Beware of the Unexpected:

Bimodal Taint Analysis

Wai Chow [1], Max Schäfer [2], Michael Pradel [1]

[1] University of Stuttgart, [2] GitHub

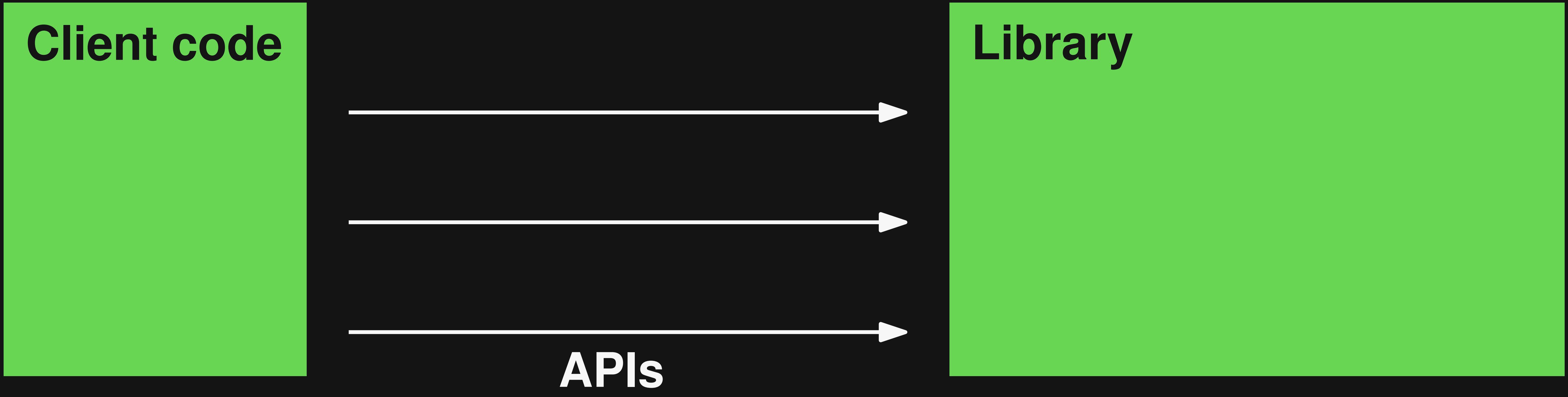
Motivation

- **Static analysis: Only as good as its specs**
- **E.g., taint analysis**
  - Need policy that specifies insecure source-sink pairs
  - Problematic flow if both
    - data flows from source to sink and
    - the flow is unexpected by developers

Example: Command Injection

Want: **Untrusted data does not flow to `exec()`**

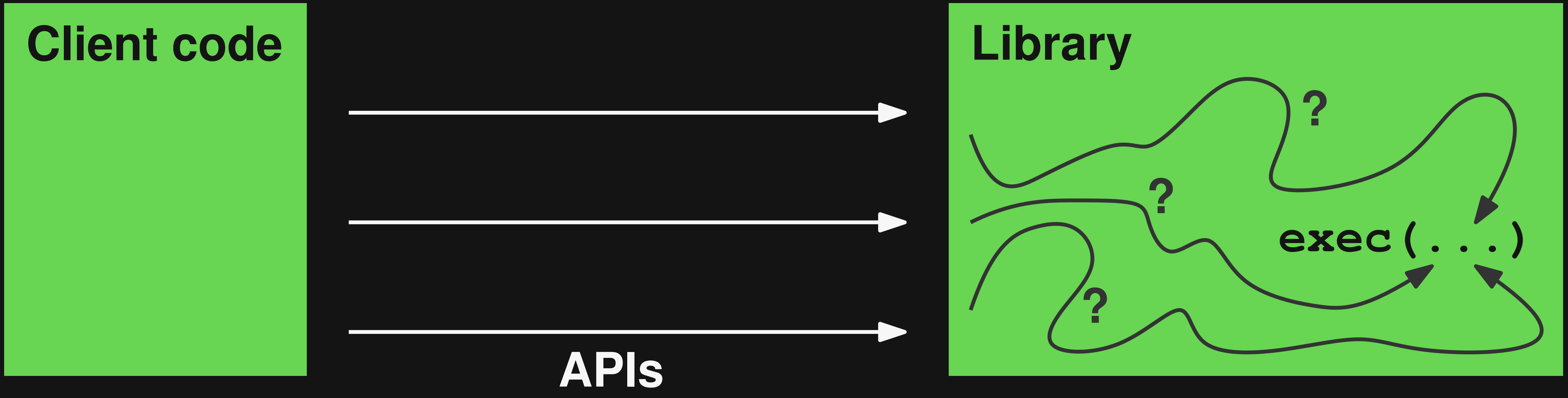
(Otherwise, command injection vulnerability)



Example: Command Injection

Want: **Untrusted data does not flow to `exec()`**

(Otherwise, command injection vulnerability)



Example: Command Injection

Want: Untrusted data does not flow to `exec()`

(Otherwise, command injection vulnerability)

Example: Command Injection

Want: Untrusted data does not flow to `exec()`

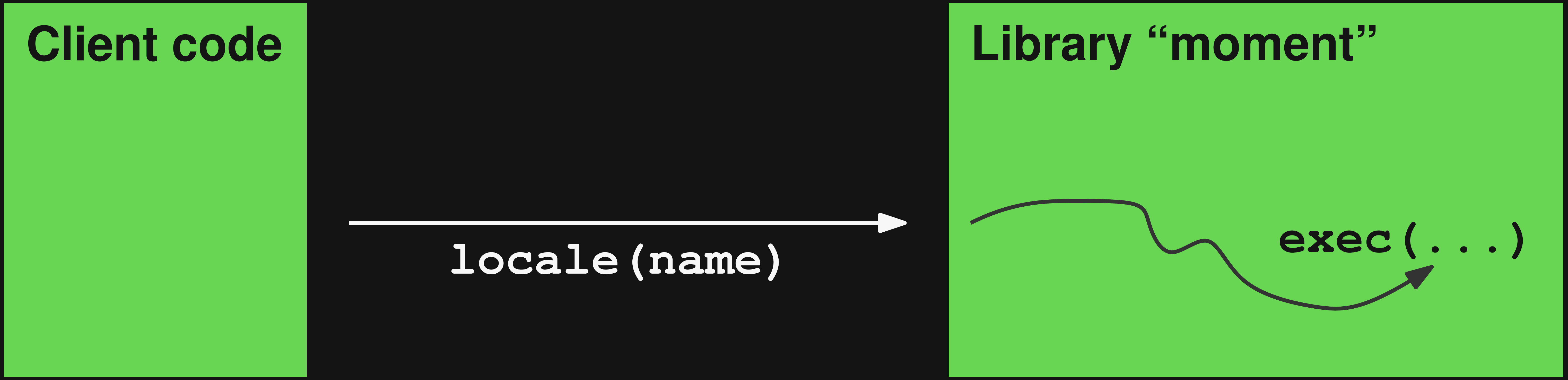
(Otherwise, command injection vulnerability)

Expected → No need to warn developer

Example: Command Injection

Want: **Untrusted data does not flow to `exec()`**

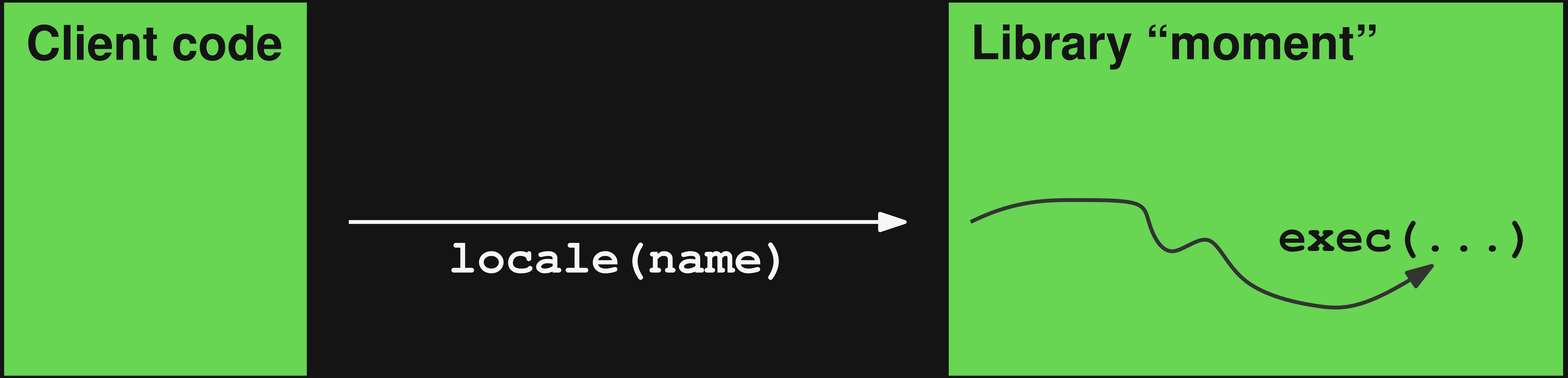
(Otherwise, command injection vulnerability)



Example: Command Injection

Want: **Untrusted data does not flow to `exec()`**

(Otherwise, command injection vulnerability)

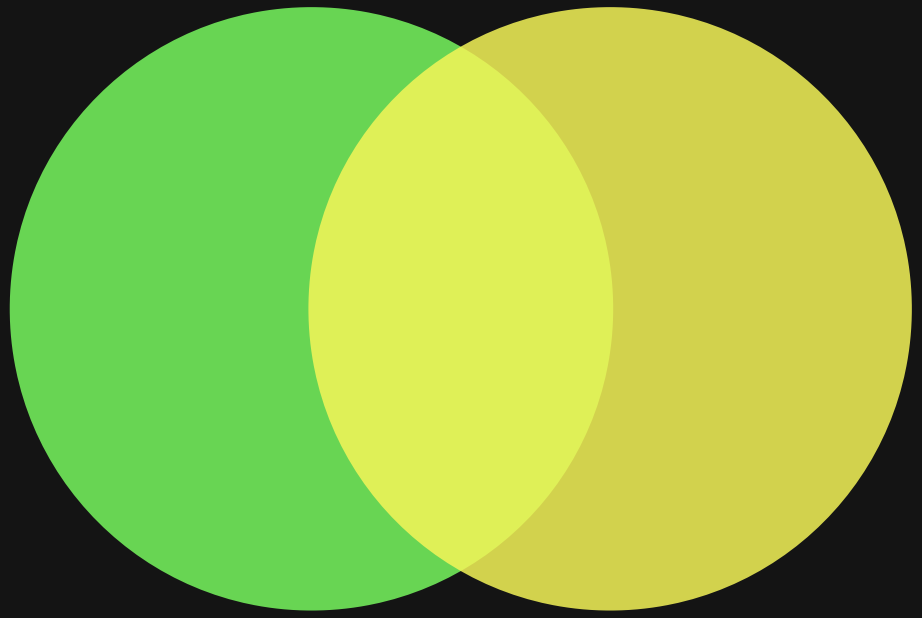


Unexpected → Should warn developer

This Talk

Bimodal program analysis

Program analysis:
Reason about PL semantics

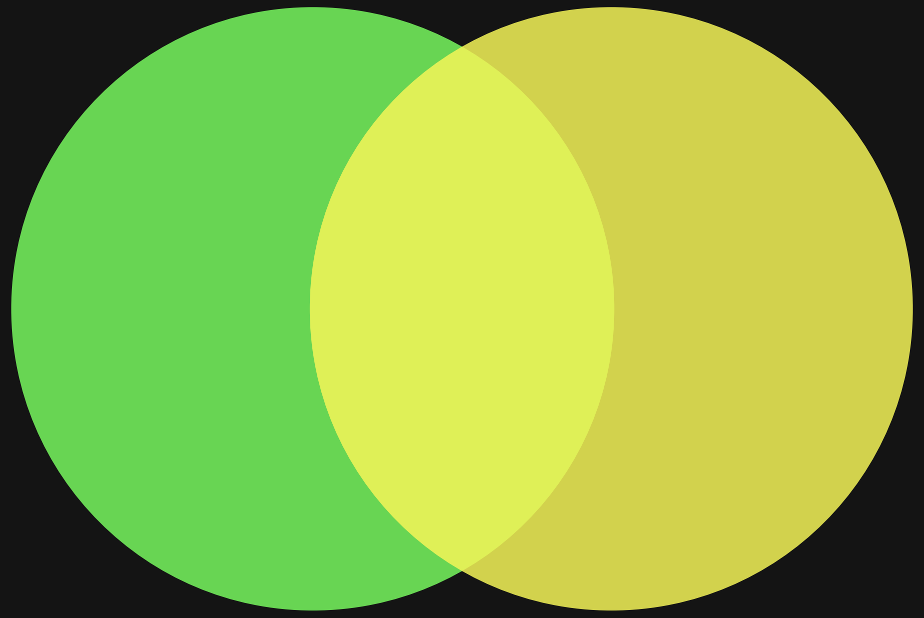


Machine learning:
Reason about NL embedded in code

This Talk

Bimodal program analysis

Program analysis:
Reason about PL semantics



Machine learning:
Reason about NL embedded in code

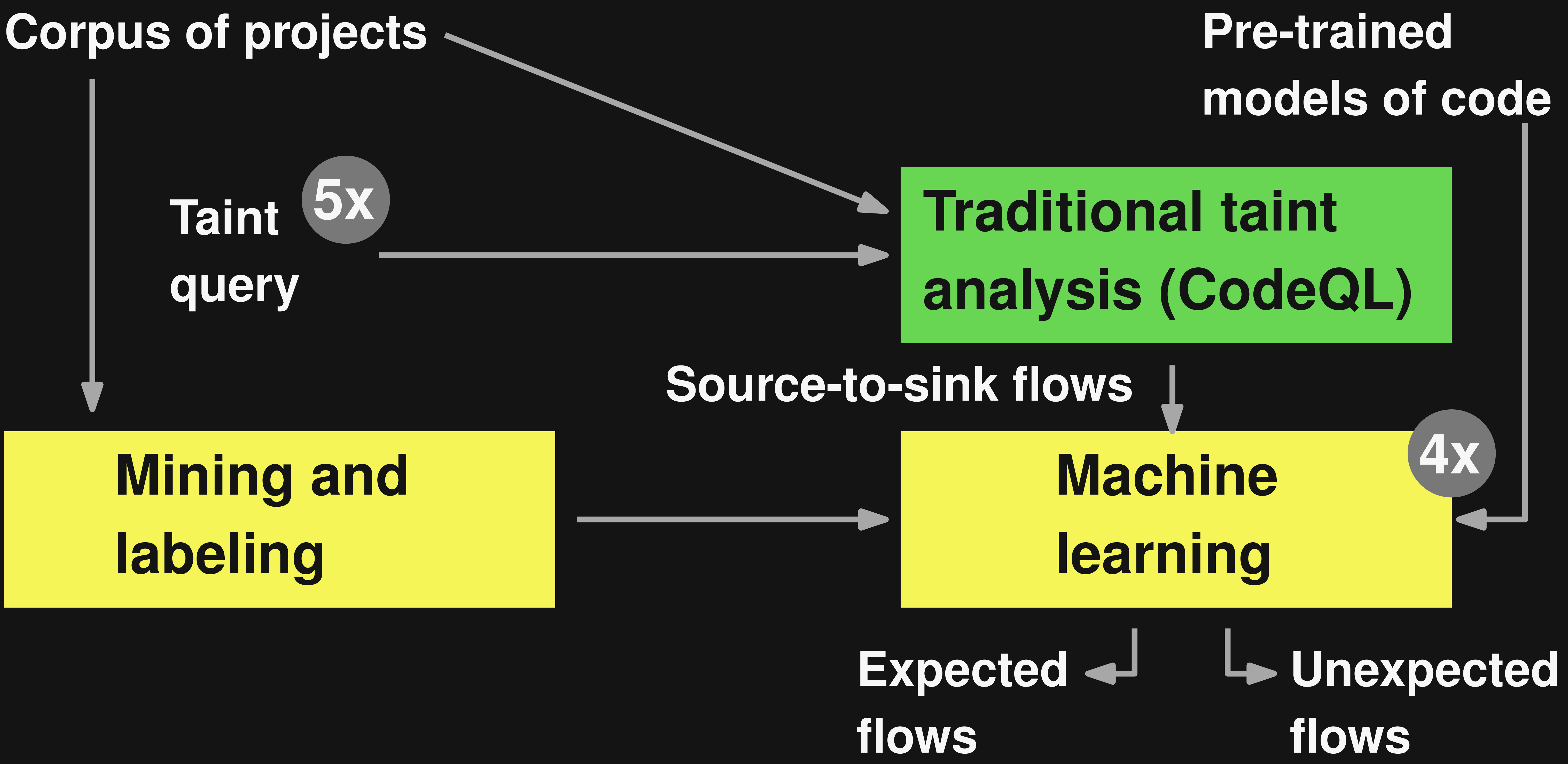
Overapproximate relevant flows

Taint analysis

Identify unexpected flows

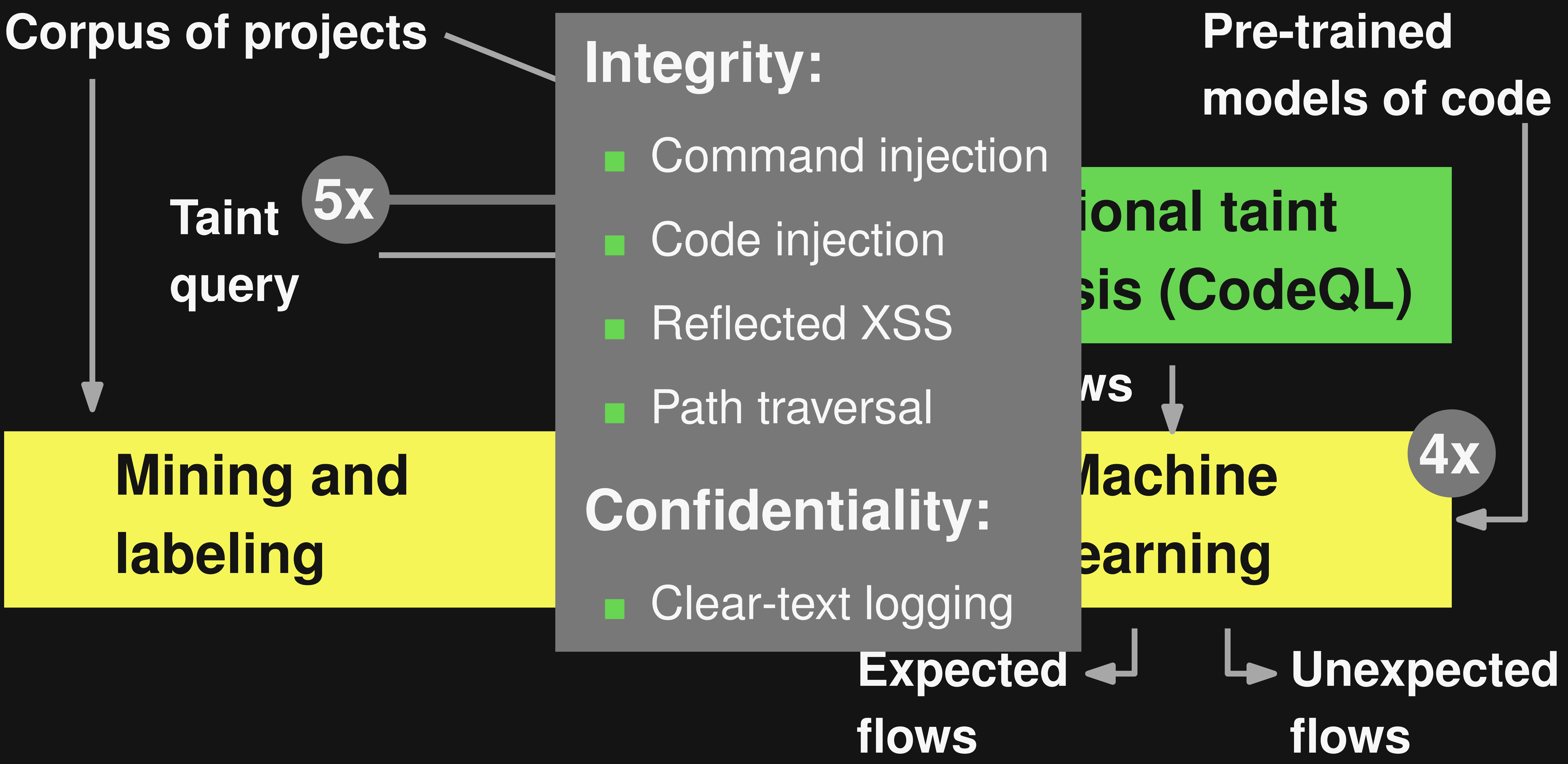
Fluffy = Flagging unexpected flows for better security

Overview

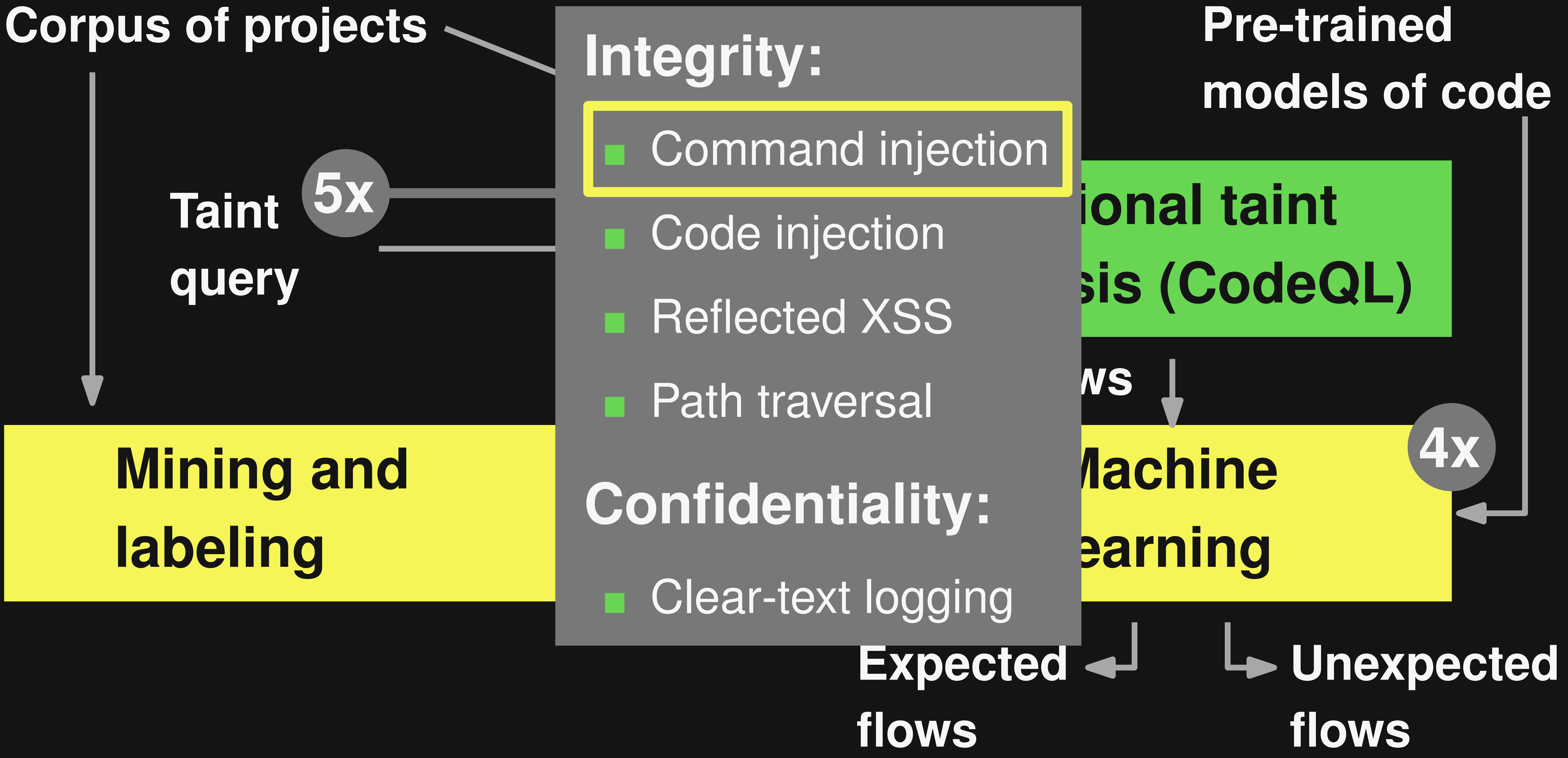


Corpus of projects
Pre-trained
models of code
Taint
5x
query
Traditional taint
analysis (CodeQL)
Source-to-sink flows
Mining and
labeling
Machine
learning
4x
Expected
flows
Unexpected
flows

Overview

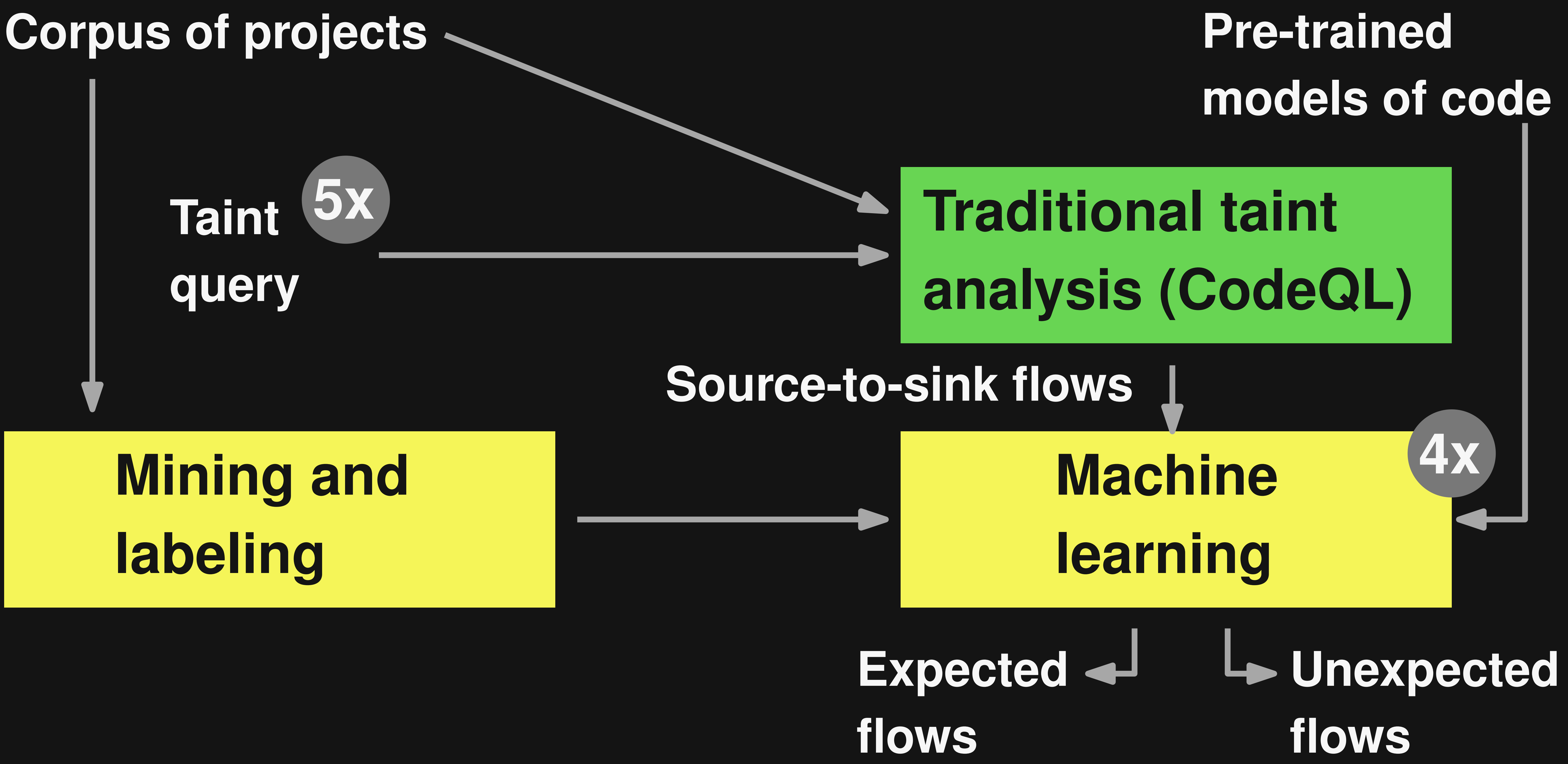

Corpus of projects
Pre-trained models of code
Taint query
5x
Integrity:
Command injection
Code injection
Reflected XSS
Path traversal
Confidentiality:
Clear-text logging
ional taint
sis (CodeQL)
ws
Mining and labeling
Machine
earning
4x
Expected flows
Unexpected flows

Overview



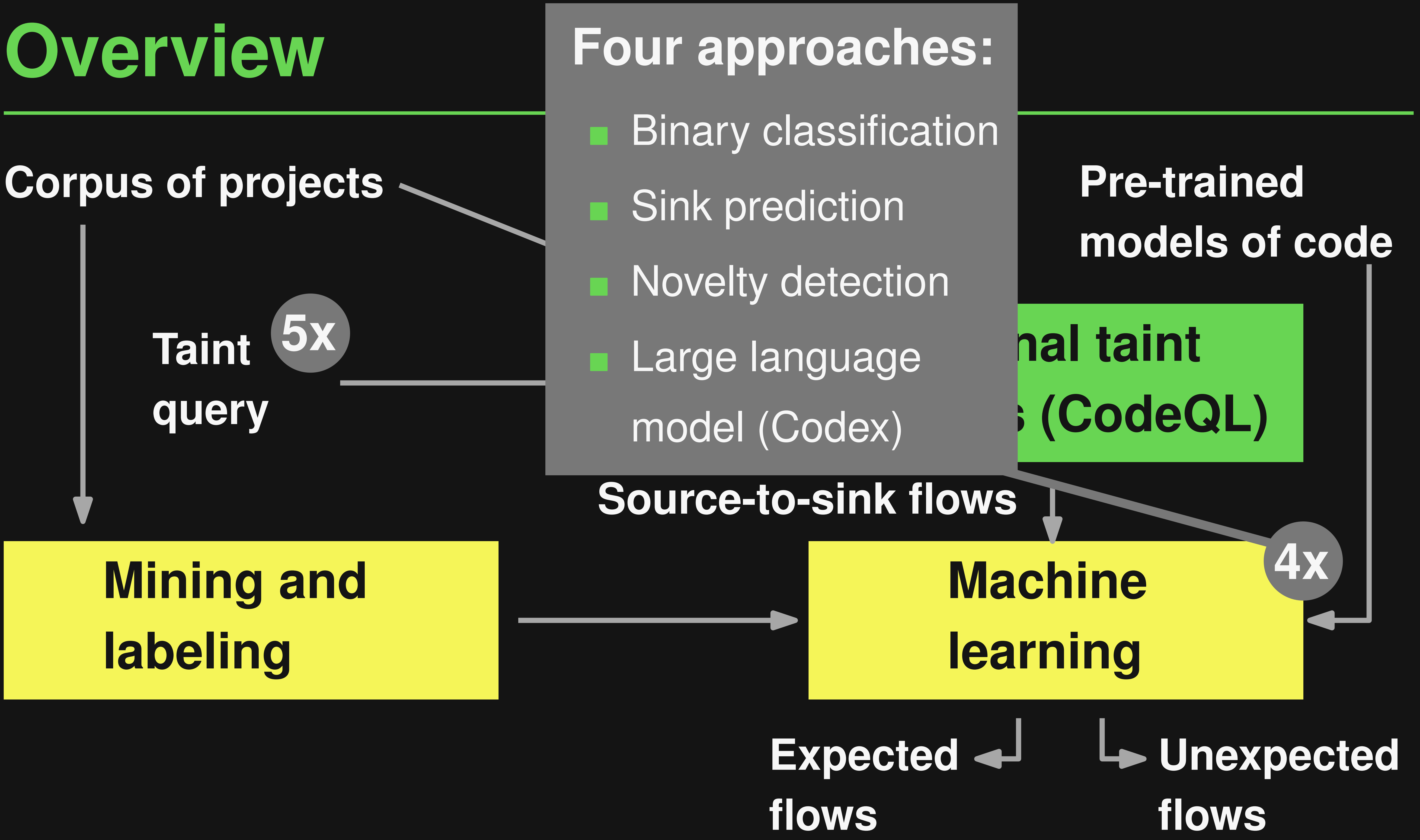
Corpus of projects
Pre-trained models of code
Taint query
5x
Integrity:
Command injection
Code injection
Reflected XSS
Path traversal
Confidentiality:
Clear-text logging
ional taint
sis (CodeQL)
ws
Mining and labeling
achine
earning
4x
Expected flows
Unexpected flows

Overview

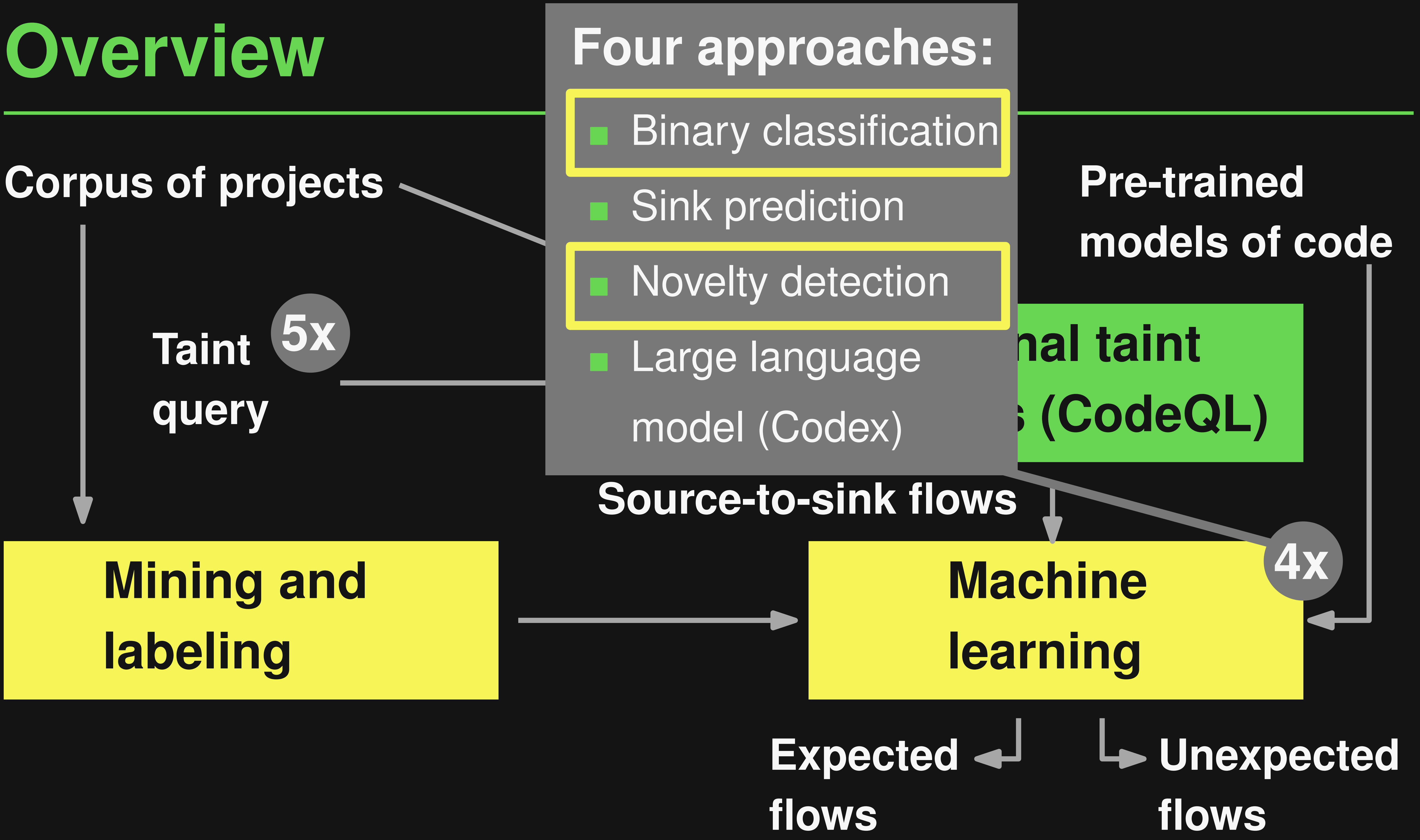


Corpus of projects
Pre-trained models of code
Taint query
5x
Traditional taint analysis (CodeQL)
Source-to-sink flows
Mining and labeling
Machine learning
4x
Expected flows
Unexpected flows

Overview


Four approaches:
Binary classification
Sink prediction
Novelty detection
Large language model (Codex)
Corpus of projects
Taint query
5x
Pre-trained models of code
nal taint
s (CodeQL)
Source-to-sink flows
Mining and labeling
Machine learning
4x
Expected flows
Unexpected flows

Overview



Four approaches:
Binary classification
Sink prediction
Novelty detection
Large language model (Codex)
Corpus of projects
Taint query
5x
nal taint
s (CodeQL)
Pre-trained models of code
Source-to-sink flows
Mining and labeling
Machine learning
4x
Expected flows
Unexpected flows

Approach 1: Binary Classification

Goal: **Predict whether a flow is expected**

$$M : N \times N_{fct} \times D \rightarrow \{Expected, Unexpected\}$$

Approach 1: Binary Classification

Goal: **Predict whether a flow is expected**

$$M : N \times N_{fct} \times D \rightarrow \{Expected, Unexpected\}$$

Name of the source (e.g., parameter)

Name of the API function

Documentation of the API function

Approach 1: Binary Classification

Goal: **Predict whether a flow is expected**

$$M : N \times N_{fct} \times D \rightarrow \{Expected, Unexpected\}$$

Model:

- Bi-directional RNN with LSTMs
- Input tokens embedded with pre-trained model
- Training data: 1,398 manually labeled examples (total across five taint queries)

Approach 3: Novelty Detection

- Goal: Predict whether a source/sink is unusual
- One-class support vector machine applied to embedded names of source/sink

Approach 3: Novelty Detection

- Goal: Predict whether a source/sink is unusual
- One-class support vector machine applied to embedded names of source/sink

Sink type	Seed names
Integrity (names expected to flow to sink):	
Command injection	`execute`, `command`
Code injection	`eval`, `execute`, `compile`, `render`, `callback`, `function`, `fn`
Reflected XSS	`sent`, `content`
Path traversal	`file`, `directory`, `path`, `cwd`, `source`, `input`
Confidentiality (names not *expected to flow to sink):*	
Clear-text logging	`authkey`, `password`, `passcode`, `passphrase`

Evaluation

- **Datasets**
  - 250k JavaScript projects → 7.5M taint flows
  - SecBench.js [ICSE'23] → 131 known vulnerabilities
- **Baselines**
  - Simple, frequency-based approach
  - Regular expressions

Evaluation

- **Datasets**
  - 250k JavaScript projects $\rightarrow$ 7.5M taint flows
  - SecBench.js [ICSE'23] $\rightarrow$ 131 known vulnerabilities
- **Baselines**
  - Simple, frequency-based approach
  - Regular expressions

- 1,398 manually labeled flows
- Validated by four external experts ($\alpha = 0.74$)

Ground truth

Effectiveness

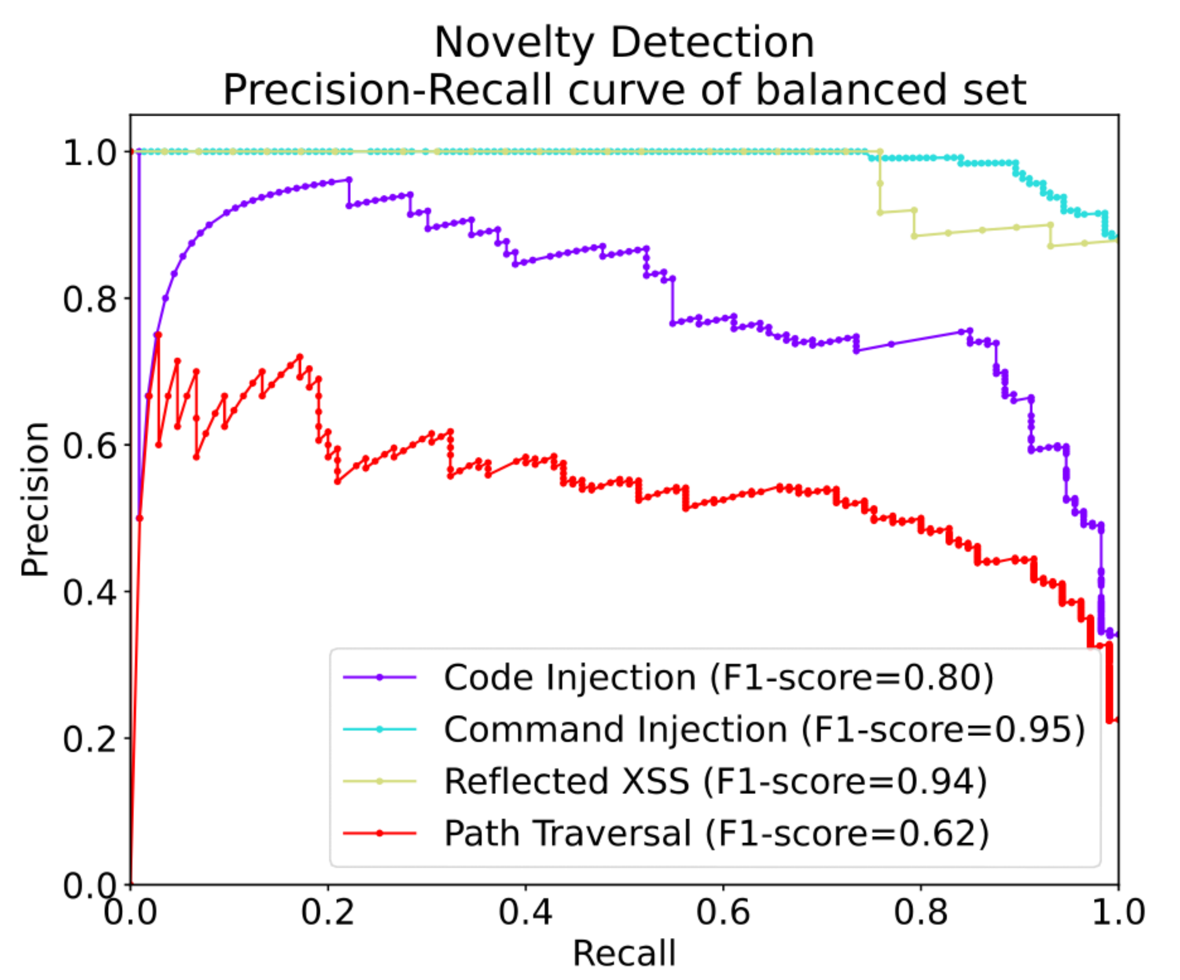
How effective is Fluffy at identifying unexpected flows?

- 81%–97% precision and 80%–100% recall
- 117/131 known vulnerabilities found

Effectiveness

How effective is Fluffy at identifying unexpected flows?

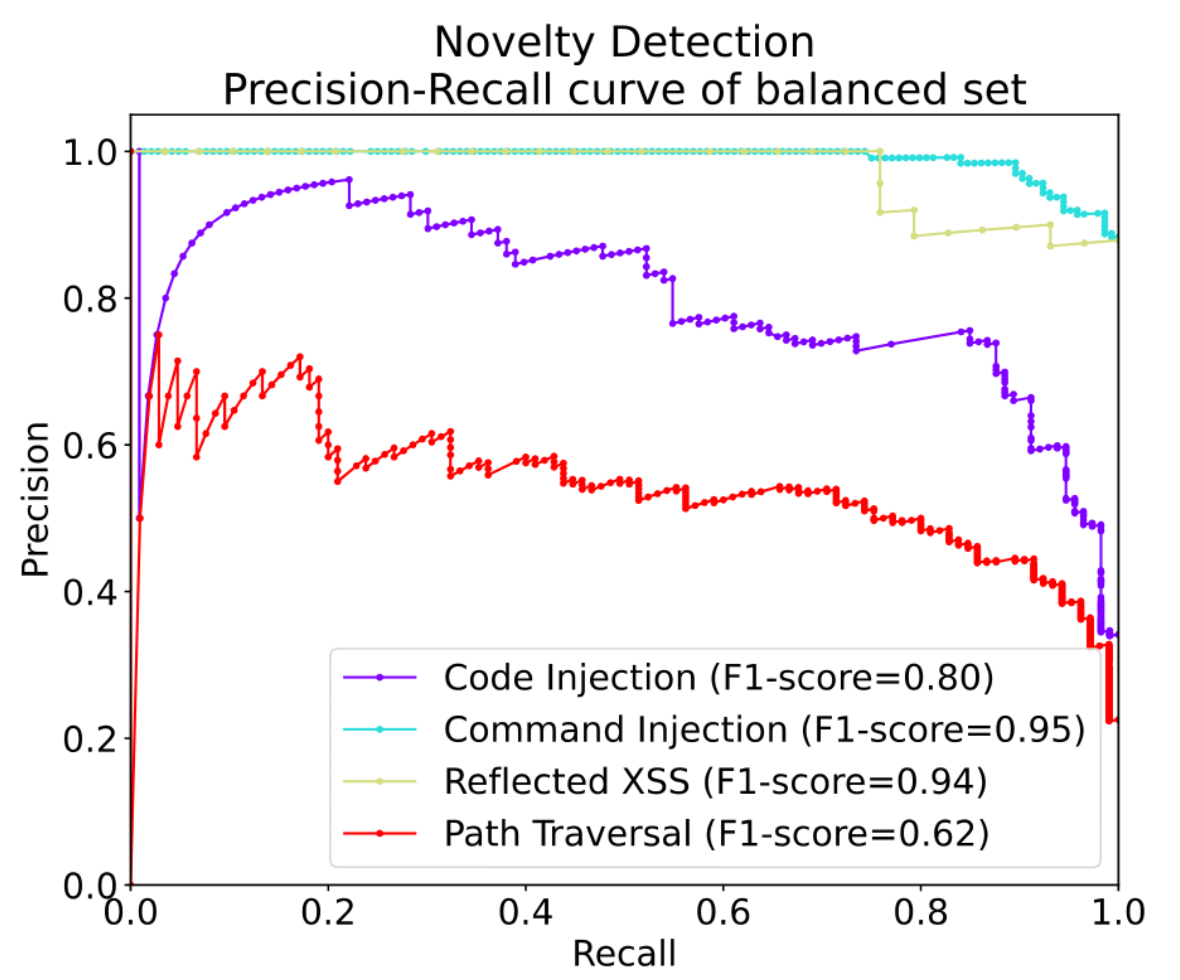
- 81%–97% precision and 80%–100% recall
- 117/131 known vulnerabilities found



Effectiveness

How effective is Fluffy at identifying unexpected flows?

- 81%–97% precision and 80%–100% recall
- 117/131 known vulnerabilities found

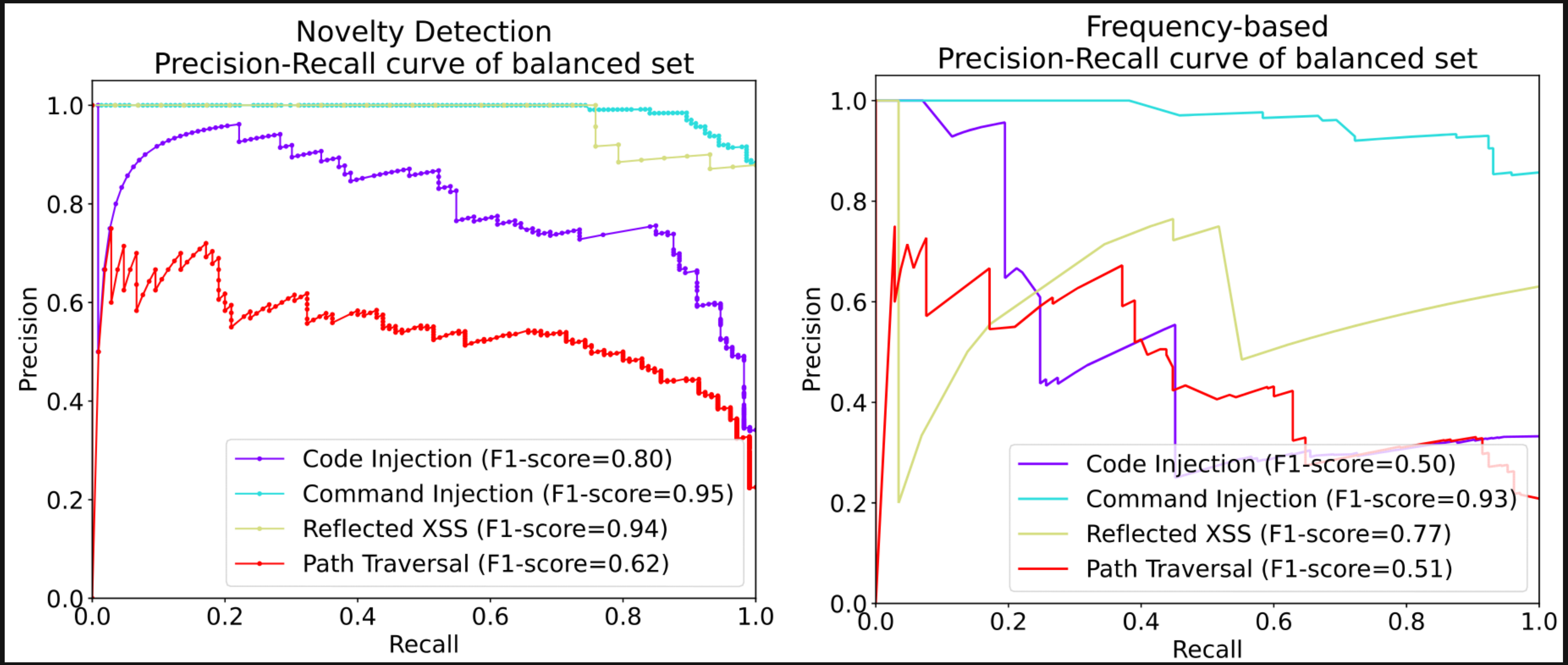


Effectiveness varies depending on taint query and ML model

Effectiveness

How effective is Fluffy at identifying unexpected flows?

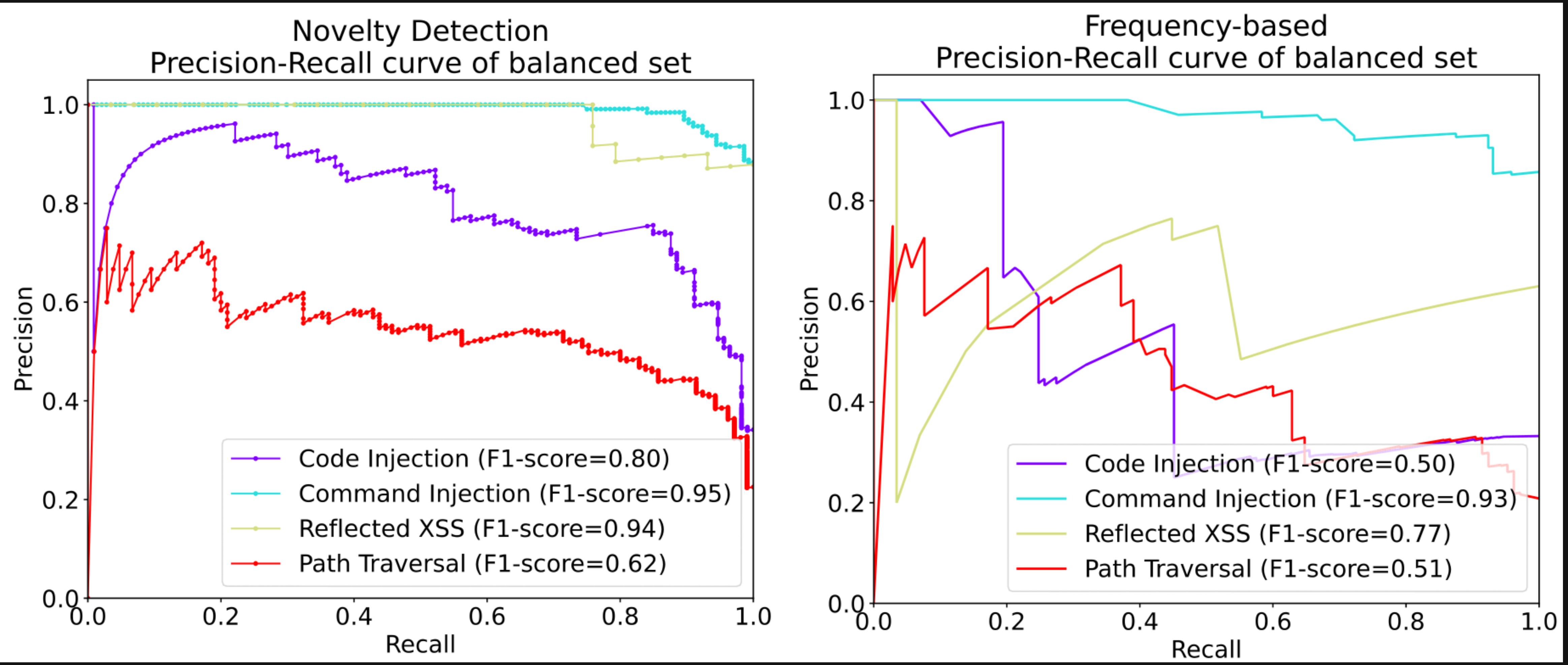
- 81%–97% precision and 80%–100% recall
- 117/131 known vulnerabilities found



Effectiveness

How effective is Fluffy at identifying unexpected flows?

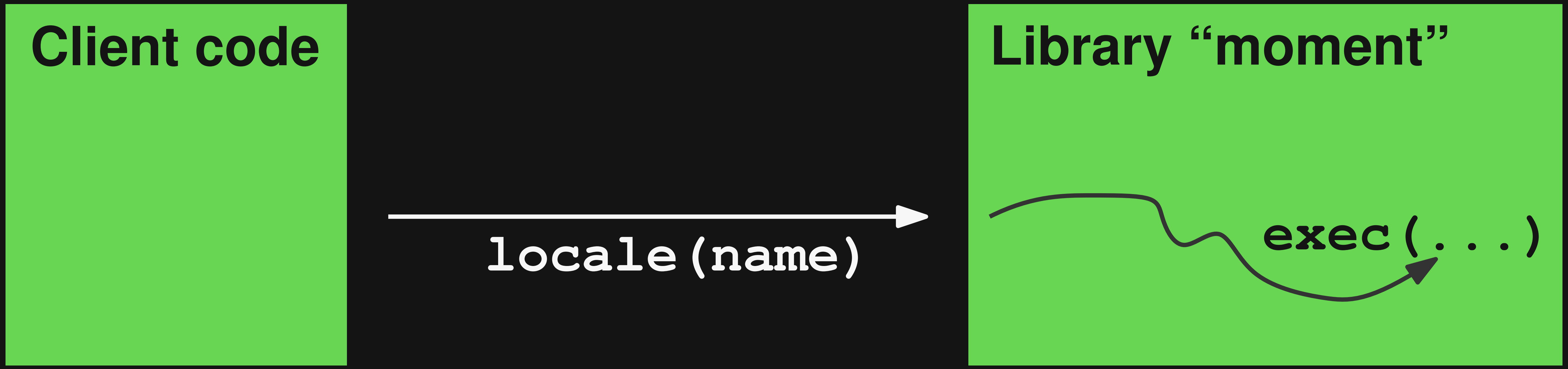
- 81%–97% precision and 80%–100% recall
- 117/131 known vulnerabilities found



Fluffy outperforms the baseline

Real-World Vulnerabilities

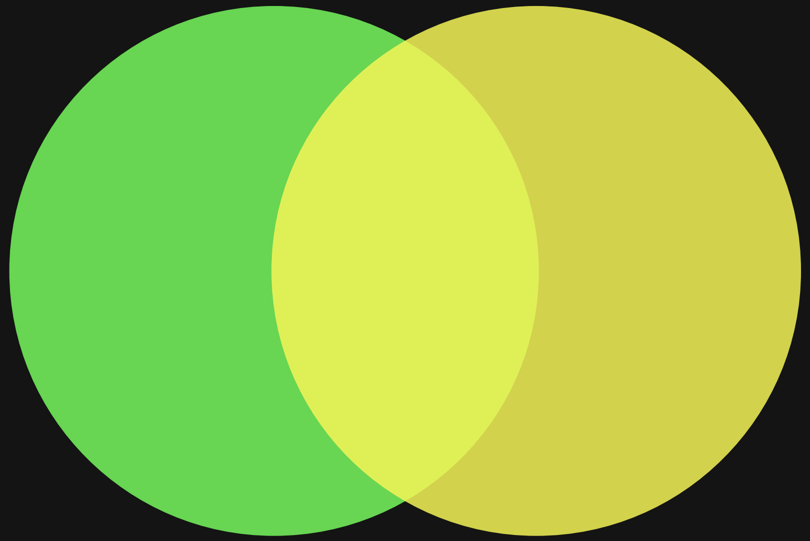
- **Found and reported 17 previously unknown vulnerabilities**
  - 10/17 confirmed and fixed so far
- **Example: CVE-2022-24785 in Moment.js**



Key Take-Aways

- **Bimodal program analysis**

Program analysis: Reason about PL semantics



Machine learning: Reason about NL embedded in code

- **Concrete application: Detecting unexpected taint flows**
  - Five kinds of vulnerabilities, four machine learning models
  - 81%–97% precision, 80%–100% recall
  - https://github.com/sola-st/fluffy