Bridging Threat Models and Detections: Formal Verification via CADP

Dumitru-Bogdan Prelipcean[1,2,3] Cătălin Dima[3]

1-Alexandru Ioan Cuza University, Iași, Romania
2-Bitdefender, Iași, Romania
3-LACL, Université Paris–Est Créteil, France

September 17, 2025
Working Formal Methods Symposium 2025

The Problem

- Detection rules (SIEM/EDR/IDS) are written in diverse DSLs.
- Threat intelligence uses attack trees, ATT&CK, and IoCs—mostly informal.
- **Gap:** No formal assurance that rules *cover* the intended threat behavior.

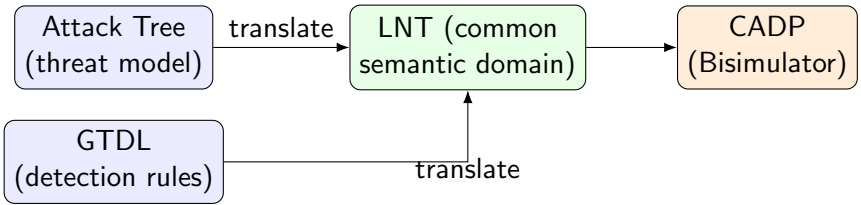
The Problem

- Detection rules (SIEM/EDR/IDS) are written in diverse DSLs.
- Threat intelligence uses attack trees, ATT&CK, and IoCs—mostly informal.
- **Gap:** No formal assurance that rules *cover* the intended threat behavior.

Goal

Formally verify conformance between high-level threat models and executable detection logic.

Key Idea


Attack Tree
(threat model)
translate
LNT (common
semantic domain)
CADP
(Bisimulator)
GTDL
(detection rules)
translate

Contributions

1. Compositional LTS semantics for **attack trees** and **GTDL**.
2. Semantics-preserving **translations to LNT**.
3. Automated **conformance checking** (bisimulation, weak trace inclusion).
4. **Tooling:** CLI pipeline from models to CADP verification.
5. **Evaluation:** Real-world malware (LokiBot, Emotet) + parametric scalability.

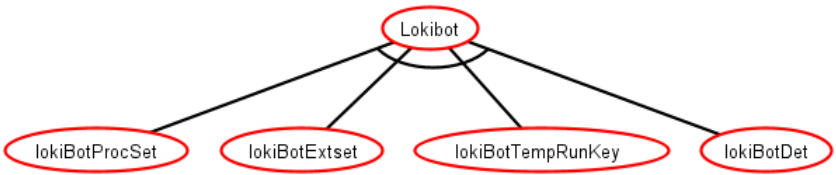
Attack Trees (AT)

- Hierarchical decomposition of an attacker goal into subgoals.
- Constructors: **LEAF**, **OR**, **AND** (unordered), **SAND** (sequential).
- Denote a *finite* set of traces over atomic actions.

Trace Semantics

$$\mathcal{T}(\mathrm{LEAF}_a) = \{a\}, \qquad \mathcal{T}(\mathrm{OR}(\dots)) = \bigcup \mathcal{T}(\cdot)$$
$$\mathcal{T}(\mathrm{AND}(\dots)) = \| \text{ (shuffle)}, \quad \mathcal{T}(\mathrm{SAND}(\dots)) = \cdot \text{ (concat)}$$

Lokibot Attack Tree

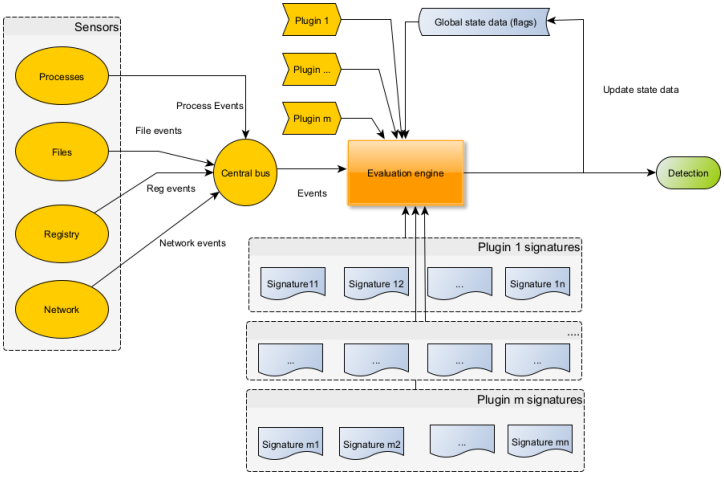


GTDL: Generic Threat Detection Language

- Declarative rules over event streams with stateful conditions.
- Building blocks: assignments, plugin calls, IF/THEN/ELSE, action `GlobalFlag.Set("D")`.
- Execution model: many rules run **in parallel**, re-evaluated per event.

Observable event: the detection action (`Set("D")`) becomes label d.

GTDL


Sensors
Processes
Files
Registry
Network
Process Events
File events
Reg events
Network events
Central bus
Plugin 1
Plugin ...
Plugin m
Global state data (flags)
Update state data
Evaluation engine
Events
Detection
Plugin 1 signatures
Signature11
Signature 12
...
Signature 1n
Plugin m signatures
Signature m1
Signature m2
Signature mn

Use Case: System-Level (Lokibot)

```
[DETECTION]
Detection_name = 'LokibotProcess'
Apply_when = "Process"
[RULE]
v_process := inPluginCall(IsProcessName, "ytpgwim");
v_location := inPluginCall(IsInProcessPath, "%TEMP%");
IF v_process AND v_location THEN
  GlobalFlag.Set("LokibotProcess");
END IF
```

Composite Signature (Correlation)

```
[DETECTION]
Detection_name = 'LokibotIncident'
Apply_when = "GlobalFlags"
[RULE]
flag1 := GlobalFlag.IsSet("LokibotProcess");
flag2 := GlobalFlag.IsSet("BotExtensions");
flag3 := GlobalFlag.IsSet("TempRunKey");
flag4 := GlobalFlag.IsSet("KnownCCAccesed");
IF flag1 AND flag2 AND flag3 AND flag4 THEN
  GlobalFlag.Set("LokibotIncident");
END IF
```

AT $\rightarrow$ LNT (Sketch)

- **LEAF**: emits non-silent action.
- **OR**: nondeterministic choice.
- **AND**: parallel composition.
- **SAND**: sequential composition.

Correctness

For any AT A: $\mathrm{Traces}(tr(A)) = \mathcal{T}(A)$.

GTDL $\rightarrow$ LNT

- Plugin-assigned variables become *process parameters*.
- Boolean logic and control flow map homomorphically to LNT.
- `GlobalFlag.Set("D")` $\mapsto$ output on channel $dSet$.
- Multiple signatures $\mapsto$ parallel composition.

Theorem (Trace Preservation)

For any GTDL rule P: $\mathrm{Traces}(\mathcal{T}(P)) = [\![P]\!]_{\mathsf{GTDL}}$.

From GTDL to LNT — Side by Side

GTDL

```
[DETECTION] Name='LokibotProcess'
[RULE]
v_process = inPluginCall(IsProcessName,"yptgwim");
v_location = inPluginCall(IsInProcessPath,"%TEMP%");
IF v_process AND v_location THEN
  GlobalFlag.Set("LokibotProcess");
END IF
```

LNT

```
process LokibotProcess [flag:FLAG_CHANNEL]
  (in var pname, ppath:String) is
  if pname == "yptgwim" and ppath == "%TEMP%" then
    flag(TRUE)
  end if
end process
```

Common Alphabet & Channels

- Both AT and GTDL models emit on the *same* observable channels.
- Internal steps in GTDL become τ (silent) actions.
- Enables CADP to decide: strong/weak simulation, (bi)simulation, (weak) trace (inclusion/equivalence).

Equivalences & Inclusions

- **Strong bisimulation:** strict stepwise matching (often too strong).
- **Weak bisimulation:** abstracts away τ.
- **Trace equivalence/inclusion:** focus on observable detections.

Interpretation

- *Inclusion* (AT $\subseteq$ DET): no false negatives.
- *Equivalence*: no false negatives nor over-approximation.

Toolchain

- `tree2lnt.py`: AT (YAML) $\rightarrow$ LNT
- `gtdl2lnt.py`: GTDL $\rightarrow$ LNT
- `verify.sh`: compile, minimize, run `bisimulator`
- `measure_times.py`: benchmark orchestration

Case Studies (Examples)

- **LokiBot:** AND-structured actions; observational equivalence achieved.
- **Emotet:** mixed AND/SAND; iterative refinement; inclusion holds.

Outcome

Framework flags semantic mismatches and guides signature refinement.

Case Study: LokiBot Tree

```
process LokibotTree [lokiBotProcSet, lokiBotExtset,
   lokiBotTempRunKey:FLAG_CHANNEL, lokiBotDet:any]  is
par
LokibotProcessLeaf [lokiBotProcSet]
|| LokibotExtensionLeaf [lokiBotExtset]
|| LokiTempExeRunKeyLeaf [lokiBotTempRunKey]
|| LokibotActions [lokiBotProcSet, lokiBotExtset, lokiBotTempRunKey,
       lokiBotDet]
end par
end process
```

Case Study: LokiBot Detection

```
process Engine [lokiBotProcSet, lokiBotExtset,
   lokiBotTempRunKey:FLAG_CHANNEL, lokiBotDet:any]  is
loop
par
LokibotProcess [lokiBotProcSet] ("yptgwim", "%TEMP%")
|| LokibotExtension [lokiBotExtset] (".exe")
|| LokiTempExeRunKey [lokiBotTempRunKey] ("Run", "Run")
|| LokiDetection [lokiBotProcSet, lokiBotExtset, lokiBotTempRunKey,
        lokiBotDet]
end par
end loop
end process
```

Parametric/Scalability Results

Setup. Attack trees & detection models with varying size and operators.
Findings.

- **AND-only:** weaktrace faster than observational.
- **SAND-only:** linear growth, similar times.
- **OR-only:** linear growth, both options feasible.
- **Mixed AND-OR, AND-SAND:** verification cost depends on operator mix.

Assumptions & Limitations

- Assurance depends on **quality of attack trees**.
- Current method is not anomaly-based; zero-days outside the model may evade.
- Extensions: automated AT and Detection synthesis, extended semantics for plugins.

Conclusion

- Unified semantic domain (LNT) for threats and detections.
- Automated conformance with CADP (bisimulation, weak trace inclusion).
- Validated on real malware and scalable synthetics.

Takeaway

Formal verification can systematically reveal detection blind spots and guide refinement before deployment.

Thank you for your attention !
Q&A