Journée du groupe IE du GDR GPL
30 / 03 / 2017, Paris, France

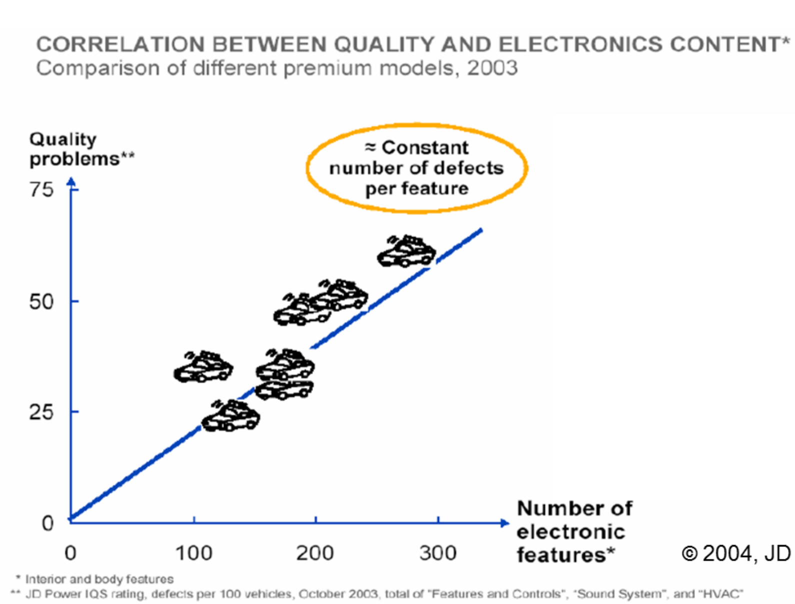
Une démarche guidée par les exigences pour la conception d'architectures de sécurité et la programmation sécurisée

Yves ROUDIER

Université Côte d'Azur - I3S – CNRS – UNS
Yves.Roudier@i3s.unice.fr

The SDLC and Security Engineering

- Multiple actors
- Separation of duties
- Secure SDLC
- What about requirements?



© 2004, JD Power

Outline

- **Introduction**
- **Security By Design**
  - **Security Objectives**
  - **Threat Analysis**
- **Security By Certification**
  - **Best Practices, Security Guidelines**
  - **Information Flow Control**
- **Conclusions and Perspectives**

Security Architectures
Security Properties
Access Control / Cryptographic Protocols

Secure Programming Guidelines
Code Scanners / Static Analysis / Code Audit
Security testing
Common Criteria / EAL

Outline

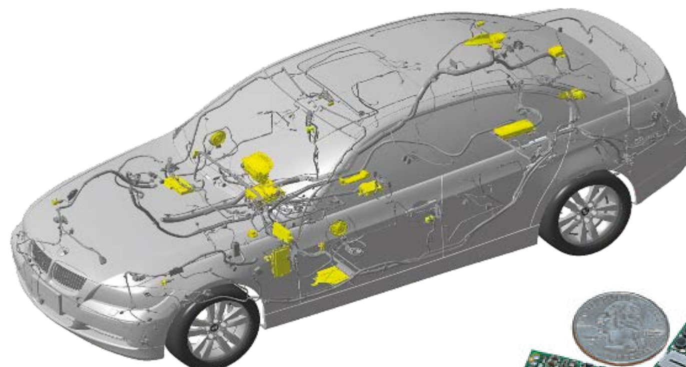
- Introduction
- Security By Design
  - Security Objectives
  - Threat Analysis
- Security By Certification
  - Best Practices, Security Guidelines
  - Information Flow Control
- Conclusions and Perspectives

Security by Design: Security Requirements?

- Requirements for security architecture and cryptographic protocol design
  - Which assets should be protected? Against which threats
  - Which mechanisms should be introduced? Are they sufficient?
- Several methods have been proposed over the years
  - KAOS: goals and anti-goals
  - UMLsec: software components and their interactions
  - Misuse Cases, SecureUML, …
- Focus mainly on IT systems
  - E.g., access control models

Embedded Systems ?

- **"Computer system with a dedicated function within a larger mechanical or electrical system" [Wikipedia]**
- **Designed on-purpose for specific control functions**
- **Integrated: Software + Hardware**
  - **Many technologies, increasingly distributed and communicating systems**



Embedded Systems: Examples of Threats

- Automotive Systems
  - Tire Pressure Monitoring System wireless link [Rouf 2010]
  - Keyfob authentication [Francillon 2011]
  - Vulnerabilities of Onboard Network [Koscher 2010]
  - HU remotely exploitable vulnerabilities [Checkoway 2011]
  - Locksmith tool(CAN/LIN injection) [MultiPick 2012]

© 2012, MultiPick

- Avionics Systems
  - Abusing the Automatic Dependent Surveillance Broadcast (ADS-B) protocol [Costin 2012]
  - Use of exploits in Flight Management System (FMS) to control ADS-B/ACARS [Teso 2013]

© 2013, Teso

- Internet of Things
  - 750000 spams sent in 2 weeks from compromised refrigerators [ProofPoint 2014]
  - Proof of concept of attack on IZON camera [Stanislav 2013]
  - Mirai DDoS attacks [2016]

Security Requirements Beyond IT Security

- **Key issues:**
  - **Collaboration between system designers and security experts**
  - **Specifying security requirements rather than security mechanisms?**
  - **Has to capture System perspective (HW / SW partitioning)**
  - **Incremental design ?**
  - **Must capture environmental constraints (e.g., real-time constraints)**
  - **Need to address functional AND safety requirements**

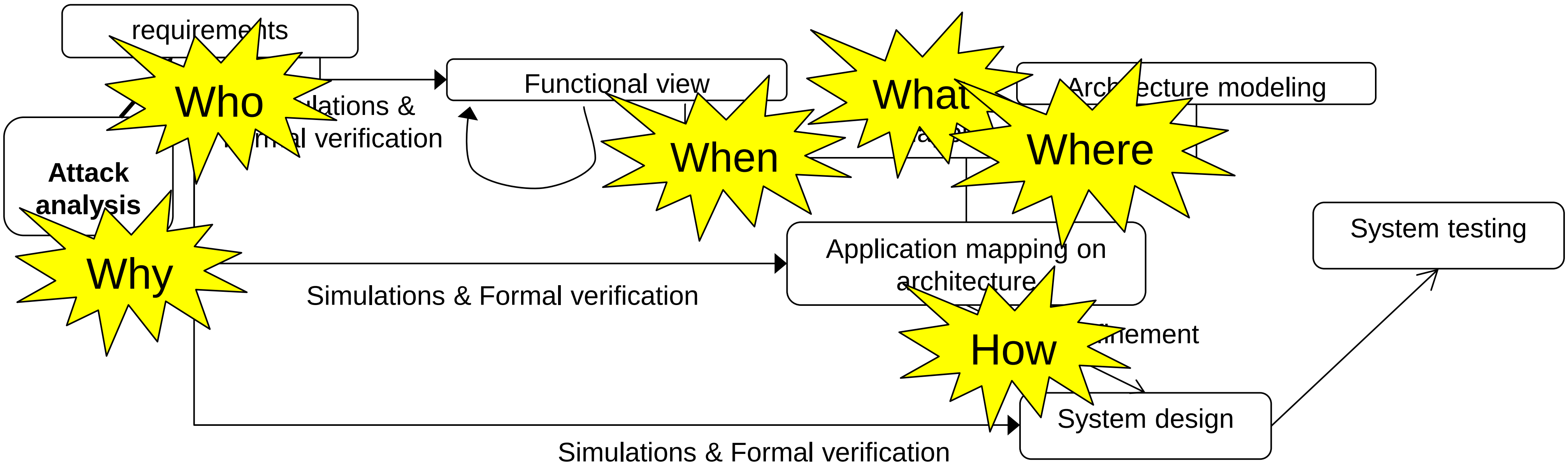
- **Model Driven Engineering as a Holistic Approach: SysML-Sec**
  - **Deployment of software components in architecture**
  - **Architecture = CPUs + memories + buses + software**
  - **Bring together system engineers & security experts**

Ongoing collaboration with Telecom ParisTech

UNIVERSITÉ CÔTE D'AZUR · i3S sophia antipolis · CNRS · Université Nice Sophia Antipolis

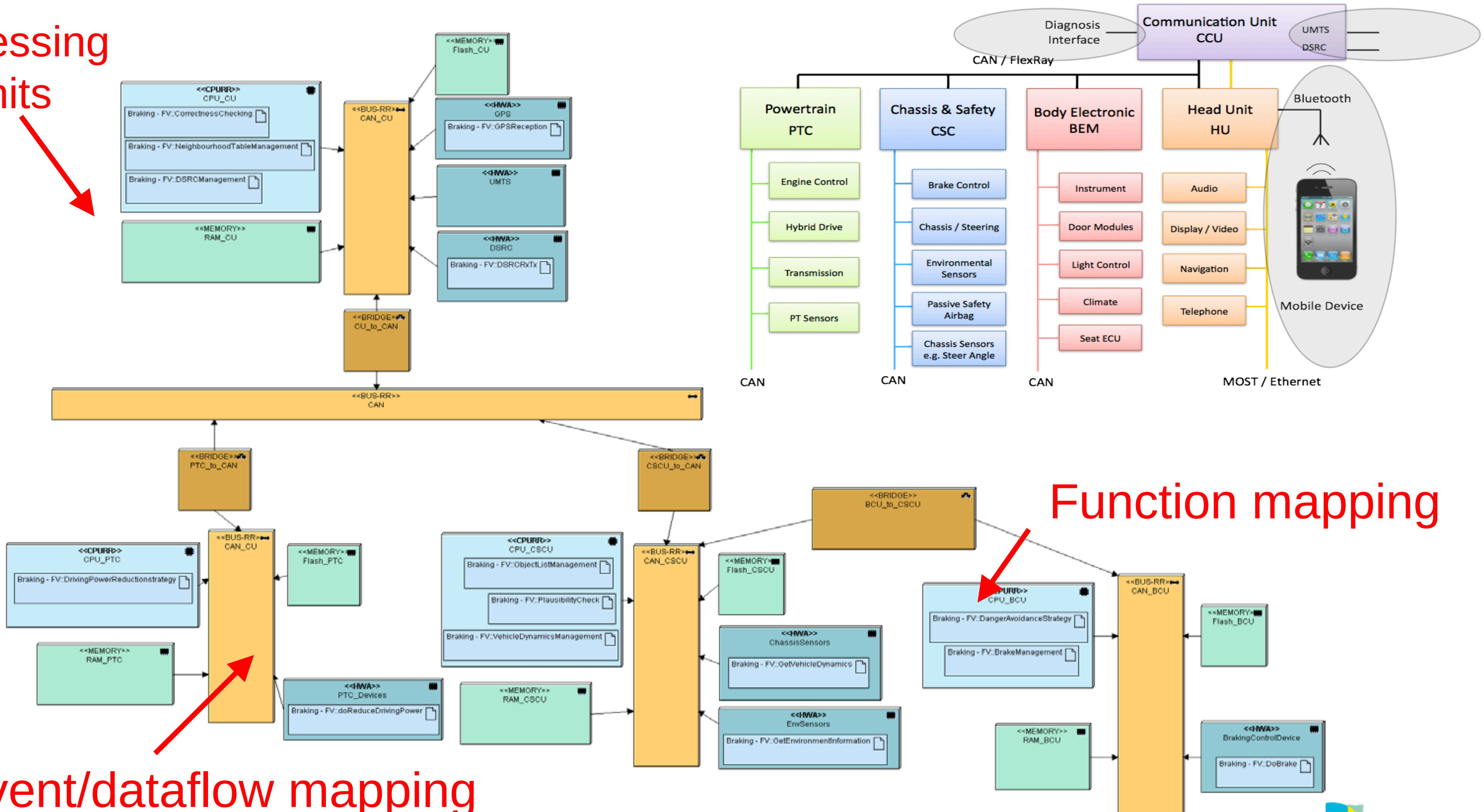
SysML-Sec : The Y-Chart Revisited

- **Security concerns:**
  - ***What***: **assets to be protected**
  - ***When***: **operation sequences in functions involving those assets**
  - ***Where***: **architecture mapping of functions involving those assets**
  - ***Why***: **attacks envisioned that motivate security countermeasures**
  - ***Who***: **stakeholders + attackers & capabilities (risk analysis)**
  - ***How***: **security objectives due to architecture (e.g., network topology, process isolation, etc.)**



Architectural Mapping Model (Deployment Diagram for Actual Topology)

Processing Units

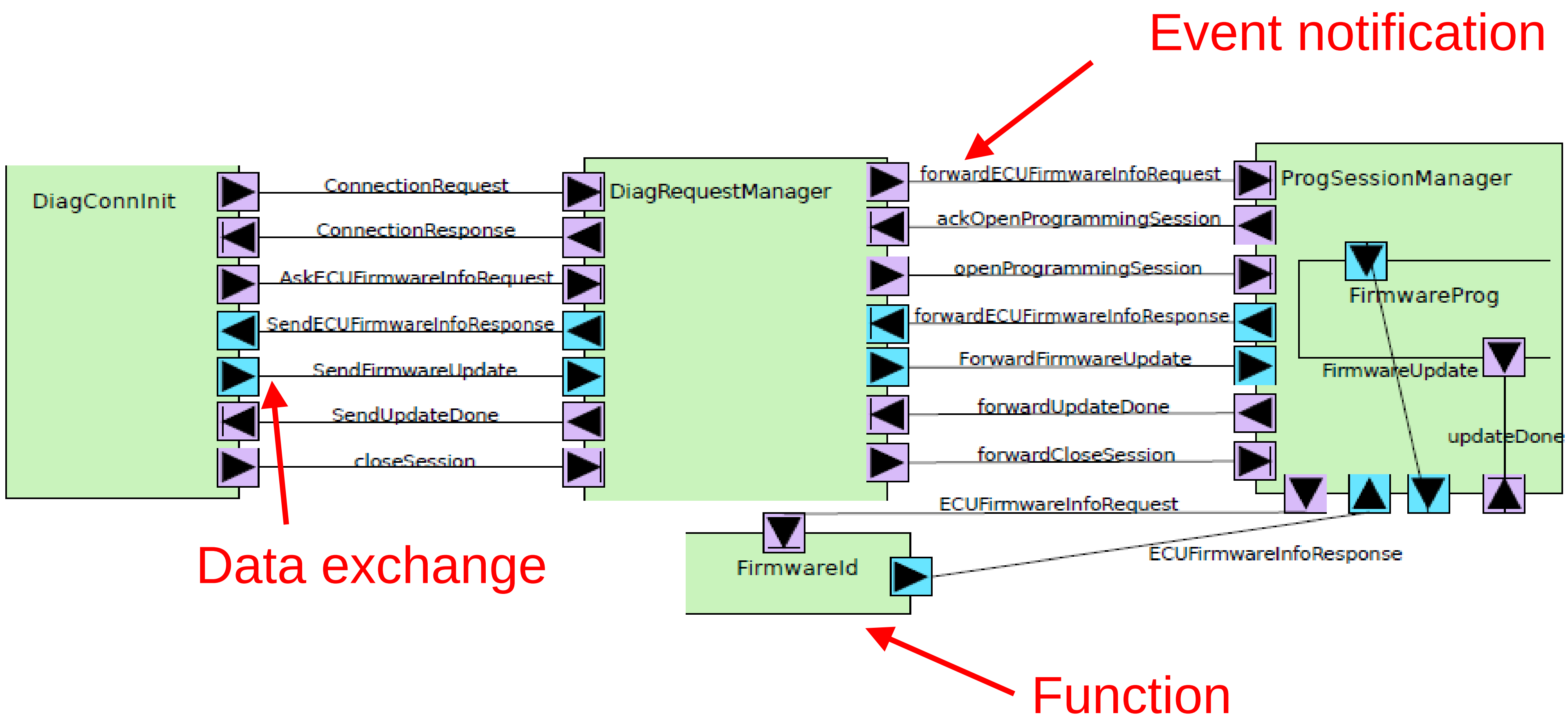


Function mapping

Event/dataflow mapping

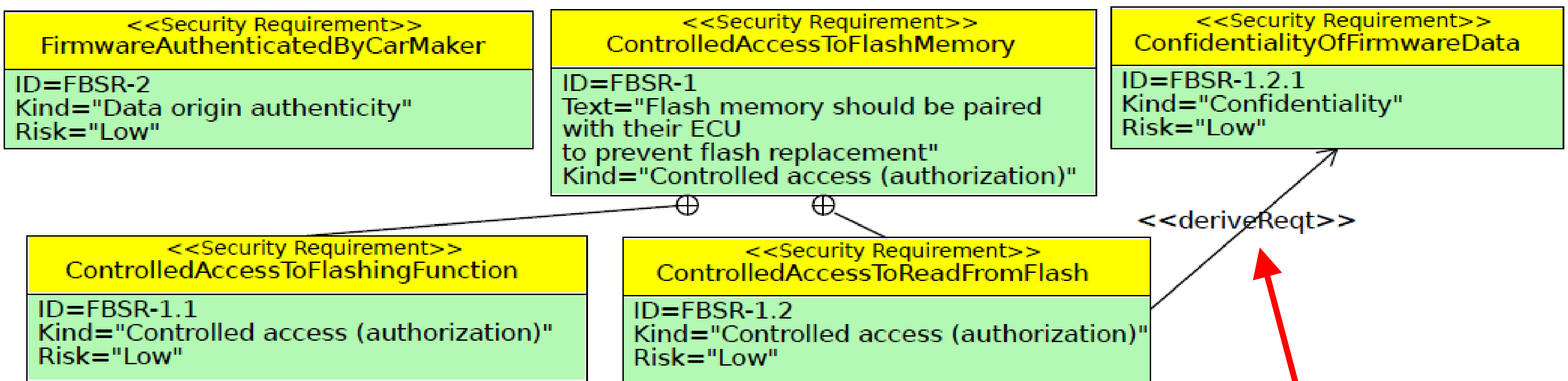
UNIVERSITÉ CÔTE D'AZUR

Functional View: Specifying Information Flows
Internal Block Diagram



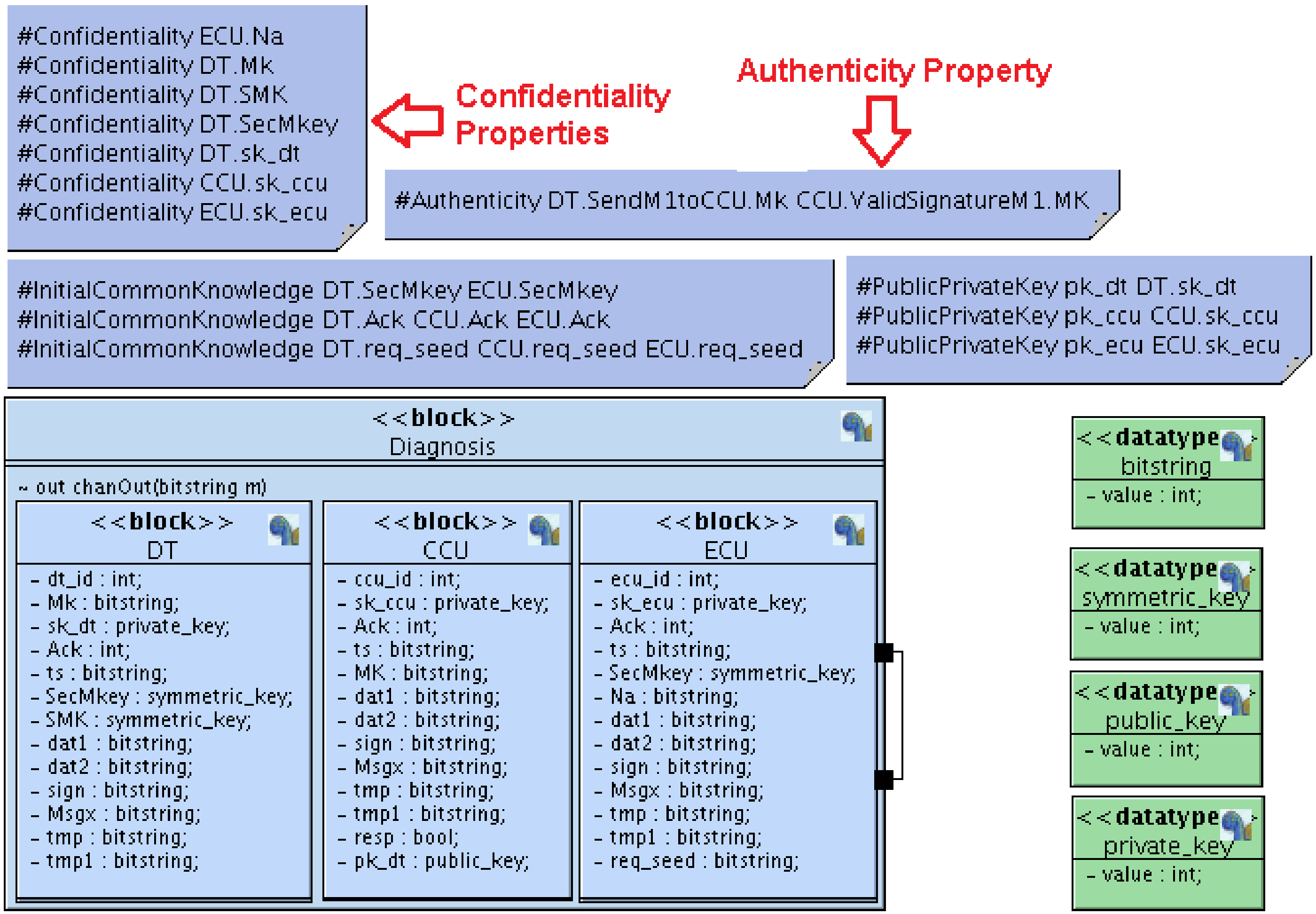
Security Properties and Types of Countermeasures Requirements Diagram

Security Properties: e.g., Confidentiality, Authenticity, Integrity, Freshness, Availability… applicable to some asset (HW/SW or more notably data or information flow)

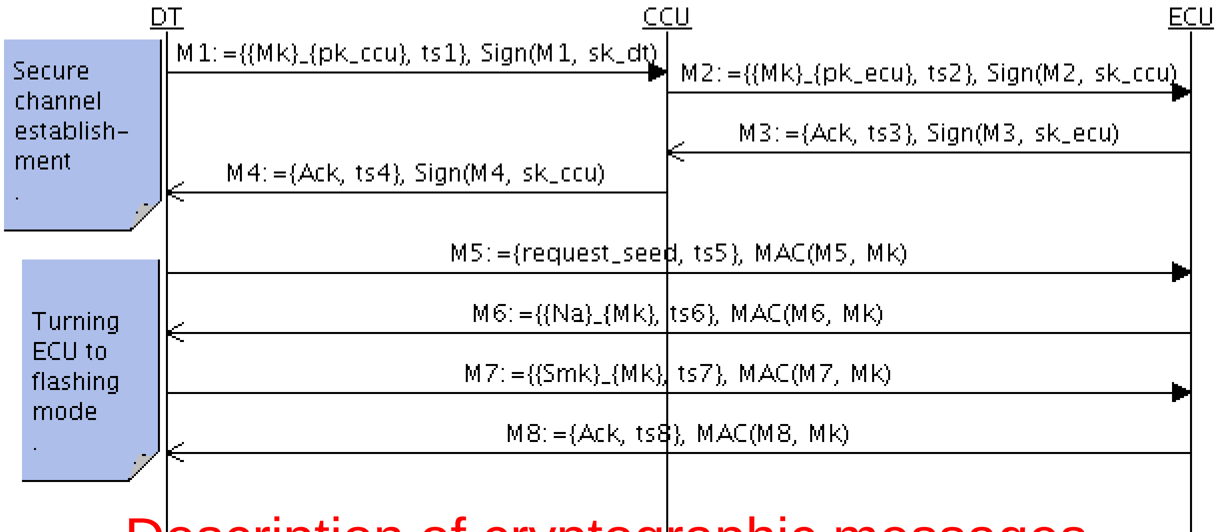


Trace refinements and dependencies

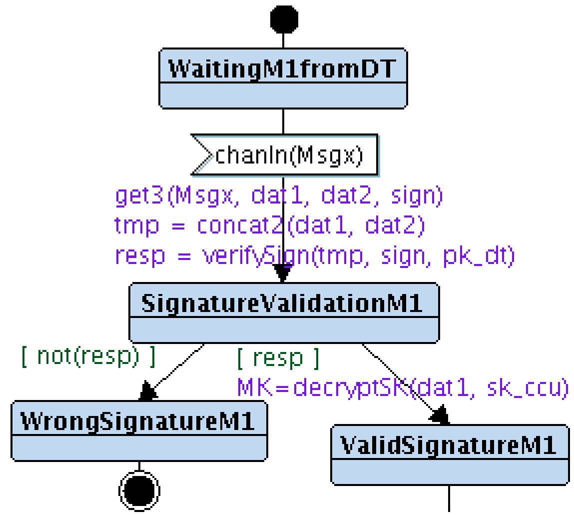
SysML Block Definition Diagram: cryptographic protocol environment support



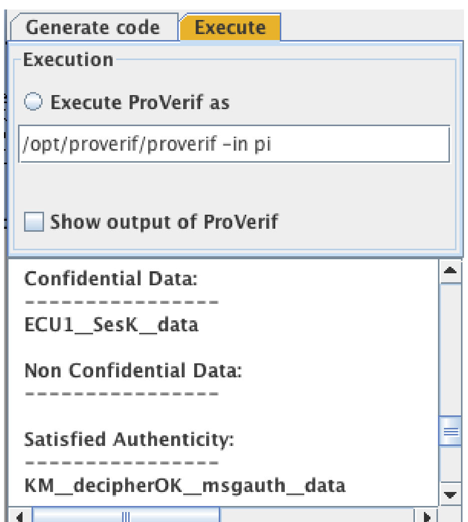
Cryptographic Protocol Messages: Content and Handling



Description of cryptographic messages

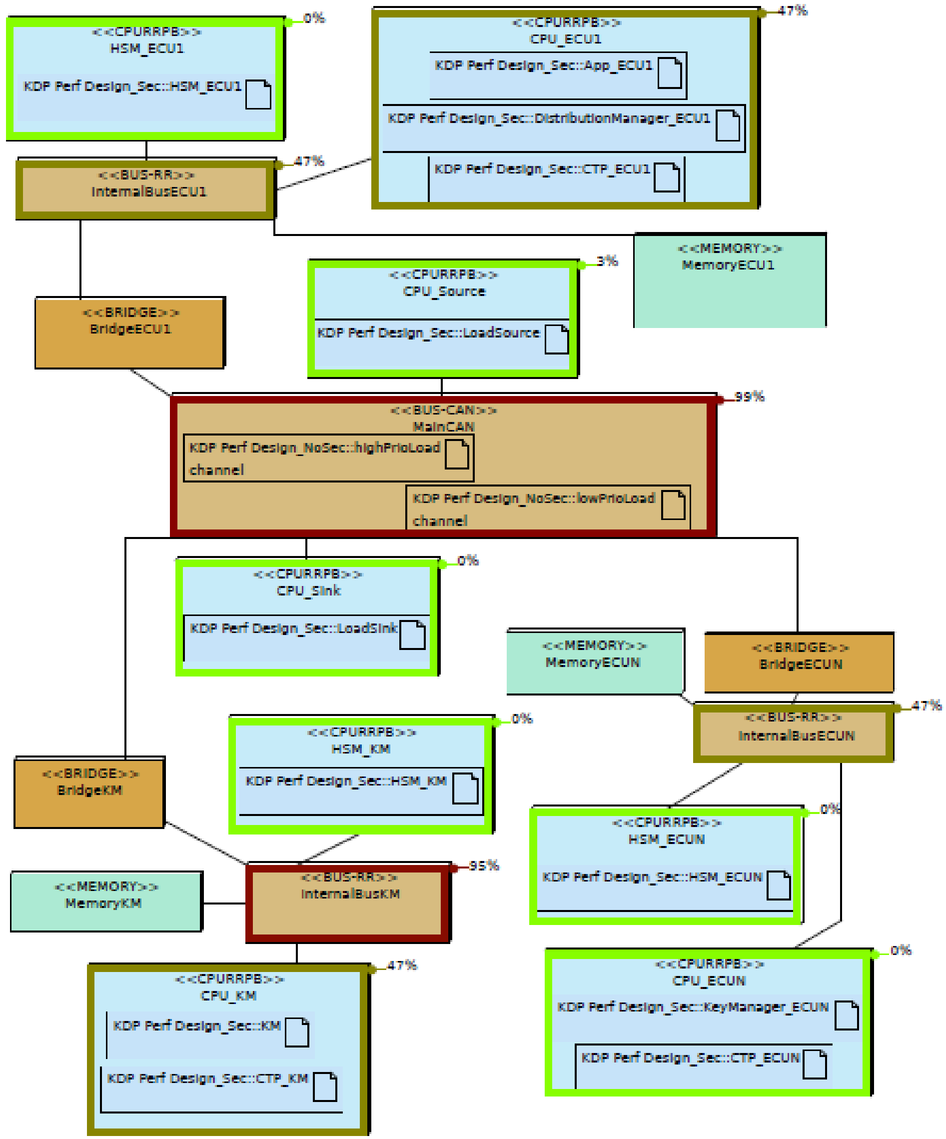


Handling of M1 at CCU



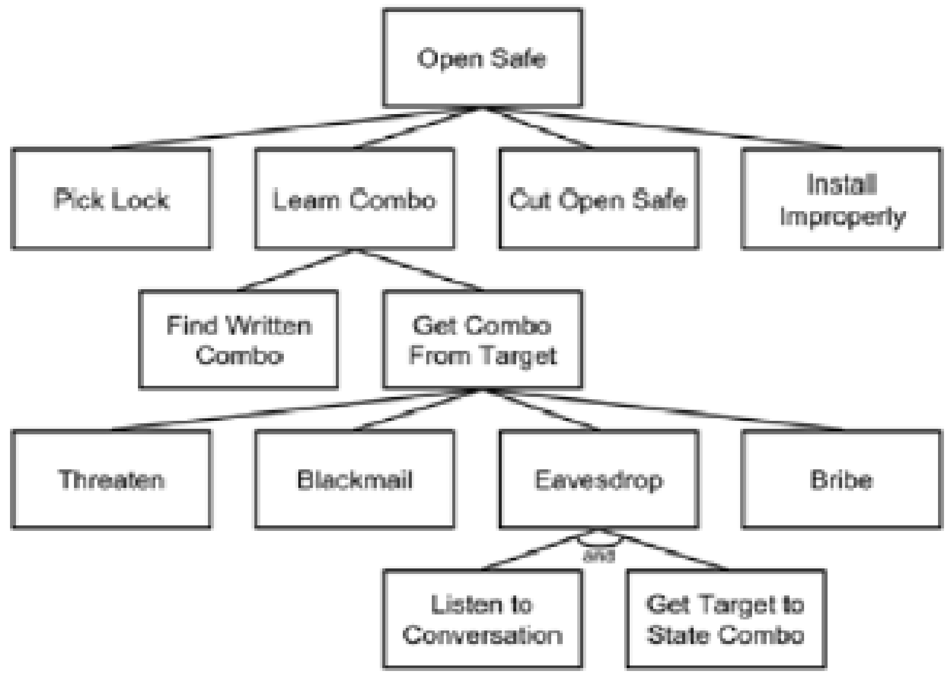
Formal verification in ProVerif (Dolev-Yao attacker)

Simulations: Analyzing the Impact of Security



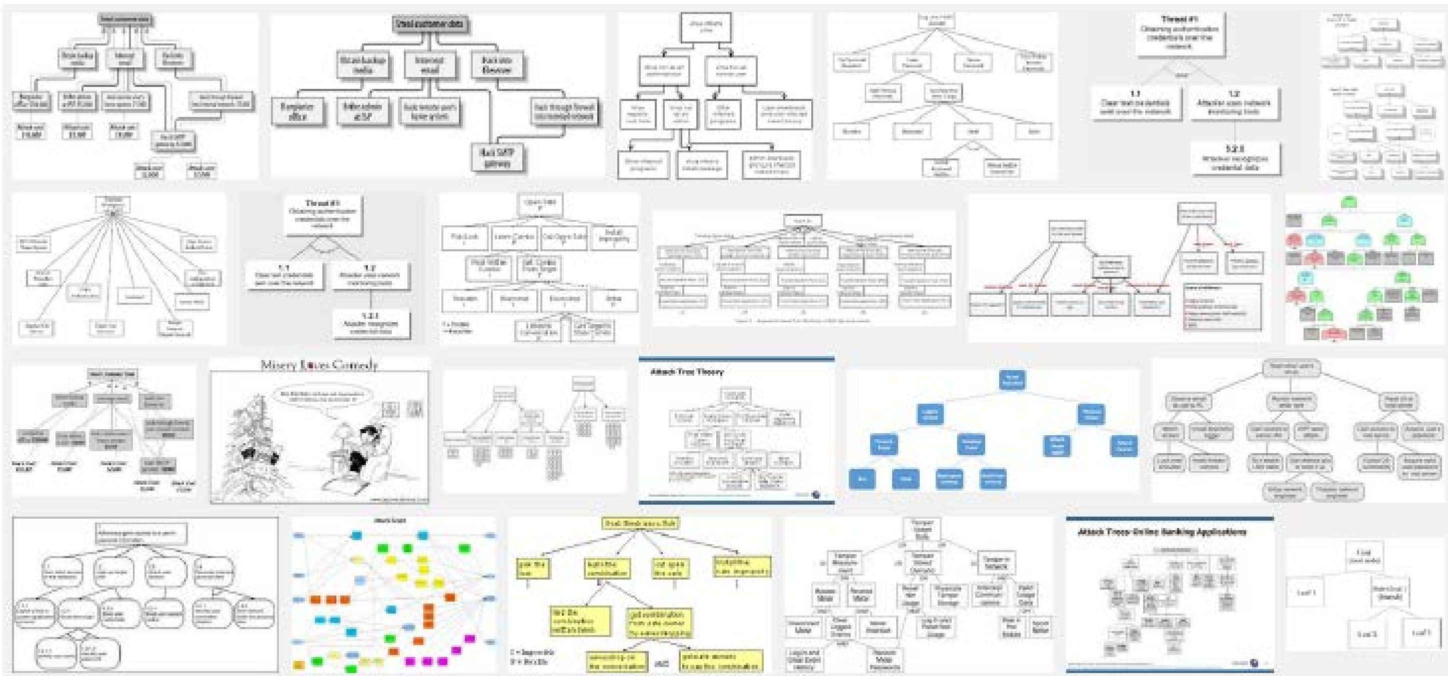
Threat Analysis: Attack Trees

- **Originate from fault trees, introduced by Bruce Schneier (1999)**
- **Depict how a system element can be attacked**
  - **Helps finding attack countermeasures**
- **Root attack, children, leaves**
- **OR and AND relations between children**



Attack Trees: not so simple!

- **Come in (too) many flavors …**

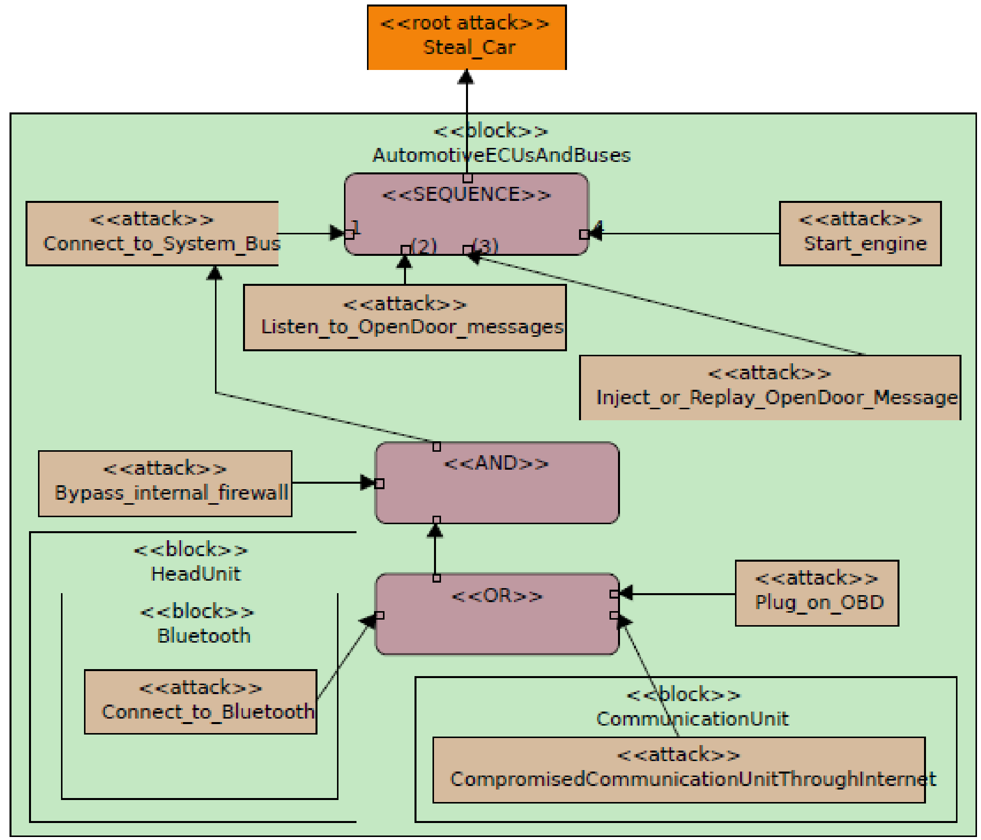


source: Google Images

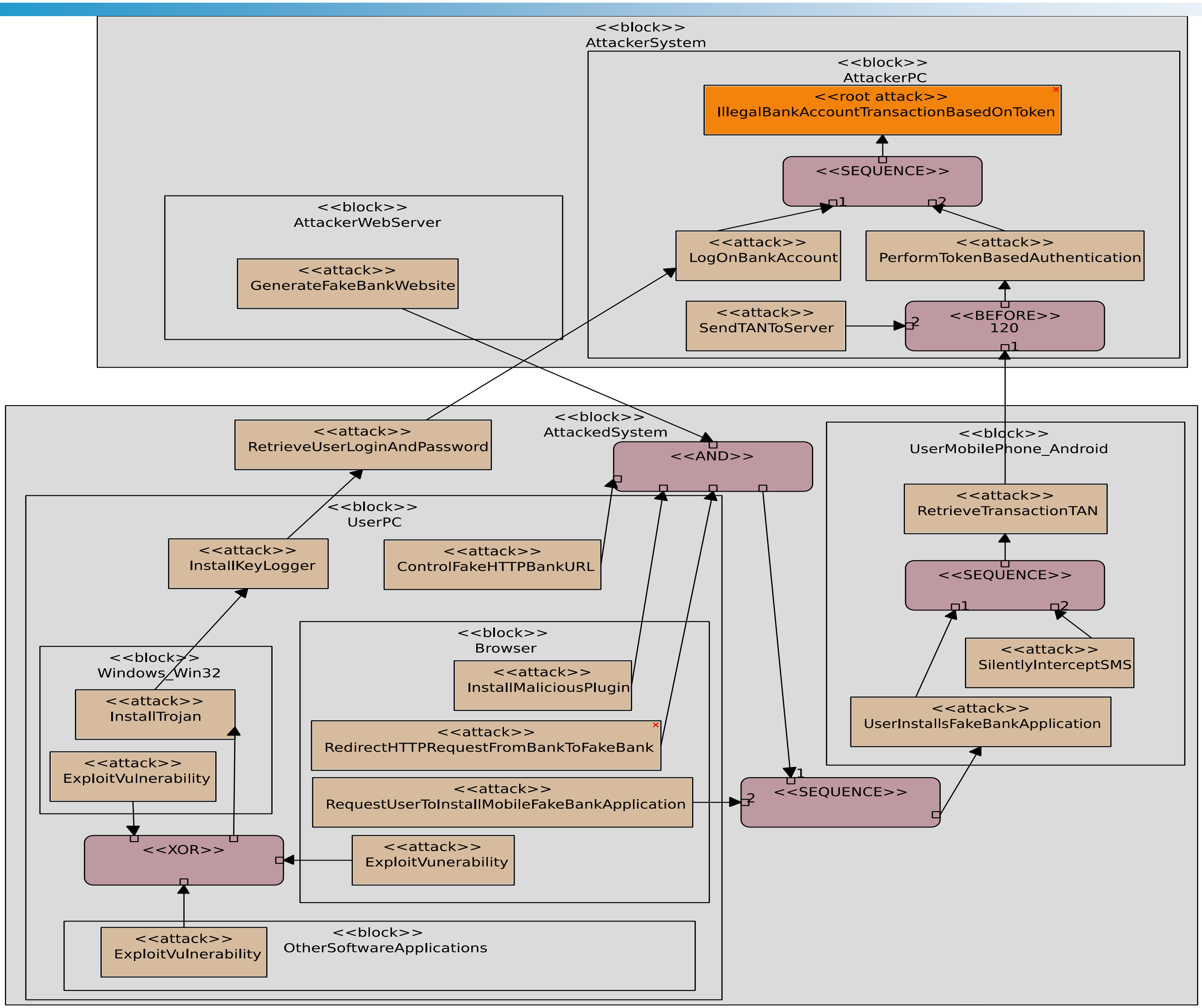
- **Complex scenario?**
- **Reusing attacks?**

From Attack Trees to Attack Graphs

- **Relations between attacks = constraints**
  - **Logical (AND, OR, XOR)**
  - **Ordering (SEQUENCE, BEFORE, AFTER)**
- **HW/SW mapping is very important**
  - **Documentation of attacks and matching countermeasures**
  - **Formal analysis of attack perimeter in architecture**
- **Reuse perspectives**
  - **E.g. better documentation for CVEs**



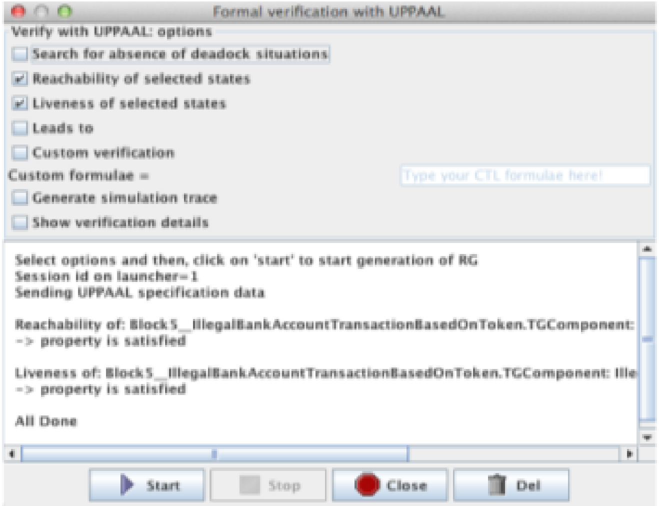
The Zeus/Zitmo malware attack



Attacker System

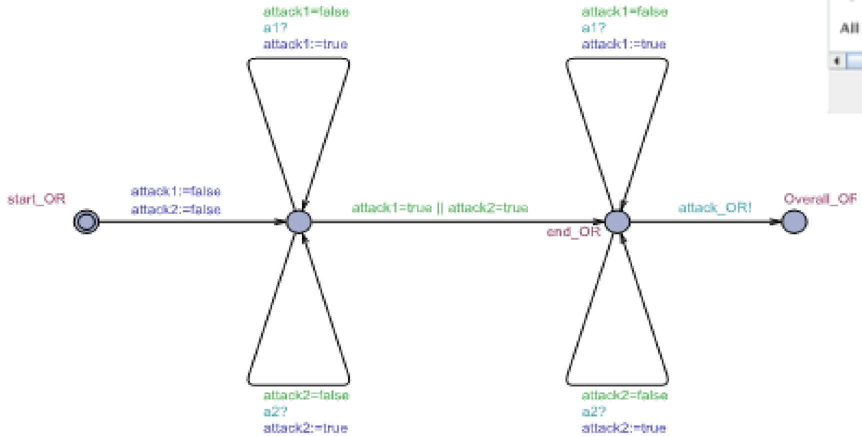
Target of Attack (Windows Host, Browsert, Mobile Phone)

Formal Attack Analysis

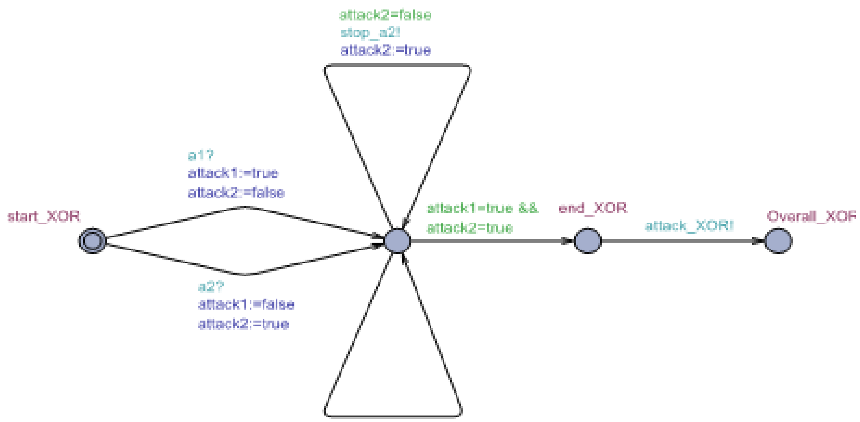


- **Attack graph semantics based on Timed Automata**
- **Objective: reachability and liveness of an attack, critical paths, etc.**

OR



XOR



UNIVERSITÉ CÔTE D'AZUR

Outline

- Introduction
- Security By Design
  - Security Objectives
  - Threat Analysis
- Security By Certification
  - Best Practices, Security Guidelines
  - Information Flow Control
- Conclusions and Perspectives

21

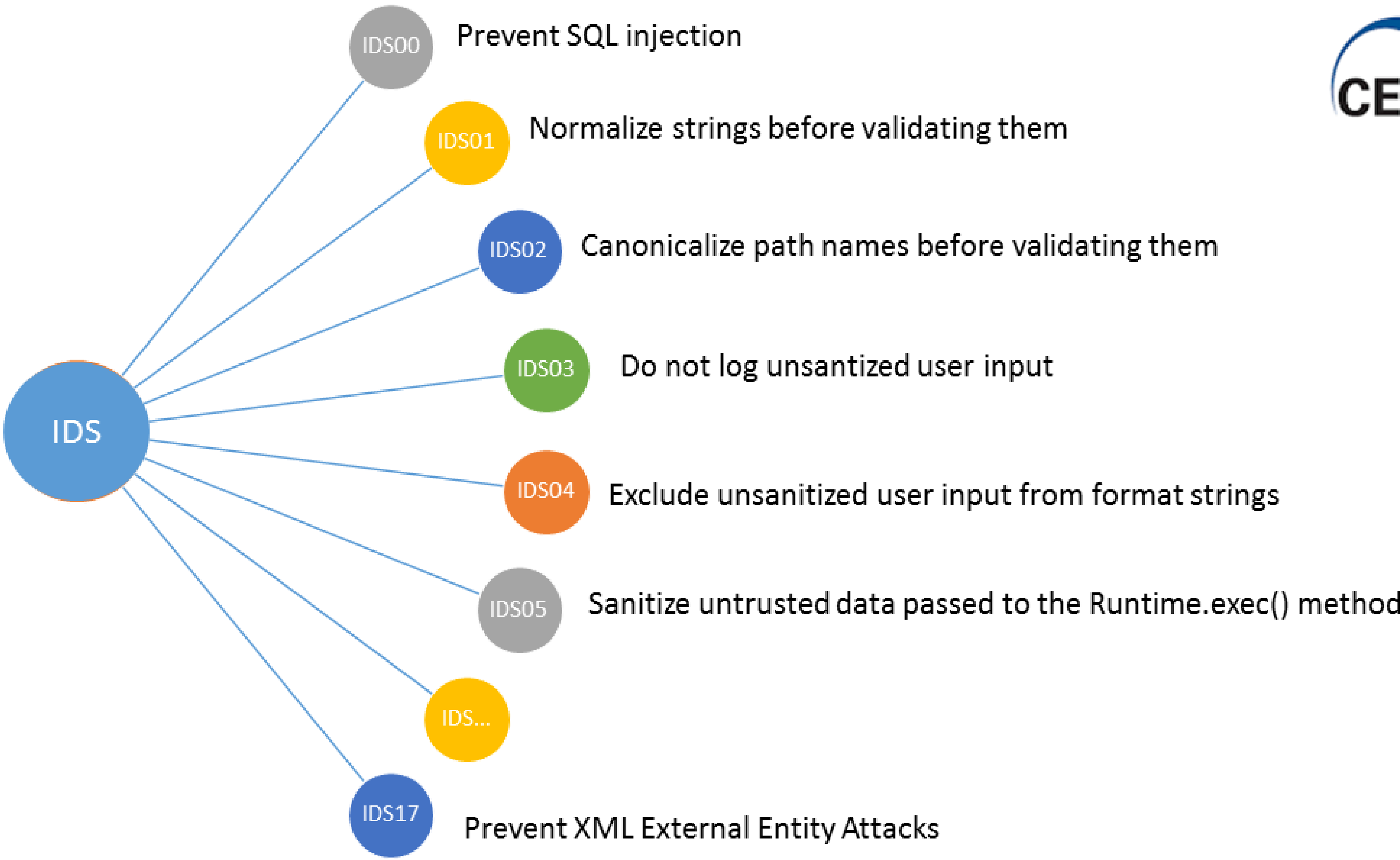
Best Practices / Secure Programming

Rules

- Input Validation and Data Sanitization
- Password Security
- Secure Communication
- Miscellaneous (MSC)

Source: https://www.securecoding.cert.org/confluence/display/java/SEI+CERT+Oracle+Coding+Standard+for+Java

Best Practices / Secure Programming



Security Guideline: Objective + Anti-Pattern

MSC62-J. Store passwords using a hash function

Créée par Matthew Wiethoff, dernière modification par David Svoboda le mai 13, 2016

Programs that store passwords as cleartext (unencrypted text data) risk exposure of those passwords in a variety of ways. Although programs generally receive passwords from users as cleartext, they should ensure that the passwords are not stored as cleartext.

An acceptable technique for limiting the exposure of passwords is the use of *hash functions*, which allow programs to indirectly compare an input password to the original password string without storing a cleartext or decryptable version of the password. This approach minimizes the exposure of the password without presenting any practical disadvantages.

Cryptographic Hash Functions

The value produced by a hash function is the *hash value* or *message digest*. Hash functions are computationally feasible functions whose inverses are computationally infeasible. In practice, a password can be encoded to a hash value, but decoding remains infeasible. The equality of passwords can be tested through the equality of their hash values.

A good practice is to always append a *salt* to the password being hashed. A salt is a unique (often sequential) or randomly generated piece of data that is stored along with the hash value. The use of a salt helps prevent brute-force attacks against the hash value, provided that the salt is long enough to generate sufficient entropy (shorter salt values cannot significantly slow down a brute-force attack). Each password should have its own salt associated with it. If a single salt were used for more than on password, two users would be able to see whether their passwords are the same.

The choice of hash function and salt length presents a trade-off between security and performance. Increasing the effort required for effective brute-force attacks by choosing a stronger hash function can also increase the time required to validate a password. Increasing the length of the salt makes brute-force attacks more difficult but requires additional storage space.

Java's `MessageDigest` class provides implementations of various cryptographic hash functions. Avoid defective functions such as the Message-Digest Algorithm (MD5). Hash functions such as Secure Hash Algorithm (SHA)-1 and SHA-2 are maintaine by the National Security Agency and are currently considered safe. In practice, many applications use SHA-256 because this hash function has reasonable performance while still being considered secure.

Noncompliant Code Example

This noncompliant code example encrypts and decrypts the password stored in `password.bin` using a symmetric key algorithm:

```
public final class Password {
  private void setPassword(byte[] pass) throws Exception {
    // Arbitrary encryption scheme
    bytes[] encrypted = encrypt(pass);
    clearArray(pass);
    // Encrypted password to password.bin
    saveBytes(encrypted,"password.bin");
    clearArray(encrypted);
  }

  boolean checkPassword(byte[] pass) throws Exception {
```

Security Guideline: Compliant Example

Compliant Solution

This compliant solution addresses the problems from the previous noncompliant code example by using a `byte` array to store the password:

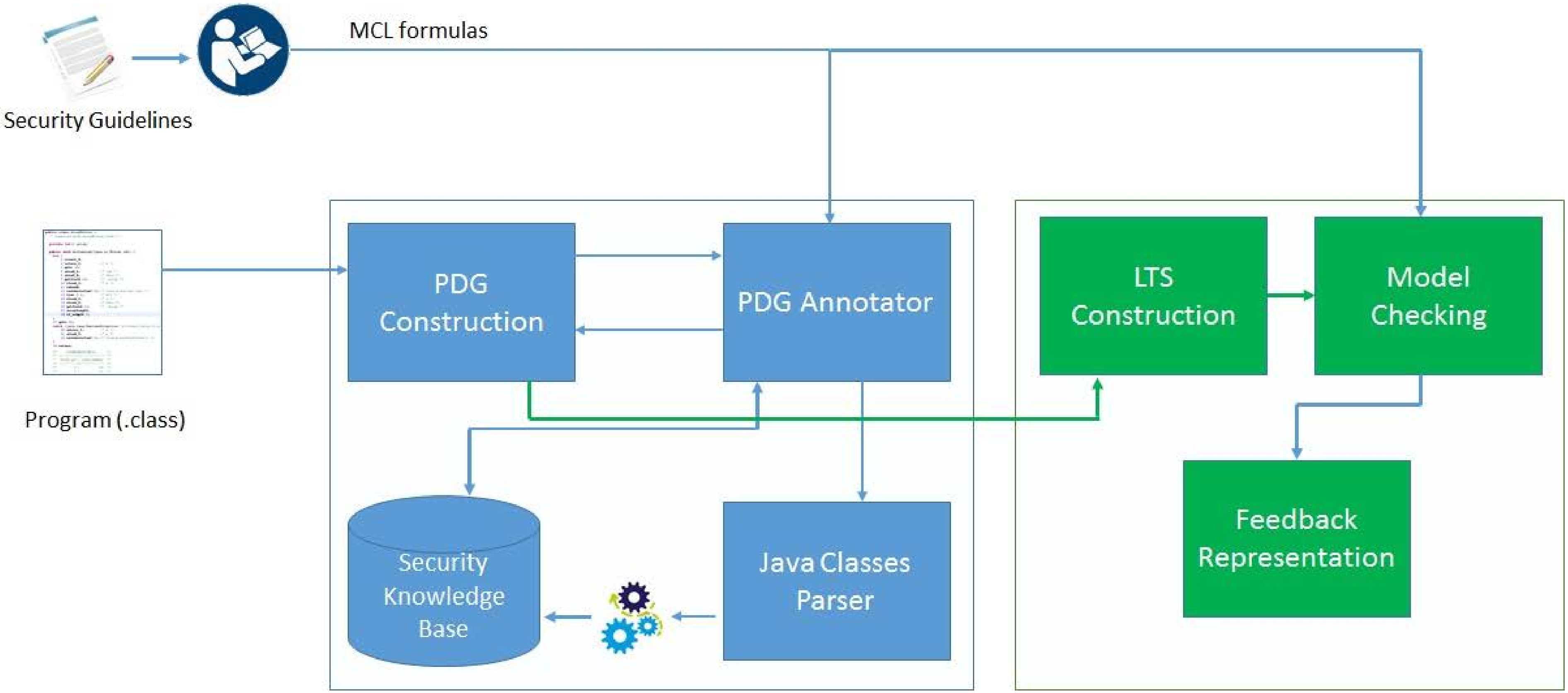
```
import java.security.GeneralSecurityException;
import java.security.SecureRandom;
import java.security.spec.KeySpec;
import javax.crypto.SecretKeyFactory;
import javax.crypto.spec.PBEKeySpec;

final class Password {
  private SecureRandom random = new SecureRandom();

  /* Set password to new value, zeroing out password */
  void setPassword(char[] pass)
      throws IOException, GeneralSecurityException  {
    byte[] salt = new byte[12];
    random.nextBytes(salt);
    saveBytes(salt, "salt.bin");
    byte[] hashVal = hashPassword( pass, salt);
    saveBytes(hashVal,"password.bin");
    Arrays.fill(hashVal, (byte) 0);
  }
  /* Indicates if given password is correct */
  boolean checkPassword(char[] pass)
      throws IOException, GeneralSecurityException  {
    byte[] salt = loadBytes("salt.bin");
    byte[] hashVal1 = hashPassword( pass, salt);
    // Load the hash value stored in password.bin
    byte[] hashVal2 = loadBytes("password.bin");
    boolean arraysEqual = timingEquals( hashVal1, hashVal2);
    Arrays.fill(hashVal1, (byte) 0);
    Arrays.fill(hashVal2, (byte) 0);
    return arraysEqual;
  }
```

Problems with Security Guidelines

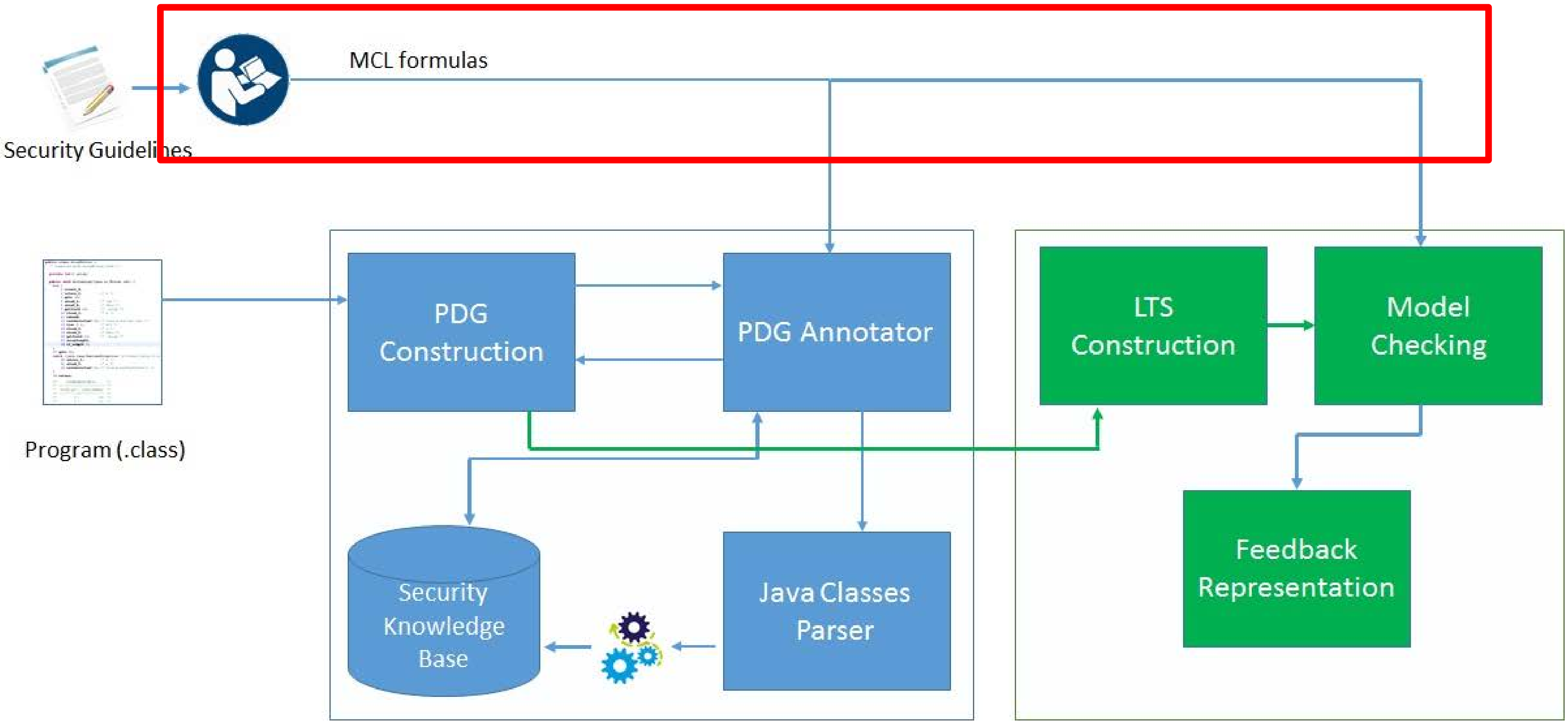
- Abstract and imprecise descriptions
- Informal specification – how to verify compliance?
- Subject to misinterpretation for developer
- Often programming language specific
- Multiple overlapping catalogs (CERT, OWASP, …)

- These are supported by code scanners which implement ad-hoc compliance checks through static analysis
  - Problem: no description of the verifications performed

Certifying Secure Programming: Approach



Approach: Specifying guidelines (requirements)



Formal Specification of Security Guidelines

- **Behavior represented as a Labelled Transition System (LTS)**
  - **Transitions = instructions**

- **Formalization of guidelines based on MCL formulas (modal mu-calculus)**

- **Example: “Store passwords using a hash function”**

  [true*.{setPassword ?msg:String}.(not ({hash !msg}))*. {log !msg}] false

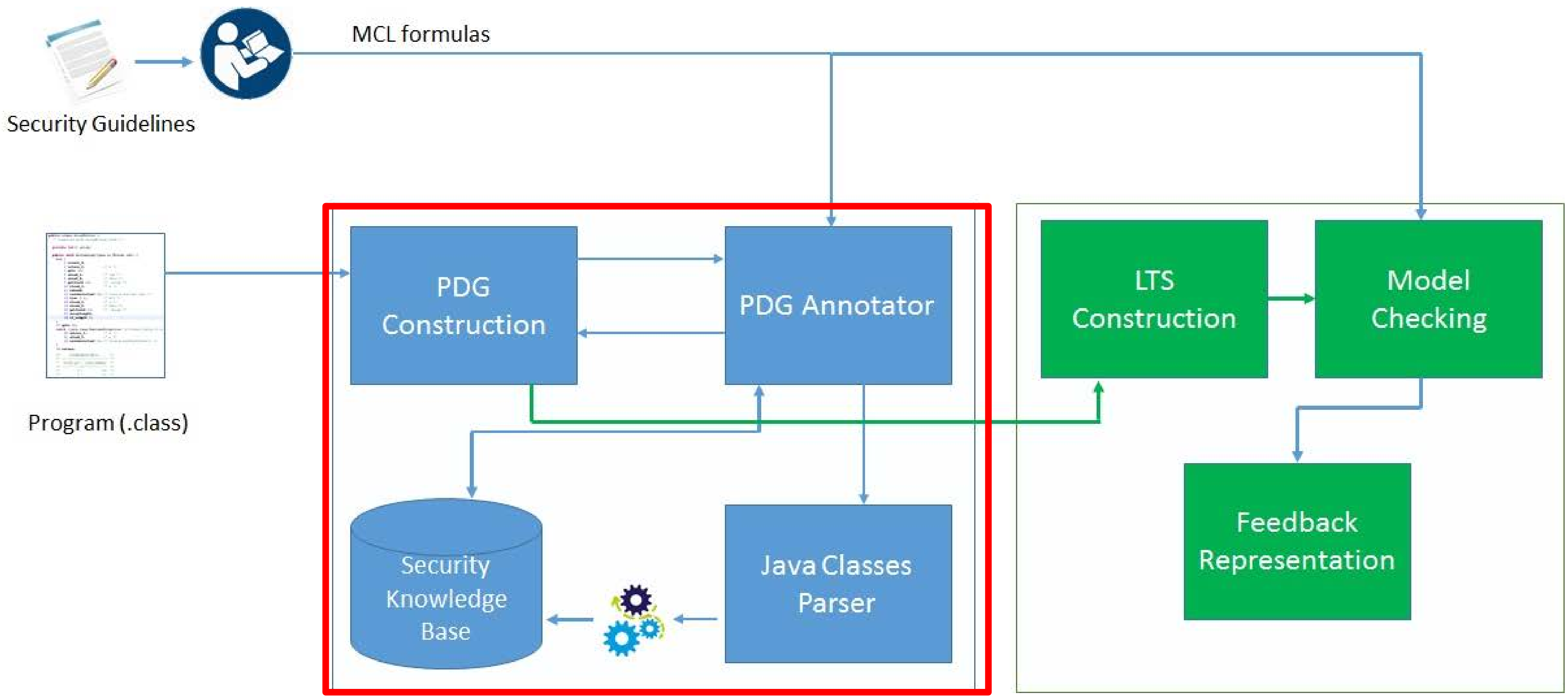
- **Analyzing further data dependencies:**

[true*.{setPassword ?msg:String }.true*. (((not {hash !msg})*.{log !msg})

|

{depend ?msg1:String !msg}.true*. ((not ({hash !msg1}))*.{log !msg1}))] false

Approach: Static Analysis of Program

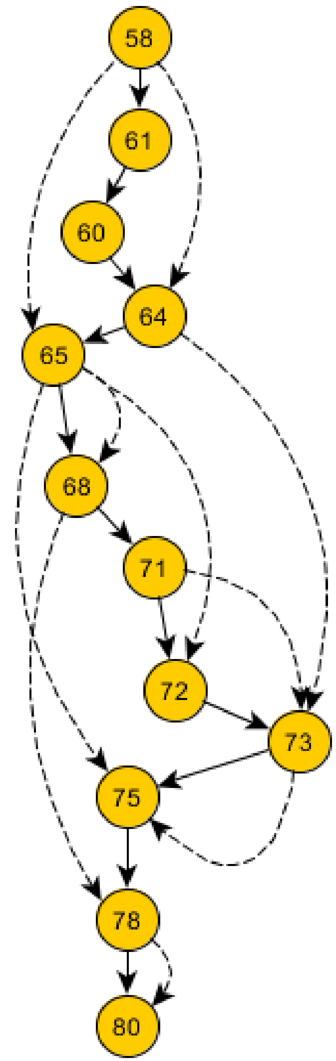


Program Dependence Graph (PDG)

- Control Dependences
- Explicit + Implicit Data Dependences
- Properties:
  - Path-sensitive
  - Context-Sensitive
  - Object-Sensitive

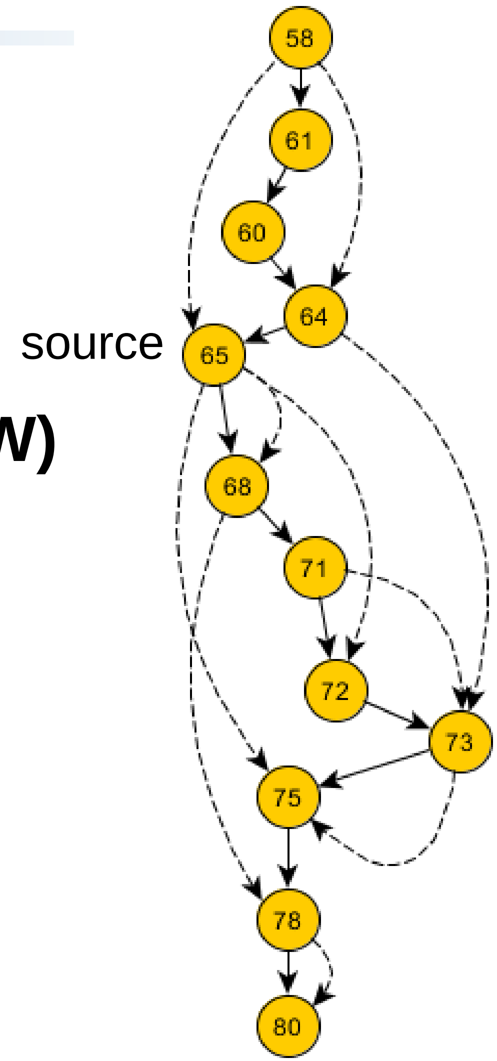
Control dependence

Data dependence



JOANA IFC tool

- **Intended for Information Flow Analysis**
- **Annotations: SINK / SOURCE**
- **Non-Interference: Security Levels (HIGH / LOW)**



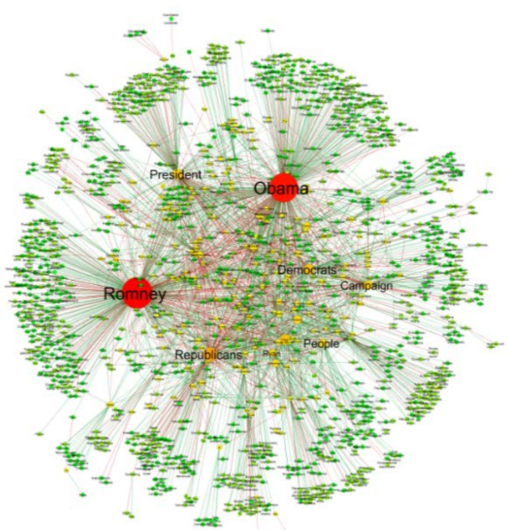
Collecting further information flow dependencies

Java
Class /
Javadoc

parser

- Class Details
- Methods
- Method Parameters

Program
Dependence
Graph



Computing
dependencies

Security
Knowledge
Base

Example

```java
// input
user.setUsername(reader.readLine());
user.setPassword(reader.readLine());

// copy
String xx = user.getPassword();

//hash
MessageDigest hash = MessageDigest.getInstance("MD5");
byte[] bytes = user.getPassword().getBytes("UTF-8");
byte[] hash_password = hash.digest(bytes);

user.setPassword(hash_password.toString());

// log
logMessage = "user name = " + user.getUsername() +
        ", password = " + user.getPassword() + xx;
logger.log(Level.INFO, logMessage);
```

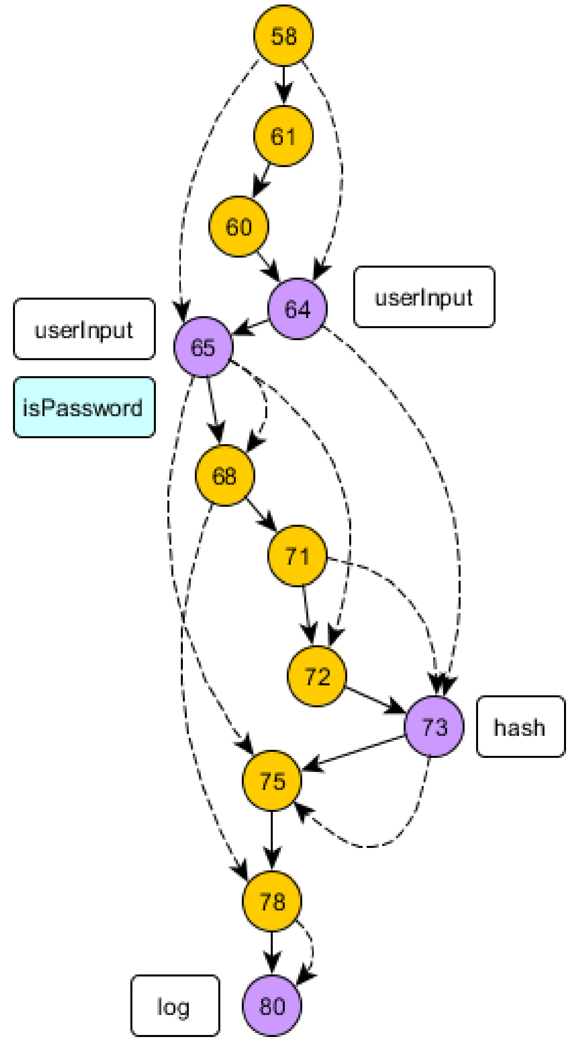
input data

hash

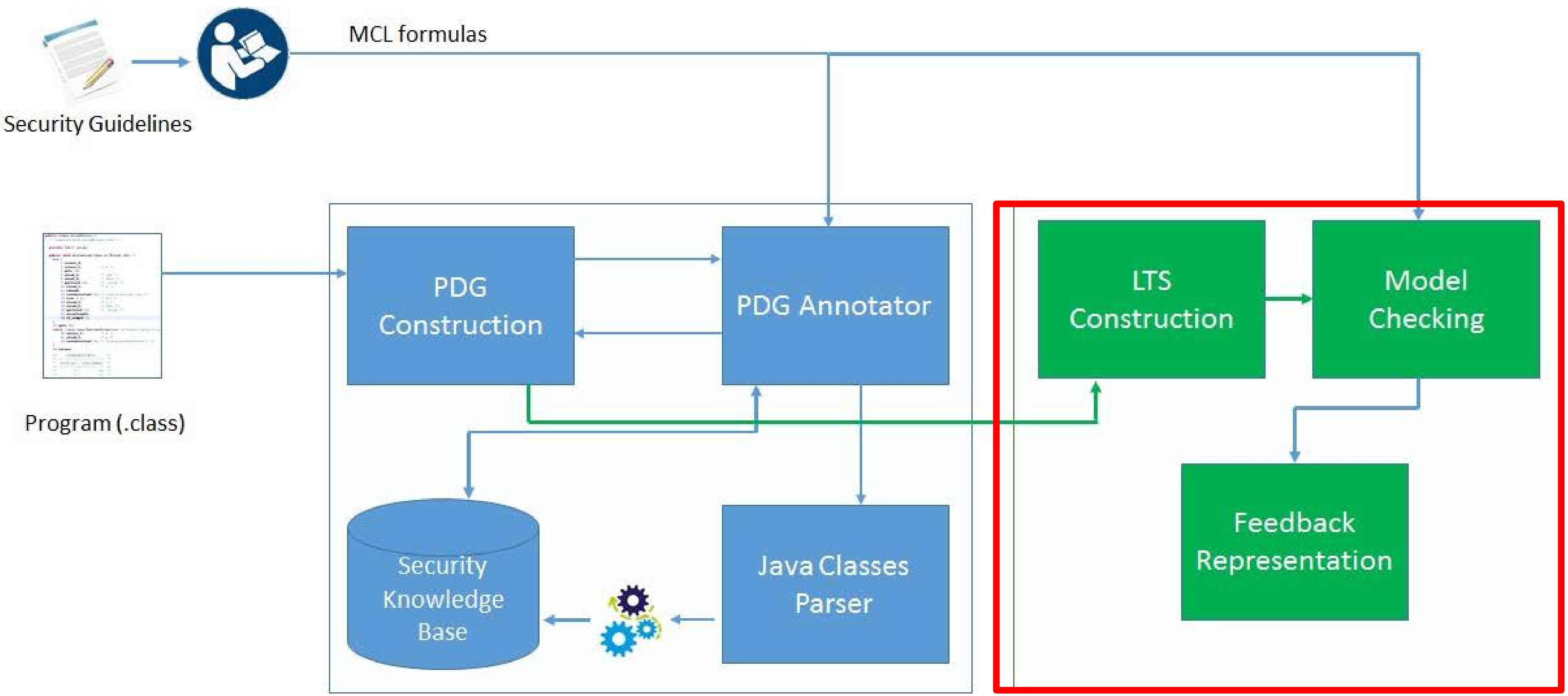
log

Augmented PDG: Automatic & Manual Annotations

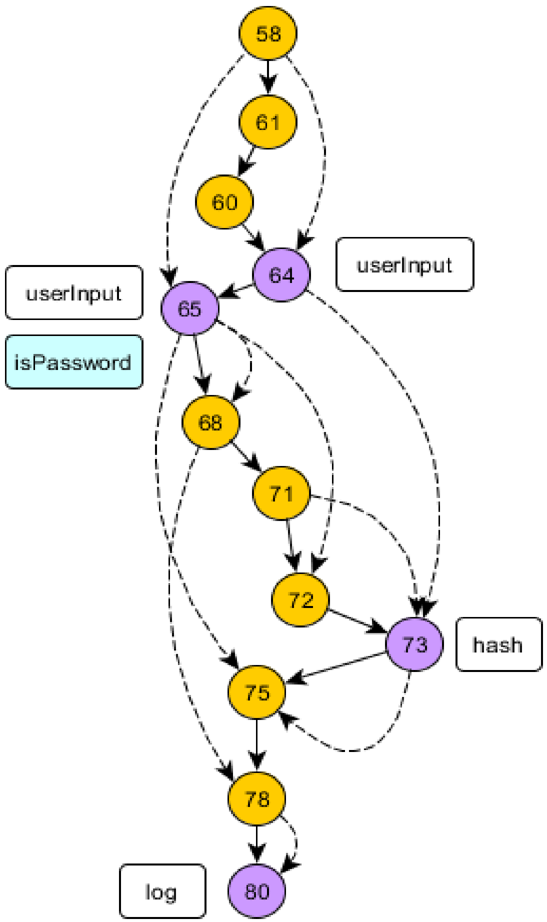
- **Manual:**
  - ➢ **Password Annotation: Requires advanced Semantic Analysis**
- **Automated (using Security Knowledge Base):**
  - ➢ **BufferedReader,read()** ⟺ **userInput**
  - ➢ **Logger,log(Level level, String msg)** ⟺ **log**
  - ➢ **MessageDigest,digest(byte[] input)** ⟺ **hash**

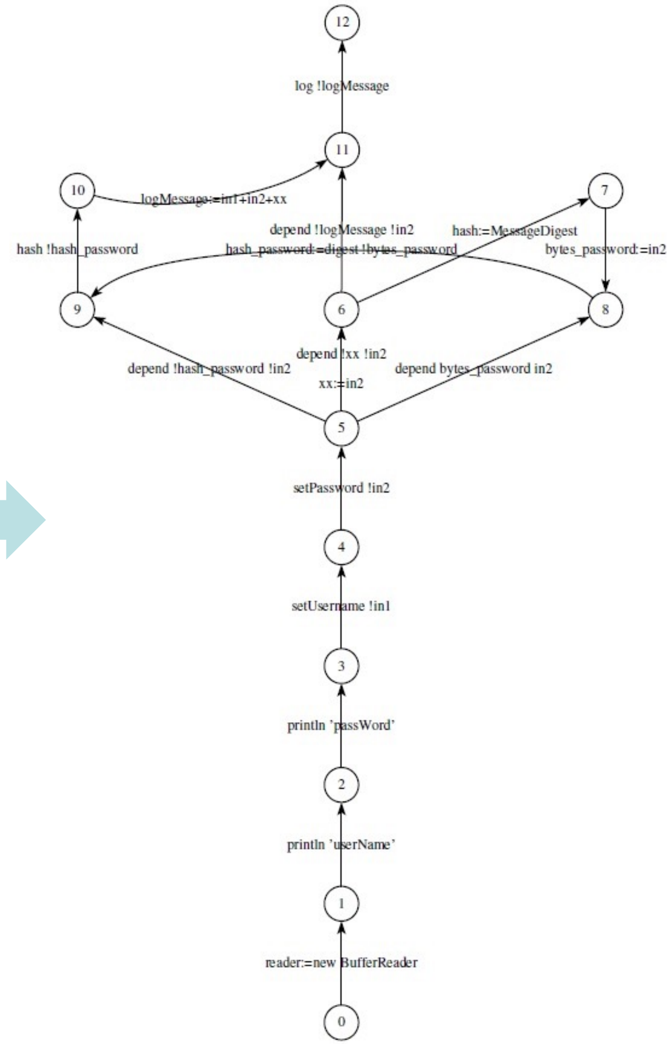


Approach: Formal Verification



Model Checking: from PDG to LTS





Model Checking: Feedback to the Developer

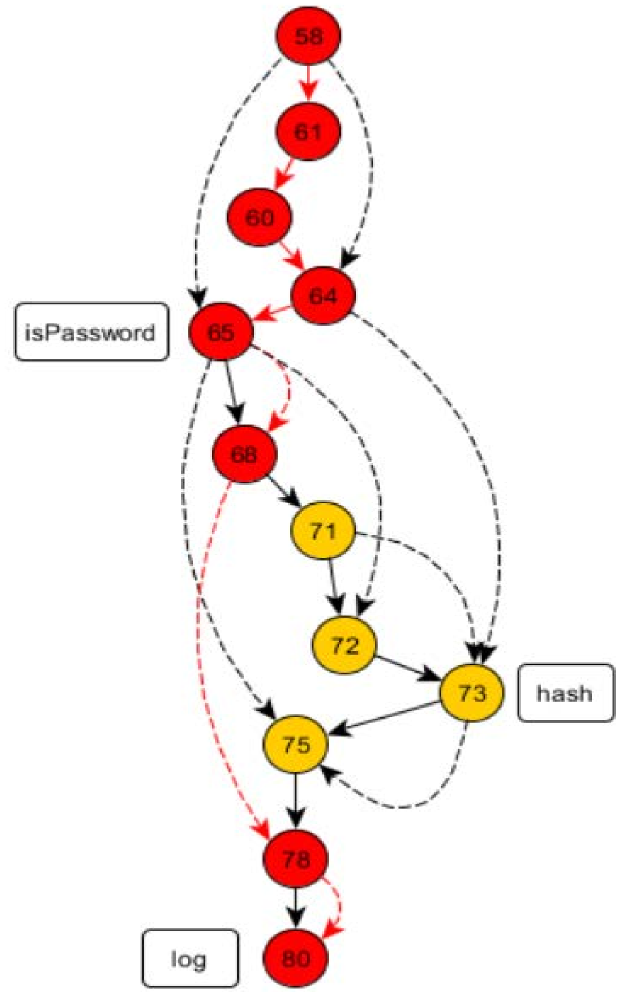
MCL

Sec_LTS

Model Checker (CADP)

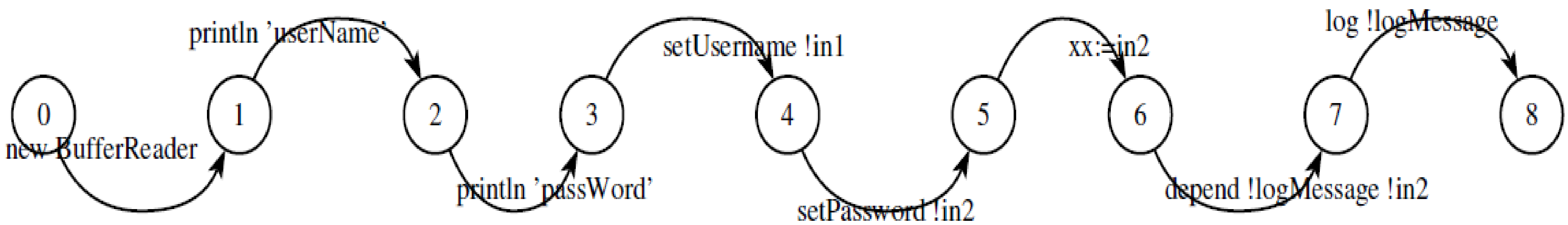
Analysis results

Violation Traces



Example: Violation Trace

- **Copy of the password into an auxiliary variable (named xx) stored into log without sanitization through hashing:**



Conclusions and Perspectives

- **Security by Design**
  - ➢ **Further developments for SysML-Sec (Ttool): Security + Safety, Connection to Vulnerability Databases, Risk Analysis**
- **Security by Certification**
  - ➢ **Ongoing work on security guidelines (collaboration with Telecom ParisTech and SAP) and prototype nearing completion**
  - ➢ **Information flow centric approach for cryptographic protocols**
- **Mixed Approach**
  - ➢ **PEPS project (INS3PECT CNRS) - IoT as the application domain**
    - ☞ **Privacy requirements**
    - ☞ **Off-the shelf components**
    - ☞ **Developer feedback**
    - ☞ **Collaboration UCA (I3S & LEAT) – IMAG (LIG) – UBS (Lab-STICC)**

THANK YOU … QUESTIONS WELCOME!

Some References: Security by Design

- L. Apvrille, L.W. Li, and Y. Roudier. *"Model-Driven Engineering for Designing Safe and Secure Embedded Systems."* Proceedings of ACVI 2016, IEEE Workshop on Architecture Centric Virtual Integration, 5-8 April 2016, Venice, Italy
- L. Apvrille, Y. Roudier and T. Tanzi, *"Autonomous Drones for Disasters Management: Safety and Security Verifications"*, Proceedings of the URSI AT-RASC Conference, May 2015
- L. Apvrille and Y. Roudier, *"SysML-Sec Attack Graphs: Compact Representations for Complex Attacks"*, The Second International Workshop on Graphical Models for Security (GramSec'2015), colocated with CSF 2015, Verona, Italy - July 13, 2015
- L. Apvrille and Y. Roudier, *"SysML-Sec: A Model Driven Approach for Designing Safe and Secure Systems"*, Special session on Security and Privacy in Model Based Engineering, 3rd International Conference on Model-Driven Engineering and Software Development (MODELSWARD'15), Angers, France, February 2015.

Some References: Security by Design

- L. Apvrille and Y. Roudier. *"Towards the Model-Driven Engineering of Secure yet Safe Embedded Systems. GRAMSEC 2014"*, 1st International Workshop on Graphical Models for Security, Co-located with ETAPS 2014, April 12, 2014, Grenoble, France
- L. Apvrille and Y. Roudier. *"SysML-sec: A SysML Environment for the Design and Development of Secure Embedded Systems."* Proceedings of APCOSEC 2013, Asia-Pacific Council on Systems Engineering, September 8-11, 2013, Yokohama, Japan
- Y. Roudier, M.S. Idrees and L. Apvrille. *"Towards the Model-Driven Engineering of Security Requirements for Embedded Systems."* MODRE 2013, International Workshop on Model-Driven Requirements Engineering, collocated with the 21st IEEE International Conference on Requirements Engineering (RE'13), 15 July 2013, Rio de Janeiro, Brazil

Some References: Security by Certification

- Zeineb Zhioua, Yves Roudier, Rabéa Ameur Boulifa, Takoua Kechiche, Stuart Short. *Tracking Dependent Information Flows.* ICISSP'17 (3rd International Conference on Information Systems Security and Privacy), Porto, Portugal, 19-21 February, 2017, held in conjunction with MODELSWARD 2017 and SENSORNETS
- Z. Zhioua, S. Short and Y. Roudier. *"Towards the Verification and Validation of Software Security Properties Using Static Code Analysis."* International Journal of Computer Science: Theory and Application, vol. 2, no. 2, pages 23-34, December 2014, ISSN: 2336-0984.
- Z. Zhioua, Y. Roudier, S. Short, and R. Boulifa Ameur. *"Security Guidelines: Requirements Engineering for Verifying Code Quality."* Proceedings of ESPRE 2016, 3rd International Workshop on Evolving Security and Privacy Requirements Engineering, September 12th, 2016, Beijing, China, co-located with the 24th IEEE International Requirements Engineering (RE'16) Conference
- Z. Zhioua, S. Short and Y. Roudier. *"Static Code Analysis for Software Security Verification: Problems and Approaches"*. STPSA 2014, 9th IEEE International Workshop on Security, Trust and Privacy for Software Applications, in COMPSAC 2014, 21-25 July 2014, Västeras, Sweden