Testing security and resilience

Ana Rosa Cavalli – Montimage
Institut Polytechnique de Paris
/Telecom Sudparis

ITEQS 2021

Outline

- Security testing
- Resilience
- H2020 VeriDevOps project
- Conclusion

Security Testing

- Testing: The process of executing software with the intent of finding and correcting faults
- Conformance testing: The process of checking if the implementation under test conforms to the specification
- Security testing: The process of checking if the implementation under test satisfies security requirements

Challenge

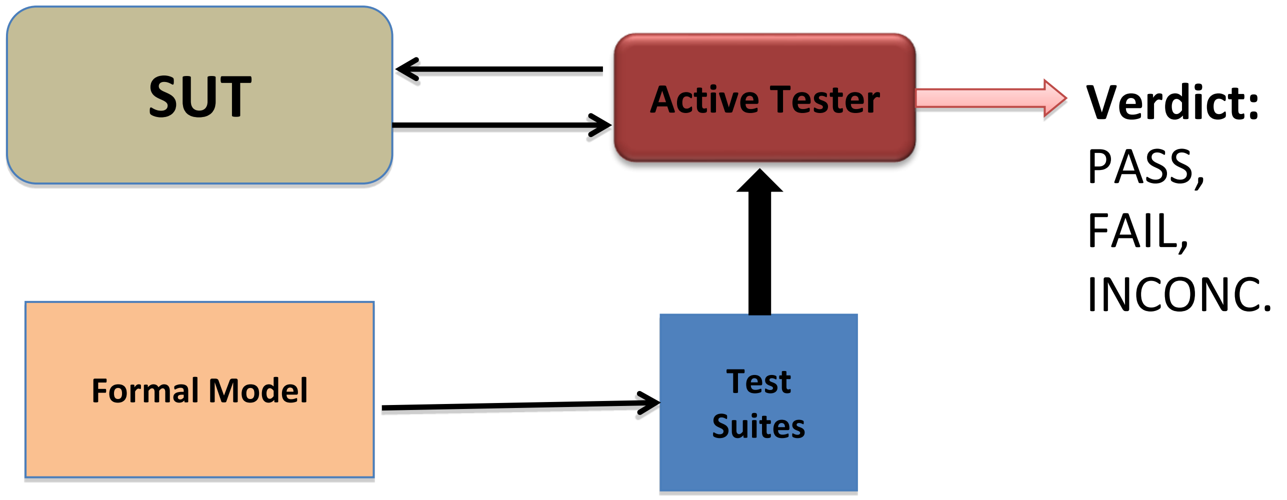
- How can we enable testing techniques to ensure security?

Security Testing

- Model based security testing
- Penetration testing
- Fuzzing testing

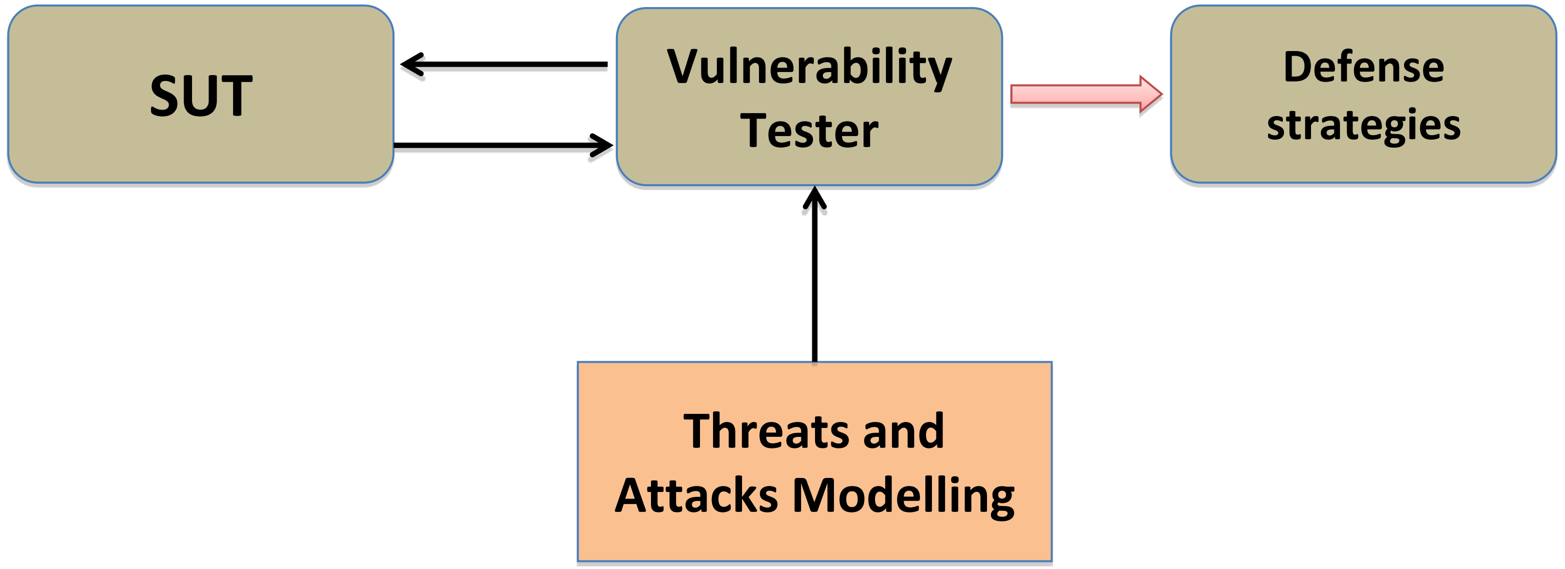
Model Based Security Testing

- It is assumed that the formal model includes security requirements
- Automatic test generation methods can be defined
- The tester can guide the implementation based on security test objectives
- Usually a test case is a set of input sequences
- Some researchers include risk analysis and assessment



Penetration testing (PenTest)

- Based on the injection of attacks and threats to detect vulnerabilites and security bugs in the system implementation (standard: PTES)
- Also it can be used to validate that mitigation mechanisms are working correctly and the vulnerability is not accessible



Penetration testing

- Active testing based on scanning (network and port based scanning, service based scanning, web application scanners)
- Active testing interacts with a target, examine responses, and determine whether a vulnerability exists based on those responses
- Passive testing used for metadata analysis (to impeach an attacker to access to data passively without directly attacking the target)

Fuzzing testing

- It is a brute-force technique for finding application flaws by submitting valid, random or unexpected inputs to the application
- Can be static (based on code review) or dynamic (based on random testing)
- Can be blackbox (not program analysis, scale well) or whitebox (program analysis, don’t scale well)

Resilience

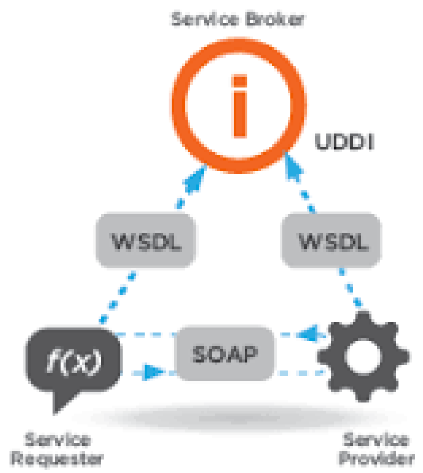
Resilience is the capability of a system to continue to function properly with minimal degradation of performance, despite intrusions or attacks. The idea is to reduce the risk of attacks and to enable a continuity of service

Challenge

- How to make systems and applications resilient ?

Example of an application domain

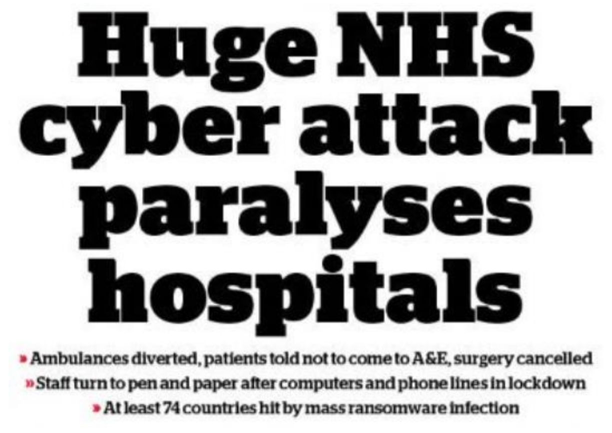
Web services



amazon
webservices™

Security issues

- Web servers and services are the targets of malicious attackers



Huge NHS cyber attack paralyses hospitals

» Ambulances diverted, patients told not to come to A&E, surgery cancelled
» Staff turn to pen and paper after computers and phone lines in lockdown
» At least 74 countries hit by mass ransomware infection

- Attack WannaCry National Health Service (UK) 2017
- Playstation Network April 2011. Millions of confidential data leaked

We focused on attacks at the network and application levels

Our approach

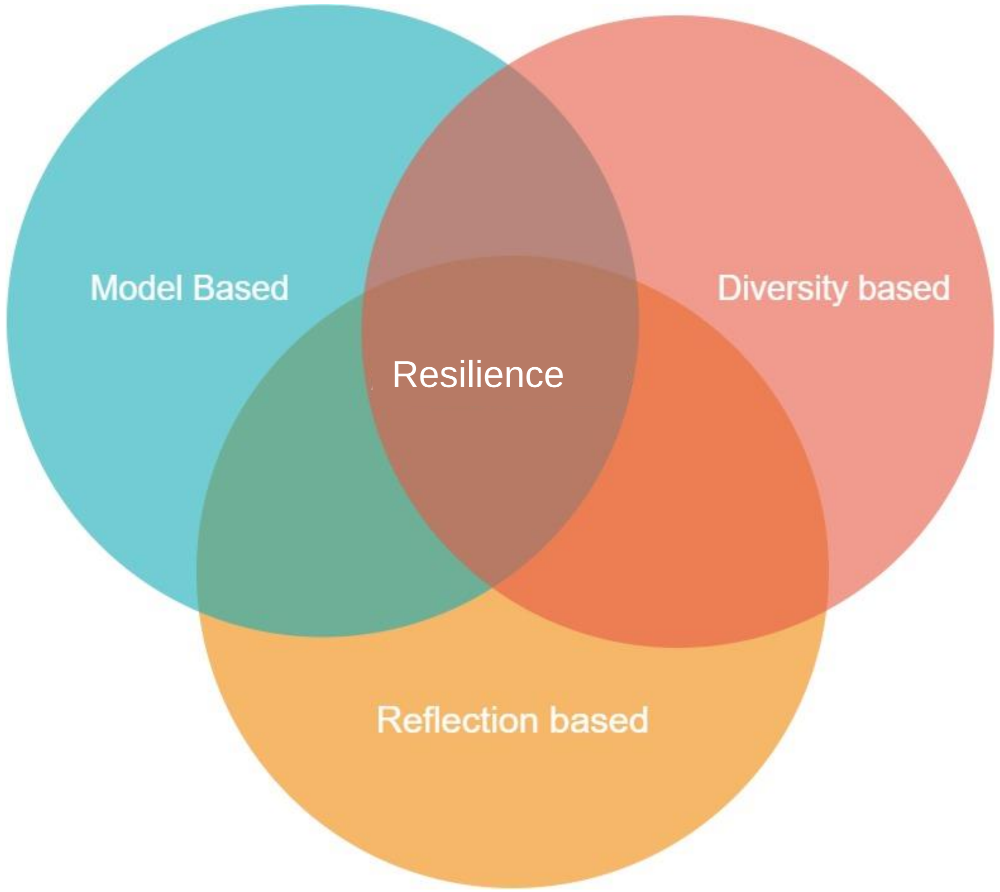
- If an attack is detected, we replace on the fly the components of our application that seems attacked, in order to mitigate the effect of the attack. In this way, the system will continue to work as previously.

Challenge

- What are the techniques to use to make replacements?

Our approach

Based on three main techniques:

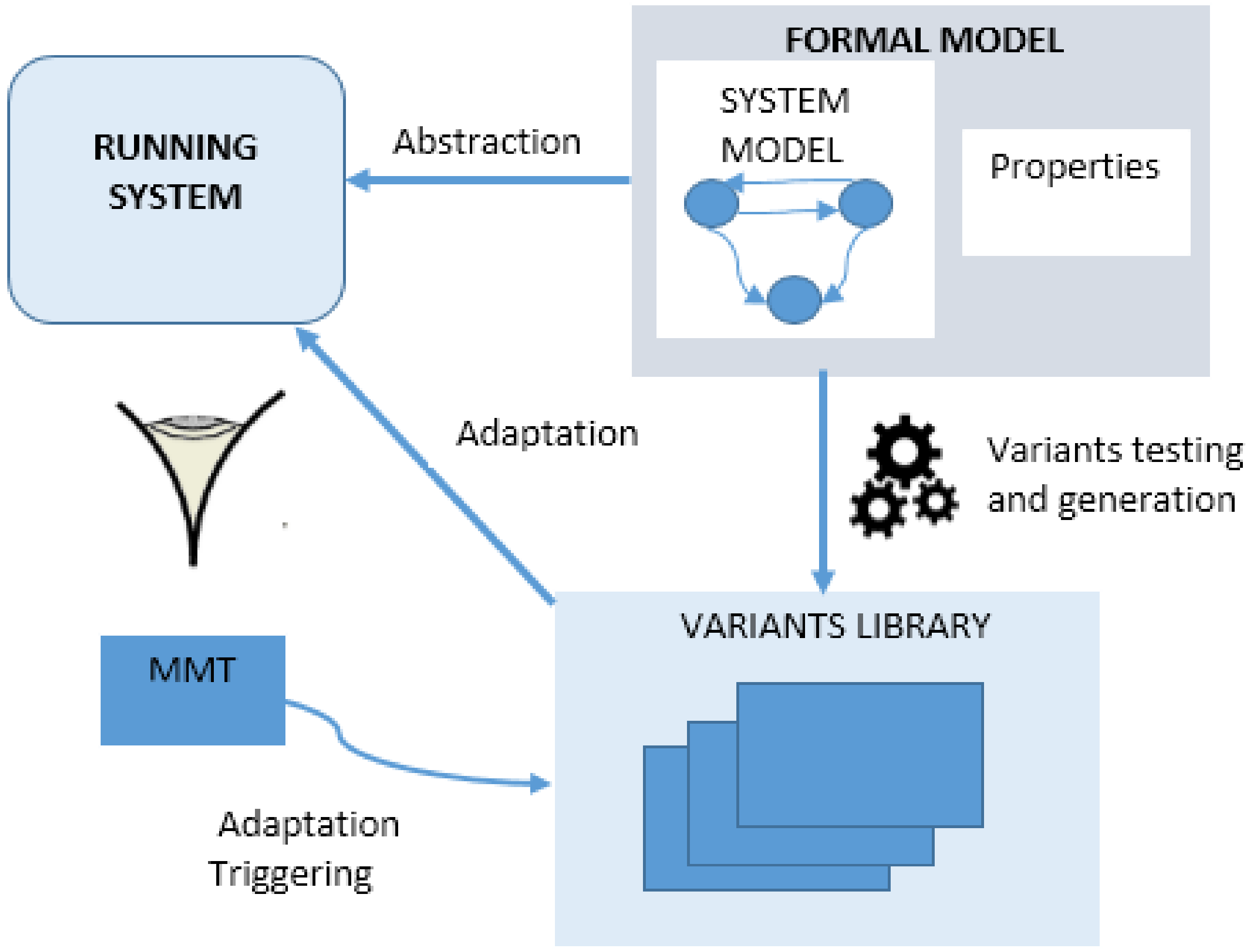


Model based

The main steps of this approach are the following :

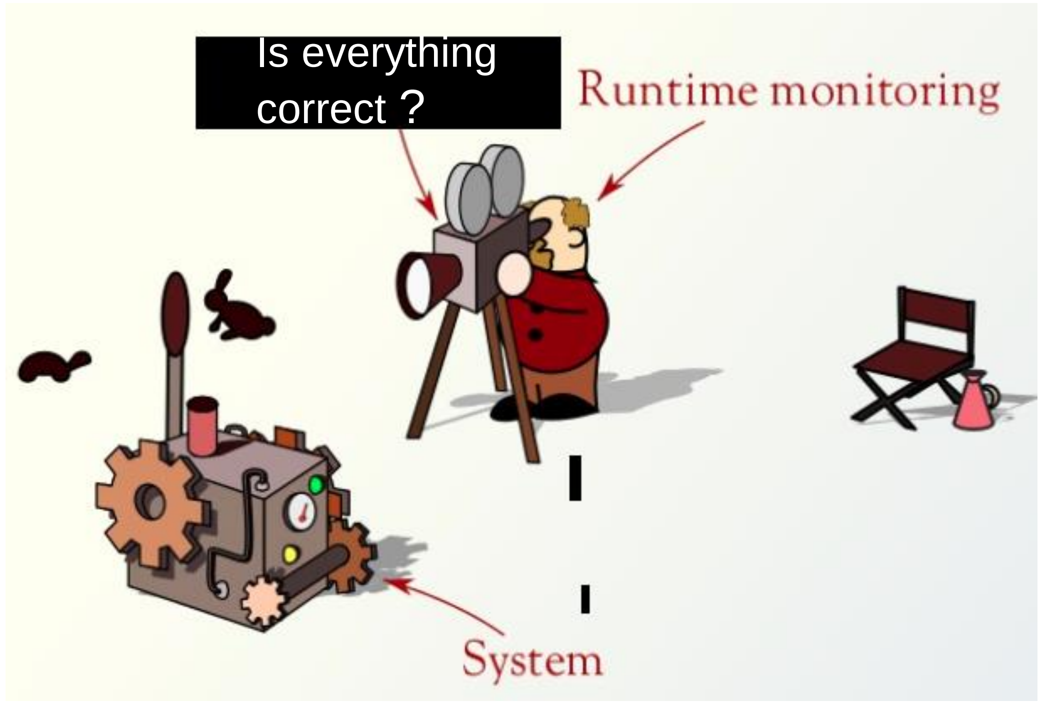
1. We design a formal model of the application;
2. We derive some functional equivalent variants of this formal model;
3. We monitor the running system in order to detect eventual attacks;
4. We replace the running application’s model with a new one if the monitoring tool detects some attacks.

Model based



FORMAL MODEL
SYSTEM MODEL
Properties
RUNNING SYSTEM
Abstraction
Adaptation
Variants testing and generation
VARIANTS LIBRARY
MMT
Adaptation Triggering

Why runtime monitoring?



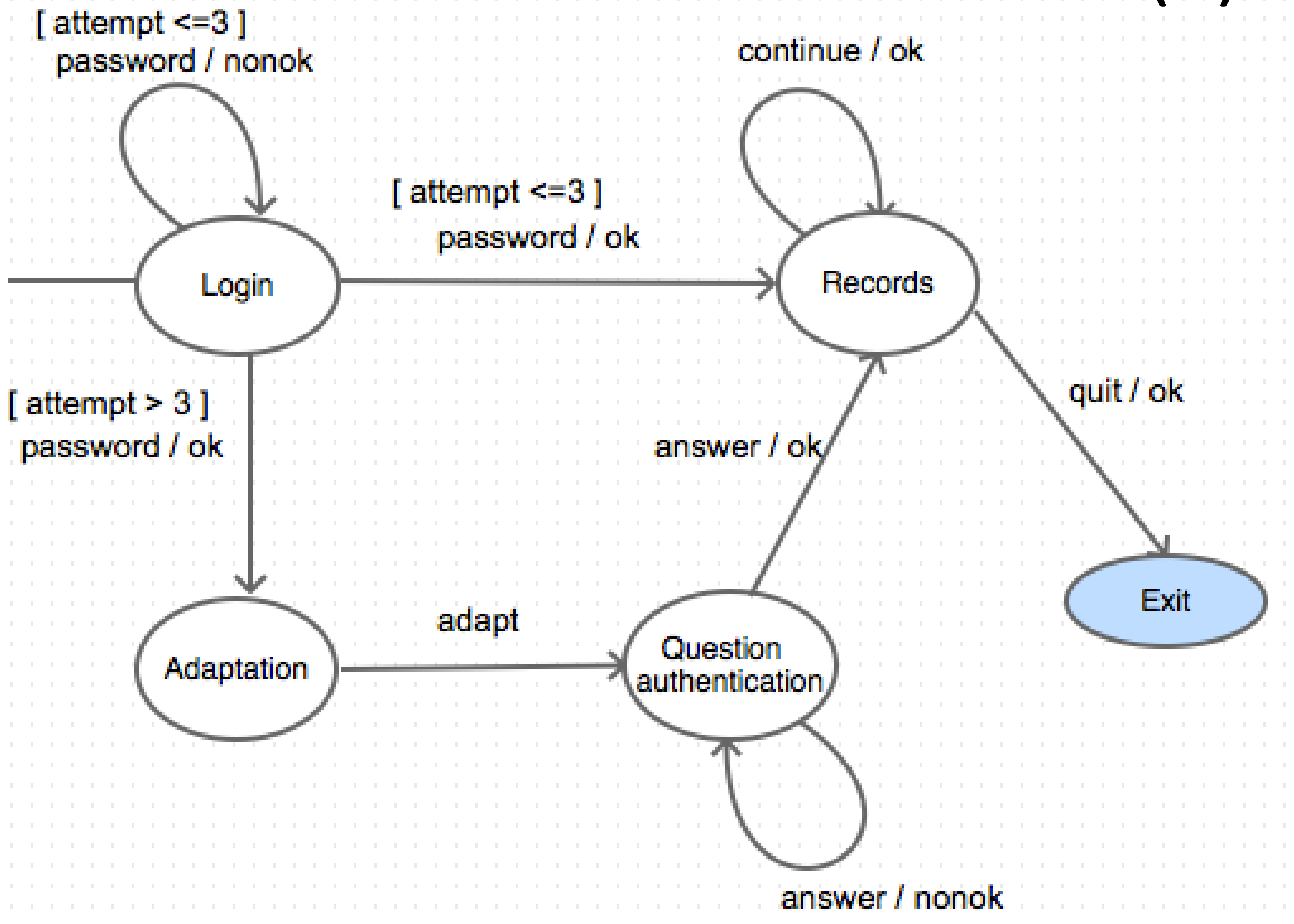
- For detecting the security threats, it is important to monitor the system at runtime.
- Runtime monitoring is the process of observing and verifying some properties of the system
- It helps to understand the behavior of the network and the application in order to detect faults and abnormal operations

Authentication example

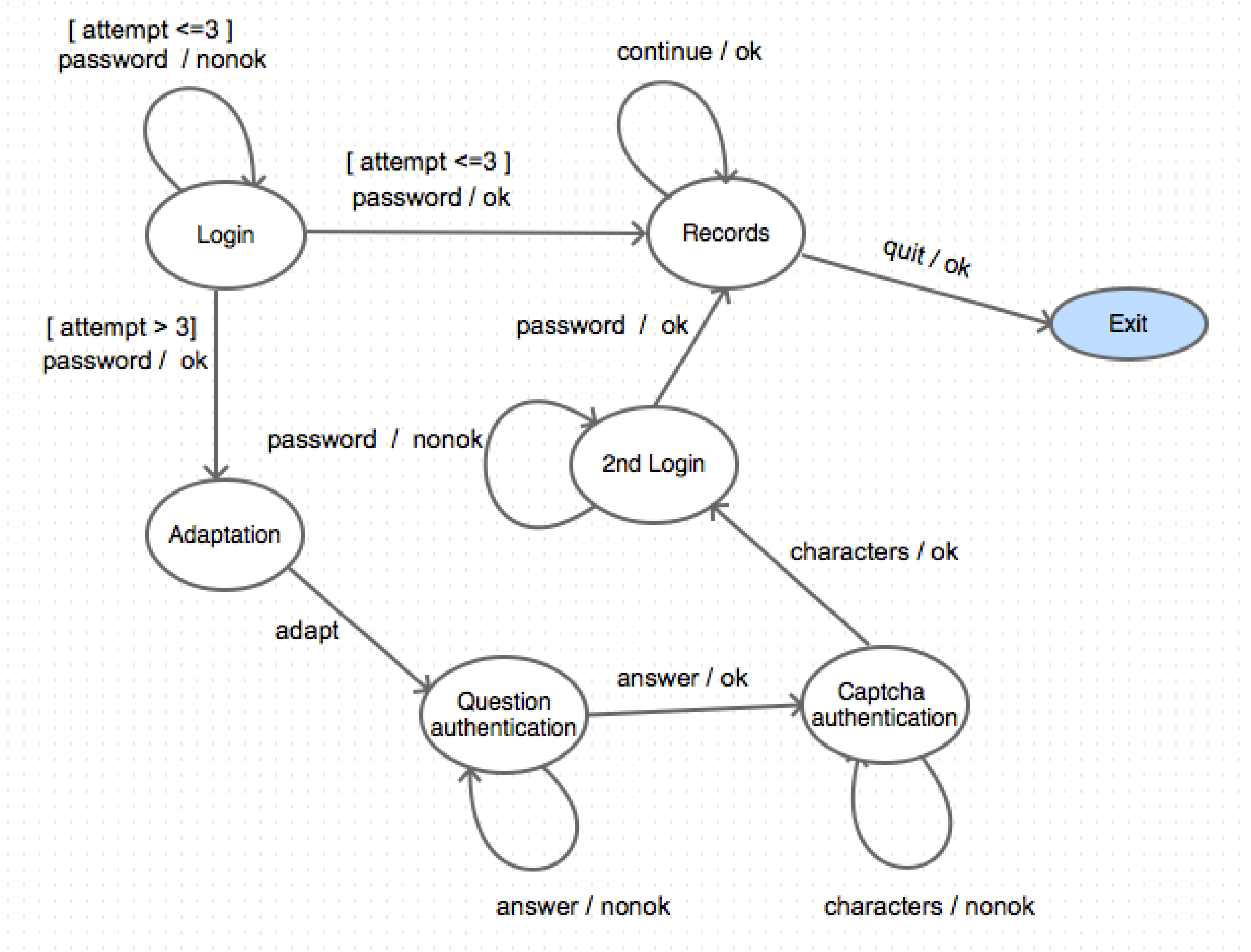
We illustrated the method with an authentication application. We made the following assumptions:

1. Three attempts as a threshold;
2. If the number of attempts is less than or equal to 3 and the password is OK, then the user is successfully connected;
3. If the number of attempts of a user is greater than the threshold but the password is OK , then, we react asking for more information.

Authentication finite state machine (2)



Authentication finite state machine(3)



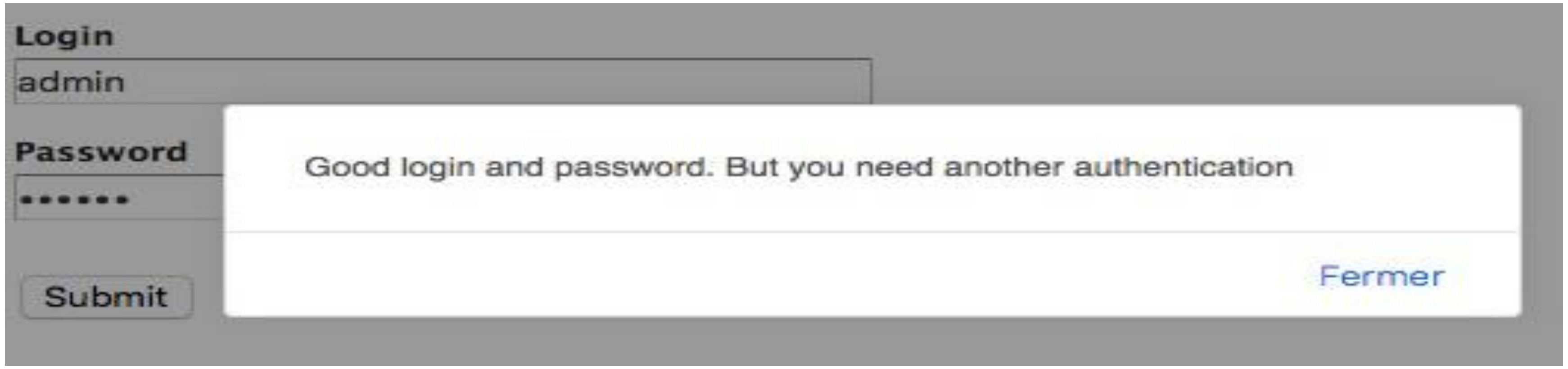
Experiments

- Experiments were performed on the authentication application
- Our framework was divided into two main parts:
  1. The specification part : We modeled the automatons in IF language and we generated the tests cases with TestGenIf;
  2. The attack execution part : We implemented these tests using Selenium IDE1 describing a brute-force attack.
- With that framework, we conducted a brute-force attack. The goal is to check the robustness of the implementations, i.e., the average time that an attacker needs to get access to the relevant information

Experiments

- **Scenario 1:** A brute-force attack is launched to verify the detection and reaction capability
- **Scenario 2:** The same brute-force attack is launched against the three implementations to check the robustness (means time needed to access the confidential data)

Results


Login

admin

Password

••••••

Submit

Good login and password. But you need another authentication

Fermer

The authentication showing the adaptation GUI

Implementations	Average time (s) to break the authentication
Implementation 1	3 384 = ~0.94 hour
Implementation 2	3 689 = ~1.02 hour
Implementation 3	21 736 = ~6 hours

This means that an attacker needs more time to break the authentication in the second and third models than in the first one

Limitations

We proposed a model-based resilience approach. However, there are some limitations that should be addressed:

- For our authentication example the design of new models was simple. But it is not easy to derive different models from a more complex application

Diversity based (1)

- Definition: Diversity is the quality or state of having many different forms, types, ideas, etc.
- The aim of the approach is to bring together diversity and monitoring for enabling resilience
- We leverage diversity for reducing the limits of model-based attack described previously
- We only design a main model and we derive the corresponding implementations

Diversity based (2)

- The aim is to enable diversity at the model level based on Model-Based Testing. This approach consists in :

  1. The design of a model of the system;
  2. The derivation of the alternatives implementations of that model and the model-based testing of these alternatives (to check they accept the same tests) ;
  3. The monitoring of the running system;
  4. The adaptation of the running system's model and implementation.

Example: e-health Web service

- We propose an e-health Web Service example that is a software for the management of patients of an hospital
- The simplified version of the API consists of 4 methods:

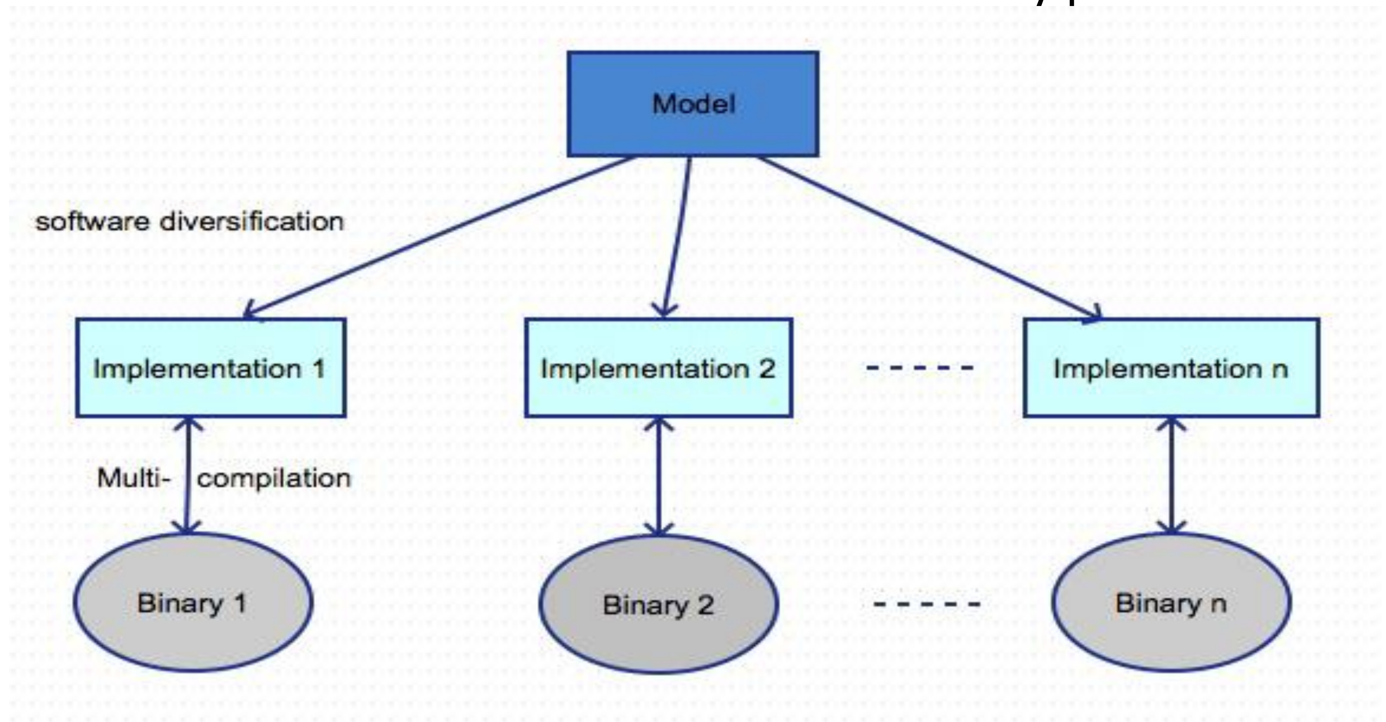
HealthOperation
dologin(login, password) : void listPatient() : boolean
createPatient() : boolean updatePatient(idPatient, token) : boolean

API of the Health Operation Centre

Diversity based (3)

We illustrated the approach with this web service that simplifies the management task in a hospital. The methodology is the following:

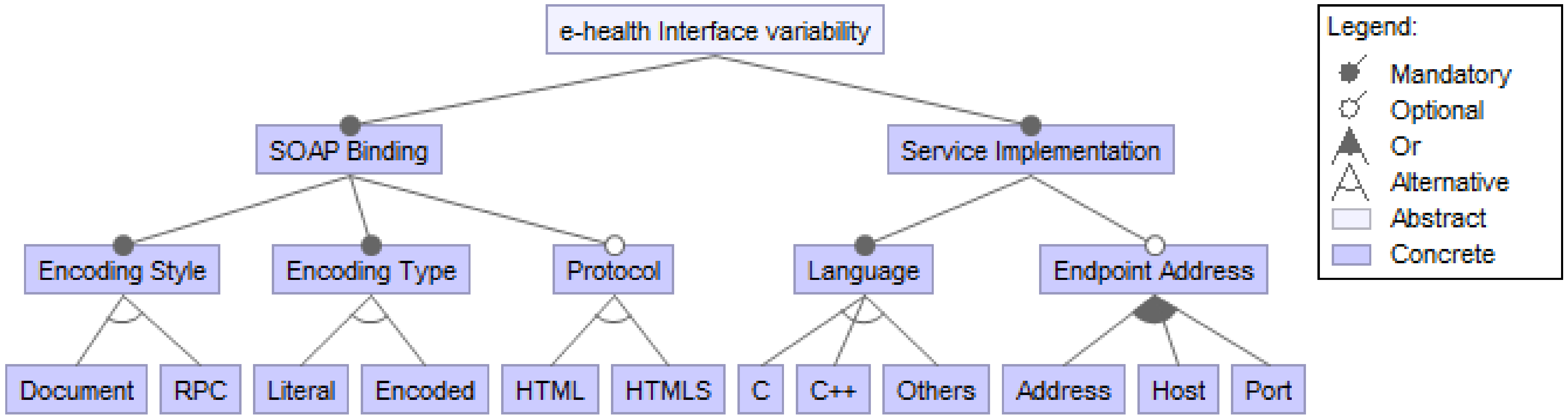
- We define a Feature Model that describes the variability points of the service.



Overview of the diversity based-approach

Diversity based (4)

- We used for that example three variability patterns (Encoding style, Encoded type, Language) and defined the corresponding WSDL files
- We used gSOAP a C/C++ compiler, to generate the corresponding skeletons of the code of our Webservices
- We used the multi-compiler to obtain different equivalent implementations of the Webservice



Diversity based (5)

- Monitoring and reaction:
  - To ensure the continuos availability of the service we configure the system in two modes:
    - Normal mode. Time is divided in *epochs*. In each *epoch* only one implementation is chosen. When the *epoch* of time elapses, another implementation is deployed to ensure continuity of the service
    - Attack mode: an attack has been detected by the monitoring tool. In this case, a change of implementation is performed before even the *epoch* has elapsed. The change is performed without changing the endpoint so the change is transparent to end user.

Reflection based

- Software reflection is a way of implementing meta programming techniques (programs manipulating themselves)

- It was designed for programs that require the ability to examine or modify their behaviours at runtime

- We propose to use software reflection to improve resilience

Assumptions

The main assumptions of the approach are the following :

- Even if the environment is safe, we assume that the unsuspecting use of employees (e.g. clicking of an email attachment) can lead to malware exposures

- The aim of the attacker is to usurp the actions, i.e., to modify the methods of the API of the platform

How to detect misbehaviours

- For detecting attacks, we use logs located on the two endpoints: on premises, on the server.
- We store the date and time, the name as well as the hash of the stack of any running code.
- Consequently, any line in all logs has the following format:
  - *Date Hour Operation Hash Host*.
- Any request has then two traces in the logs:
  - outbound (request) and
  - inbound (response)

Normal interactions

Normal entries in the log of the client application:

Date	Hour	Method	Hash	Host
07/11/2017	10:00:00 AM	!createPatient(Outbound)	2224d35250e...	a
07/11/2017	10:15:00 AM	?createPatient(Inbound)	2224d35250e...	b

Abnormal Interactions

Incorrect entries in the log of the client application

Date	Hour	Method	Hash	Host
07/11/2017	10:00:00 AM	!createPatient(Outbound)	2224d35250e...	b
07/11/2017	10:15:00 AM	?createPatient(Inbound)	345672dade2...	**a**

Remediation

To mitigate an attack, we propose:

- to dynamically change the implementation of a compromised method at runtime with the same code or with an alternative code

EXPERIMENTS

The e-health Web service has been developed in python (FLASK).
Two different experiments :

1. We measured the average time to make a client request without attack and when the attack is detected;

2. We evaluated the ability of the framework to detect viruses attack in comparison to a conventional anti-virus

Security rules

Some security rules are applied. For example:

- **Rule 1**: Any createPatient request should have hashes for its operations (outbound and inbound) on the log files and these hashes must correspond

- **Rule 2**: Any outbound operations should be followed by an inbound operation

Related works on attack tolerance

- **Some techniques** (Indirection, recovery, voting, ...); and architectures (MAFTIA [Stroud et al., 2004], [Deswarte and Powell, 2004],...); simulation and testbeds [Conti et al., 2021]
- **Some frameworks** [Karande et al., 2011] ...; Constable et al. 2010 proposed an approach based on deductive verification ; protocols are specified correct and secure by construction; these protocols are members of an attack tolerance library
- **Few works** on resilience for Web Services [Sadegh and Azgomi, 2015], [Ficco and Rak, 2011]
- **Resilience for CPSs**: [Segovia et al 2021], A Survey on Taxonomy and Systematic Review of Cyber-Resilience Techniques for Attack Tolerance in Cyber-Physical Systems, submitted for publication

Remarks

- Existing solutions are mainly attack-specific;
- No evidence of their protection against harmful silent attacks and insider attacks

New: Moving Target Defense approach

- Based on Moving Target Defense approaches that provide strategies that change the system over time to increase its complexity, attack cost and limit the exposure of vulnerabilities
- Can be applied at the network and node level (randomly selects routing nodes to change the forwarding paths while ensuring reachability, many forms of the same object, changing randomly the route of the multiple flows, changes the availability of the sensor data)
- In our case applied to Cyber Physical Systems (CPSs)

Challenge

- How to design a system that is able to adapt itself to achieve resilience?

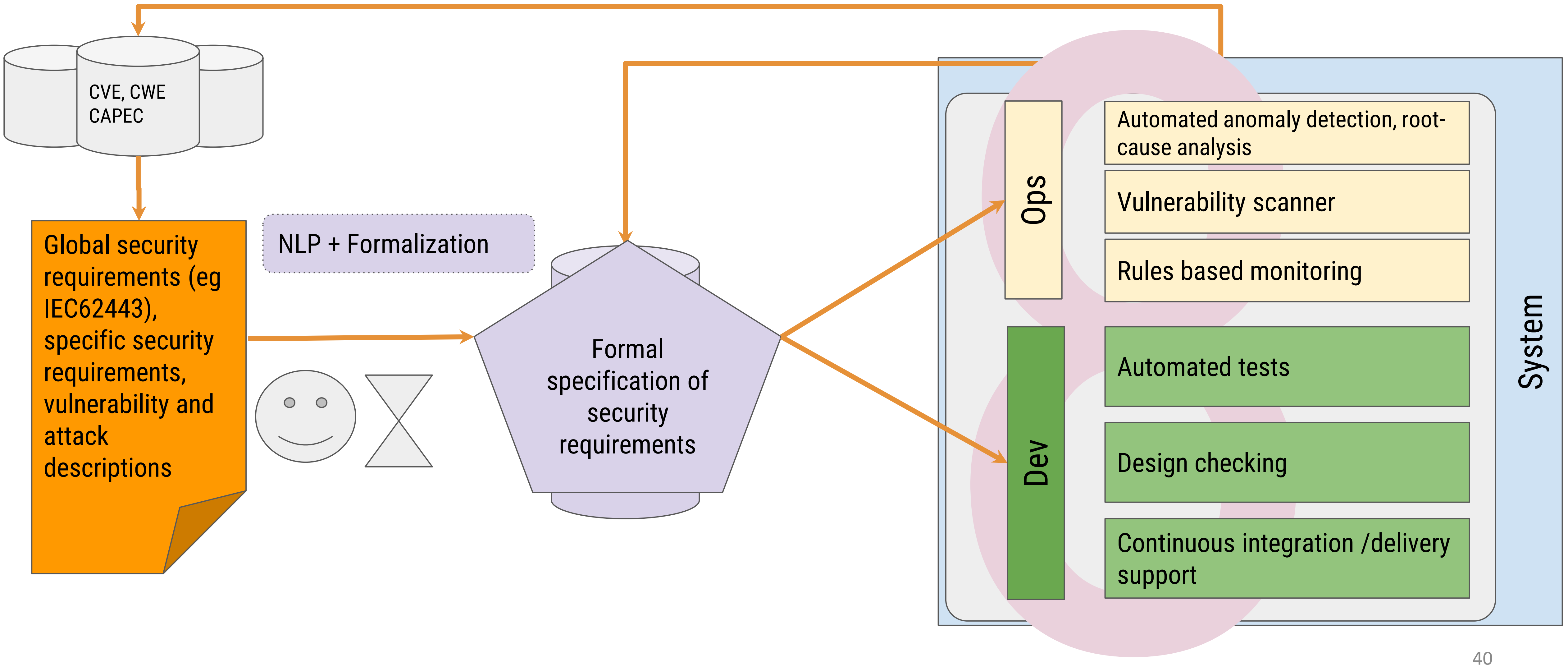
H2020 VeriDevOps project

MÄLARDALEN UNIVERSITY SWEDEN SOFTEAM UNE MARQUE DE DOCAPOSTE Åbo Akademi University ikerlan montimage FAGOR ABB

- Main objective: to improve automation to protect and preventing security issues based on:
  - formalizing security requirements from its formulation in natural language
  - validation based on model checking and penetration testing
  - to perform security monitoring at runtime, to ensure app/system/network monitoring at different layers based on a passive capture of traces and ML/AI based techniques
  - locating root causes of vulnerabilities, and identifying security flaws in code and design
- This will improve productivity and contribute to the design of trusted systems.

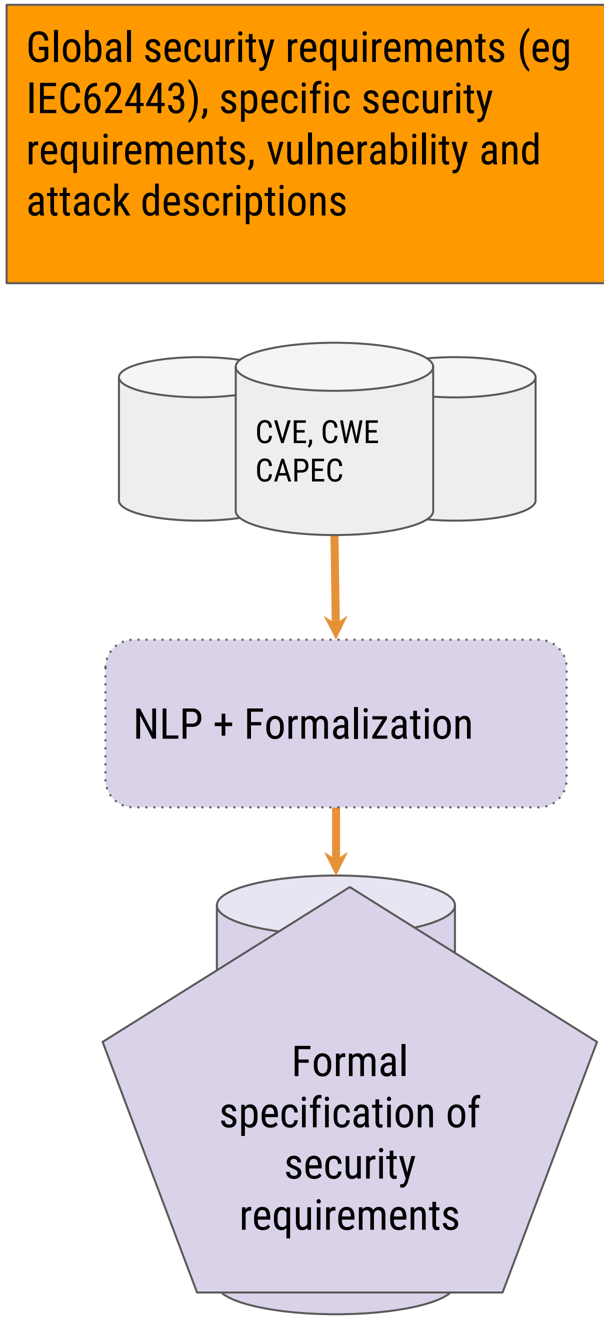
Main idea

Active vulnerability discovery, reporting and recommendations


CVE, CWE
CAPEC
Global security requirements (eg IEC62443), specific security requirements, vulnerability and attack descriptions
NLP + Formalization
Formal specification of security requirements
Ops
Automated anomaly detection, root-cause analysis
Vulnerability scanner
Rules based monitoring
Dev
Automated tests
Design checking
Continuous integration /delivery support
System

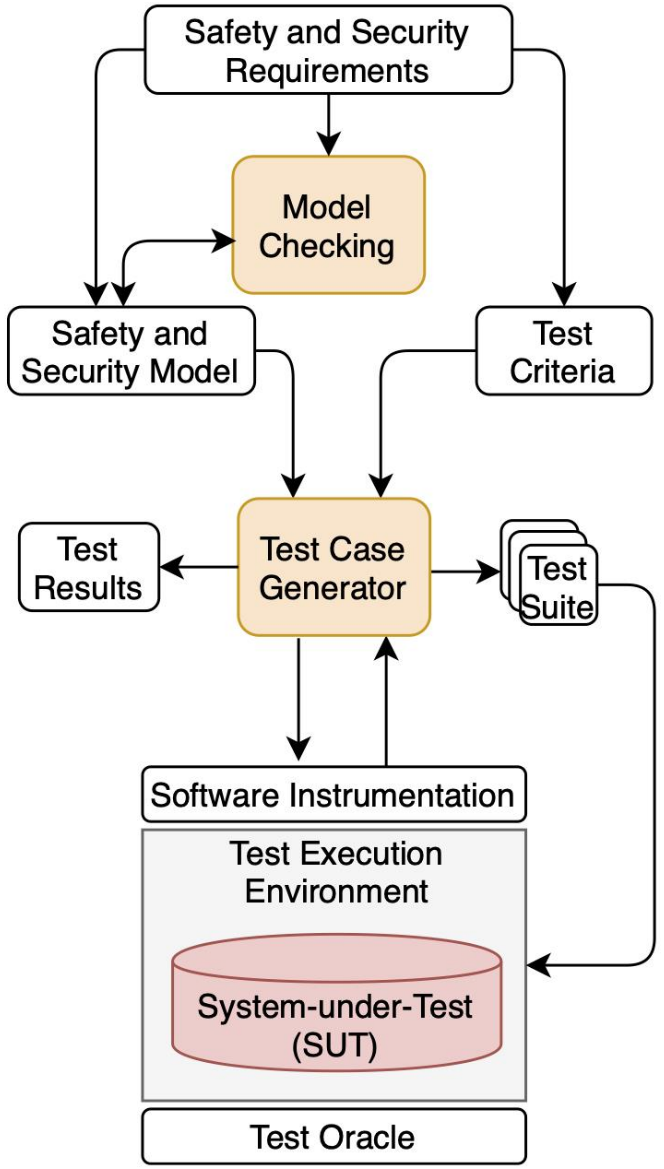
NLP for security specification modelling

- **Main objectives:**
  - Extract security requirements from unstructured text
  - Classify security requirements
  - Identify entities and properties
  - Apply formal specification patterns
- **Results:**
  - Security formal specification


Global security requirements (eg IEC62443), specific security requirements, vulnerability and attack descriptions
CVE, CWE CAPEC
NLP + Formalization
Formal specification of security requirements

Security at development

- **Main objectives:**
  - (Prerequisite) semi-automated extraction of relevant security information from safety assessments and security requirements.
  - (Prerequisite) model-checking of the resulting model or antimodel.
  - Test case generation and execution based on security test criteria.
- **Results:**
  - Methodology to validate safety/security functional requirements.
  - Full integration of the set of components

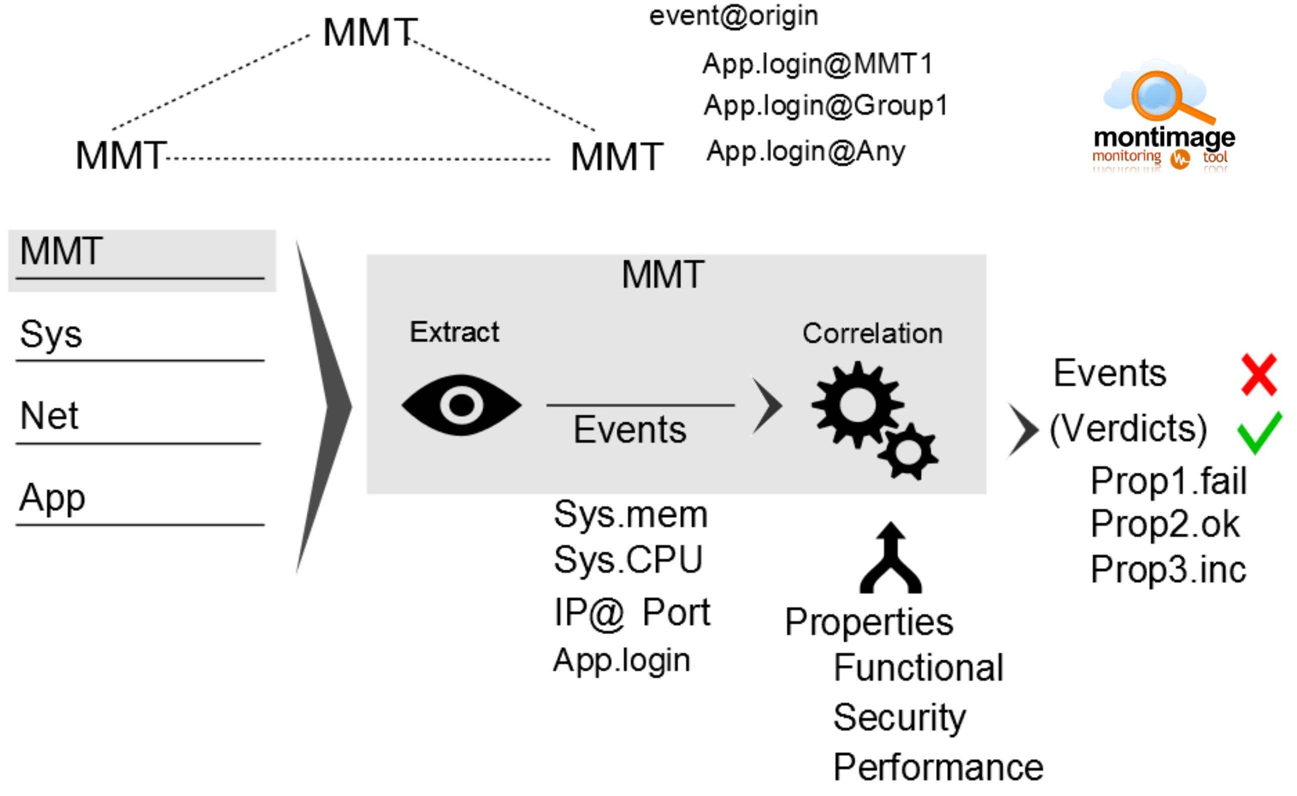


MÄLARDALEN UNIVERSITY SWEDEN

Åbo Akademi University

Security monitoring at runtime

- **Main objective:** passively observing or inspecting the app/system/network at different layers
- **With the objective of**
  - Drawing operation baselines
  - Produce reports
  - Notify on intrusions and anomalies
  - Provide input to management
- **Passive capture of traces (logs, network traffic, etc.), extracting metadata and correlating events**
  - Based on predefined rules
  - Based on ML/AI based techniques

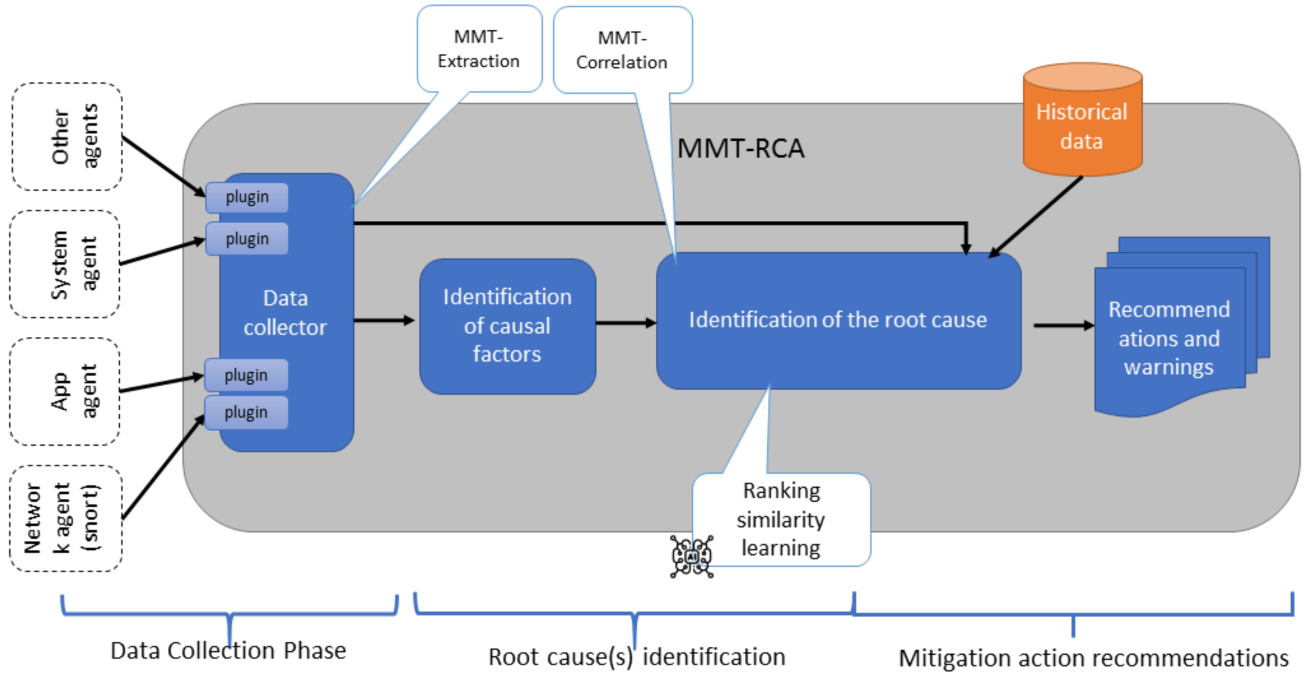


Root cause analysis

Main objective: identifying the root cause of incidents and alerts (when it is not obvious to be done)

We investigate:

- The suitable data inputs than can help identifying a cause of a detected issue
- Rely on similarity learning to be able to identify the probability of a root cause
- Provide accurate recommendations about the most adapted corrective action to mitigate the risk



MMT-Extraction
MMT-Correlation
Historical data
MMT-RCA
Other agents
System agent
App agent
Network agent (snort)
plugin
plugin
plugin
plugin
Data collector
Identification of causal factors
Identification of the root cause
Recommendations and warnings
Ranking similarity learning
Data Collection Phase
Root cause(s) identification
Mitigation action recommendations

Conclusion

- Experiments have been performed on the different approaches that showed good performance and scalability
- The proposed approaches have been successfully applied to Web services and CPSs

Future work:

- Investigate new approaches: application of ML techniques to improve the selection of the components to be replaced
- Investigate digital twins to improve attacks prediction
- Adapt the approaches to check resilience of other networks and systems (5G networks…)

References

- Georges Ouffoue, Antonio M. Ortiz, Ana R. Cavalli, Wissam Mallouli, Josep Domingo-Ferrer, David Sánchez, Fatiha Zaïdi. Intrusion Detection and Attack Tolerance for Cloud Environments: The CLARUS Approach. ICDCS Workshops 2016: 61-66
- Georges Ouffoue, Fatiha Zaïdi, Ana R. Cavalli, Mounir Lallali. How Web Services Can Be Tolerant to Intruders through Diversification. ICWS 2017: 436-443
- Georges Ouffoué, Fatiha Zaïdi, Ana R. Cavalli. Attack Tolerance for Services-Based Applications in the Cloud. ICTSS 2019: 242-258
- Georges L. A. Ouffoue, Fatiha Zaïdi, Ana R. Cavalli, Mounir Lallali. An Attack-Tolerant Framework for Web Services. SCC 2017: 503-506

References

- M. Segovia, A. Cavalli, N. Cuppens, J. Garcia-Alfaro. A Study on Mitigation Techniques for SCADA-driven Cyber-Physical Systems, Foundations and Practice of Security (FPS 2018), LNCS 11358, pp. 257-264, Springer, Montreal, Canada, November 2018.
- M. Segovia, A. Cavalli, N. Cuppens, J. Rubio-Hernan, J. Garcia-Alfaro. Reflective Techniques to Attenuate Cyber-Physical Attacks, 5th Workshop on the Security of Industrial Control Systems & of Cyber-Physical Systems (CyberICPS 2019), ESORICS 2019, LNCS11980, pp. 19-34, Springer, Luxembourg, September 2019.
- M. Segovia, J. Rubio-Hernan, A. Cavalli, J. Garcia-Alfaro. Cyber-Resilience Evaluation of Cyber-Physical Systems. NCA 2020, November 26-28 2020.
- M. Segovia, J. Rubio-Hernan, A. Cavalli, J. Garcia-Alfaro. Switched-based Resilient Control of Cyber-Physical Systems. IEEE Access journal, DOI:10.1109/ACCESS.2020.3039879, Nov. 2020.