Parser Instrumentation for Semantic-Aware Applicative Intrusion Detection

Grégor QUETEL - Télécom Paris
Pierre-François GIMENEZ - Inria
Thomas ROBERT - Télécom Paris
Laurent PAUTET - Télécom Paris

10th June 2026

IFIP SEC 2026, Perth, Australia

Intrusion Detection Systems

In 2025, system intrusions account for **53% of all data breaches** (+17% since 2024) [1].

Intrusion Detection Systems (IDSes) monitor system behaviour to detect malicious activity.

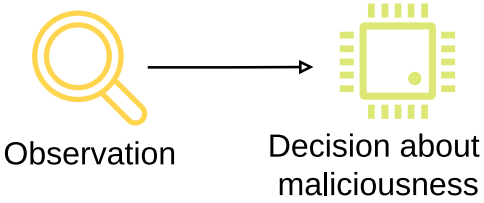


Figure 1: IDSes consist of a data collector and a decision engine

Focus on the detection of **applicative** attacks: SQL Injection, insider attack, ...

Key open challenges include:

📉 **Detection performance**
⏳ **Runtime overhead**
⚠️ **Robustness against evasion attacks**

Providing Better Observations: Where ?

Richer observations from a better-positioned collector address all three challenges.

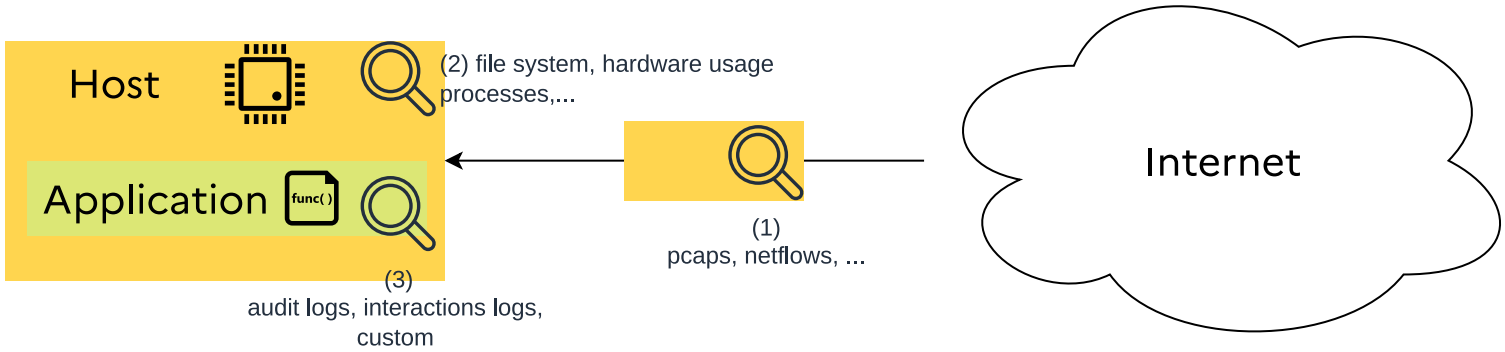


Figure 2: Typical locations to collect observations for IDSes

🔍 **Where to observe?**

- (1) **Network**: traffic is encrypted
- (2) **Host**: semantic gap, application intent is lost
- (3) **Application**: direct access to state or interactions

Providing Better Observations: What ?

🔡 What to observe?

- **State dynamics analysis** requires software-specific adaptation
- **Interactions** are easier to monitor
  - **Lexical**, **Syntactic** and **Semantic** features can be derived.

Lexical	Surface properties of tokens
Syntactic	Token organization according to the grammar
Semantic	Effect of the input on application behaviour

  - Semantics inferred via **implicit** representations (i.e. embeddings): high overhead, brittle against adversarial inputs [2], [3]

Our goal: designing a data collector at the application level providing **explicit semantic information** from user interactions.

Research Objectives

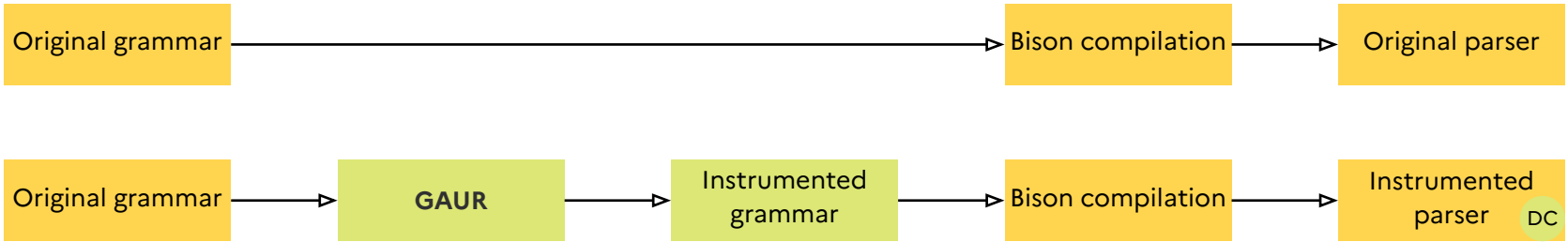
RO1	**Low-overhead** data collection via application instrumentation, transparent to deployment, no maintenance overhead
RO2	**Full characterization**: capture lexical, syntactic and **semantic** information directly during query processing
RO3	**Explicit** semantic annotations for better **explainability** and **robustness**

Parser-based Data Collection

The parser is a **strategic** data collection point:

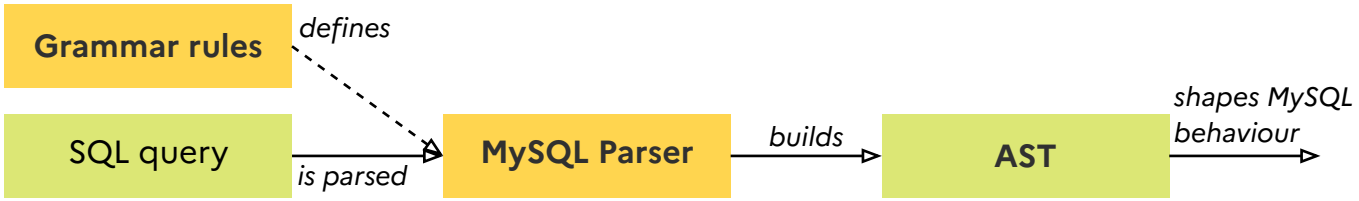
Low overhead: already in the processing pipeline	**RO1**
Automatable: generated from grammar	**RO1**
Full coverage: all inputs pass through it	**RO1**
Natively provides **lexical** & **syntactic** information	**RO2**

We designed **GAUR** to integrate data collection logic into Bison grammars.



But how to produce semantic information?

Semantic Information Production



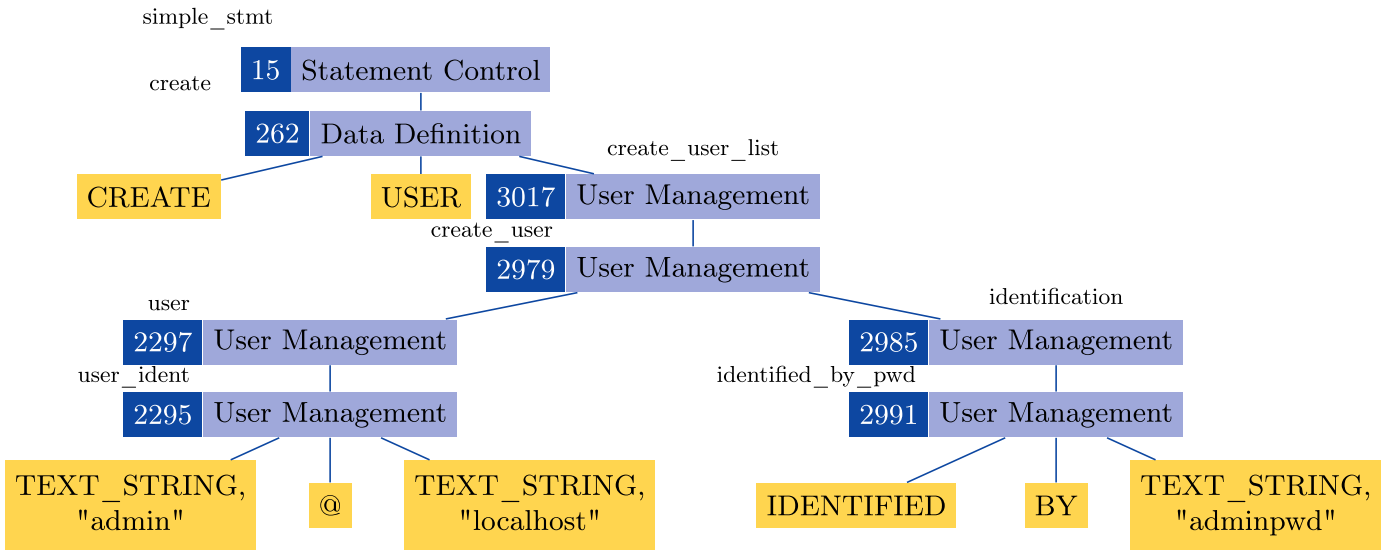
- We can associate **explicit semantic tags** to parser rules
- When a rule is used, we collect its tag
- The set of allowed semantic tags are defined through **semantic models**

Model	Semantic Tags
Expert	*actions*: Create, Delete, Modify, Execute, Read. *objects*: Table, Index, View, User, Database, …
Mistral	Data definition, Data manipulation, Data query, Security privileges, Transaction control, User management, …

Table 1: Semantic models for MySQL defined by an expert / generated by Mistral.

GAUR DC at runtime

For each query, the GAUR DC produces a **tree-structured observation**. Here for the MySQL query: `CREATE USER "admin"@"localhost" IDENTIFIED BY "adminpwd"` and the semantic model defined by Mistral.



Lexical	Token types and string values (e.g., `TEXT_STRING`, `"admin"`)
Syntactic	Parser rule identifiers and tree hierarchy
Semantic	Mistral-defined semantic tags associated to the rules

Evaluation

- ⏳ **RQ1**: What is the runtime overhead?
- 📈 **RQ2**: Does GAUR improve detection effectiveness?
- 💪 **RQ3**: Is explicit semantics more robust against adversarial mutations?

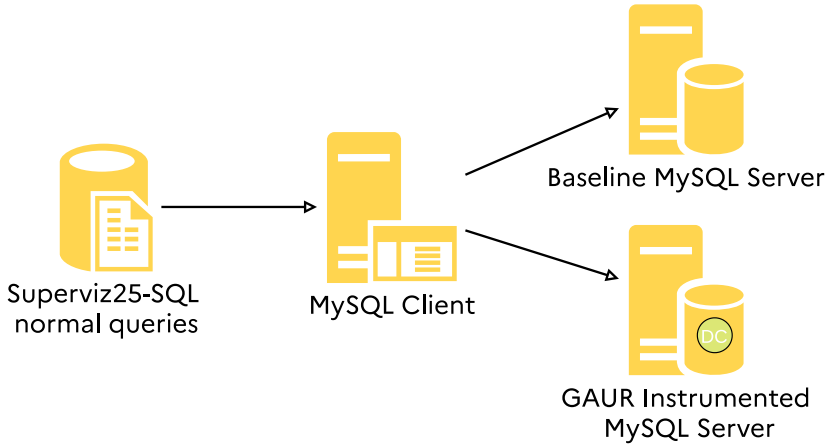
Experimental Setup

We study GAUR through the emblematic SQL attack detection tasks

- **Dataset**: Superviz25-SQL [4]. Unsupervised, 3.7M queries, SQLIAs + insider attacks
- **Decision engines**: One-Class SVM (OCSVM) and AutoEncoder (AE)
- **Baselines**:
  - **Li et al.** [5]: manually selected SQL features
  - **SecureBERT** [6]: Sentence-BERT fine-tuned on cybersecurity data

RQ1: Runtime Overhead

What is the runtime overhead of the data collector?



We compare the average query execution time over 3M+ queries on a baseline vs. GAUR-instrumented MySQL server.

Low mean overhead: $\mathbf{0.113 \pm 0.006}$ **ms** per query (0.31% of execution time)

Enabled by:

- 🔍 Data collection logic **embedded in parser subroutines**
- 🔍 No runtime NLP, tags are **explicit**

RQ2.1: Semantic Model Instantiation

What is the impact of the semantic model instantiation on detection performance?

Classifier	Metric	Expert	ChatGPT	Claude	Llama	Mistral	OSS-GPT	Naive
OCSVM	AUPRC	92.64	93.00	92.79	91.94	92.41	92.65	**93.27**
	AUROC	95.81	96.06	94.98	95.12	95.13	94.77	**96.81**
AE	AUPRC	92.76	**95.57**	92.91	94.04	93.66	94.13	92.41
	AUROC	96.50	**98.06**	97.34	97.89	97.74	97.61	97.98

Table 2: AUPRC and AUROC (%) for SQL attack detection using different semantic model instantiations

We compare GAUR with 6 semantic models + a **naive** baseline (one unique tag per rule = 1,012 features).

🔍 Semantic model has **marginal influence** (AUROC between 95.81 and 98.06)
🔍 The naive approach also exhibits similar performances

RQ2.2: Detection Performance vs. Baselines

How performant are pipelines using GAUR relative to existing approaches?

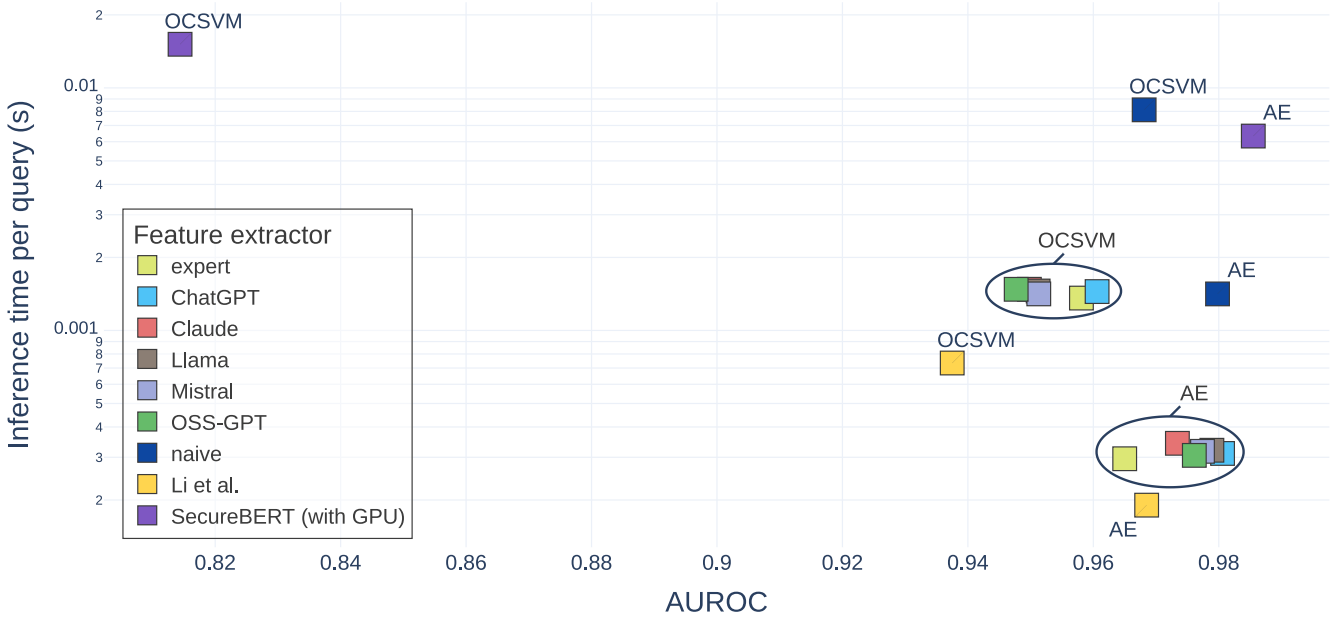
Model	AUPRC	AUROC	F1	Bool.	Error	Union	Stack.	Time	Inline	Insider
GAUR (Claude) + AE	92.91	97.34	**93.79**	94.69	**80.58**	94.26	71.85	65.69	**88.77**	79.06
GAUR (ChatGPT) + OCSVM	93.00	96.06	93.72	**94.74**	80.17	**94.29**	**72.31**	**66.51**	85.89	75.94
GAUR (Mistral) + AE	93.66	97.74	93.68	94.69	80.51	93.69	71.19	65.22	86.90	**80.17**
Li et al. + OCSVM	89.65	93.75	91.82	91.49	77.04	93.77	67.12	61.98	66.74	**0.00**
Li et al. + AE	92.57	96.85	92.06	91.80	79.80	93.54	67.61	60.53	74.44	**0.00**
SecureBERT + OCSVM	48.88	81.44	14.02	7.96	1.52	20.50	0.98	1.31	1.22	**0.00**
SecureBERT + AE	**95.65**	**98.55**	91.78	92.74	79.10	83.66	64.84	64.43	77.11	**0.00**

Table 3: Detection performance (%). Recall per attack technique shown.

🔍 GAUR achieves the best **F1-score** (93.79%) and, due to its location, is able to capture insider attacks.
🔍 SecureBERT + AE obtains the highest AUROC score (98.55%)

RQ2.3: Inference Overhead

What inference latency do decision engines incur compared to existing methods?



🔍 GAUR collection + inference (0.377 ms) $<$ SecureBERT inference (6.46 ms).
🔍 Naive approach (1.41–8.11 ms): **semantic abstraction reduces overhead**

RQ3: Robustness to Adversarial Mutations

Can an attacker mutate a malicious query to evade detection while preserving its effect?

- **WAF-A-MoLE** [7]: guided mutational fuzzing with a budget.
- Mutations range from **lexical** (keyword casing, comments) to **semantic** (tautology rewrite).

Type	Original	Mutated
Case swap	`admin' OR 1=1#`	`admin' oR 1=1#`
Operator swap	`admin' OR 1=1#`	`admin' OR 1 LIKE 1#`

Table 4: Examples of WAF-A-MoLE mutations

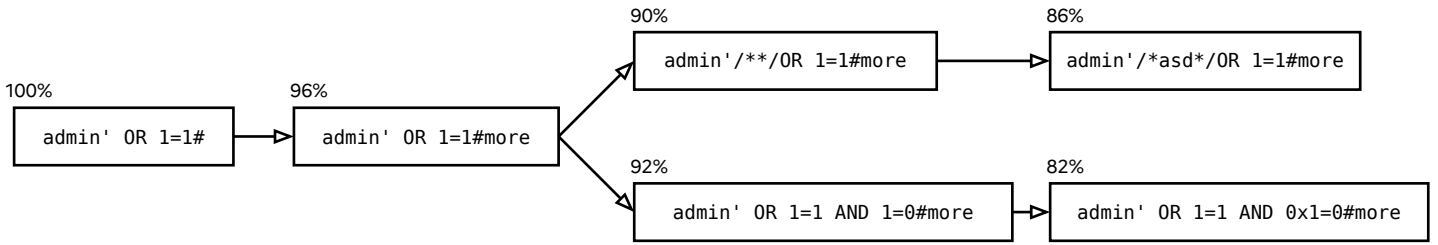


Figure 4: A possible mutation tree of an initial payload

RQ3: Robustness to Adversarial Mutations

WAF-A-MoLE is used with:

- 100 random attacks from Superviz25-SQL
- A budget of 1000 mutations per query (default)

Model	Evaded queries	Avg. rounds to evasion
GAUR (Mistral) + AE	**0 / 100**	Not evaded
SecureBERT + AE	14 / 100	123.14

🔍 Explicit features are **stable** under semantic-preserving mutations
🔍 Implicit embeddings encode semantics **opaquely**: small surface perturbations shift the latent space

Conclusion

Summary

GAUR is a novel type of data collector for AppIDS that

- 🤖 is **easy to deploy**
- ⏳ exhibits **low overhead**
- 📈 allows **sota-level detection**
- 💪 provides **robust** semantic representations

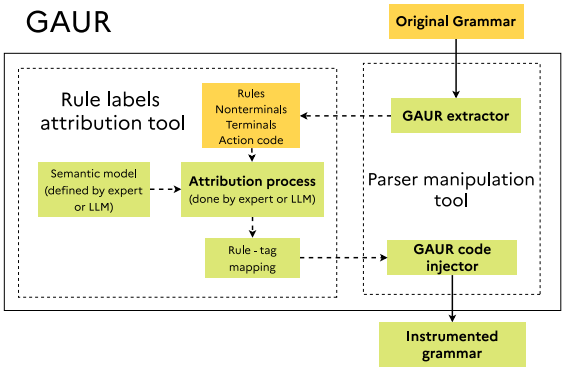
Future Works

- **Intrusion prevention**: block semantically malicious inputs before execution
- **Generalizability**: instrument other Bison-based parsers beyond MySQL
- **Semantic refinement**: incorporate static analysis or contextual information

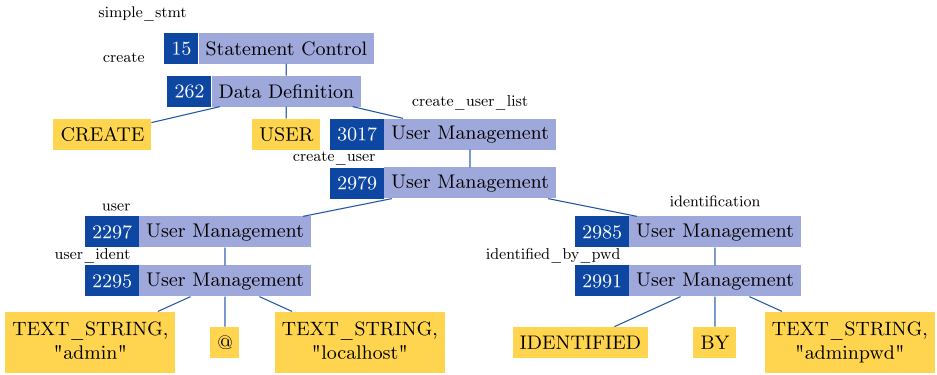
Thanks

This work has been partially supported by the French National Research Agency under the France 2030 label (Superviz ANR-22-PECY-0008). The views reflected herein do not necessarily reflect the opinion of the French government.

- GAUR: https://github.com/gquetel/gaur
- Experiments: https://github.com/gquetel/gaur-sql-detect
- Dataset: Superviz25-SQL [4]



GAUR Instrumentation process



GAUR-produced observation

References

[1] Verizon, "Data Breach Investigations Report." [Online]. Available: https://www.verizon.com/business/resources/reports/dbir/

[2] L. Li, R. Ma, Q. Guo, X. Xue, and X. Qiu, "BERT-ATTACK: Adversarial Attack Against BERT Using BERT," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Association for Computational Linguistics, 2020, pp. 6193–6202. doi: 10.18653/v1/2020.emnlp-main.500.

[3] S. Garg and G. Ramakrishnan, "BAE: BERT-based adversarial examples for text classification," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2020, pp. 6174–6181.

[4] G. Quetel, E. Alata, P.-F. Gimenez, T. Robert, and L. Pautet, "Superviz25-SQL: High-Quality Dataset to Empower Unsupervised SQL Injection Detection Systems," in *1st International Workshop on Assessment with New Methodologies, Unified Benchmarks, and Environments, of Intrusion Detection and Response Systems (ANUBIS)*, Toulouse, France, Sept. 2025.

[5] Q. Li, W. Li, J. Wang, and M. Cheng, "A SQL Injection Detection Method Based on Adaptive Deep Forest," *IEEE Access*, vol. 7, pp. 145385–145394, 2019, doi: 10.1109/ACCESS.2019.2944951.

[6] E. Aghaei, X. Niu, W. Shadid, and E. Al-Shaer, "SecureBERT: A Domain-Specific Language Model for Cybersecurity," in *Security and Privacy in Communication Networks*, Springer Nature Switzerland, 2023, pp. 39–56. doi: 10.1007/978-3-031-25538-0_3.

[7] L. Demetrio, A. Valenza, G. Costa, and G. Lagorio, "WAF-A-MoLE: Evading Web Application Firewalls through Adversarial Machine Learning," in *Proceedings of the 35th Annual ACM Symposium on Applied Computing*, ACM, 2020, pp. 1745–1752. doi: 10.1145/3341105.3373962.

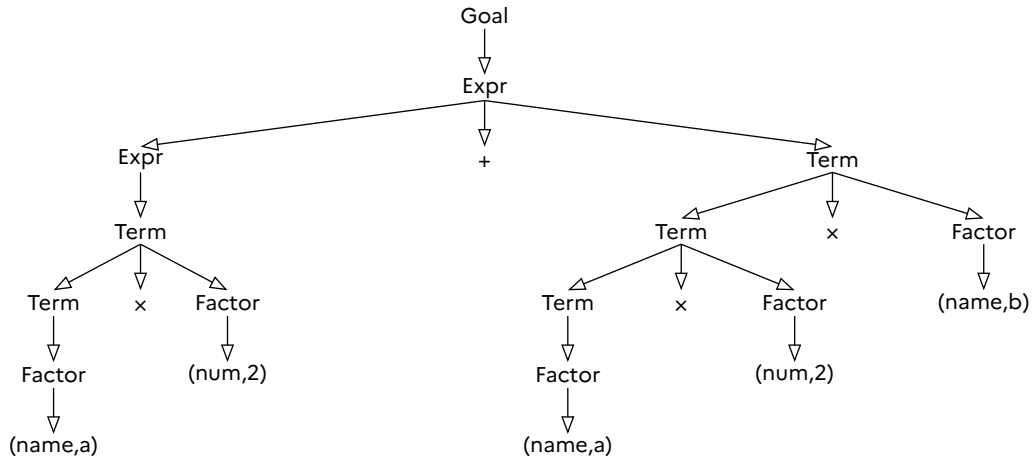
Appendix

Grammar structure

A **grammar** defines valid inputs through production **rules**. The **parse tree** records which rules the parser applied to derive a given input.

Goal → *Expr*
Expr → *Expr* + *Term*
| *Expr* – *Term*
| *Term*
Term → *Term* × *Factor*
| *Term* ÷ *Factor*
| *Factor*
Factor → (*Expr*)
| num
| name

(a) Classic Expression Grammar



(b) Parse Tree for $a \times 2 + a \times 2 \times b$

Appendix

Bison Grammar File

A Bison rule has a **name**, **alternatives** (separated by `|`), and **action code** `{ ... }` executed on reduction.

```
term
    : term TIMES factor  { $$ = $1 * $3; }
    | term DIV   factor  { $$ = $1 / $3; }
    | factor             { $$ = $1;      }
    ;
```

GAUR extracts per rule:

- **Rule name**: `term`
- **Terminals**: `TIMES`
- **Action code bodies**: `$$ = $1 * $3;`

Appendix

Features used for detection

A subset of the features provided by the GAUR data collector, used for the evaluation.

Lexical

len_query, c_comparison, has_ascii, c_num, c_upper, c_space, c_special, c_arith, c_square_brackets, c_round_brackets, c_curly_brackets

Syntactic

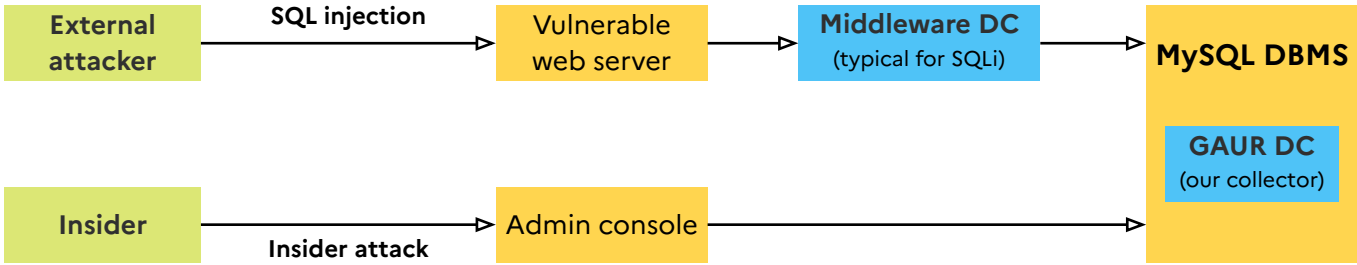
n_terminal, n_nonterminal, is_syntax_error, depth, n_parser_invoc

Semantic

Data Definition, Data Import/Export, Data Manipulation, Data Query, Database Management, Locking & Concurrency, Miscellaneous Operations, Replication & Clustering, Resource Management, Security & Privileges, Stored Procedures & Functions, System Information, System Maintenance, System Variables, Temporary Objects, Triggers & Events, User Management, Views, Statement Control, Transaction Control

Appendix

Attack Origins and Observation Points



🔍 The **Middleware DC** observes only the web-server ⇋ DBMS channel: it sees **SQL injections** but is **blind** to insider attacks.
🔍 Our **GAUR DC**, embedded in the DBMS, observes **all interactions**.