Kernel Trace Analysis

Hashem Waly and Béchir Ktari

FACULTÉ DES SCIENCES ET DE GÉNIE
Université Laval, Québec, Canada

December 9, 2011
École Polytechnique de Montréal, Canada

Introduction

Work since last meeting

- Hashem Waly
  - Complete the coding/documentation of the tool.
  - Writing thesis and preparing for his defence.
  - Publishing a paper and presenting the work in the Canadian Conference on Electrical Engineering (CCEE 2011) [WK11].
- Also
  - New student, Rimeh Zribi, was recruited at the M.Sc. level.
  - Currently working on a policy-based approach.

Motivation

Signature-Based

- A new scenario description language.
- An Eclipse framework is developed on top of TMF.

Anomaly-Based

- Re-use the AFI language for the purpose.
- Generate models from the source code of programs.
- Detect anomalies between models and the execution of the programs.

Policy-Based

- Define policies for the access of different resources of the system.
- Detect sequences of events that violate these policies.

Anomaly-Based Detection in LTTng Traces

Hashem Waly.

Plan

1. Anomaly-Based detection
   - Introduction
   - Implementation
   - Demo
   - Conclusion
2. Policy-based techniques
   - Introduction
   - Methodology
   - Proposed model
   - System architecture
   - Conclusion

Plan

Anomaly-based

Detecting the deviation of the actual system to a pre-defined **model**.

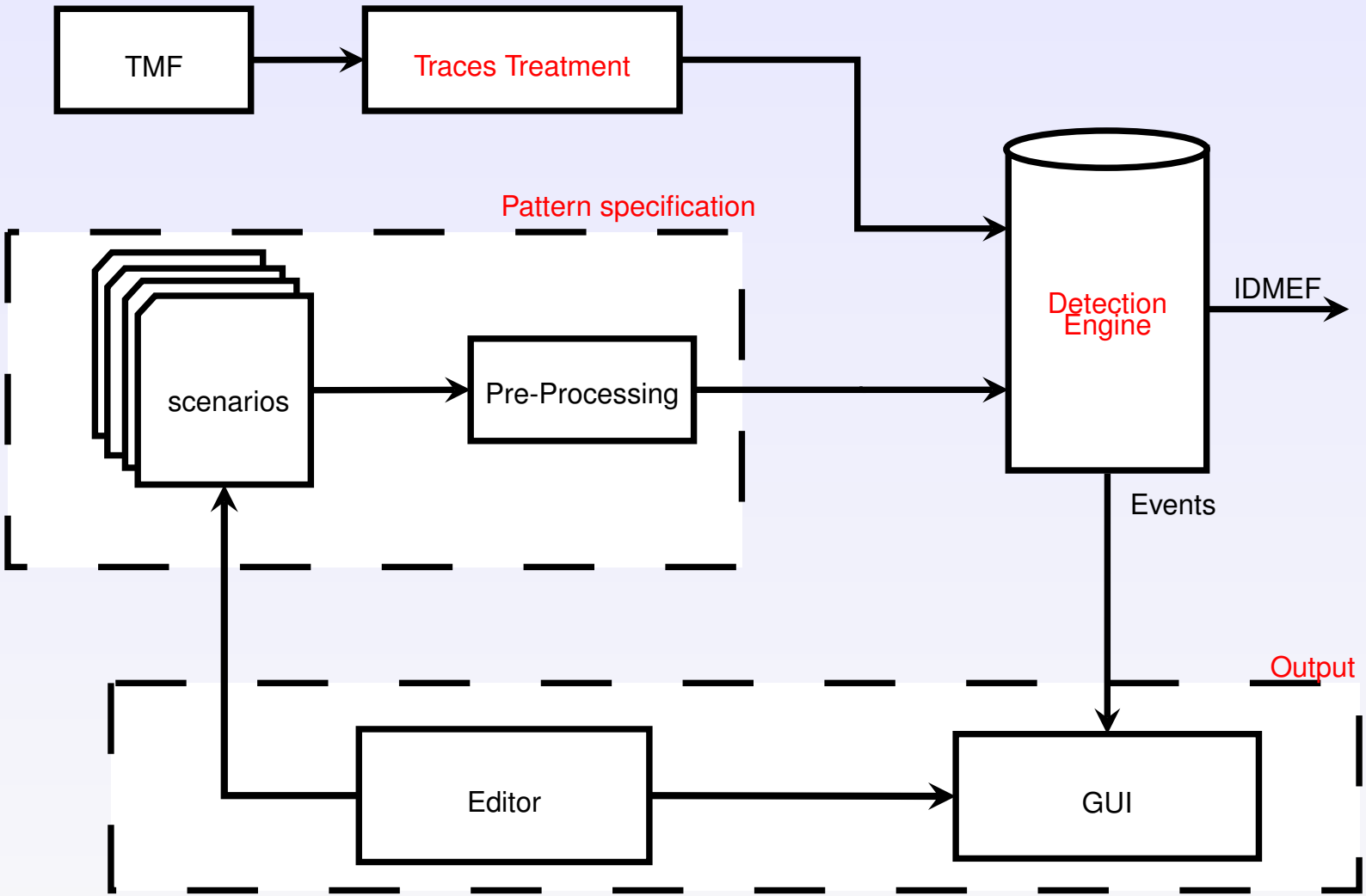
1. Learning phase.
   - Dynamic: Execute multiple times the program and progressively construct the model.
     Hidden Markov Models (HMM) [CC09], Improved (HMM), Gao et al. approach, etc.
   - Static: Profit from the availability of the source code to construct the model by using *static analysis*.

   Both approaches could also be combined by completing the static model by the program execution.
2. Detection phase.

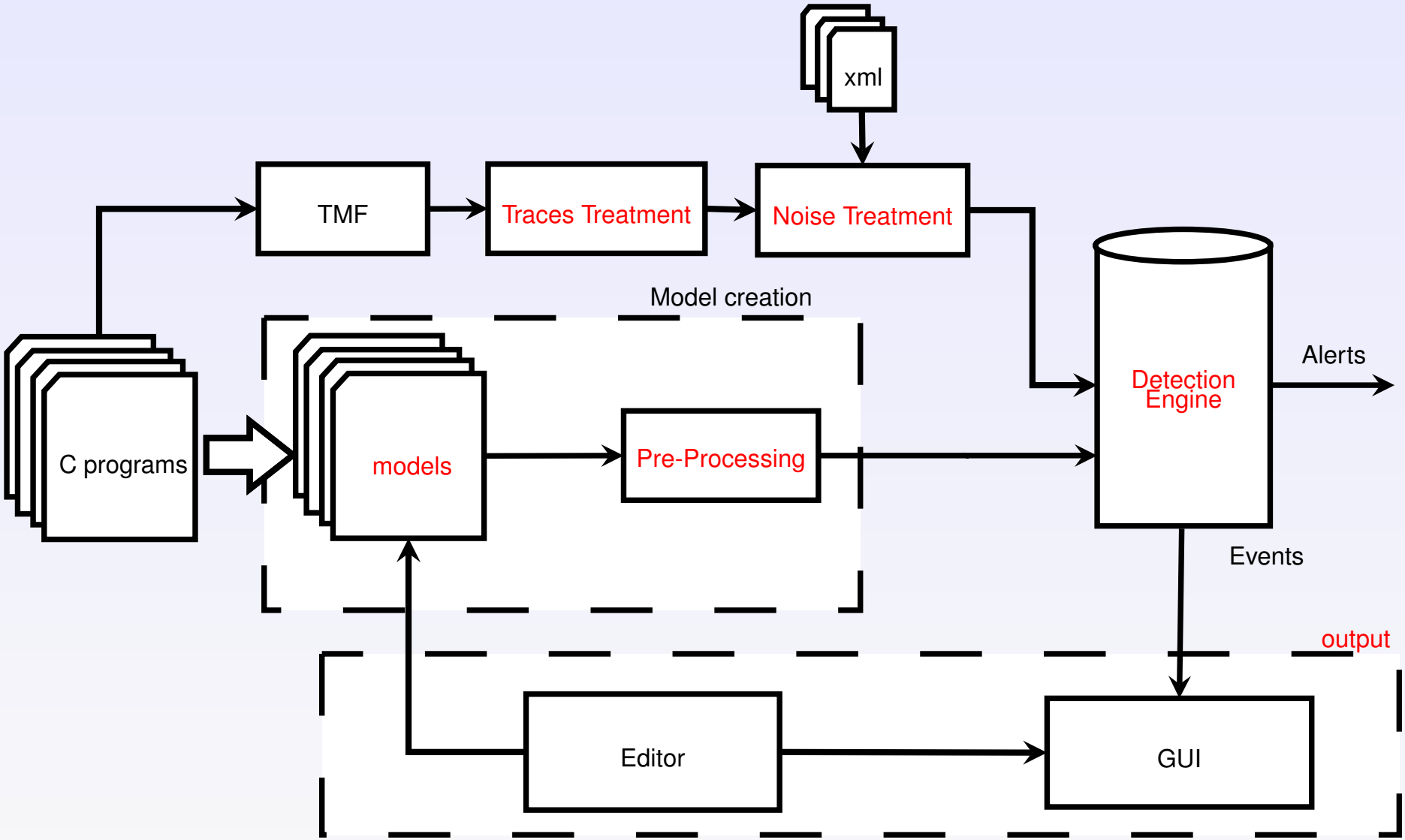
Objectives

- Generate behaviour *models* from the source code of C programs.
- Adapt and enrich the syntax of AFI language.
- Adapt the detection engine to detect anomalies in the execution of the programs in the traces.
- Integrate the developed work within *Eclipse* environment.

System architecture (old)



System architecture (updated)



Plan

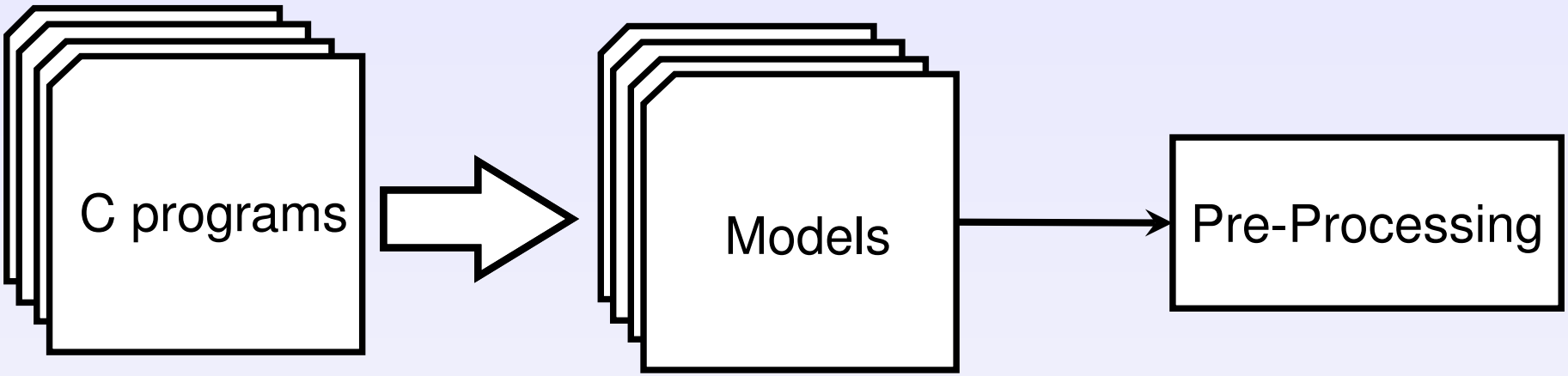
Static Analysis tools

- Bug Checking: analyse the code to detect programming bugs/threats.
  - CppCheck: Open source C/C++ static analyser (Bounds checking, memory leaks, resource leaks, etc) [cpp11].
  - Sparse: Detect coding faults in Linux kernel [spa11].
  - Coccinelle: Collateral evolution by doing transformations on the source code based on pre-defined patterns [Coc11].
  - Coverity: A commercial tool to detect software bugs and programming errors [Cov11].
  - CODe ANlysis (CODAN): Eclipse framework to easily ”plugin” end user checks [Cod11].
  - CDT: A fully C/C++ Development tooling that parses, compiles, executes C/C++ code [cdt11].

Traces Treatment

- `LttngSyntheticEvents` (pid, ppid, tgid, etc).
- Treat traces per event which increased the performance of the overall system.
- Filter non-concerned system-calls (metadata, memory operations, etc).
- The noise is defined as a separate XML files.

Models creation



- The Eclipse C/C++ Development Tooling (CDT) is be used to read the C programs.
- The Abstract Syntax Tree (AST) from CDT is then parsed by our tool.
- The model encapsulates the function calls.

File operation example

```c
#include <stdio.h>

int main()
{
  FILE *file;
  file = fopen("file.txt","a+");
  for(int i=0;i<10;i++){
    fprintf(file,"%s\n",i);
  }
  fclose(file);
  return 0;
}
```

```
include "./file_header.scn";
model main()
{
    event e1:fopen;
    repeat(10){
      event e2:fprintf;
    }
    event e3:fclose;
}
```

- The basic constructs are treated: while/for loops, if/switch case, function calls, return, variables definitions, functions definitions, etc.

Challenges

- Executing the program generates 462 system-calls.

#	Channel	Type	#	Description
1.	FS	exec	1	Executes the program.
		open	3	/etc/ld.so.cache,/lib/i386linuxgnu/libc.so.6, file.
		close	3	Closing the above files.
		read	1	/lib/i386linuxgnu/libc.so.6
2.	Kernel	page_fault_entry/exit	238	Trap functions.
		syscall_entry/exit	72	Entry/Exit system calls.
		timer_set	1	Sets a timer for certain time.
		kernel.sched_try_wakeup	3	Scheduler related.
		sched_schedule	1	Scheduler related.
		process_exit	1	Exiting the process
		send_signal	1	Send signal to terminate process.
3.	MM	page_free	140	Freeing a page from memory.

Challenges

- Executing the program generates 462 system-calls.
- The model contains only 3 system calls.
- The solution could be:
  - Filter un-necessary system calls such as: memory operations, metadata, page faults, etc.
  - Modelling all behaviours of loading the program, linking with libraries, process creation/termination.
  - This process changes from a process to another.
  - Another solution is highlighting the start/end of the program by compiling the code using `finstrument` option.
  - Inserting them as a templates in the beginning/ending of the model.

Updating the old system

- The key word `model` separates models from scenarios.
- The syntax/semantic of the **language** has been updated to deal with different operators between statements: AND, OR, SEQ and NOT.
- The **GUI** has been updated to generate models, specify noise, and insert templates in the models.
- The **Engine** has been updated to compare models in the execution traces.

Plan

Plan

Conclusion

- We have updated the AFI project to deal with models creation/comparison.
- A lot of challenges are involved in the project.
- The model creation should be enhanced to deal with data flow.
- The engine should be enhanced to deal with uncertainty.
- Precise models lead to better results in detecting anomalies and enrich the Linux Knowledge Base (LKB).

Knowledge-Based Model

- Structure knowledge and analysis execution within a computer through a Linux Knowledge Base (LKB) [Des11].
- Enriching the model by combining the 3 major detection approaches (signature, anomaly-based and signature based).
- Combine their strengths to lower the impact of their weaknesses.
- The models generated from the static analysis of the code could be combined by the execution of the programs to enrich the knowledge base.

Discussion

- Complete the generated models by treating more C/C++ instructions.
- Analyse data by tracking the values of the different variables, pointers, function pointers, arrays, etc.
- For the uncertainty about the values of certain variables, we have added range of values and '$' operator for that purpose.
- These variables could be substituted by their actual values from the dynamic execution of the code.

Bibliography I

Yu-Shu Chen and Yi-Ming Chen.

Combining incremental hidden markov model and adaboost algorithm for anomaly intrusion detection.

In *Proceedings of the ACM SIGKDD Workshop on CyberSecurity and Intelligence Informatics*, CSI-KDD '09, pages 3–9, New York, NY, USA, 2009. ACM.

Cdt, 2011.

`http://eclipse.org/cdt/`.

Coccinelle, 2011.

`http://coccinelle.lip6.fr/`.

Codan, 2011.

`http://wiki.eclipse.org/CDT/designs/StaticAnalysis`.

Bibliography II

Coverity, 2011.

`http://www.coverity.com/.`

Cppcheck, 2011.

`http://cppcheck.sourceforge.net/.`

Mathieu Desnoyers.

Knowledge base model for the linux kernel.

Technical report, EfficiOS Inc., March 2011.

Sparse, 2011.

`http://cppcheck.sourceforge.net/.`

Hashem Waly and Béchir Ktari.

A complete framework for kernel trace analysis.

In *Canadian Conference on Electrical Engineering (CCEE)*, May 2011.

Thanks

- We would like to thank DRDC Valcartier and Ericsson for their financial support.
  1. Mario Couture from **DRDC** Valcartier.
  2. Francois Chouinard, Bernd Hufmann, and Matthew Khouzam from **Ericsson** Montréal.
  3. Papa Maleye from **Laval** University.
- A big thanks to everyone helped in that project along the past two and half years.

Policy-Based Detection in LTTng Traces

Béchir Ktari

Plan

Motivation

- Specify all the suspicious behavior is a far away target.
- Exploit the work of Hashem to detect or report new malicious or suspicious behavior.
- Helping the system administrator to specify activities that fit the security policy.

Objectives

Signature based approach

Attacks are identified by a scenario language (pattern) that model malicious activity.

Policy based approach

A logical security policy specification : any sequence of actions (events) that leads to the violation of security policy should been identified and intercepted.

Plan

Methodology

- Using expert systems to capture the logic necessary to identify the sequences that violate security policy :
  - The security policy is characterized by predicates.
  - The reasoning performed by a security expert to deduce that there is a violation is represented by facts and rules.
  - A rule can specify multiple new suspicious behaviors.

Methodology

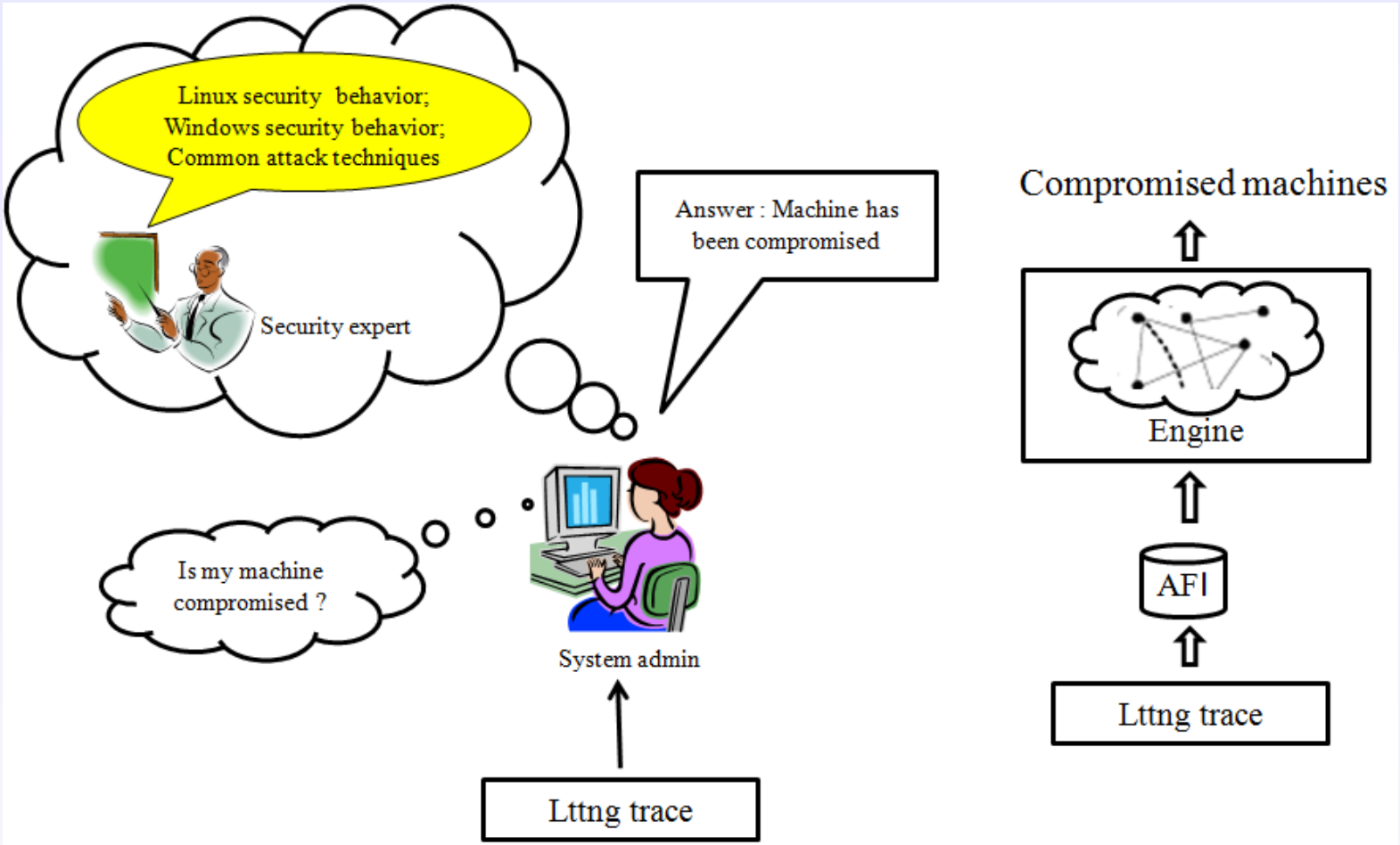


Figure: Administrator behavior and system behavior

Plan

Description

Proposed solution: consider actions (events) identified in a trace and deduce their effects in terms of security.

- The events captured in LTTng trace are represented by predicates after the identification of the effect of each action.
- The knowledge base used to identify a security violation is fed by a set of facts and rules.
- The facts and rules are represented by predicates.

Characteristics

- Fact: specify the effects (semantics) of each action on the system resources.

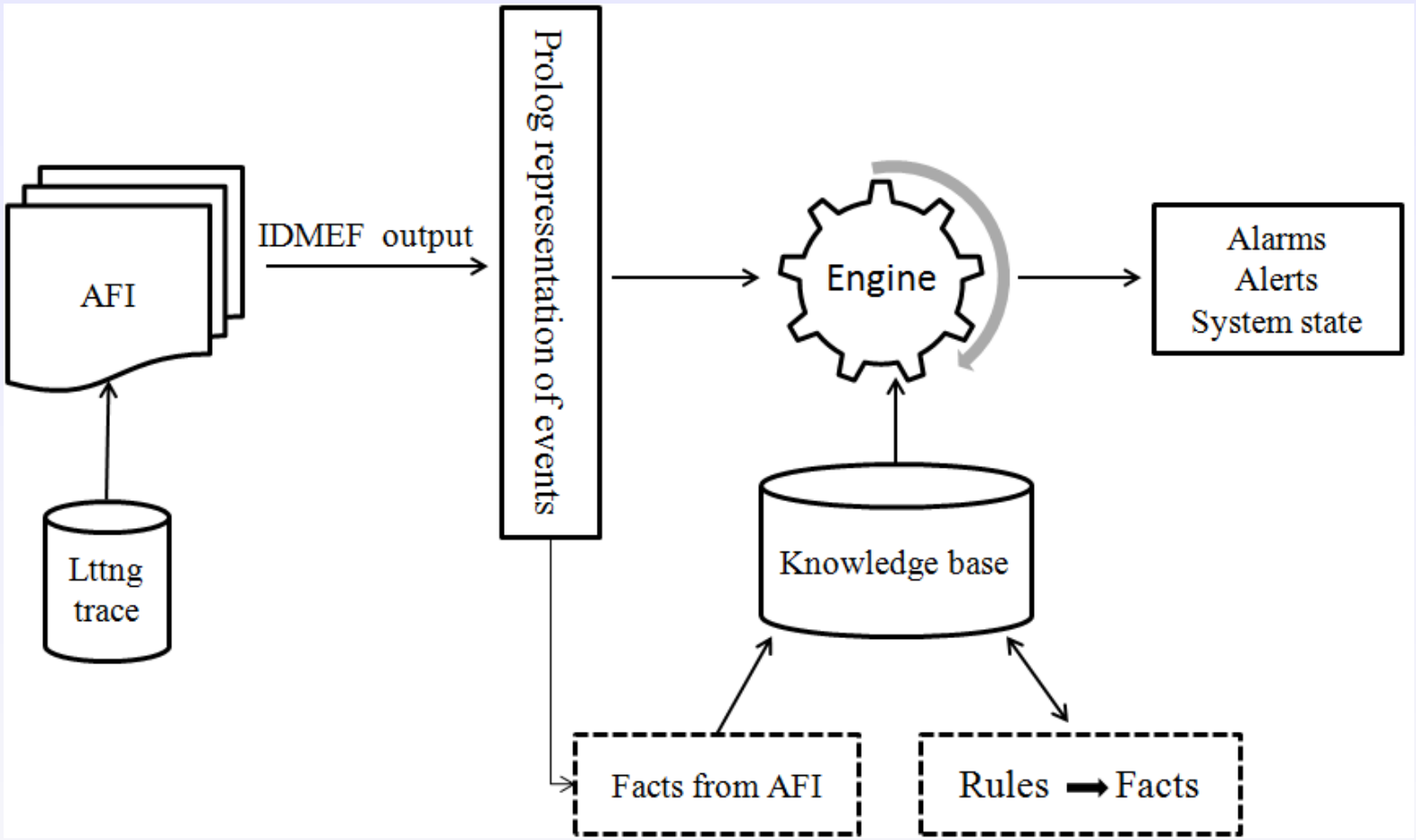
$$\circledast \textit{ What can we infer } \circledast$$

- Rule: express logical relation among various events.

$$\textit{event1} \rightarrow \textit{event2}$$
$$\textit{"lead to" relation}$$
$$\textit{event1 may cause event2}$$

Plan

System architecture



Work progress

The current work is to model the system for the detection of behavior that violates the system security. This model indicates:

- The identification of the involved entities: resources (files, directories, memory, process, net, service), actions, events, etc. Predicates must be defined to represent all This entities.
- The identification of rules used by a security expert for reasoning.
- The determination of the syntax and semantics used by different entities.

Remark

Signature based approach	Policy based approach
Add all the suspicious behaviors in the database	Add and improve the system of knowledge (rules) to identify a new suspect behavior

The policy based detection identifies a set of new behaviors $\Rightarrow$ *Reduce of false negatives*

Plan

Conclusion

- In the field of kernel tracing, this is a proposed method to detect new malicious activities.
- Using AFI results to improve intrusion detection.
- Need the security expert knowledge for building rules.
- The implemented plugin could be used independently of AFI (thanks to the IDMEF interface).

Future work

- Complete this system modelling.
- Implement a plugin that clearly identifies behaviours that violate security policy.
- Enhance the knowledge base of rules in order to detect new suspicious behaviours.
- Possibility of integrating the notion of uncertainty in the modelling of facts and rules.

Questions

Example : illegal file access

Rules

1. touch(agent, file) $\rightarrow$ file(file) && authorized(agent, file)
2. remove(agent, file) $\rightarrow$!file(file)
3. ln-s(agent, link, file) $\rightarrow$ file(file) && linked(link, file)
4. print-process(printer, link, file) $\rightarrow$ printed(printer, file) && !queued(link, printer)
5. get-file(agent, file, printer)$\rightarrow$ read-access(agent, file)

$\rightarrow$: ”lead to” relation
&& : AND

! : suppression of the fact from the knowledge base

Example : illegal file access

security policy

No read access to a secret file

read-access(agent, secret_file) && authorized(agent, secret_file) $\Rightarrow$ security policy violation

Example : illegal file access