Scalable Taint Specification Inference with Big Code

Victor Chibotaru  Ben Bichsel  Veselin Raychev  Martin Vechev
DeepCode  ETH Zurich  DeepCode  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

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

```python
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)
def upload():
    fname = flask.request.files['f'].filename
    path = os.path.join(upload_dir, fname)
    flask.request.files['f'].save(path)
def upload():
    fname = flask.request.files['f'].filename
    path = os.path.join(upload_dir, fname)
    flask.request.files['f'].save(path)
def upload():
    fname = flask.request.files['f'].filename
    path = os.path.join(upload_dir, fname)
    flask.request.files['f'].save(path)
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)
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)
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)
Completeness of taint specifications is crucial

Missing source or sink $\rightarrow$ undetected vulnerabilities

Missing sanitizer $\rightarrow$ false positive reports
Creating taint specifications is labour-intensive
Creating taint specifications is labour-intensive
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
Learning Taint Specifications from Big Code

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

Sanitizers:
- flask.escape()

Sources:
- request.form.get()

Known specifications

Data flow graphs

Static analysis
Learning Taint Specifications from Big Code

Known specifications

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

- **Sanitizers:**
  - `flask.escape()`

- **Sources:**
  - `request.form.get()`

Static analysis

Data flow graphs
Learning Taint Specifications from Big Code

Known specifications

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

Sanitizers:
- flask.escape()

Sources:
- request.form.get()

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()
- ...

Static analysis

Data flow graphs
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

➢ 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

Inference based on factor graphs

Semi-supervised learning
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)

- 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)

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

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

- The system has to work with few known specifications.
The system has to be **fast** enough to learn from Big Code

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
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

Static analysis ➔ Data flow graphs ➔ Instantiations of beliefs ➔ Global Constraint System

Known specifications

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

Sanitizers:
- flask.escape()

Sources:
- request.form.get()

Hard Constraints:
- SRC₁ = 1
- SAN₂ = 0
- SNK₇ = 0

Soft Constraints:
- SAN₃ + C ≥ SRC₁ + SNK₄
- SNK₇ + SNK₈ + C ≥ SRC₄ + SAN₂
- SRC₁ + SRC₂ + SRC₃ + C ≥ SAN₄ + SNK₅

Global Constraint System

Inferred Specifications

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

Static analysis

Data flow graphs

Instantiations of beliefs

Beliefs about taint flow

Global Constraint System

Soft Constraints

\[ \text{SAN}_1 + C \geq \text{SRC}_1 + \text{SNK}_1 \]
\[ \text{SNK}_1 + \text{SNK}_2 + C \geq \text{SRC}_2 + \text{SAN}_2 \]
\[ \text{SRC}_3 + \text{SRC}_4 + \text{SRC}_5 + C \geq \text{SAN}_3 + \text{SNK}_6 \]

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()`
  - ...  

Linear optimization

- **Hard Constraints**
  - \( \text{SRC}_2 = 1 \)
  - \( \text{SAN}_2 = 0 \)
  - \( \text{SNK} = 0 \)

- **Soft Constraints**
  - \( \text{SAN}_3 + C \geq \text{SRC}_1 + \text{SNK}_1 \)
  - \( \text{SNK}_1 + \text{SNK}_2 + C \geq \text{SRC}_2 + \text{SAN}_2 \)
  - \( \text{SRC}_3 + \text{SRC}_4 + \text{SRC}_5 + C \geq \text{SAN}_3 + \text{SNK}_6 \)

Global Constraint System
Candidate scores

\[ 0 \leq \text{SRC}_A, \text{SAN}_A, \text{SNK}_A \leq 1 \]
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

\[ \text{SRC}_X + \text{SRC}_Y + \text{SRC}_Z + 1 \geq \text{SAN}_B + \text{SNK}_C \]
Instantiations of beliefs

\( \text{SRC}_x + \text{SRC}_y + \text{SRC}_z + 1 \geq \text{SAN}_b + \text{SNK}_c \)
Instantiations of beliefs

\[ \text{SRC}_x + \text{SRC}_y + \text{SRC}_z + 1 \geq \text{SAN}_b + \text{SNK}_c \]

\[ \text{SAN}_x + \text{SAN}_y + \text{SAN}_z + 1 \geq \text{SRC}_A + \text{SNK}_c \]
Instantiations of beliefs

\[ \text{SRC}_A + \text{SRC}_B + \text{SRC}_C + 1 \geq \text{SAN}_A + \text{SNK}_C \]

\[ \text{SAN}_A + \text{SAN}_B + \text{SAN}_C + 1 \geq \text{SRC}_A + \text{SNK}_C \]

\[ \text{SNK}_A + \text{SNK}_B + \text{SNK}_C + 1 \geq \text{SRC}_A + \text{SNK}_B \]
Instantiations of beliefs

Project_1

Project_2

Project_N

Global Constraint System
Hard constraints for known specifications

**Known specifications**

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

- **Sanitizers:**
  - `flask.escape()`

- **Sources:**
  - `request.form.get()`

**Hard Constraints**

- `SRC_i = 1`
- `SAN_i = 0`
- `SNK_i = 0`

**Soft Constraints**

- `SAN_j + C ≥ SRC_i + SNK_i` (2)
- `SNK_j + SNK_i + C ≥ SRC_i + SAN_i` (2)
- `SRC_i + SRC_j + SRC_k + C ≥ SAN_i + SNK_i` (2)

**Global Constraint System**

**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()`
  - ...

**Static analysis**

**Data flow graphs**

**Instantiations of beliefs**

**Beliefs about taint flow**

**Static analysis**
Hard constraints for known specifications

**Known specifications**

**Hard Constraints**

- SRC\(_A\) = 1
- SAN\(_A\) = 0
- SNK\(_A\) = 0

**Sanitizers**

- flask.escape()

**Sources**

- request.form.get()

**Sanctions**

- db.session().execute()

**Inferred Specifications**

**Hard Constraints**

- SRC\(_1\) = 1
- SAN\(_1\) = 0
- SNK\(_1\) = 0

**Soft Constraints**

\[ \text{SAN}_2 + C \geq \text{SRC}_1 + \text{SNK}_4 \]
\[ \text{SNK}_3 + \text{SNK}_1 + C \geq \text{SRC}_4 + \text{SAN}_1 \]
\[ \text{SRC}_1 + \text{SRC}_2 + \text{SRC}_3 + C \geq \text{SNK}_3 + \text{SAN}_1 \]

**Global Constraint System**

- SRC\(_A\) = 1
- SAN\(_A\) = 0
- SNK\(_A\) = 0

**Sanitizers**

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

**Sources**

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

**Static analysis**

- Data flow graphs
- Instantiations of beliefs
- Beliefs about taint flow

**Linear optimization**
Hard constraints for known specifications

Known specifications

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

Sanitizers:
- flask.escape()

Sources:
- request.form.get()

Hard Constraints
- \( SRC = 1 \)
- \( SAN = 0 \)
- \( SNK = 0 \)

Soft Constraints
- \( SAN_j + C \geq SRC_i + SNK \)
- \( SNK + SNK + C \geq SRC_i \)
- \( SRC_i + SRC_j + SRC_k + C \geq SAN_j + SNK \)

Global Constraint System

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()
- ...

Static analysis

Data flow graphs

Instantiations of beliefs

Beliefs about taint flow

B

\( SRC_B = 0 \)
\( SAN_B = 1 \)
\( SNK_B = 0 \)
Hard constraints for known specifications

**Known specifications**

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

**Sanitizers:**
- `flask.escape()`

**Sources:**
- `request.form.get()`

**Hard Constraints**

- \[ \text{SRC}_C = 0 \]
- \[ \text{SAN}_C = 0 \]
- \[ \text{SNK}_C = 1 \]

**Soft Constraints**

- \[ \text{SAN}_1 + C \geq \text{SRC}_1 + \text{SNK}_1 \]
- \[ \text{SNK}_2 + \text{SNK}_3 + C \geq \text{SRC}_4 + \text{SAN}_3 \]
- \[ \text{SRC}_1 + \text{SRC}_2 + \text{SRC}_3 + C \geq \text{SAN}_4 + \text{SNK}_5 \]

**Global Constraint System**

**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()`
- ...

**Static analysis**

**Data flow graphs**

**Instantiations of beliefs**

**Beliefs about taint flow**
Seldon overview

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 ≥ SRC_1 + SNK_4`
- `SNK_7 + SNK_8 + C ≥ SRC_1 + SAN_2`
- `SRC_1 + SRC_2 + SRC_3 + C ≥ SAN_1 + SNK_5`

Global Constraint System

Data flow graphs

Static analysis

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()`
  - ...

Inferred Specifications
Seldon in production at deepcode.ai
Seldon for Python (this work)

Data flow graphs

Instantiations of beliefs

Beliefs about taint flow

Known specifications

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

Sanitizers:
- flask.escape()

Sources:
- request.form.get()

Hard Constraints
- SRC₂ = 1
- SAN₃ = 0
- SNK₄ = 0

Soft Constraints
- SAN₃ + C ≥ SRC₁ + SNK₄
- SNK₃ + SNK₄ + C ≥ SRC₁ + SAN₃
- SRC₂ + SRC₃ + SRC₄ + C ≥ SAN₃ + SNK₄

Global Constraint System

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 for Python (this work)

- **Known specifications**
  - **Sinks**: db.session().execute()
  - **Sanitizers**: flask.escape()
  - **Sources**: request.form.get()

- **Hard Constraints**
  - \[ \text{SRC}_2 = 1 \]
  - \[ \text{SAN}_3 = 0 \]
  - \[ \text{SNK}_4 = 0 \]

- **Soft Constraints**
  - \[ \text{SAN}_1 + C \geq \text{SRC}_1 + \text{SNK}_2 + \text{SNK}_3 \]
  - \[ \text{SNK}_2 + \text{SNK}_3 + C \geq \text{SRC}_4 + \text{SAN}_2 \]
  - \[ \text{SRC}_1 + \text{SRC}_2 + \text{SRC}_3 + C \geq \text{SAN}_4 + \text{SNK}_6 \]

- **Linear optimization**

- **Global Constraint System**

- **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

Known specifications

- **Sinks:**
  - `db.session().execute()`
- **Sanitizers:**
  - `flask.escape()`
- **Sources:**
  - `request.form.get()`

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()`
  - ...

Global Constraint System

- Hard Constraints:
  - `SRC_1 = 1`
  - `SAN_2 = 0`
  - `SNK_2 = 0`

- Soft Constraints:
  - `SAN_3 + C ≥ SRC_1 + SNK_4`
  - `SNK_5 + SNK_7 + SNK_8 + C ≥ SRC_3 + SAN_2`
  - `SRC_1 + SRC_2 + SRC_3 + C ≥ SAN_4 + SNK_5`

- Linear optimization

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

---

65
Seldon for Python (this work)

106 specifications

Known specifications

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

- **Sanitizers:**
  - flask.escape()

- **Sources:**
  - request.form.get()

Hard Constraints

- \( \text{SRC}_1 = 1 \)
- \( \text{SAN}_1 = 0 \)
- \( \text{SNK}_1 = 0 \)

Soft Constraints

- \( \text{SAN}_2 + C \geq \text{SRC}_3 + \text{SNK}_3 \)
- \( \text{SNK}_4 + \text{SNK}_5 + C \geq \text{SRC}_4 + \text{SAN}_6 \)
- \( \text{SRC}_6 + \text{SRC}_7 + \text{SRC}_8 + C \geq \text{SAN}_3 + \text{SNK}_2 \)

Global Constraint System

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()

- **Static analysis**
  - 44 250 files from web-related projects on Github

- **Dynamic analysis**
  - 106 specifications
  - 210 864 candidates
  - 504 982 constraints

- **Linear optimization**

66
Seldon for Python (this work)

- 106 specifications

**Known specifications**

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

- **Sanitizers:**
  - `flask.escape()`

- **Sources:**
  - `request.form.get()`

**Instantiations of beliefs**

**Hard Constraints**

- `SRC_1 = 1`
- `SAN_2 = 0`
- `SNK_2 = 0`

**Soft Constraints**

- `SAN_3 + C ≥ SRC_1 + SNK_4 + SNK_7 + SNK_8 + C ≥ SAN_4 + SNK_5`
- `SRC_1 + SRC_2 + SRC_3 + C ≥ SAN_4 + SNK_5`

**Global Constraint System**

**Linear optimization**

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

**Static analysis**

**Data flow graphs**

**Instantiations of beliefs**

**Beliefs about taint flow**

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

**Inferred Specifications**

- **210 864 candidates**
- **504 982 constraints**

**Current Spec Compliance**

**67**
Seldon’s scalability

Number of files

Learning time (h)

- 0.43 at 100,000 files
- 0.9 at 200,000 files
- 1.5 at 300,000 files
- 2.22 at 400,000 files
- 2.65 at 500,000 files
- 3.3 at 600,000 files
- 4.0 at 700,000 files
- 4.4 at 800,000 files
Impact of learning from Big Code

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

<table>
<thead>
<tr>
<th>Model</th>
<th>Total number of true candidates inferred</th>
<th>Precision of inferred candidates</th>
</tr>
</thead>
<tbody>
<tr>
<td>Individual model for each project</td>
<td>5</td>
<td>45.5%</td>
</tr>
<tr>
<td>Model trained on Big Code</td>
<td>23</td>
<td>65.7%</td>
</tr>
</tbody>
</table>
## Predicted Vulnerabilities with Taint Analysis

<table>
<thead>
<tr>
<th></th>
<th>Predicted vulnerabilities</th>
<th>Estimated true vulnerabilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seed Specifications</td>
<td>662</td>
<td>159 (24%)</td>
</tr>
<tr>
<td>Inferred Specifications</td>
<td>21318</td>
<td>5969 (28%)</td>
</tr>
</tbody>
</table>
Breakdown of reports

**Seed Specifications**
- Missing sanitizer (40%)
- True vulnerabilities (24%)
- Other reasons (36%)

**Inferred Specifications**
- False source or sink (40%)
- True vulnerabilities (28%)
- Other reasons (24%)
- Missing sanitizer (8%)
Vulnerabilities in real projects detected using Seldon Specs

5. https://github.com/earthgecko/skyline/issues/86
6. https://github.com/gestorpsi/gestorpsi/pull/75
13. https://github.com/MinnPost/election-night-api/issues/1
15. https://github.com/MLTSHP/mltshp/pull/509
17. https://github.com/PadamSethia/shorty/pull/4
18. https://github.com/sharadbhat/VideoHub/issues/3

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

Try online at [deepcode.ai](https://deepcode.ai)

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()

Data flow graphs

- 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()

Instantiations of beliefs

Beliefs about taint flow

Constraint System

\[
\text{SAN}_1 + C \geq \text{SRC}_1 + \text{SNK}_4
\]

\[
\text{SRC}_1 + \text{SRC}_2 + \text{SRC}_3 + C \geq \text{SAN}_4 + \text{SNK}_5
\]