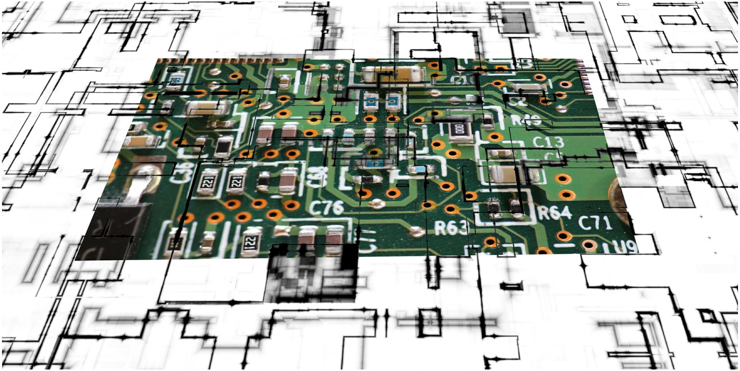


PyCG: Practical Call Graph Generation in Python

HE PEILIN

2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)[1]

[2]

Contents

Call Graph

Program Analysis, Inter-procedural Analysis, and Vulnerability Propagation Analysis.

Contributions

- *Static approach for pragmatic call graph generation in Python.*
- *Micro-benchmark suite used as a standard to evaluate this methods in Python.*
- *Evaluating the effectiveness of the approach through Micro-benchmark and Macro-benchmarks*
- *How the approach can aid dependency impact analysis through a potential enhancement of GitHub's "security advisory"*

Contents

Contents

Challenges List

Python Features

- Higher-order Functions
- Nested Definitions
- Classes
  - Inherit attributes and methods
  - Method Resolution Order (MRO)
- Modules
- Dynamic Features
  - Meta-programming
- Duck Typing

Contents

crypto module

crypto.py

```python
import cryptops

class Crypto:
    def __init__(self, key):
        self.key = key

    def apply(self, msg, func):
        return func(self.key, msg)

crp=Crypto('secretkey')
encrypted=crp.apply('hello␣world', cryptops.encrypt)
decrypted=crp.apply(encrypted, cryptops.decrypt)
```

1

[1] Jump to Scope Tree(Figure.5), Assignment Graph(Page 6)

Call graphs for *crypto* module

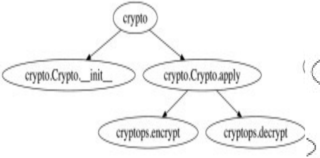


Figure 1. Precise call graph.

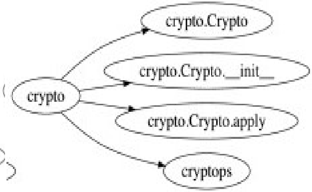


Figure 2. Pyan-generated call graph

Figure 3. Depends-generated call graph.

Contents

Contents

Contents

Syntax Analysis – AST

$$
\begin{aligned}
e \in \mathit{Expr} &::= o \mid x \mid x := e \mid \textbf{function}\ x\ (y\ldots)\ e \mid \textbf{return}\ e \mid \\
&\quad\ e(x{=}e\ldots) \mid \textbf{class}\ x\ (y\ldots)\ e \mid e.x \mid e.x := e \mid \\
&\quad\ \textbf{new}\ x\ (y = e\ldots) \mid \textbf{import}\ x\ \textbf{from}\ m\ \textbf{as}\ y \mid \\
&\quad\ \textbf{iter}\ x \mid e;e \\
o \in \mathit{Obj} &::= n, v \\
v \in \mathit{Definition} &::= x, \tau \\
\tau \in \mathit{IdentType} &::= \textbf{func} \mid \textbf{var} \mid \textbf{cls} \mid \textbf{mod} \\
n \in \mathit{Namespace} &::= (v)^* \\
x, y \in \mathit{Identifier} &::= \textit{is the set of program identifiers} \\
m \in \mathit{Modules} &::= \textit{is the set of modules}
\end{aligned}
$$

Figure 4. The syntax for representing the input Python programs along with the evaluation contexts

[2]

[2] Jump to Scope Tree(Figure.5), Assignment Graph(Page 6)

Syntax Analysis – Evaluation

Evaluation E

Use evaluation contexts [3], [4] that describe the order in which sub-expressions are evaluated.

$$
\begin{aligned}
E ::= [\,]\ |x := E|\text{return } E|E(x = e \ldots)| \\
o(x = E \ldots)|\text{new } x(y = E)|E.x|E.x := e| \\
o.x := E|\text{iter } o|E; e|o; E
\end{aligned}
$$

Contents

State Analysis

After the AST (Figure.4), the analysis maintains a state consisting of four domains as shown:

$$
\begin{aligned}
\pi &\in AssignG = Obj \hookrightarrow P(Obj)\\
s &\in Scope = Definition \hookrightarrow P(Definition)\\
h &\in ClassHier = Obj \hookrightarrow Obj^{*}\\
\sigma &\in State = AssignG \times Scope \times Namespace \times ClassHier
\end{aligned}
$$

[3]

[3] Jump to AST(Figure.4)

Scope Tree

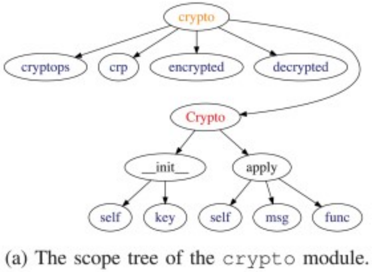
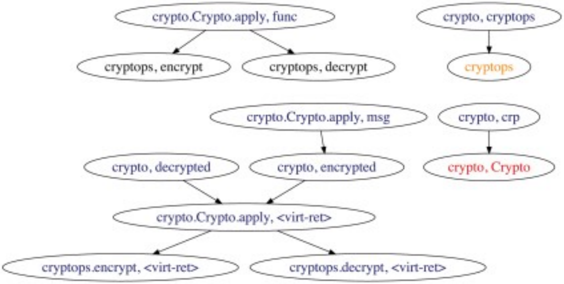


Figure 5. The scope tree of the *crypto* module

4

[4] Return to Code demo(Page 8), AST(Figure.4)

Assignment Graph



(b) The assignment graph of the `crypto` module.

Figure 6. The assignment graph of the *crypto* module

5

[5] Return to Code demo(Page 8), AST(Figure.4)

Contents

Analysis Rules

Demonstrating the state transition rules of analysis, the rules follow the form:

$$\langle \pi, s, n, h, E[e] \rangle \rightarrow \left\langle \pi', s', n', h', E\left[e'\right] \right\rangle \tag{1}$$

When having an expression e in the evaluation context E

$$\text{E-CTX}: \frac{\langle \pi, s, n, h, E[e] \rangle \hookrightarrow \left\langle \pi', s', n', h', e' \right\rangle}{\langle \pi, s, n, h, E[e] \rangle \rightarrow \left\langle \pi', s', n', h', E\left[e'\right] \right\rangle} \tag{2}$$

getObject(s, n, x)

The function iterates every element y of the namespace n in the reverse order, checking whether the element node y has any child matching the identifier x

addScope(s, n, x, t)

This function adds an edge from the node accessed by the path n to the target node given by the definition (x ,t)

getClassAttrObject(o, x, h)

This function deals with multiple inheritance, retrieving the object corresponding to the attribute x of the receiver object o, the analysis by examining the hierarchy of classes h

Python Source Code

main.py

```python
class A:
    def func():
        pass
class B:
    def func():
        pass
a=A()
b=B()
a.func()
b.func()
```

main.py

```python
class A:
    def func():
        pass

class B:
    def func():
        pass

class C(B,A):
    pass

c=C()
c.func()
```

Analysis Rules

COMPOUND

$$\frac{}{\langle \pi, s, n, h, E[o_1; o_2] \rangle \rightarrow \langle \pi, s, n, h, E[o_2] \rangle}$$

IDENT

$$\frac{o = \texttt{getObject}(s, n, x)}{\langle \pi, s, n, h, E[x] \rangle \rightarrow \langle \pi, s, n, h, E[o] \rangle}$$

ASSIGN

$$\frac{\begin{array}{c} s' = \texttt{addScope}(s, n, x, \textbf{var}) \\ o' = (n, (x, \textbf{var})) \qquad \pi' = \pi[o' \rightarrow \pi(o') \cup \{o\}] \end{array}}{\langle \pi, s, n, h, E[x := o] \rangle \rightarrow \langle \pi', s', n, h, E[o'] \rangle}$$

FUNC

$$\frac{\begin{array}{c} s' = \texttt{addScope}(s, n, x, \textbf{func}) \\ n' = n \cdot (x, \textbf{func}) \qquad s'' = \texttt{addScope}(s', n', \texttt{ret}, \textbf{var}) \\ s^{(3)} = \texttt{addScope}(s'', n', y, \textbf{var}) \end{array}}{\langle \pi, s, n, h, E[\textbf{function}\ x\ (y \dots)\ e] \rangle \rightarrow \langle \pi, s^{(3)}, n', h, E[e] \rangle}$$

RETURN

$$\frac{o' = (n \cdot x, (\texttt{ret}, \textbf{var})) \qquad \pi' = \pi[o' \rightarrow \pi(o') \cup \{o\}]}{\langle \pi, s, n \cdot x, h, E[\textbf{return}\ o] \rangle \rightarrow \langle \pi', s, n, h, E[o'] \rangle}$$

CALL

$$\frac{\begin{array}{c} o_1 = (n', (f, \textbf{func})) \\ o_2' = (n' \cdot f, (y, \textbf{var})) \qquad \pi' = \pi[o_2' \rightarrow \pi(o_2') \cup \{o_2\}] \end{array}}{\langle \pi, s, n, h, E[o_1(y = o_2 \dots)] \rangle \rightarrow \langle \pi', s, n, h, (n' \cdot f, (\texttt{ret}, \textbf{var})) \rangle}$$

Figure 7. Rules of the analysis (1)

Analysis Rules

CLASS

$$\frac{\begin{array}{c} s' = \mathtt{addScope}(s, n, x, \mathbf{cls}) \qquad t = \langle \mathtt{getObject}(\mathtt{s}, \mathtt{n}, \mathtt{b}) \mid b \in (y \ldots)\rangle \\ h' = h[(n, (x, \mathbf{cls})) \to t] \qquad n' = n \cdot (x, \mathbf{cls}) \end{array}}{\langle \pi, s, n, h, E[\mathbf{class}\ x\ (y \ldots)\ e]\rangle \to \langle \pi, s', n', h', E[e]\rangle}$$

ATTR

$$\frac{o' = \mathtt{getClassAttrObject}(o, x, h)}{\langle \pi, s, n, h, E[o.x]\rangle \to \langle \pi, s, c, h, E[o']\rangle}$$

NEW

$$\frac{\begin{array}{c} o_3 = \mathtt{getObject}(s, n, x) \\ o_2 = \mathtt{getClassAttrObject}(o_3, \_\_\mathtt{init}\_\_, h) \end{array}}{\langle \pi, s, n, h, E[\mathbf{new}\ x(y = o_1 \ldots)]\rangle \to \langle \pi, s, n, h, E[o_2(y = o_1 \ldots); o_3]\rangle}$$

ATTR-ASSIGN

$$\frac{o_3 = \mathtt{getClassAttrObject}(o_1, x, h) \qquad \pi' = \pi[o_3 \to \pi(o_3) \cup \{o_2\}]}{\langle \pi, s, n, h, E[o_1.x := o_2]\rangle \to \langle \pi', s, n, h, E[o_3]\rangle}$$

IMPORT

$$\frac{\begin{array}{c} o_2 = \mathtt{getObject}(\mathtt{s}, \mathtt{m}, \mathtt{x}) \qquad s' = \mathtt{addScope}(s, n, y, \mathbf{var}) \\ o_1 = (n, (y, \mathbf{var})) \qquad \pi' = \pi[o_1 \to \pi(o_1) \cup \{o_2\}] \end{array}}{\langle \pi, s, n, h, E[\mathbf{import}\ x\ \mathbf{from}\ m\ \mathbf{as}\ y]\rangle \to \langle \pi', s', n, h, E[o_1]\rangle}$$

ITER-ITERABLE

$$\frac{o' = \mathtt{getClassAttrObject}(o, \_\_\mathtt{next}\_\_, h)}{\langle \pi, s, n, h, E[\mathbf{iter}\ o]\rangle \to \langle \pi, s, n, h, E[o'()]\rangle}$$

ITER-GENERATOR

$$\frac{\mathtt{getClassAttrObject}(o, \_\_\mathtt{next}\_\_, h) = \text{undefined}}{\langle \pi, s, n, h, E[\mathbf{iter}\ o]\rangle \to \langle \pi, s, n, h, E[o()]\rangle}$$

Figure 8. Rules of the analysis (2)

Contents

Producing a call graph:

$$cg \in CallGraph = Obj \hookrightarrow P(Obj) \tag{3}$$

Algorithm (Figure.9) takes two elements as input:

1. a program $p \in Program$ of the model language whose syntax is shown in Figure.4.
2. the final state $\sigma \in State$ stemming from the analysis step.

Algorithm

```
Input  : p ∈ Program
         σ ∈ State
Output: cg ∈ CallGraph
foreach e in Program do
    while e ∉ Obj do
        ⟨σ, E[e]⟩ → ⟨σ′, E[e′]⟩
        if e′ = o1(y = o2 ...) then // Call Expression
            (π, s, n · f, h) ← σ′
            c ← getReachableFuns(π, o1)
            o3 ← getObject(s, n, f)
            cg ← cg[o3 → cg(o3) ∪ c] // Add Call Edges
        end
        e ← e′
    end
end
return cg
```

Figure 9. Algorithm for call graph construction.

Contents

Outlook

Evaluating the approach based on three research questions:

- RQ1 Is the proposed approach effective in constructing call graphs for Python programs?
- RQ2 How does the proposed approach stand in comparison with existing open-source, static-based approaches for Python?
- RQ3 What is the performance of our approach?

Setup

Experiments on a Debian 9 host with 16 CPUs and 16 GBs of RAM.

1. a micro-benchmark suite containing 112 minimal Python programs.
2. a macro-benchmark suite of five popular real-world Python packages.

Contents

Micro-benchmark

Category	#tests	Description
`parameters`	6	Positional arguments that are functions
`assignments`	4	Assignment of functions to variables
`built-ins`	3	Calls to built in functions and data types
`classes`	22	Class construction, attributes, methods
`decorators`	7	Function decorators
`dicts`	12	Hashmap with values that are functions
`direct calls`	4	Direct call of a returned function (`func()()`)
`exceptions`	3	Exceptions
`functions`	4	Vanilla function calls
`generators`	6	Generators
`imports`	14	Imported modules, functions classes
`kwargs`	3	Keyword arguments that are functions
`lambdas`	5	Lambdas
`lists`	8	Lists with values that are functions
`mro`	7	Method Resolution Order (MRO)
`returns`	4	Returns that are functions

Figure 10. Micro-benchmark for the suite categories.

Addressing Validity Threats

Asking two Python developers to rank the suite (from 1 to 10) based on the following criteria:

1. Completeness : Does it cover Python features?
2. Code Quality : Are the tests unique and minimal?
3. Description : Does the description adequately describe the given test case?

Micro-benchmark Suite Results

Category	PyCG		Pyan	
	Complete	Sound	Complete	Sound
`assignments`	4/4	3/4	4/4	4/4
`built-ins`	3/3	1/3	2/3	0/3
`classes`	22/22	22/22	6/22	10/22
`decorators`	6/7	5/7	4/7	3/7
`dicts`	12/12	11/12	6/12	6/12
`direct calls`	4/4	4/4	0/4	0/4
`exceptions`	3/3	3/3	0/3	0/3
`functions`	4/4	4/4	4/4	3/4
`generators`	6/6	6/6	0/6	0/6
`imports`	14/14	14/14	10/14	4/14
`kwargs`	3/3	3/3	0/3	0/3
`lambdas`	5/5	5/5	4/5	0/5
`lists`	8/8	7/8	3/8	4/8
`mro`	7/7	5/7	0/7	2/7
`parameters`	6/6	6/6	0/6	0/6
`returns`	4/4	4/4	0/4	0/4
Total	**111/112**	**103/112**	**43/112**	**36/112**

Figure 11. Micro-benchmark results for PyCG and Pyan. Depends is unsound in all cases and complete in 110/112 cases and is omitted.

Contents

Macro-benchmark

Project	LoC	Stars	Forks	Description
fabric	3,236	12.1k	1.8k	Remote execution & deployment
autojump	2,662	10.8k	530	Directory navigation tool
asciinema	1,409	7.9k	687	Terminal session recorder
face_classification	1,455	4.7k	1.4k	Face detection & classification
Sublist3r	1,269	4.4k	1.1k	Subdomains enumeration tool

Figure 12. Macro-benchmark suite project details.

Macro-benchmark Results

Project	Precision (%)			Recall (%)		
	PyCG	Pyan	Depends	PyCG	Pyan	Depends
autojump	99.5	66.5	99.2	68.2	28.5	22.5
fabric	98.3	-	100	61.9	-	6.3
asciinema	100	-	98.1	68	-	15.5
face_classification	99.5	86.8	96.2	89.7	7.6	5.7
Sublist3r	98.8	69.8	100	61.6	25.6	21.9
Average	**99.2**	**74.4**	**98.7**	**69.9**	**20.6**	**14.4**

Figure 13. Macro-benchmark results and tool comparison.

Contents

Time and Memory Performance

Command

UNIX command : *time*, *pmap* (average out of 20 runs)

Project	Time (sec)			Memory (MB)		
	PyCG	Pyan	Depends	PyCG	Pyan	Depends
autojump	0.76	0.42	2.37	62.7	37.8	27.1
fabric	0.77	-	1.83	60.9	-	18.5
asciinema	0.87	-	2	61.6	-	19.4
face_classification	0.92	0.38	2.49	60.9	35.3	25.6
Sublist3r	0.51	0.33	2.01	60	35.8	19.4
Average	**0.77**	**0.38**	**2.14**	**61.2**	**36.3**	**22**

Figure 14. Time and Memory Performance.

Contents

Case Study: A Fine-grained Tracking of Vulnerable Dependencies

Further, we show a potential application through the enhancement of GitHub's "security advisory"[5] notification service.

Cases contained functional vulerability

1. **PyYAML**[6] (versions before 5.1), a YAML parser affected by CVE-2017-18342 [7].
2. **Paramiko**[8] (multiple versions before 2.4.1), an implementation of the SSHv2 protocol affected by CVE-2018-7750 [9].

Results

- The vulnerable function in PyYAML(i.e., *load*) was invoked by 42/106 projects.
- In Paramiko, method (*start_server*) was not utilized at all by any of the 76 projects. We also observed that 12 projects did not invoke any library coming from Paramiko.

Contents

References (1)

Paper

[1] Vitalis Salis et al. "PyCG: Practical Call Graph Generation in Python". In: *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. 2021, pp. 1646–1657. DOI: 10.1109/ICSE43902.2021.00146.

Related Work and Additional Literature

[3] Matthias Felleisen, Robert Bruce Findler, and Matthew Flatt. *Semantics engineering with PLT Redex*. Mit Press, 2009.

[4] Magnus Madsen, Ondřej Lhoták, and Frank Tip. "A model for reasoning about JavaScript promises". In: *Proceedings of the ACM on Programming Languages* 1.OOPSLA (2017), pp. 1–24.

References (2)

Website Sources

Title page: drocheam. *About LaTeX Beamer Template in TH Koeln Style*. Online. https://github.com/drocheam/th-koeln-beamer-template

[5] *GitHub advisory database*. Online. https://github.com/advisories. 2020.

[6] *PyYAML: The next generation YAML parser and emitter for Python*. Online. https://github.com/yaml/pyyaml/. 2020.

[7] *CVE-2017-18342*. Online. https://nvd.nist.gov/vuln/detail/CVE-2017-18342. 2017.

[8] *Paramiko: The leading native Python SSHv2 protocol library*. Online. https://github.com/paramiko/paramiko/. 2020.

[9] *CVE-2018-7750*. Online. https://nvd.nist.gov/vuln/detail/CVE-2018-7750. 2018.

Thank you for your attention

Please feel free to ask any questions

Contact:
timhh991022@gmail.com