEP6: concrete test cases are built from implied scenario traces and race condition traces.


Add Race conditions as negative scenarios and repeat checking for implied scenarios caused by the addition of race conditions
Assumes complete knowledge
HMSC & MSCs
MSC1
MSC2
MSC3
MSC4
Start
Check Each MSC for potential race conditions
User
Control
Heating
start_ack
toast_ack
heatCmd
noBreadError
Behaviour model
localReqA
endAction
localReplyA
forward
localReqB
0
1
2
3
E
5
return
Maximal Traces
Generate MSCs from maximal traces

Attibute comparison between SecArch and Uchitel2003 and Alur2000

Criterion	Uchitel	Alur	SecArch
Assumes complete knowledge of environment		✔	
Assumes incomplete knowledge of environment	✔		✔
Produces all possible executions of modelled system	✔		✔
Multi-scenario analysis (i.e. produces maximal traces)	✔		✔
Single scenario analysis (i.e. produces sub-traces)		✔	
Searches for specification gaps (i.e. implied scenarios)	✔		✔
Searches for race conditions		✔	✔
Supports High-level MSC to infinite traces	✔		✔
Finite traces (i.e. bounded MSCs)		✔	✔
Synctactic analysis		✔	✔
Semantics analysis	✔		✔
Models timing		✔	✔

Figure 4. Attribute comparison between Uchitel's algorithm [23], and Alur's [3], and SecArch

Case Study: Architecture interfacing the cloud

- A bank adopting SaaS cloud provider, Salesforce.com to process their risk data;
- Started with 11 scenarios representing the requirements
- The architecture consists of 7 components with two types of users, registered-users and administrators

Searching for implied scenarios and race conditions on every scenario individually

	UBET	LTSA-MSC
Scen1: Synchronise	0	0
Scen2: User Registeration	0	0
Scen3: Subscribe	0	0
Scen4: Set Fields Encrypted	0	0
Scen5: Set Fields Decrypted	0	0
Scen6: Revoke user	0	0
Scen7: View Regulated Data	0	0
Scen8: View Plain Data	0	0
Scen9: Revoke Key	1	0
Scen10: Save Regulated Data	0	0
Scen11: Save Plain Data	0	0

Figure 6. System scenarios tested using LTSA-MSC [23], and UBET [14]. Results indicate one race conditions found

Results

	Scenario No.				Traces	Alur	Uchitel	Single Cycle	Positive	Security IS
Combo1	Scen10	Scen9	Scen1	Scen2	9	2	1	4	3	6
Combo2	Scen3	Scen6	Scen5	Scen1	17	29	8	4	16	4
Combo3	Scen5	Scen4	Scen6	Scen10	17	33	10	7	21	8
Combo4	Scen7	Scen11	Scen10	Scen8	9	0	5	N/A	3	2
Combo5	Scen1	Scen8	Scen6	Scen5	11	7	4	2	5	4

gure 7. Results from the composition of sets of scenarios using LTSA-MSC [23], and UBET [3]. Scenario names are listed in Figure

	Scenario No.				Confidentiality	Integrity	Availability	Security Scenarios
Combo1	Scen10	Scen9	Scen1	Scen2	0	3	3	6
Combo2	Scen3	Scen6	Scen5	Scen1	1	2	1	4
Combo3	Scen5	Scen4	Scen6	Scen10	2	4	2	8
Combo4	Scen7	Scen11	Scen10	Scen8	1	0	1	2
Combo5	Scen1	Scen8	Scen6	Scen5	1	2	1	4

Figure 10. Results of potential security consequences from the composition of scenarios using LTSA-MSC [23], and UBET [3]

Usage Scenarios

- Early test case and test suite generation
  - demonstrated our ability to enrich current existing test suite to include security related test cases
- Architecture refinement
  - Including detected positive scenarios
  - Correcting design errors
- Security risk assessment

Further…

- Automation
  - The process can be easily automated
  - Requires translating the inputs between the tools forwards and backwards
  - Omitting repetitions
- Generality and Applicability
  - Working at the architecture offers an adequate level of generality
  - Applicable for threaded systems, such as concurrent and real time systems
  - Tried on Identity Management Systems, web script design, Smart camera distributed system, etc…
- Phases of Application
  - Approach can begin at analyses and design phase
  - Supports early test case generation for test-driven applications
- Scalability
  - Lack of scalability in the LTSA-MSC tool requires repetition
  - Any tool that is capable of composing scenarios and searching for hidden implied scenarios can be adapted.

Summarising!

- Proposed a systematic architecture evaluation methodology to search for potential vulnerabilities in the specification
  - Reduce the search time and the chances of overlooking vulnerabilities
  - Semantics and syntactic implication of specifications
  - Reduces subjectivity
  - Takes into account incompleteness of specifications
  - Do not make assumptions on how the implementation might prevent an issue from occurring
  - Design-specific to search for design vulnerabilities.
  - Evaluating the security posture with presence of negative behaviour
    - Might be expensive to redesign a system
  - Forms building blocks for searching for multi-step attacks

€10.00 per question please! ☺

S.Al-Azzani@cs.bham.ac.uk