Scalable Taint Specification Inference with Big Code

Victor Chibotaru
DeepCode

Benjamin Bichsel
ETH Zurich

Veselin Raychev
DeepCode

Martin Vechev
ETH Zurich

OWASP top 10 security threats to web apps

1. Injection (SQL, NoSQL, OS command, Code, …)
2. Broken Authentication
3. Sensitive Data Exposure
4. XML External Entities (XXE)
5. Broken Access Control
6. Security Misconfiguration
7. Cross-Site Scripting (XSS)
8. Insecure Deserialization
9. Using Components with Known Vulnerabilities
10. Insufficient Logging & Monitoring

Injection vulnerabilities

1. Injection (SQL, NoSQL, OS command, Code, …)
2. Broken Authentication
3. Sensitive Data Exposure
4. XML External Entities (XXE)
5. Broken Access Control
6. Security Misconfiguration
7. Cross-Site Scripting (XSS)
8. Insecure Deserialization
9. Using Components with Known Vulnerabilities
10. Insufficient Logging & Monitoring

A typical injection vulnerability

```python
def upload():
```

A typical injection vulnerability

```
def upload():
  fname = flask.request.files['f'].filename
```

A typical injection vulnerability

```
def upload():
  fname = flask.request.files['f'].filename
  path = os.path.join(upload_dir, fname)
```

A typical injection vulnerability

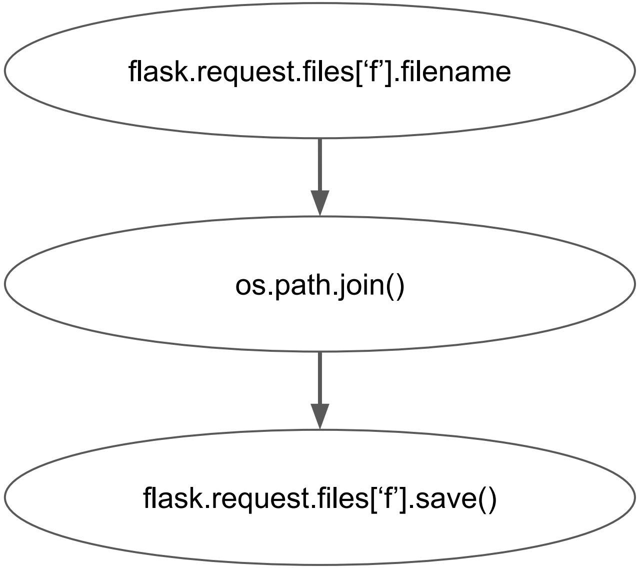
```
def upload():
  fname = flask.request.files['f'].filename
  path = os.path.join(upload_dir, fname)
  flask.request.files['f'].save(path)
```

A typical injection exploit

```
def upload():
  fname = "../../../etc/passwd"
  path = "/var/www/app/../../../etc/passwd"
  flask.request.files['f'].save("/etc/passwd")
```

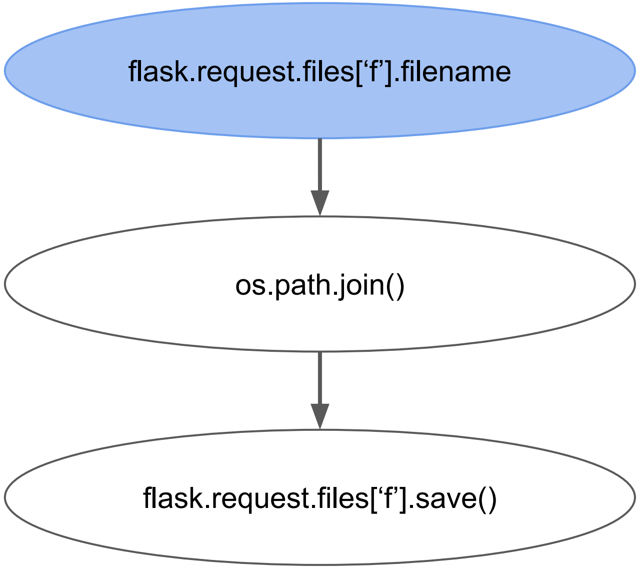
Taint Analysis: detecting injection vulnerabilities

```
def upload():
  fname = flask.request.files['f'].filename
  path = os.path.join(upload_dir, fname)
  flask.request.files['f'].save(path)
```


flask.request.files[‘f’].filename
os.path.join()
flask.request.files[‘f’].save()

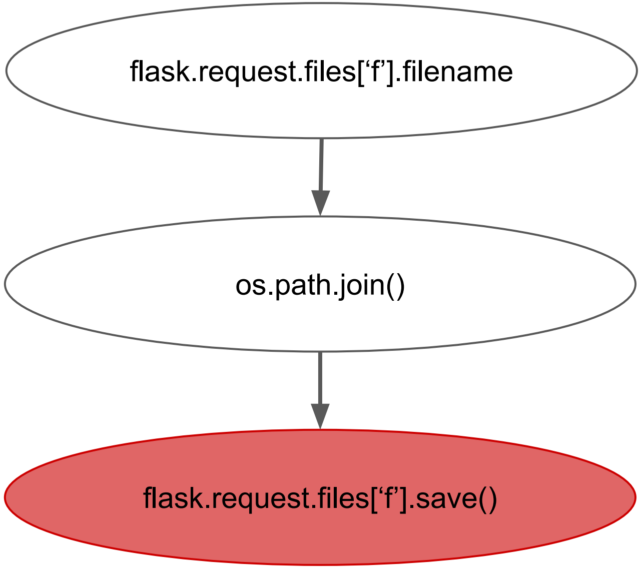
Taint Analysis: sources

```
def upload():
  fname = flask.request.files['f'].filename
  path = os.path.join(upload_dir, fname)
  flask.request.files['f'].save(path)
```


flask.request.files[‘f’].filename
os.path.join()
flask.request.files[‘f’].save()

Taint Analysis: sinks

```
def upload():
  fname = flask.request.files['f'].filename
  path = os.path.join(upload_dir, fname)
  flask.request.files['f'].save(path)
```



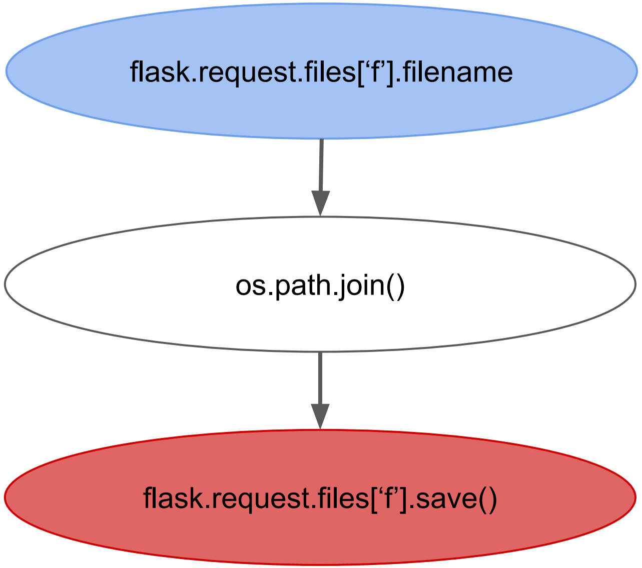
flask.request.files[‘f’].filename

os.path.join()

flask.request.files[‘f’].save()

Taint Analysis: vulnerability model

```
def upload():
  fname = flask.request.files['f'].filename
  path = os.path.join(upload_dir, fname)
  flask.request.files['f'].save(path)
```



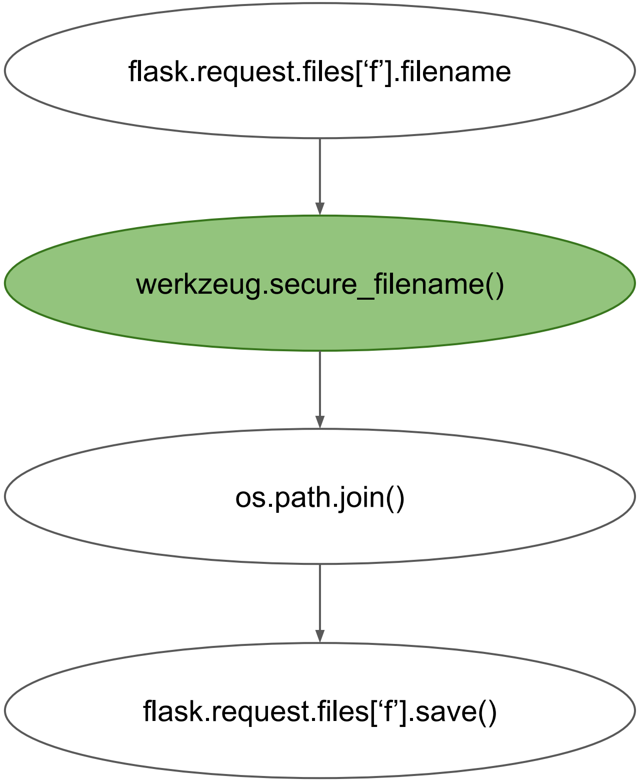
flask.request.files[‘f’].filename
os.path.join()
flask.request.files[‘f’].save()

A typical fix

```
def upload():
  fname = flask.request.files['f'].filename
  fname = werkzeug.secure_filename(fname)
  path = os.path.join(upload_dir, fname)
  flask.request.files['f'].save(path)
```

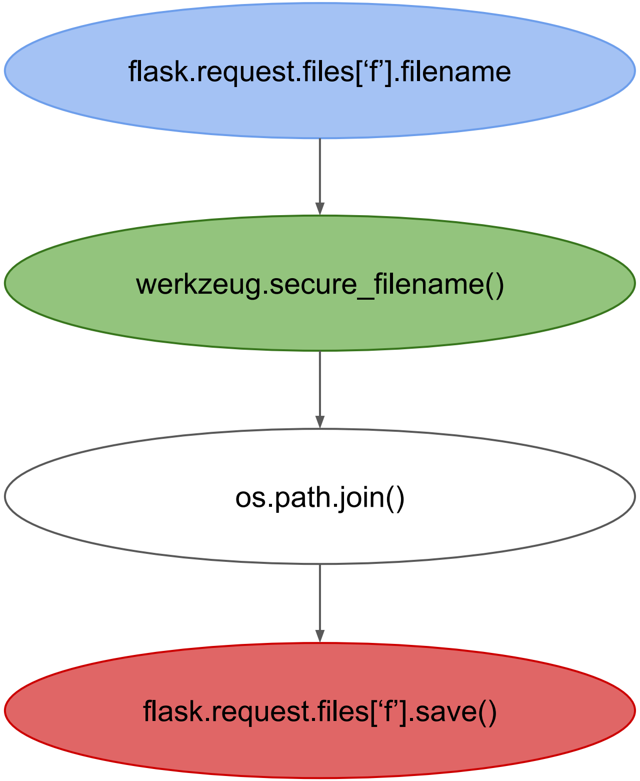
Taint Analysis: sanitizers

```
def upload():
  fname = flask.request.files['f'].filename
  fname = werkzeug.secure_filename(fname)
  path = os.path.join(upload_dir, fname)
  flask.request.files['f'].save(path)
```


flask.request.files[‘f’].filename
werkzeug.secure_filename()
os.path.join()
flask.request.files[‘f’].save()

Taint Analysis: vulnerability model

```python
def upload():
    fname = flask.request.files['f'].filename
    fname = werkzeug.secure_filename(fname)
    path = os.path.join(upload_dir, fname)
    flask.request.files['f'].save(path)
```


flask.request.files[‘f’].filename
werkzeug.secure_filename()
os.path.join()
flask.request.files[‘f’].save()

Completeness of taint specifications is crucial

Missing source or sink → undetected vulnerabilities

Missing sanitizer → false positive reports

Creating taint specifications is labour-intensive

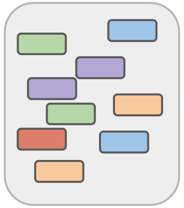
Creating taint specifications is labour-intensive

Express JS
Bottle
STRUTS
JavaServer™ Faces
JSF
Flask
django
node JS
GWT
spring

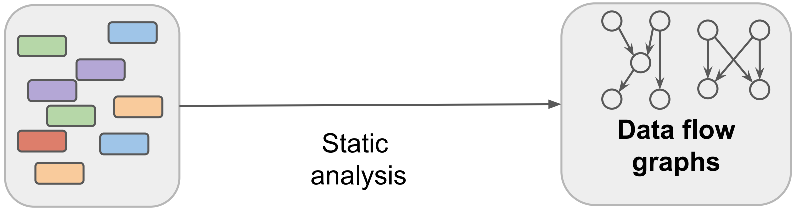
Creating taint specifications is labour-intensive

Goal: Automatically learn Taint Specifications from Big Code

Learning Taint Specifications from Big Code



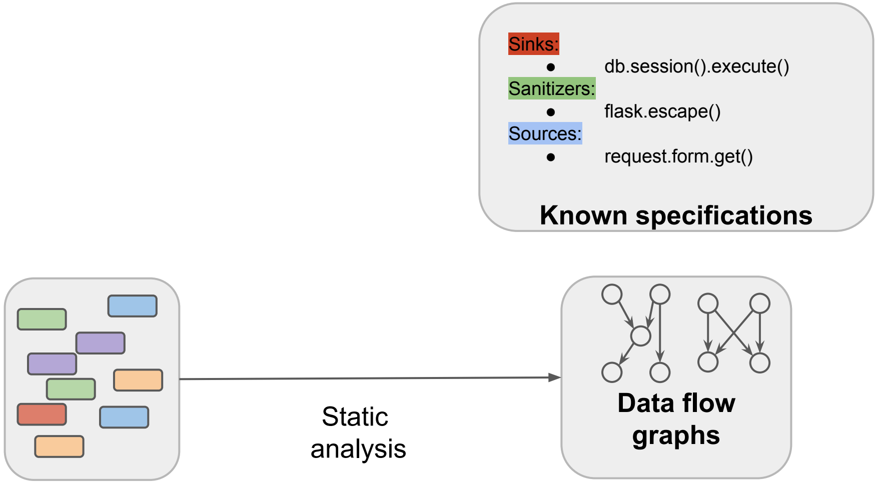
Learning Taint Specifications from Big Code



Static analysis

Data flow graphs

Learning Taint Specifications from Big Code



Sinks:

- db.session().execute()

Sanitizers:

- flask.escape()

Sources:

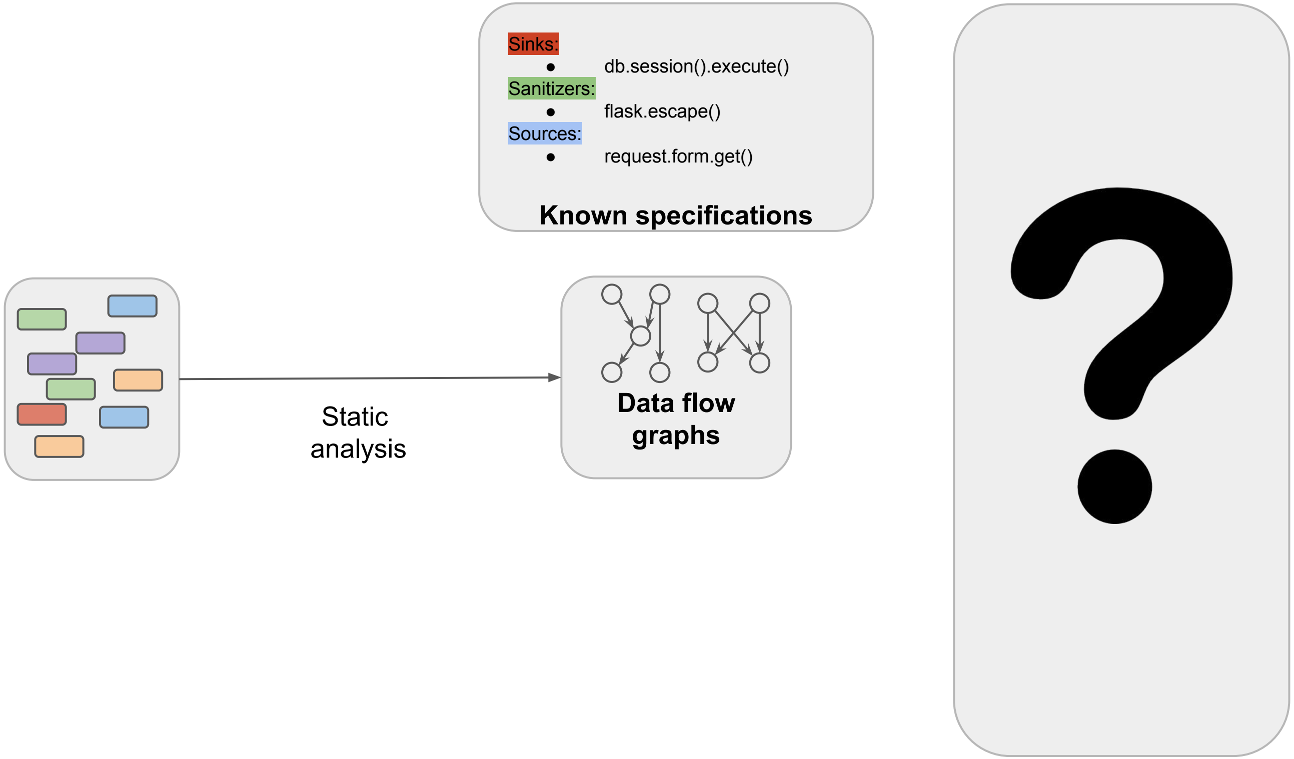
- request.form.get()

Known specifications

Static analysis

Data flow graphs

Learning Taint Specifications from Big Code



Sinks:

- db.session().execute()

Sanitizers:

- flask.escape()

Sources:

- request.form.get()

Known specifications

Static analysis

Data flow graphs

?

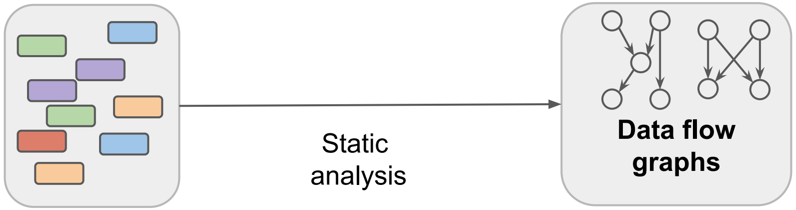
Learning Taint Specifications from Big Code

Sinks:
- db.session().execute()

Sanitizers:
- flask.escape()

Sources:
- request.form.get()

Known specifications



?

Sinks:
- flask.Markup()
- django.HttpResponse()
- pymysql.connect().execute()
- …

Sanitizers:
- cgi.escape()
- werkzeug.secure_filename()
- MySQLdb.escape_string()
- …

Sources:
- cherrypy.request.params
- flask.request.args
- request.GET.get()
- …

Inferred Specifications

System requirements

- ➢ The system has to be **fast** enough to learn from Big Code

- ➢ The system has to work with **few** known specifications

?

Merlin (Livshits et al., PLDI ‘09)

- ➢ The system has to be **fast** enough to learn from Big Code

- ➢ The system has to work with **few** known specifications

Merlin (Livshits et al., PLDI ‘09)

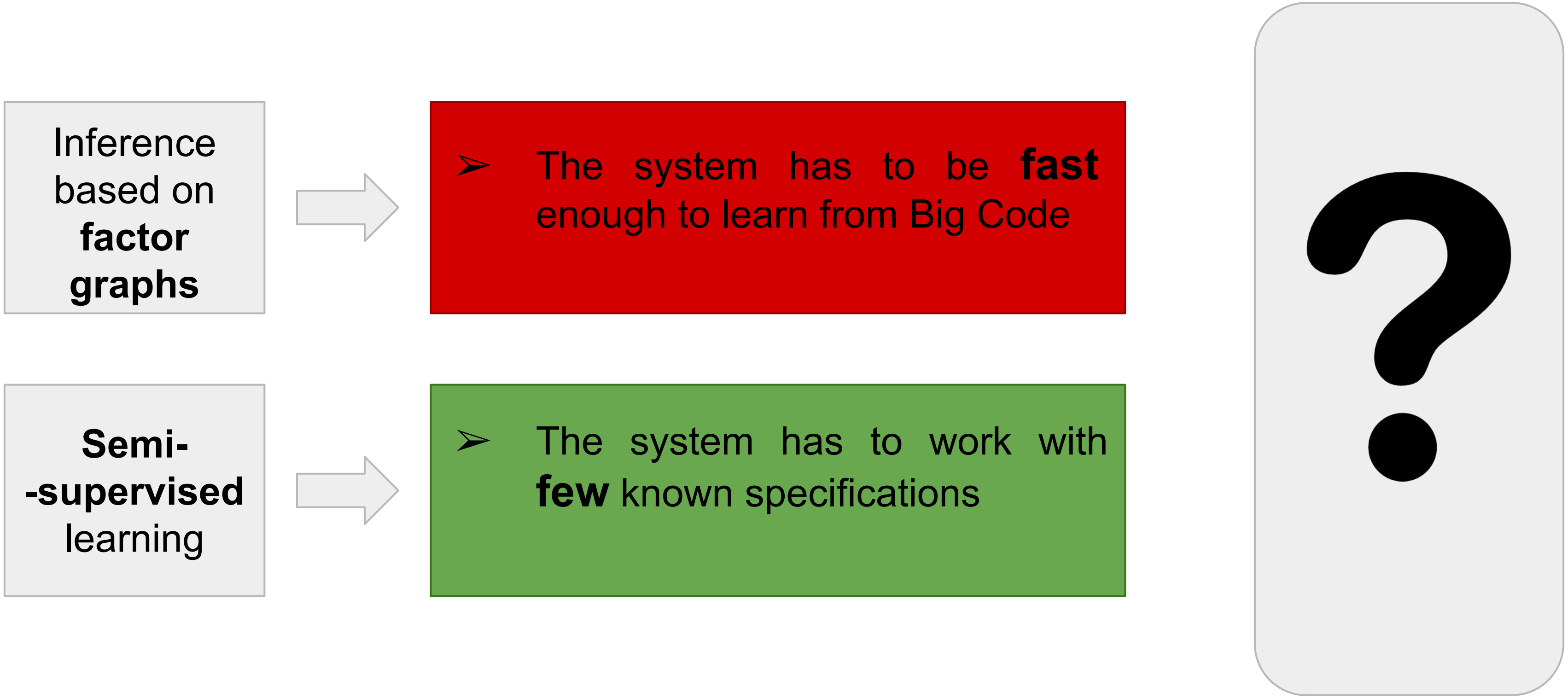
- ➢ The system has to be **fast** enough to learn from Big Code

Semi-
-supervised
learning

- ➢ The system has to work with **few** known specifications

?

Merlin (Livshits et al., PLDI ‘09)



Inference based on **factor graphs**

➢ The system has to be **fast** enough to learn from Big Code

Semi--supervised learning

➢ The system has to work with **few** known specifications

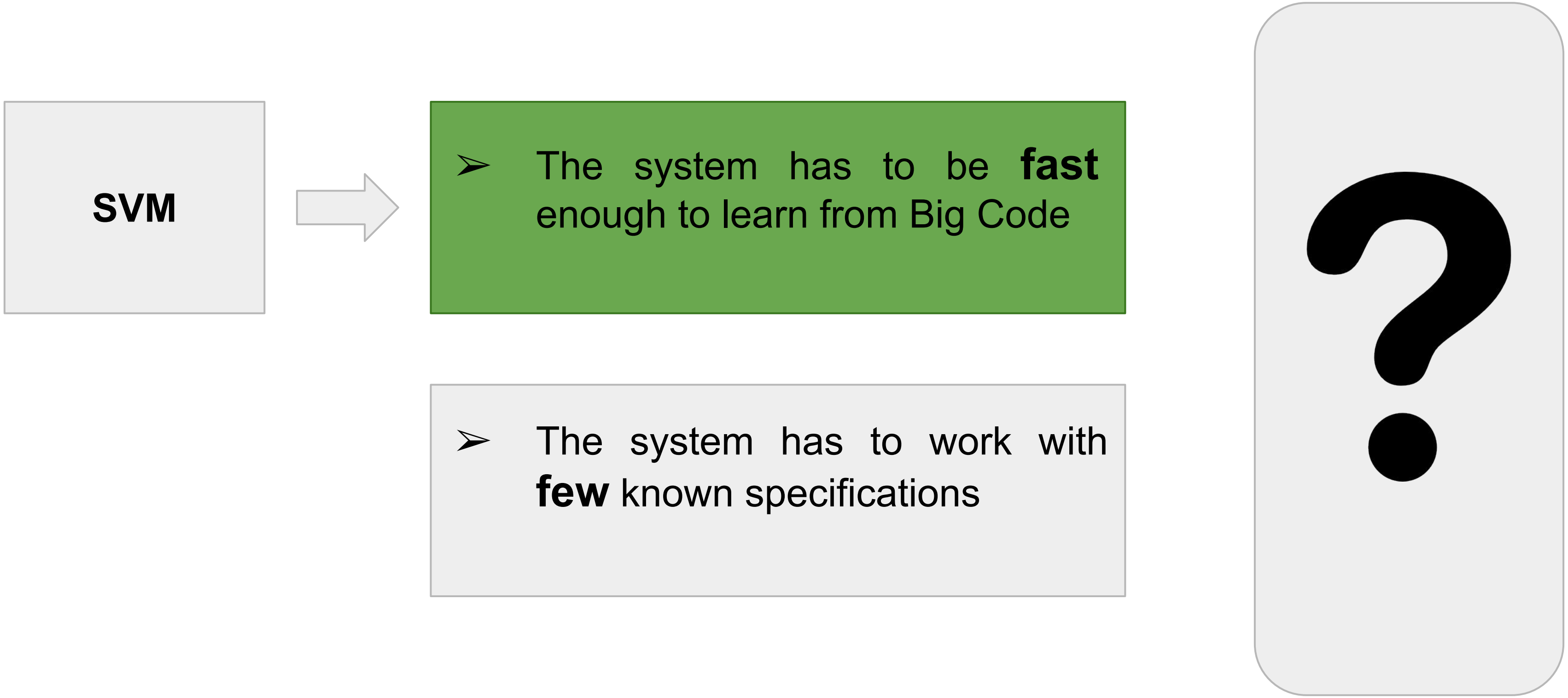
?

SuSi (Rasthofer et al., NDSS Symposium 2014)

➢ The system has to be **fast** enough to learn from Big Code

➢ The system has to work with **few** known specifications

SuSi (Rasthofer et al., NDSS Symposium 2014)

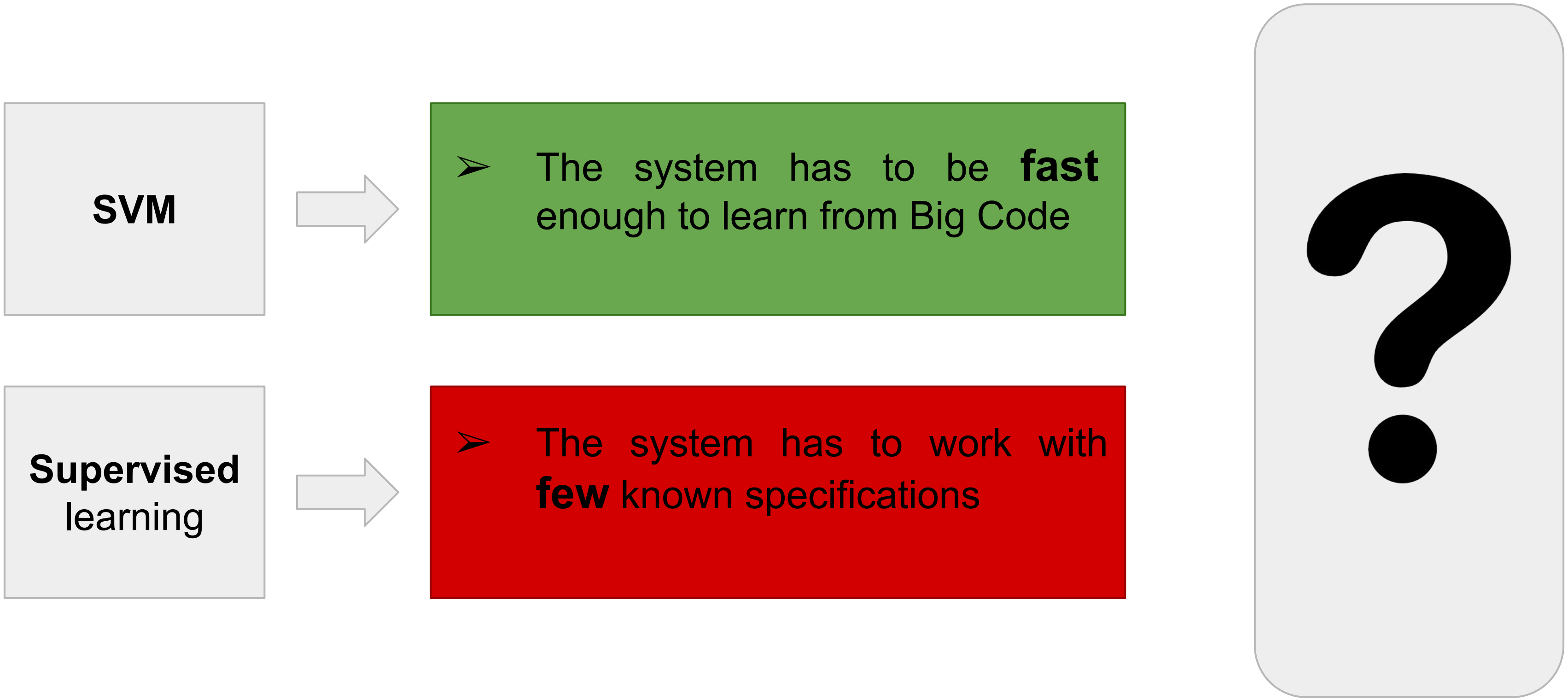


SVM

- ➢ The system has to be **fast** enough to learn from Big Code

- ➢ The system has to work with **few** known specifications

SuSi (Rasthofer et al., NDSS Symposium 2014)



SVM

- The system has to be **fast** enough to learn from Big Code

Supervised learning

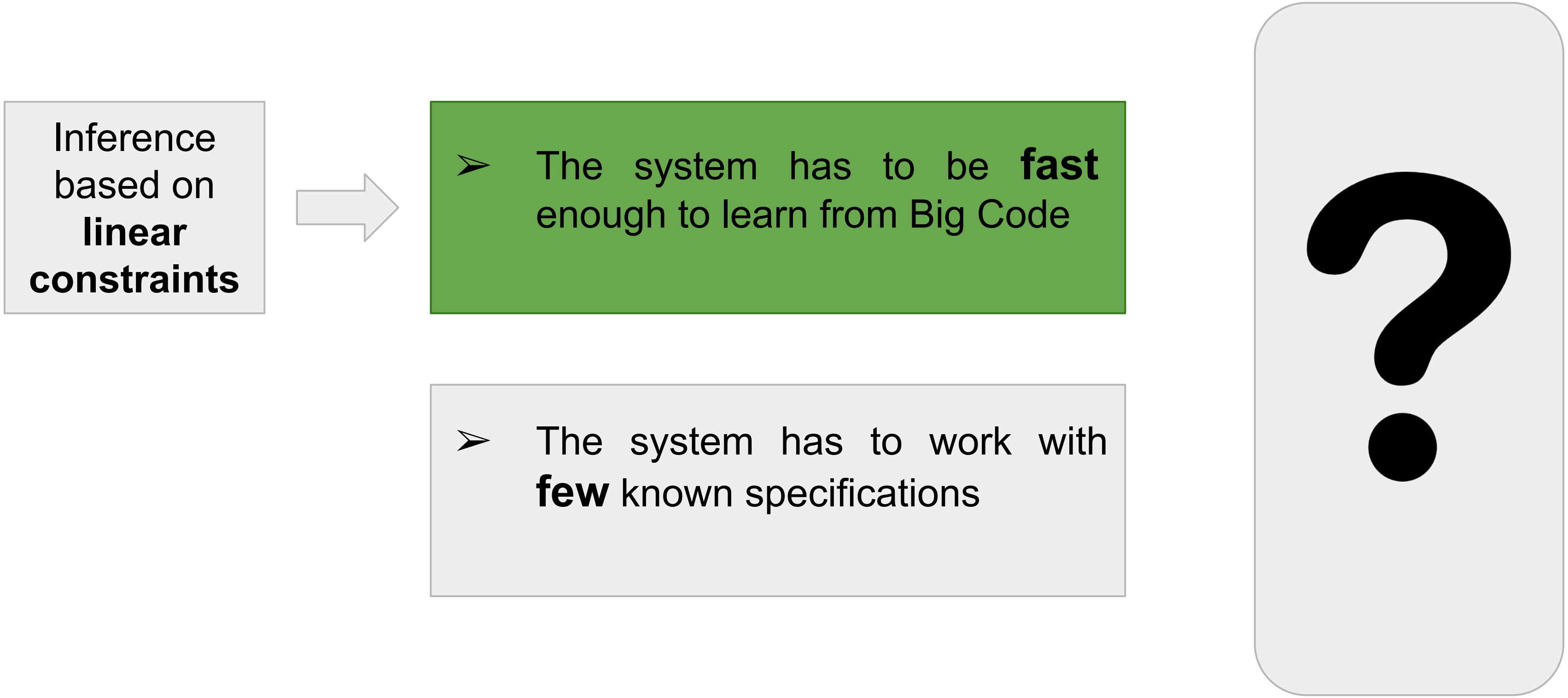
- The system has to work with **few** known specifications

Seldon

- ➢ The system has to be **fast** enough to learn from Big Code

- ➢ The system has to work with **few** known specifications

Seldon

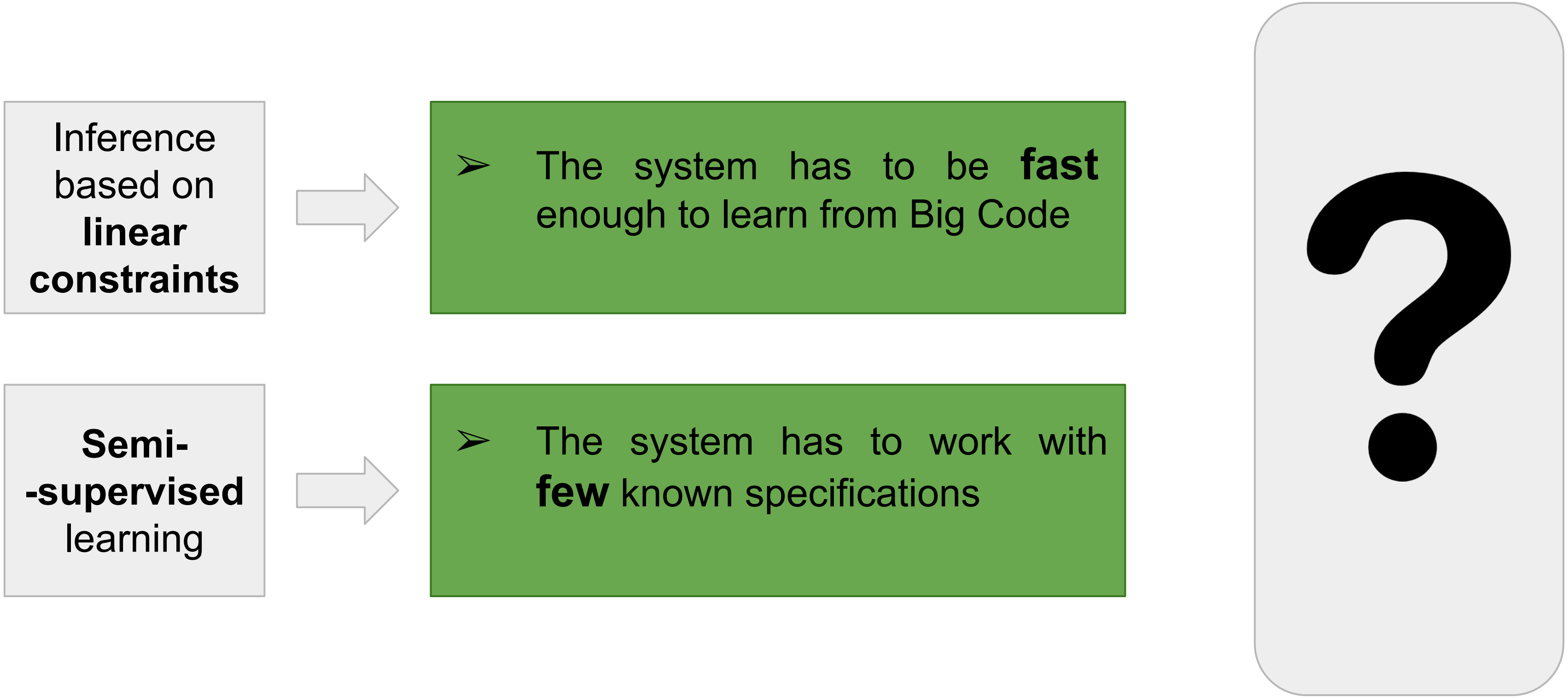


Inference based on **linear constraints**

- ➢ The system has to be **fast** enough to learn from Big Code

- ➢ The system has to work with **few** known specifications

Seldon



Inference based on **linear constraints**

- ➢ The system has to be **fast** enough to learn from Big Code

Semi--supervised learning

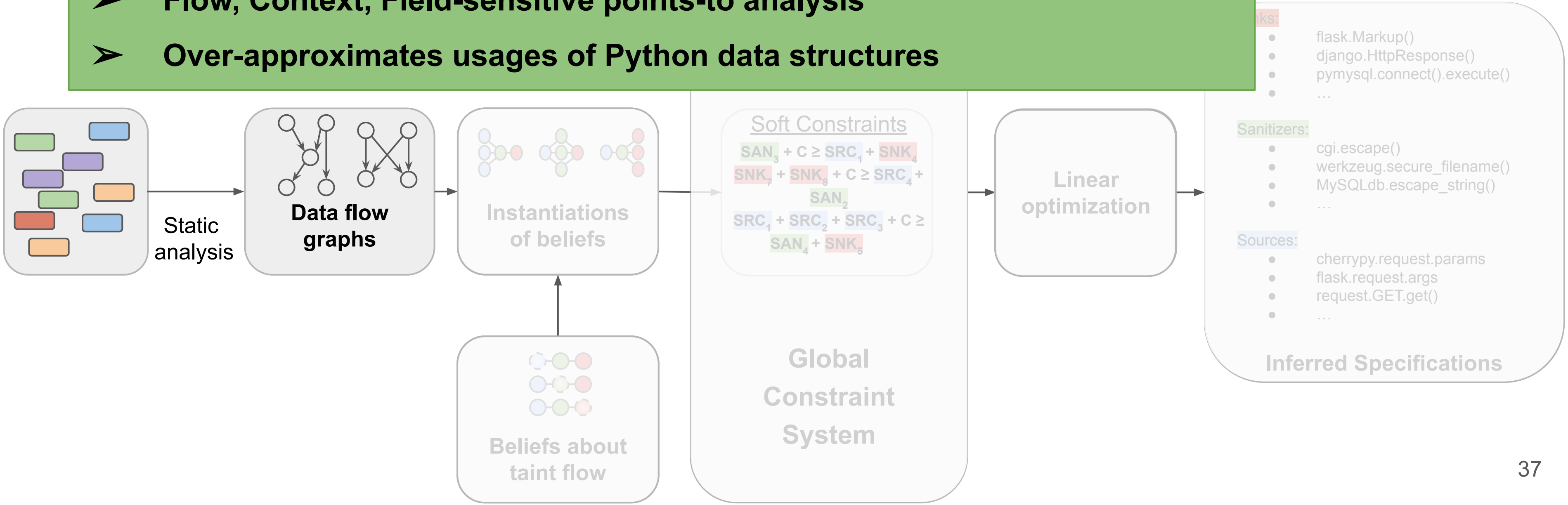
- ➢ The system has to work with **few** known specifications

?

Solution: Seldon

Static Code Analysis

- ➢ Static analysis is used to build data flow graphs for training programs
- ➢ Nodes are events in program (e.g. function calls, parameter loads)
- ➢ Edges represent the data flow between events
- ➢ Flow, Context, Field-sensitive points-to analysis
- ➢ Over-approximates usages of Python data structures



Static analysis

Data flow graphs

Instantiations of beliefs

Beliefs about taint flow

Soft Constraints

$SAN_3 + C \geq SRC_1 + SNK_4$

$SNK_7 + SNK_8 + C \geq SRC_4 + SAN_2$

$SRC_1 + SRC_2 + SRC_3 + C \geq SAN_4 + SNK_5$

Global Constraint System

Linear optimization

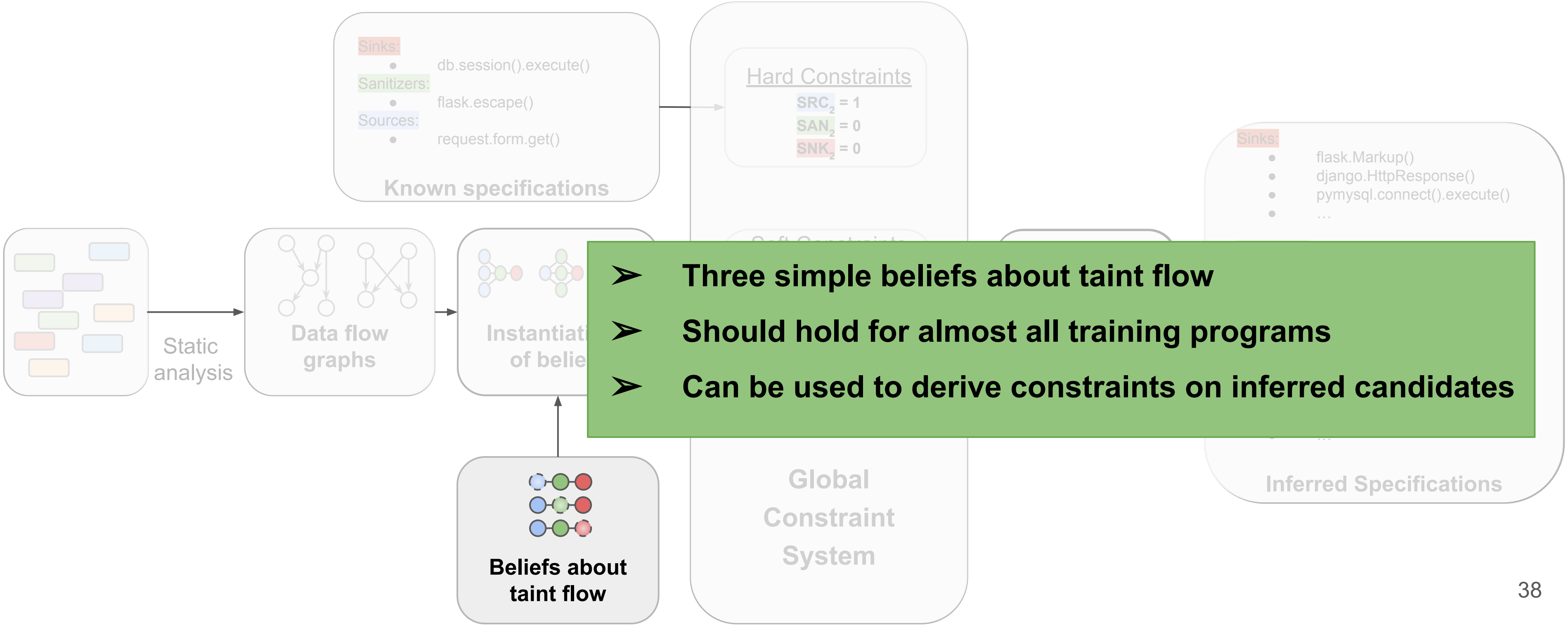
nks:
- flask.Markup()
- django.HttpResponse()
- pymysql.connect().execute()
- …

Sanitizers:
- cgi.escape()
- werkzeug.secure_filename()
- MySQLdb.escape_string()
- …

Sources:
- cherrypy.request.params
- flask.request.args
- request.GET.get()
- …

Inferred Specifications

Beliefs about taint flow



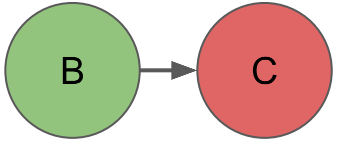
- ➢ Three simple beliefs about taint flow
- ➢ Should hold for almost all training programs
- ➢ Can be used to derive constraints on inferred candidates

Beliefs about taint flow

“Sanitizers secure sinks from untrusted input”

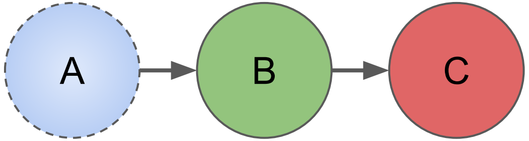
Beliefs about taint flow

“Sanitizers secure sinks from untrusted input”



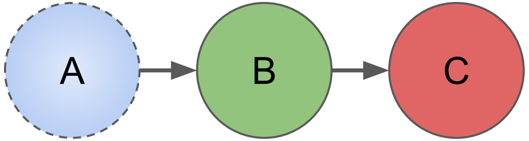
Beliefs about taint flow

“Sanitizers secure sinks from untrusted input”



Beliefs about taint flow

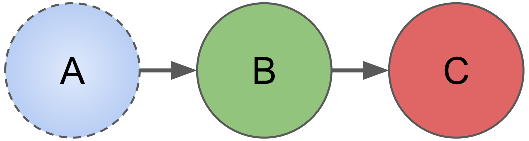
“Sanitizers secure sinks from untrusted input”



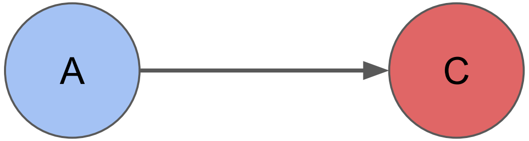
“Vulnerabilities do not occur often”

Beliefs about taint flow

“Sanitizers secure sinks from untrusted input”

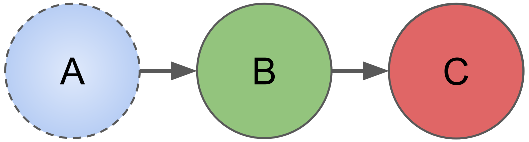


“Vulnerabilities do not occur often”

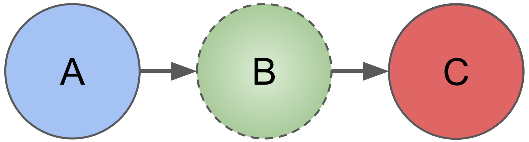


Beliefs about taint flow

“Sanitizers secure sinks from untrusted input”

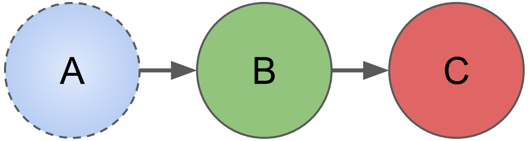


“Vulnerabilities do not occur often”

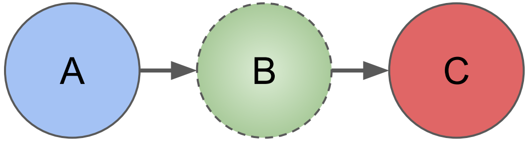


Beliefs about taint flow

“Sanitizers secure sinks from untrusted input”



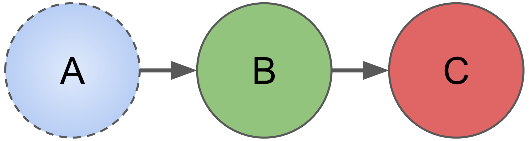
“Vulnerabilities do not occur often”



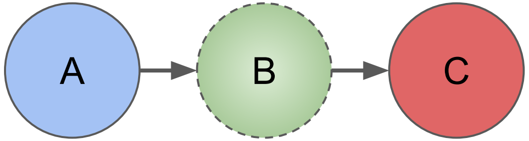
“Sanitizers clean untrusted input before it reaches a sink”

Beliefs about taint flow

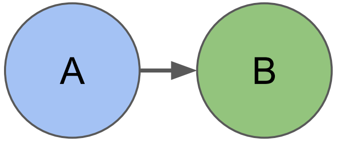
“Sanitizers secure sinks from untrusted input”



“Vulnerabilities do not occur often”

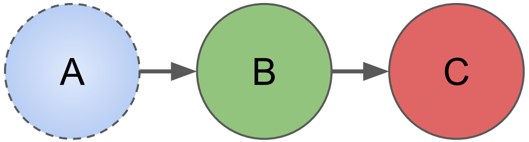


“Sanitizers clean untrusted input before it reaches a sink”

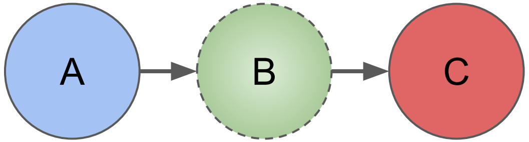


Beliefs about taint flow

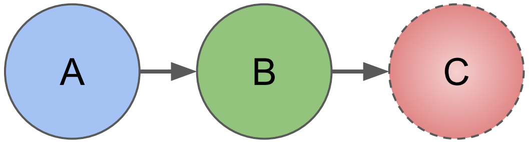
“Sanitizers secure sinks from untrusted input”



“Vulnerabilities do not occur often”

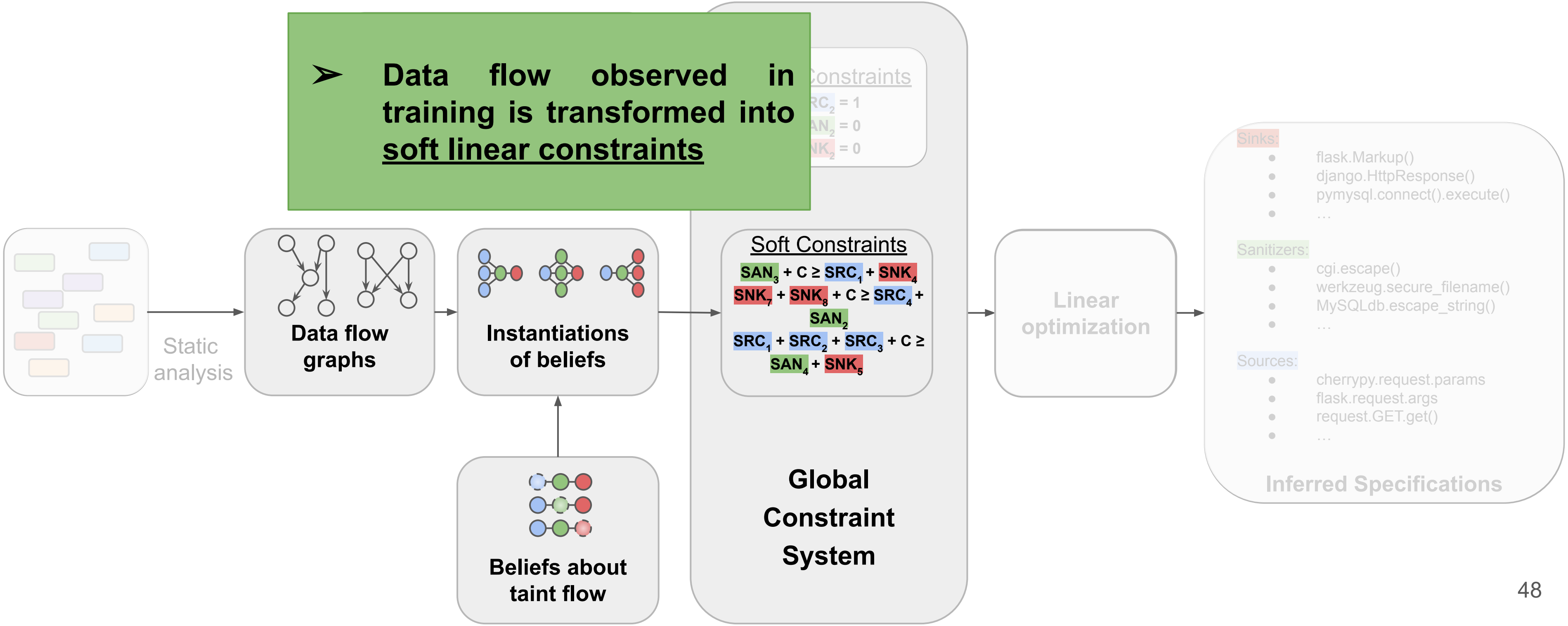


“Sanitizers clean untrusted input before it reaches a sink”



Seldon overview

➢ Data flow observed in training is transformed into soft linear constraints

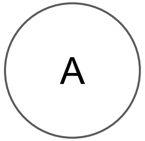


$SAN_3 + C \geq SRC_1 + SNK_4$

$SNK_7 + SNK_8 + C \geq SRC_4 + SAN_2$

$SRC_1 + SRC_2 + SRC_3 + C \geq SAN_4 + SNK_5$

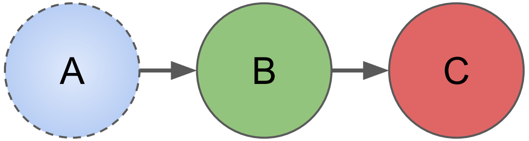
Candidate scores



$0 \le SRC_A, SAN_A, SNK_A \le 1$

Instantiations of beliefs

“Sanitizers secure sinks from untrusted input”



Instantiations of beliefs

If

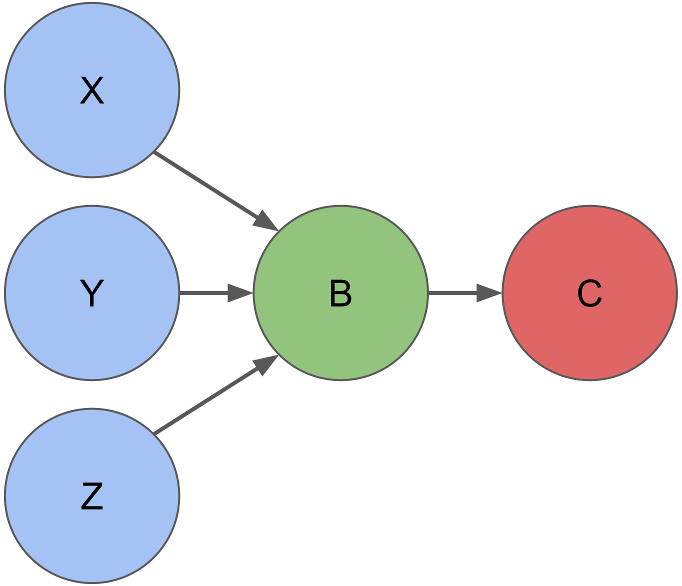
B is a sanitizer

And

C is a sink

Then

At least one of X, Y, Z is a source



Instantiations of beliefs

If

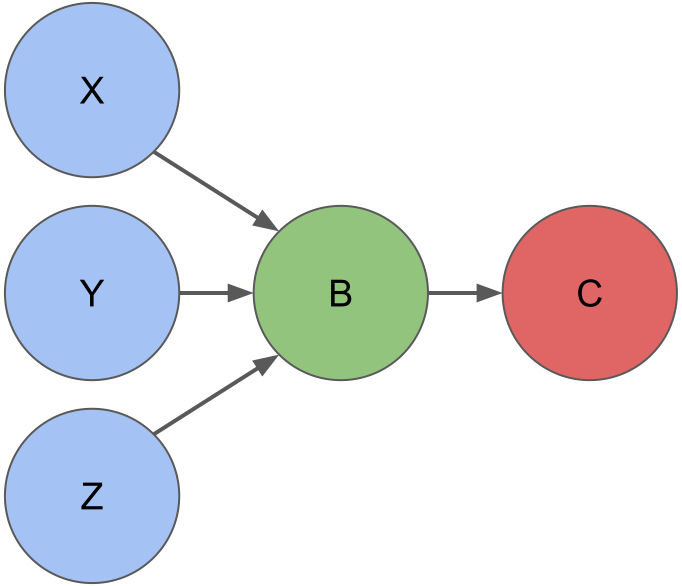
B is a sanitizer

And

C is a sink

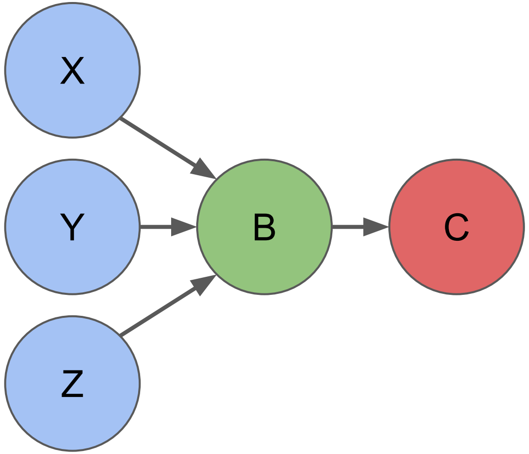
Then

At least one of X, Y, Z is a source



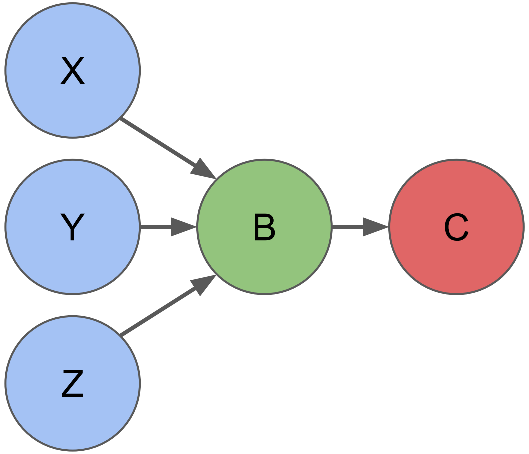
$$\mathbf{SRC_X + SRC_Y + SRC_Z + 1 \geq SAN_B + SNK_C}$$

Instantiations of beliefs

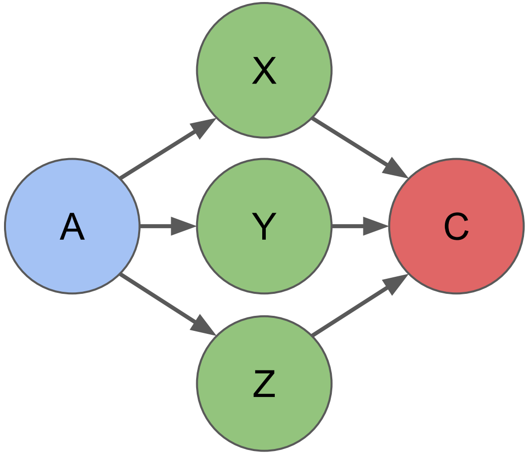


$SRC_X + SRC_Y + SRC_Z + 1 \geq SAN_B + SNK_C$

Instantiations of beliefs

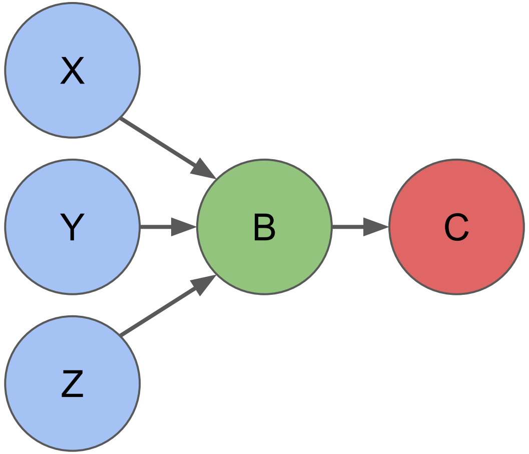


$SRC_X + SRC_Y + SRC_Z + 1 \geq SAN_B + SNK_C$

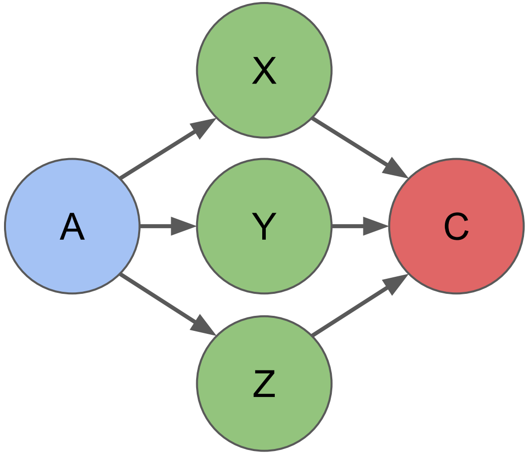


$SAN_X + SAN_Y + SAN_Z + 1 \geq SRC_A + SNK_C$

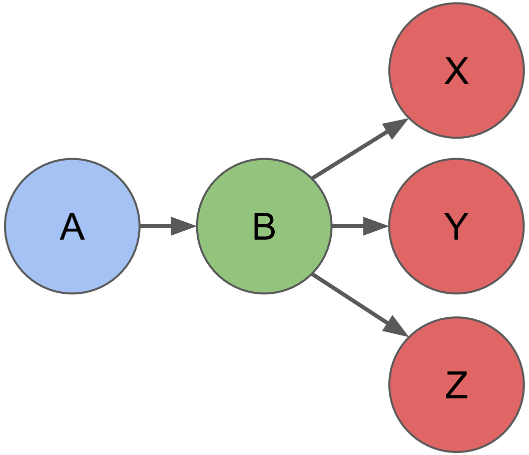
Instantiations of beliefs



$\mathrm{SRC}_X + \mathrm{SRC}_Y + \mathrm{SRC}_Z + 1 \geq \mathrm{SAN}_B + \mathrm{SNK}_C$

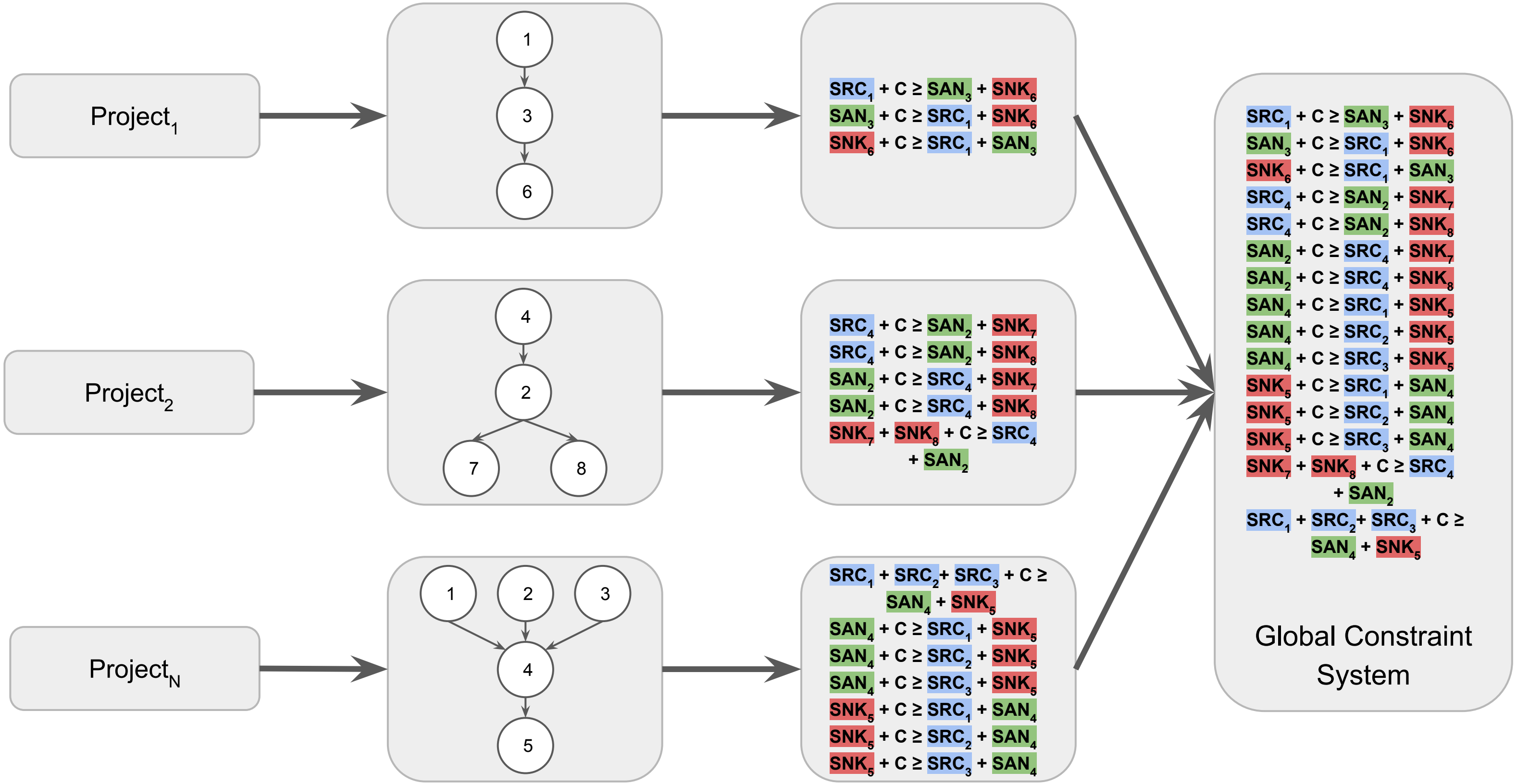


$\mathrm{SAN}_X + \mathrm{SAN}_Y + \mathrm{SAN}_Z + 1 \geq \mathrm{SRC}_A + \mathrm{SNK}_C$



$\mathrm{SNK}_X + \mathrm{SNK}_Y + \mathrm{SNK}_Z + 1 \geq \mathrm{SRC}_A + \mathrm{SAN}_B$

Instantiations of beliefs



Project_1

$SRC_1 + C \geq SAN_3 + SNK_6$
$SAN_3 + C \geq SRC_1 + SNK_6$
$SNK_6 + C \geq SRC_1 + SAN_3$

Project_2

$SRC_4 + C \geq SAN_2 + SNK_7$
$SRC_4 + C \geq SAN_2 + SNK_8$
$SAN_2 + C \geq SRC_4 + SNK_7$
$SAN_2 + C \geq SRC_4 + SNK_8$
$SNK_7 + SNK_8 + C \geq SRC_4 + SAN_2$

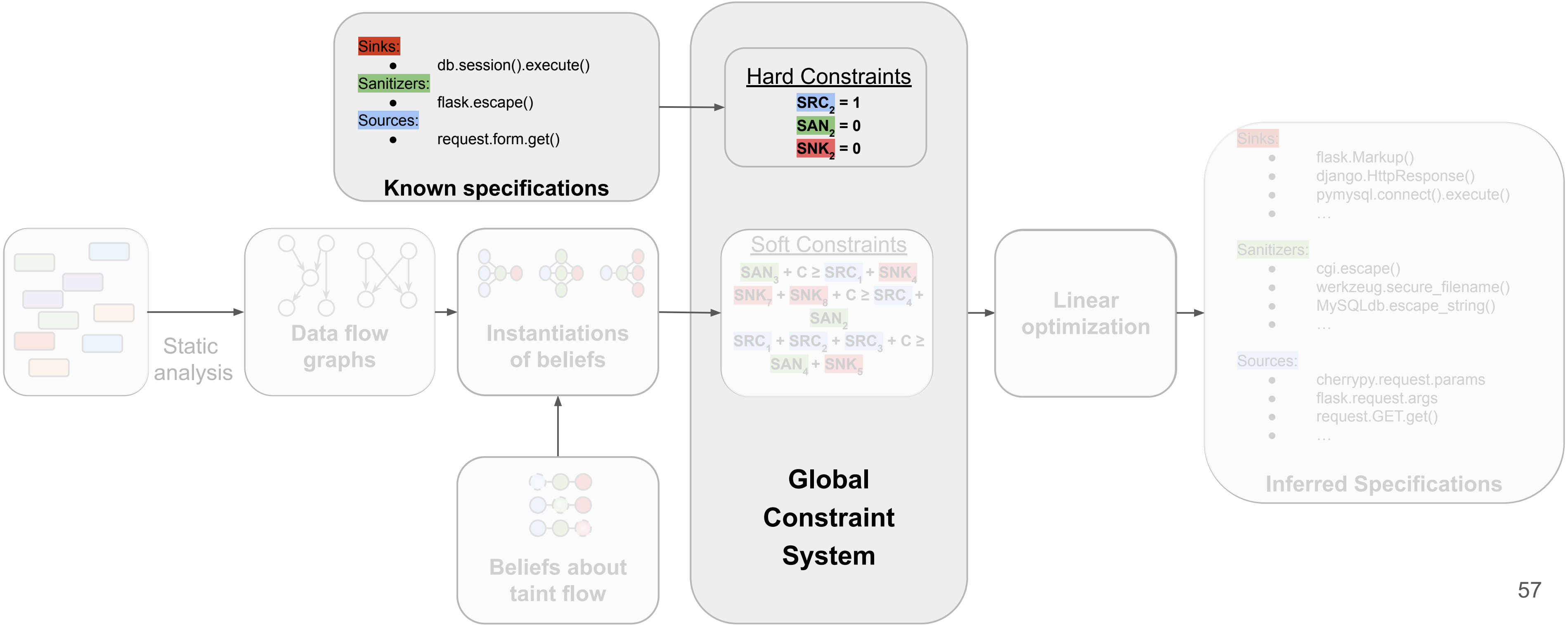
Project_N

$SRC_1 + SRC_2 + SRC_3 + C \geq SAN_4 + SNK_5$
$SAN_4 + C \geq SRC_1 + SNK_5$
$SAN_4 + C \geq SRC_2 + SNK_5$
$SAN_4 + C \geq SRC_3 + SNK_5$
$SNK_5 + C \geq SRC_1 + SAN_4$
$SNK_5 + C \geq SRC_2 + SAN_4$
$SNK_5 + C \geq SRC_3 + SAN_4$

$SRC_1 + C \geq SAN_3 + SNK_6$
$SAN_3 + C \geq SRC_1 + SNK_6$
$SNK_6 + C \geq SRC_1 + SAN_3$
$SRC_4 + C \geq SAN_2 + SNK_7$
$SRC_4 + C \geq SAN_2 + SNK_8$
$SAN_2 + C \geq SRC_4 + SNK_7$
$SAN_2 + C \geq SRC_4 + SNK_8$
$SAN_4 + C \geq SRC_1 + SNK_5$
$SAN_4 + C \geq SRC_2 + SNK_5$
$SAN_4 + C \geq SRC_3 + SNK_5$
$SNK_5 + C \geq SRC_1 + SAN_4$
$SNK_5 + C \geq SRC_2 + SAN_4$
$SNK_5 + C \geq SRC_3 + SAN_4$
$SNK_7 + SNK_8 + C \geq SRC_4 + SAN_2$
$SRC_1 + SRC_2 + SRC_3 + C \geq SAN_4 + SNK_5$

Global Constraint System

Hard constraints for known specifications



Sinks:
- db.session().execute()

Sanitizers:
- flask.escape()

Sources:
- request.form.get()

Known specifications

Hard Constraints

$SRC_2 = 1$

$SAN_2 = 0$

$SNK_2 = 0$

Soft Constraints

$SAN_3 + C \geq SRC_1 + SNK_4$

$SNK_7 + SNK_8 + C \geq SRC_4 + SAN_2$

$SRC_1 + SRC_2 + SRC_3 + C \geq SAN_4 + SNK_5$

Global Constraint System

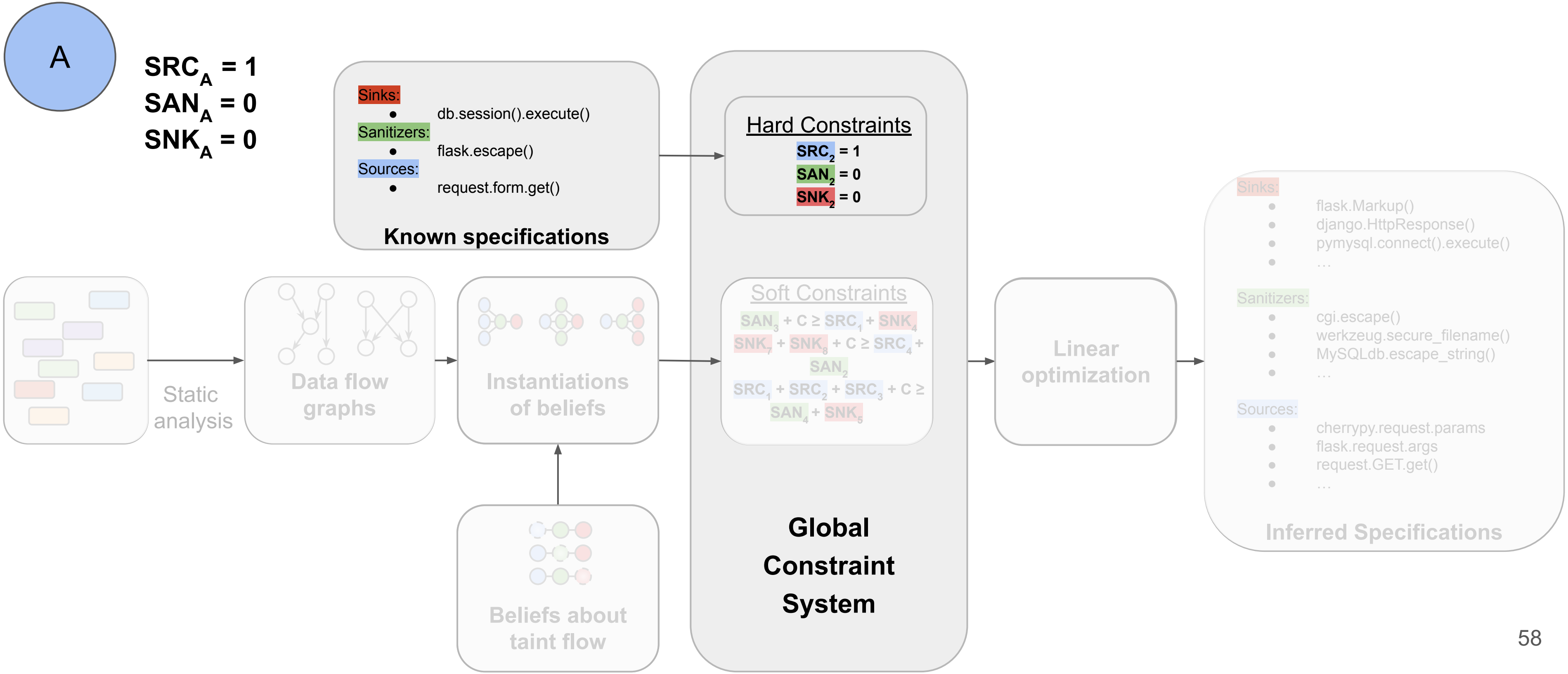
Sinks:
- flask.Markup()
- django.HttpResponse()
- pymysql.connect().execute()
- …

Sanitizers:
- cgi.escape()
- werkzeug.secure_filename()
- MySQLdb.escape_string()
- …

Sources:
- cherrypy.request.params
- flask.request.args
- request.GET.get()
- …

Inferred Specifications

Hard constraints for known specifications



$SRC_A = 1$
$SAN_A = 0$
$SNK_A = 0$

Sinks:
- db.session().execute()

Sanitizers:
- flask.escape()

Sources:
- request.form.get()

Known specifications

Hard Constraints

$SRC_2 = 1$
$SAN_2 = 0$
$SNK_2 = 0$

Soft Constraints

$SAN_3 + C \geq SRC_1 + SNK_4$
$SNK_7 + SNK_8 + C \geq SRC_4 + SAN_2$
$SRC_1 + SRC_2 + SRC_3 + C \geq SAN_4 + SNK_5$

Global Constraint System

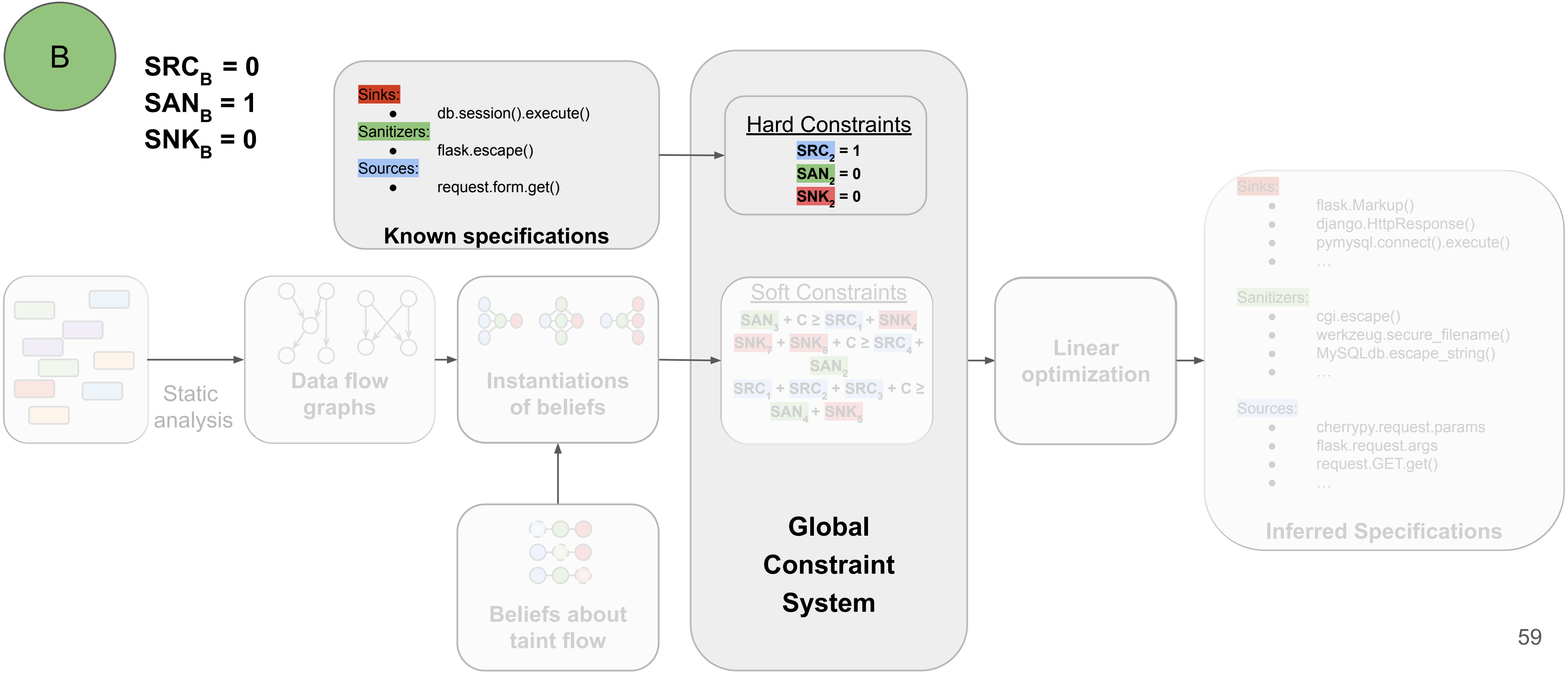
Sinks:
- flask.Markup()
- django.HttpResponse()
- pymysql.connect().execute()
- …

Sanitizers:
- cgi.escape()
- werkzeug.secure_filename()
- MySQLdb.escape_string()
- …

Sources:
- cherrypy.request.params
- flask.request.args
- request.GET.get()
- …

Inferred Specifications

Hard constraints for known specifications



$SRC_B = 0$
$SAN_B = 1$
$SNK_B = 0$

Sinks:
- db.session().execute()

Sanitizers:
- flask.escape()

Sources:
- request.form.get()

Known specifications

Hard Constraints

$SRC_2 = 1$
$SAN_2 = 0$
$SNK_2 = 0$

Soft Constraints

$SAN_3 + C \geq SRC_1 + SNK_4$
$SNK_7 + SNK_8 + C \geq SRC_4 + SAN_2$
$SRC_1 + SRC_2 + SRC_3 + C \geq SAN_4 + SNK_5$

Global Constraint System

Static analysis

Data flow graphs

Instantiations of beliefs

Beliefs about taint flow

Linear optimization

Sinks:
- flask.Markup()
- django.HttpResponse()
- pymysql.connect().execute()
- …

Sanitizers:
- cgi.escape()
- werkzeug.secure_filename()
- MySQLdb.escape_string()
- …

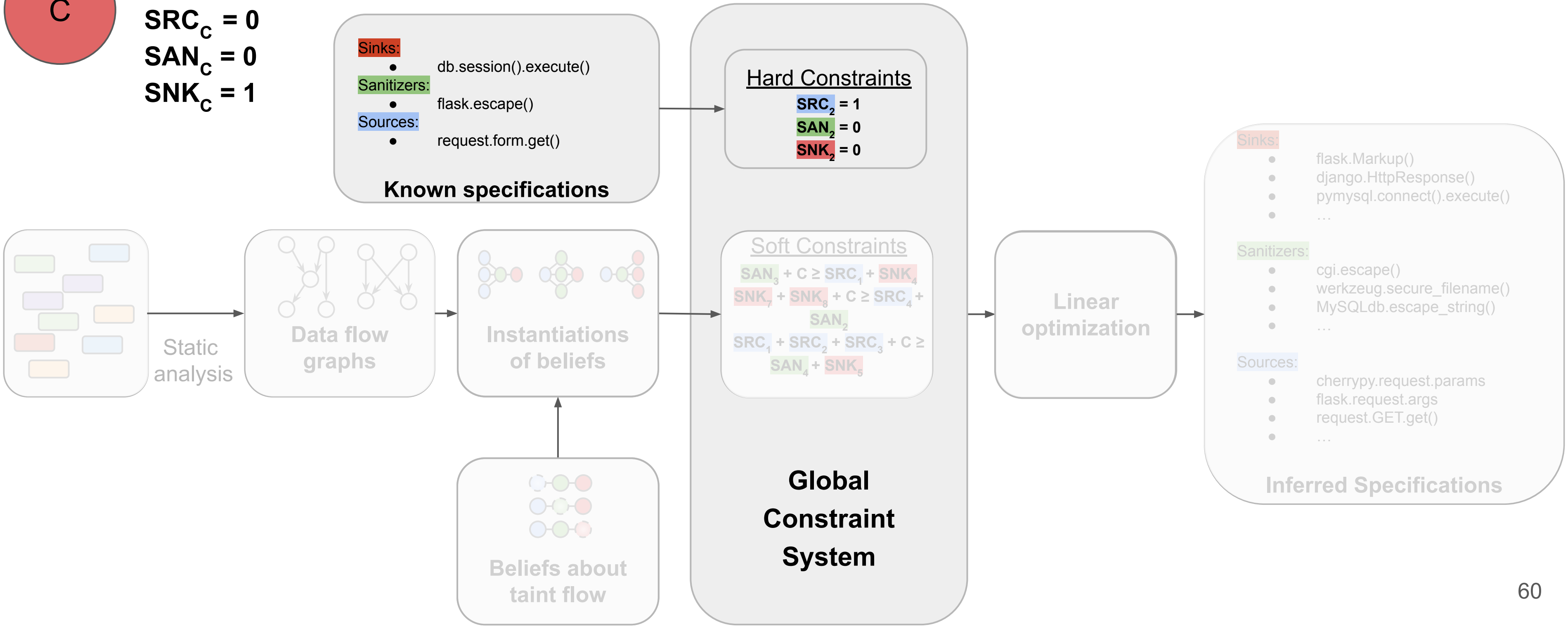
Sources:
- cherrypy.request.params
- flask.request.args
- request.GET.get()
- …

Inferred Specifications

Hard constraints for known specifications

C

$SRC_C = 0$
$SAN_C = 0$
$SNK_C = 1$



Sinks:
- db.session().execute()

Sanitizers:
- flask.escape()

Sources:
- request.form.get()

Known specifications

Hard Constraints

$SRC_2 = 1$
$SAN_2 = 0$
$SNK_2 = 0$

Soft Constraints

$SAN_3 + C \geq SRC_1 + SNK_4$

$SNK_7 + SNK_8 + C \geq SRC_4 + SAN_2$

$SRC_1 + SRC_2 + SRC_3 + C \geq SAN_4 + SNK_5$

Global Constraint System

Static analysis

Data flow graphs

Instantiations of beliefs

Beliefs about taint flow

Linear optimization

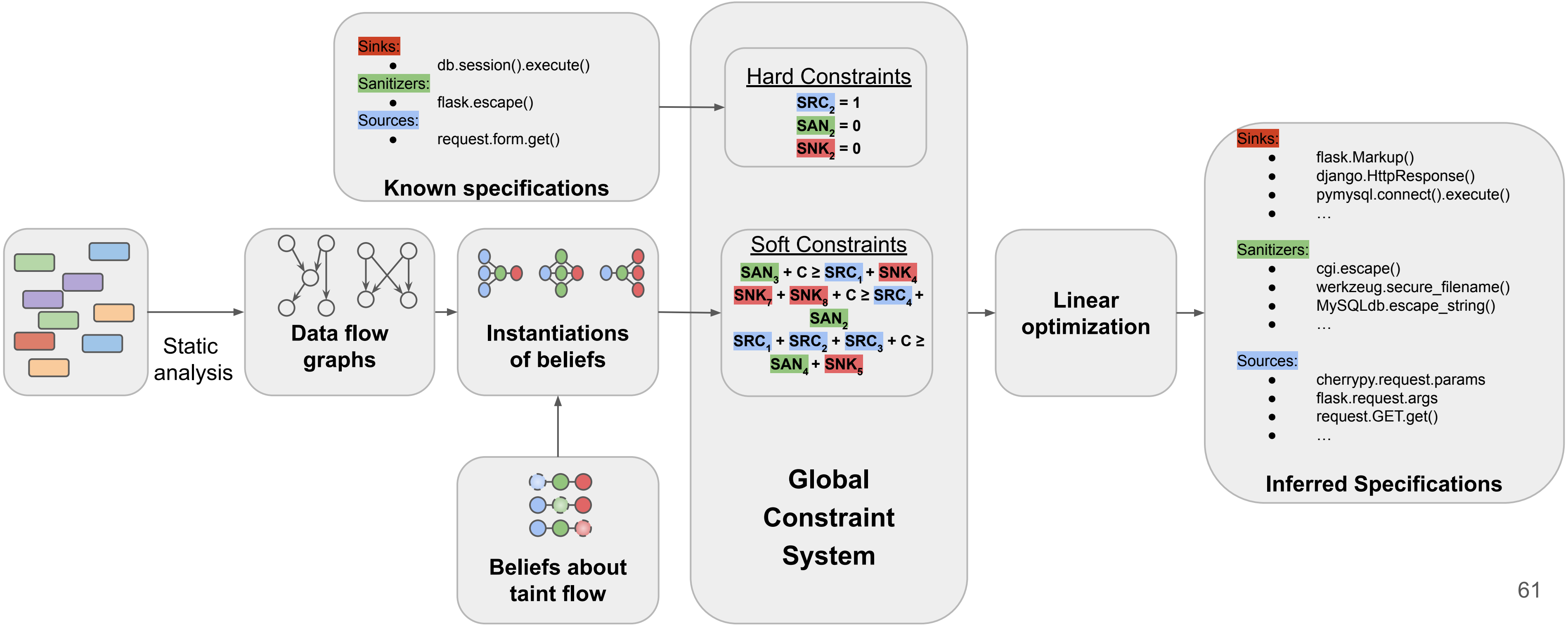
Sinks:
- flask.Markup()
- django.HttpResponse()
- pymysql.connect().execute()
- …

Sanitizers:
- cgi.escape()
- werkzeug.secure_filename()
- MySQLdb.escape_string()
- …

Sources:
- cherrypy.request.params
- flask.request.args
- request.GET.get()
- …

Inferred Specifications

Seldon overview



Known specifications

Sinks:
- db.session().execute()

Sanitizers:
- flask.escape()

Sources:
- request.form.get()

Static analysis

Data flow graphs

Instantiations of beliefs

Beliefs about taint flow

Global Constraint System

Hard Constraints

$SRC_2 = 1$

$SAN_2 = 0$

$SNK_2 = 0$

Soft Constraints

$SAN_3 + C \geq SRC_1 + SNK_4$

$SNK_7 + SNK_8 + C \geq SRC_4 + SAN_2$

$SRC_1 + SRC_2 + SRC_3 + C \geq SAN_4 + SNK_5$

Linear optimization

Inferred Specifications

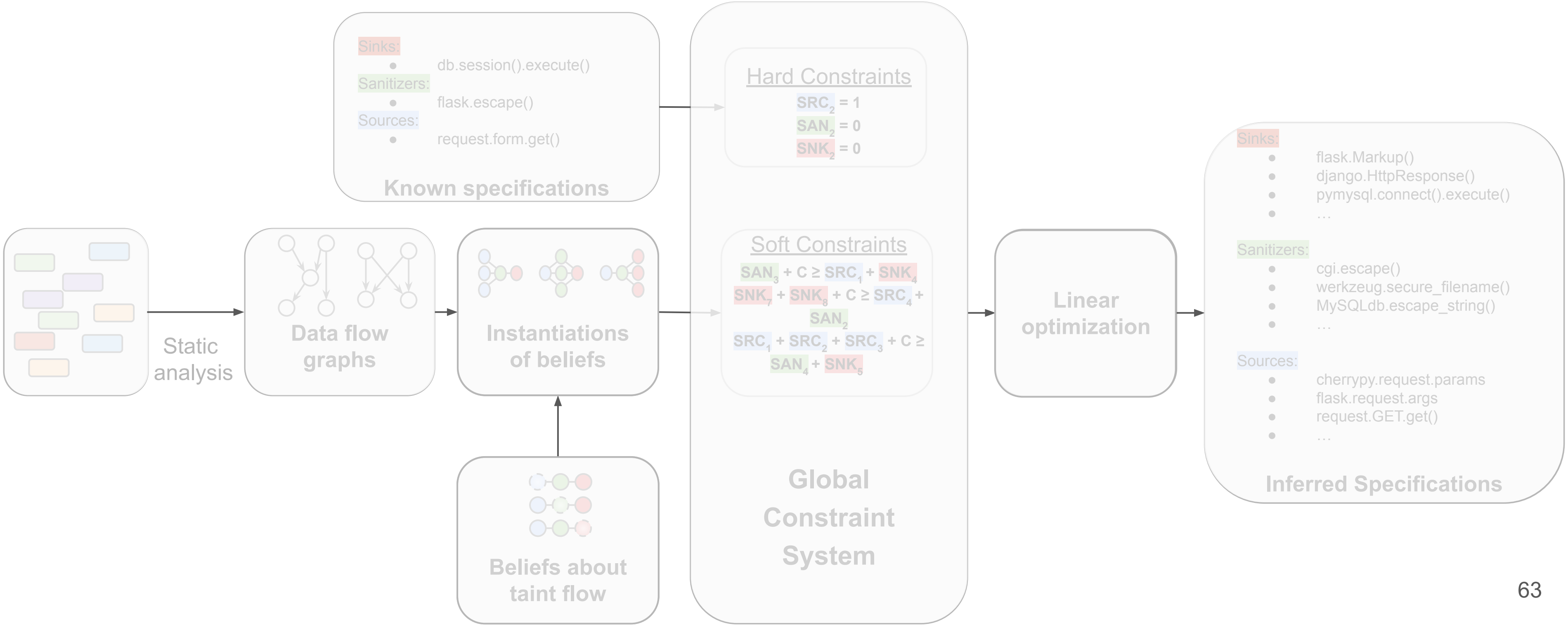
Sinks:
- flask.Markup()
- django.HttpResponse()
- pymysql.connect().execute()
- …

Sanitizers:
- cgi.escape()
- werkzeug.secure_filename()
- MySQLdb.escape_string()
- …

Sources:
- cherrypy.request.params
- flask.request.args
- request.GET.get()
- …

Seldon in production at deepcode.ai

Seldon for Python (this work)



Sinks:
- db.session().execute()

Sanitizers:
- flask.escape()

Sources:
- request.form.get()

Known specifications

Static analysis

Data flow graphs

Instantiations of beliefs

Beliefs about taint flow

Hard Constraints

$SRC_2 = 1$

$SAN_2 = 0$

$SNK_2 = 0$

Soft Constraints

$SAN_3 + C \geq SRC_1 + SNK_4$

$SNK_7 + SNK_8 + C \geq SRC_4 + SAN_2$

$SRC_1 + SRC_2 + SRC_3 + C \geq SAN_4 + SNK_5$

Global Constraint System

Linear optimization

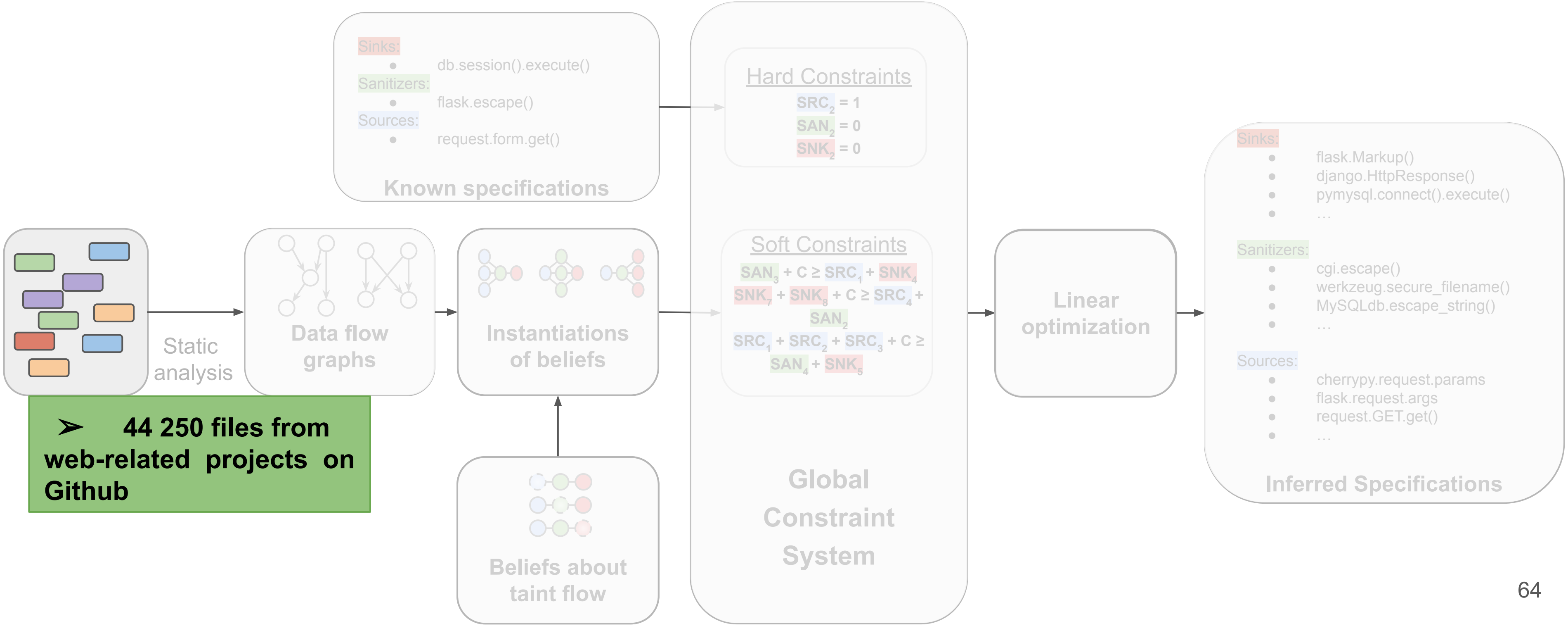
Sinks:
- flask.Markup()
- django.HttpResponse()
- pymysql.connect().execute()
- …

Sanitizers:
- cgi.escape()
- werkzeug.secure_filename()
- MySQLdb.escape_string()
- …

Sources:
- cherrypy.request.params
- flask.request.args
- request.GET.get()
- …

Inferred Specifications

Seldon for Python (this work)



Known specifications

Sinks:
- db.session().execute()

Sanitizers:
- flask.escape()

Sources:
- request.form.get()

Hard Constraints

$SRC_2 = 1$
$SAN_2 = 0$
$SNK_2 = 0$

Soft Constraints

$SAN_3 + C \geq SRC_1 + SNK_4$
$SNK_7 + SNK_8 + C \geq SRC_4 + SAN_2$
$SRC_1 + SRC_2 + SRC_3 + C \geq SAN_4 + SNK_5$

Global Constraint System

Static analysis

Data flow graphs

Instantiations of beliefs

Beliefs about taint flow

Linear optimization

Inferred Specifications

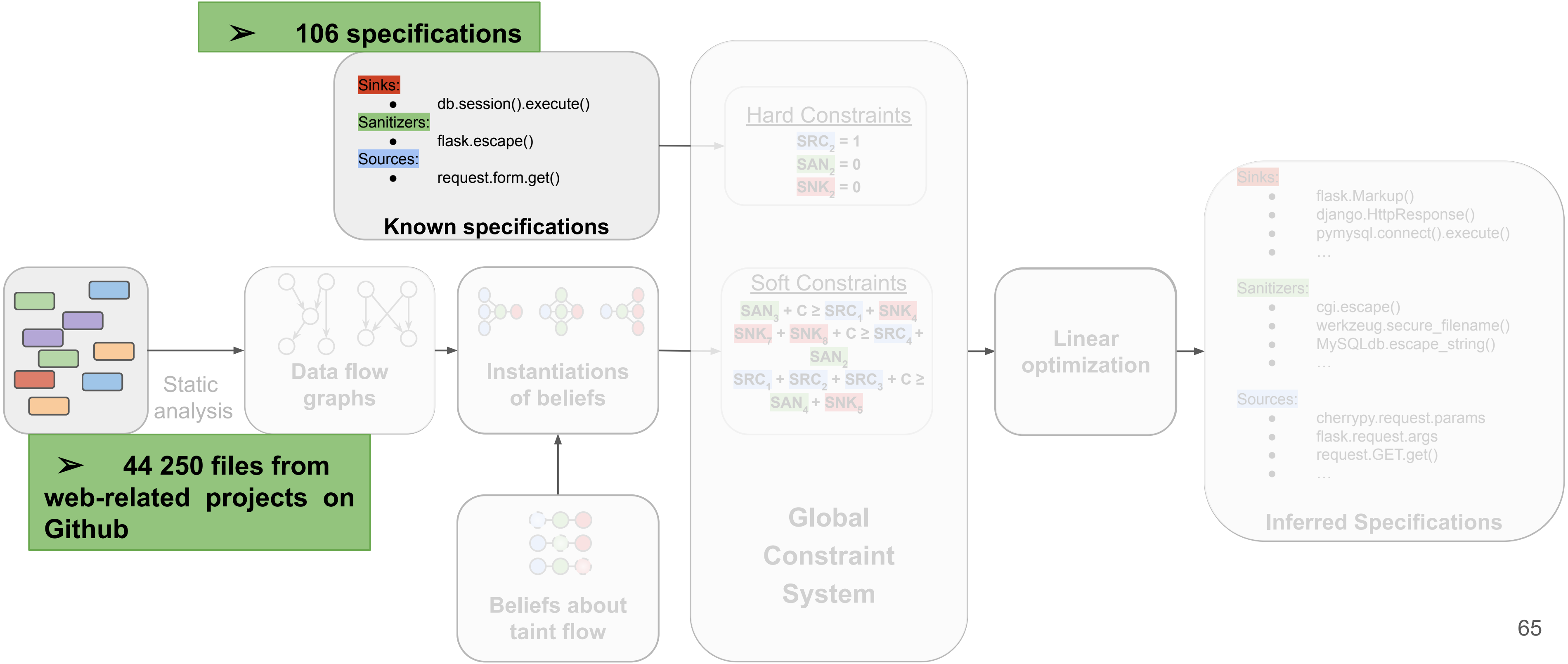
Sinks:
- flask.Markup()
- django.HttpResponse()
- pymysql.connect().execute()
- …

Sanitizers:
- cgi.escape()
- werkzeug.secure_filename()
- MySQLdb.escape_string()
- …

Sources:
- cherrypy.request.params
- flask.request.args
- request.GET.get()
- …

➢ **44 250 files from web-related projects on Github**

Seldon for Python (this work)



- ➢ 106 specifications

Sinks:

- db.session().execute()

Sanitizers:

- flask.escape()

Sources:

- request.form.get()

Known specifications

Static analysis

Data flow graphs

Instantiations of beliefs

Beliefs about taint flow

Hard Constraints

$SRC_2 = 1$

$SAN_2 = 0$

$SNK_2 = 0$

Soft Constraints

$SAN_3 + C \geq SRC_1 + SNK_4$

$SNK_7 + SNK_8 + C \geq SRC_4 + SAN_2$

$SRC_1 + SRC_2 + SRC_3 + C \geq SAN_4 + SNK_5$

Global Constraint System

Linear optimization

Sinks:

- flask.Markup()
- django.HttpResponse()
- pymysql.connect().execute()
- …

Sanitizers:

- cgi.escape()
- werkzeug.secure_filename()
- MySQLdb.escape_string()
- …

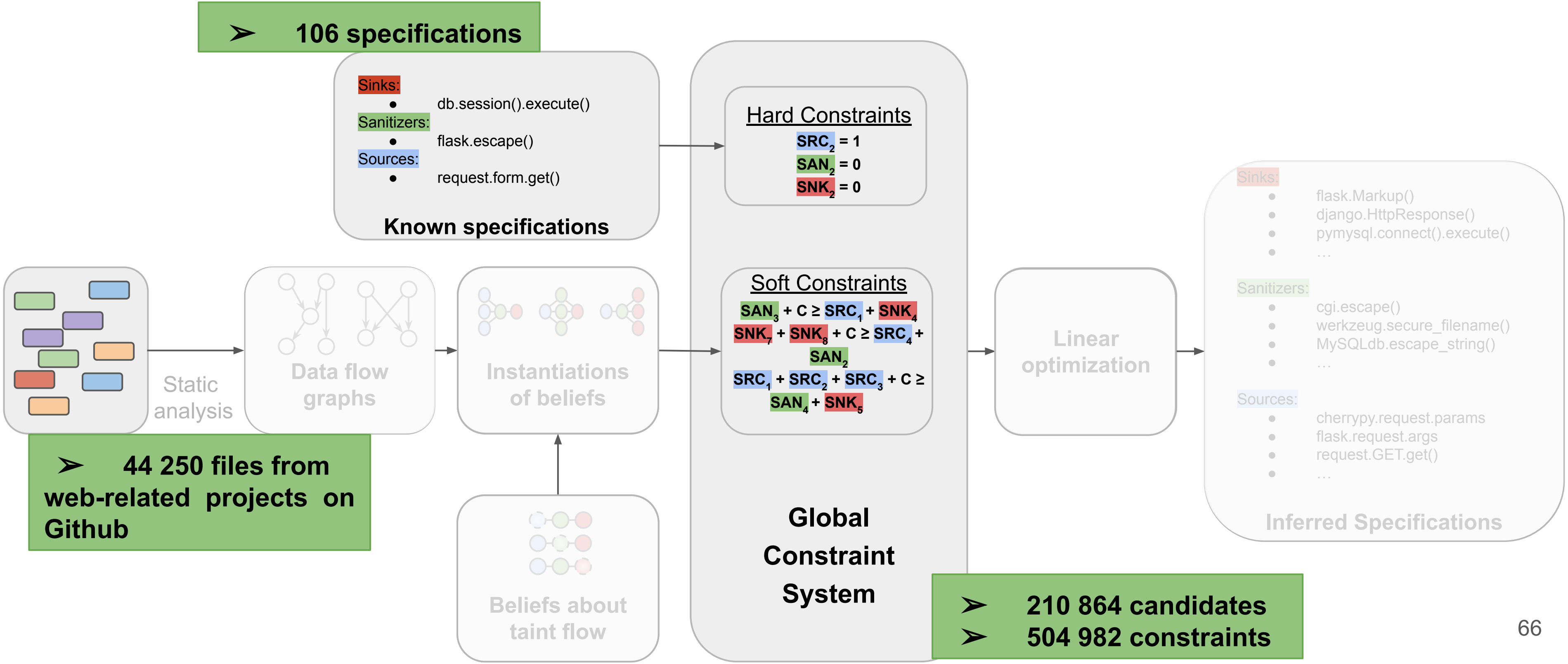
Sources:

- cherrypy.request.params
- flask.request.args
- request.GET.get()
- …

Inferred Specifications

- ➢ 44 250 files from web-related projects on Github

Seldon for Python (this work)



➢ 106 specifications

Sinks:
- db.session().execute()

Sanitizers:
- flask.escape()

Sources:
- request.form.get()

Known specifications

Hard Constraints

$SRC_2 = 1$

$SAN_2 = 0$

$SNK_2 = 0$

Soft Constraints

$SAN_3 + C \geq SRC_1 + SNK_4$

$SNK_7 + SNK_8 + C \geq SRC_4 + SAN_2$

$SRC_1 + SRC_2 + SRC_3 + C \geq SAN_4 + SNK_5$

Global Constraint System

Sinks:
- flask.Markup()
- django.HttpResponse()
- pymysql.connect().execute()
- …

Sanitizers:
- cgi.escape()
- werkzeug.secure_filename()
- MySQLdb.escape_string()
- …

Sources:
- cherrypy.request.params
- flask.request.args
- request.GET.get()
- …

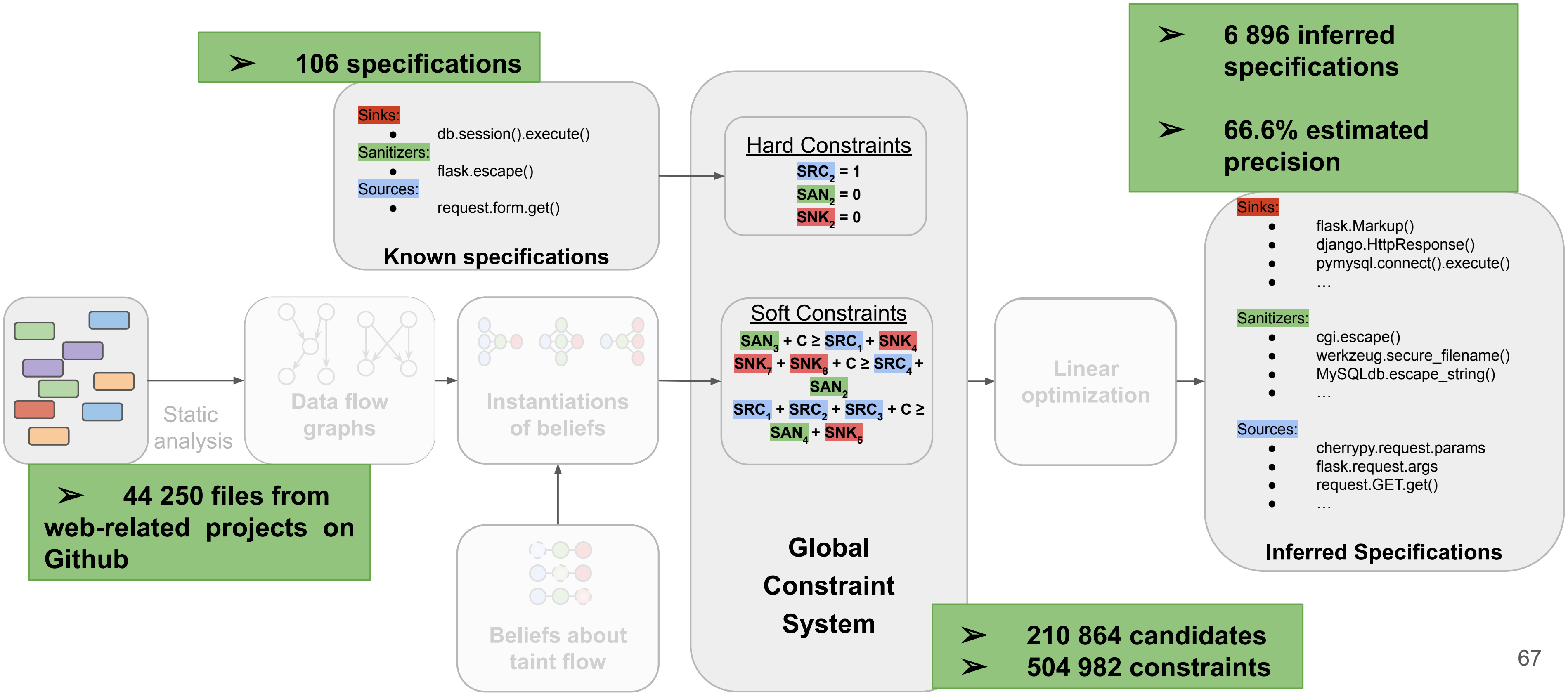
Inferred Specifications

➢ 44 250 files from web-related projects on Github

➢ 210 864 candidates

➢ 504 982 constraints

Seldon for Python (this work)



- ➢ 106 specifications

Sinks:
- db.session().execute()

Sanitizers:
- flask.escape()

Sources:
- request.form.get()

Known specifications

Static analysis

Data flow graphs

Instantiations of beliefs

Beliefs about taint flow

Global Constraint System

Hard Constraints

$SRC_2 = 1$

$SAN_2 = 0$

$SNK_2 = 0$

Soft Constraints

$SAN_3 + C \geq SRC_1 + SNK_4$

$SNK_7 + SNK_8 + C \geq SRC_4 + SAN_2$

$SRC_1 + SRC_2 + SRC_3 + C \geq SAN_4 + SNK_5$

Linear optimization

- ➢ 6 896 inferred specifications
- ➢ 66.6% estimated precision

Sinks:
- flask.Markup()
- django.HttpResponse()
- pymysql.connect().execute()
- …

Sanitizers:
- cgi.escape()
- werkzeug.secure_filename()
- MySQLdb.escape_string()
- …

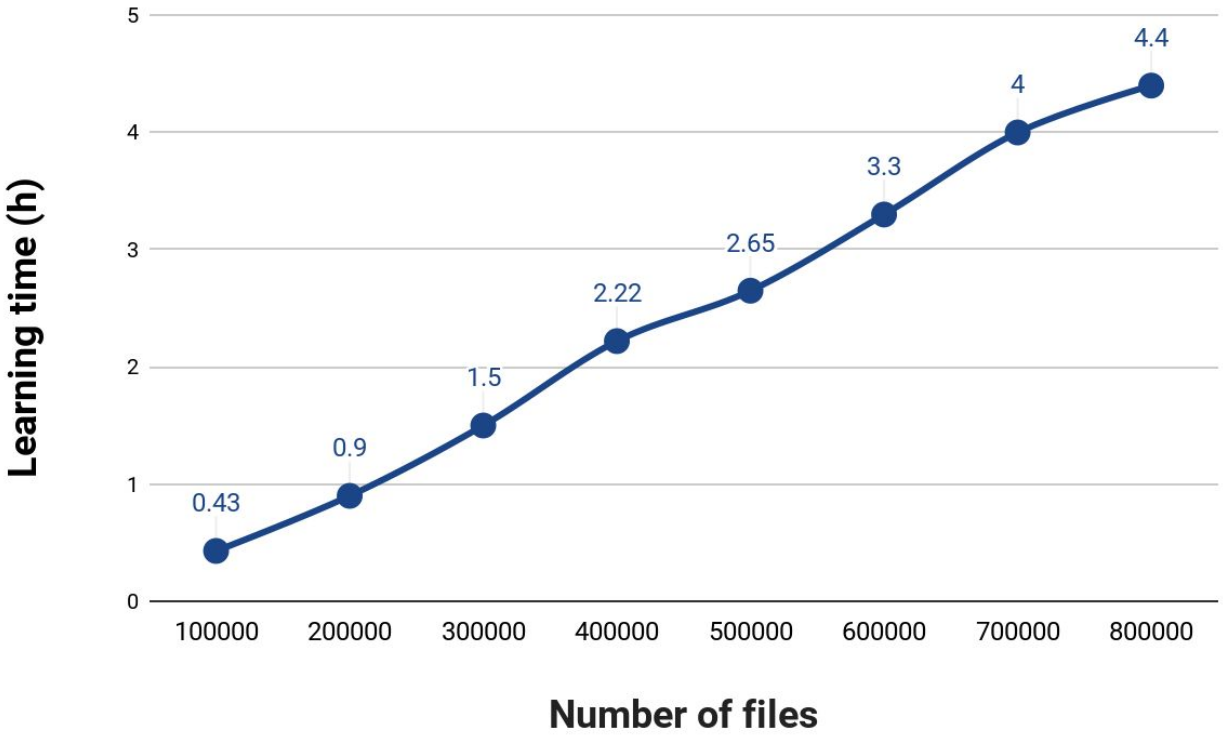
Sources:
- cherrypy.request.params
- flask.request.args
- request.GET.get()
- …

Inferred Specifications

- ➢ 44 250 files from web-related projects on Github

- ➢ 210 864 candidates
- ➢ 504 982 constraints

Seldon’s scalability



Impact of learning from Big Code

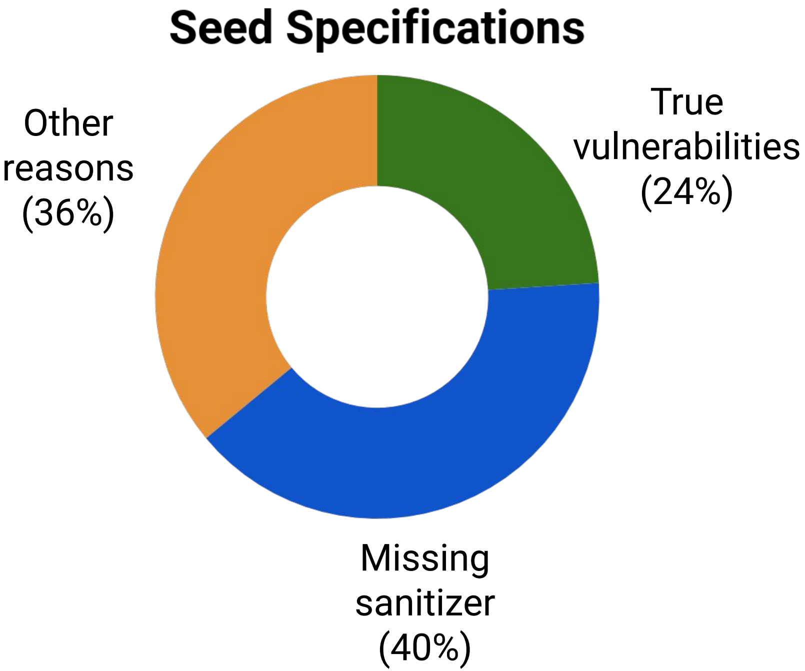
Evaluated on three randomly chosen projects separately (Patchwork, find_link, Django FileBrowser)

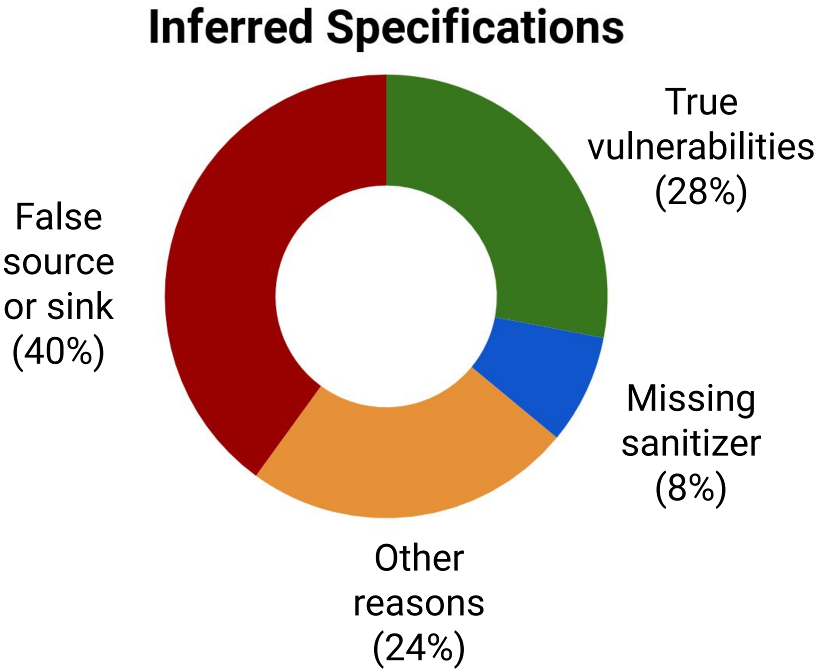
	Total number of true candidates inferred	Precision of inferred candidates
Individual model for each project	5	45.5%
Model trained on Big Code	23	65.7%

Predicted Vulnerabilities with Taint Analysis

	Predicted vulnerabilities	Estimated true vulnerabilities
Seed Specifications	662	159 (24%)
Inferred Specifications	21318	5969 (28%)

Breakdown of reports





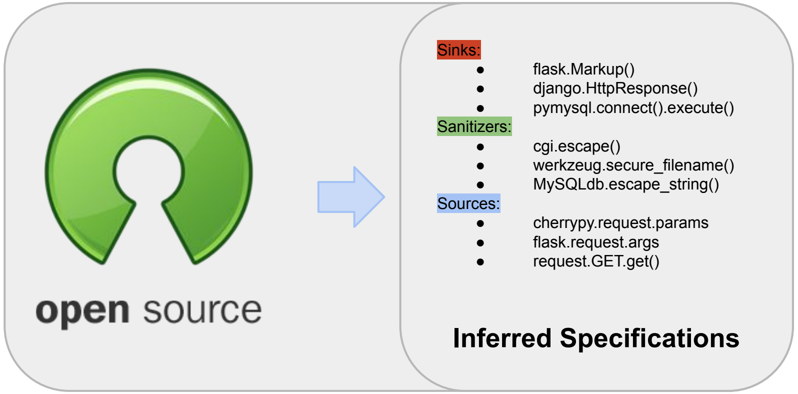
Vulnerabilities in real projects detected using Seldon Specs

1. https://github.com/anyaudio/anyaudio-server/pull/163
2. https://github.com/DataViva/dataviva-site/issues/1661
3. https://github.com/DataViva/dataviva-site/issues/1662
4. https://github.com/earthgecko/skyline/issues/85
5. https://github.com/earthgecko/skyline/issues/86
6. https://github.com/gestorpsi/gestorpsi/pull/75
7. https://github.com/HarshShah1997/Shopping-Cart/pull/2
8. https://github.com/kylewm/silo.pub/issues/57
9. https://github.com/kylewm/woodwind/issues/77
10. https://github.com/LMFDB/lmfdb/pull/2695
11. https://github.com/LMFDB/lmfdb/pull/2696
12. https://github.com/mgymrek/pybamview/issues/52
13. https://github.com/MinnPost/election-night-api/issues/1
14. https://github.com/mitre/multiscanner/issues/159
15. https://github.com/MLTSHP/mltshp/pull/509
16. https://github.com/mozilla/pontoon/pull/1175
17. https://github.com/PadamSethia/shorty/pull/4
18. https://github.com/sharadbhat/VideoHub/issues/3
19. https://github.com/UDST/urbansim/issues/213
20. https://github.com/viaict/viaduct/pull/5
21. https://github.com/yashbidasaria/Harry-s-List-Friends/issues/1

- **17** projects, **49** severe vulnerabilities (Cross-Site Scripting, SQL Injection, Path Traversal, OS Command Injection, Code Injection)

- Only **3** vulnerabilities could be detected using the seed specifications

Summary

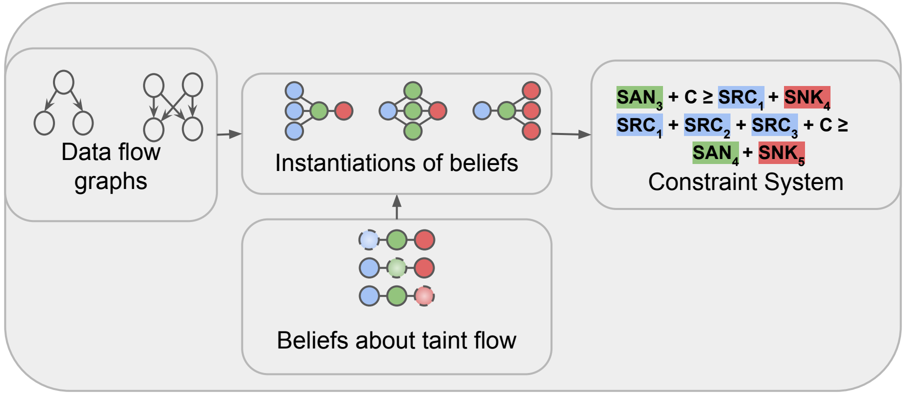


Sinks:
- flask.Markup()
- django.HttpResponse()
- pymysql.connect().execute()

Sanitizers:
- cgi.escape()
- werkzeug.secure_filename()
- MySQLdb.escape_string()

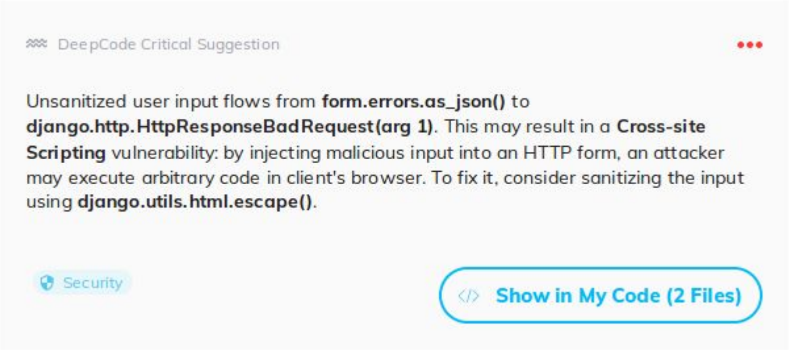
Sources:
- cherrypy.request.params
- flask.request.args
- request.GET.get()

Inferred Specifications



$SAN_3 + C \geq SRC_1 + SNK_4$

$SRC_1 + SRC_2 + SRC_3 + C \geq SAN_4 + SNK_5$



Unsanitized user input flows from **form.errors.as_json()** to **django.http.HttpResponseBadRequest(arg 1)**. This may result in a **Cross-site Scripting** vulnerability: by injecting malicious input into an HTTP form, an attacker may execute arbitrary code in client's browser. To fix it, consider sanitizing the input using **django.utils.html.escape()**.

Try online at deepcode.ai