Serverless Computing and Python: Security Challenges and Ongoing Research

Speaker: Giuseppe Raffa

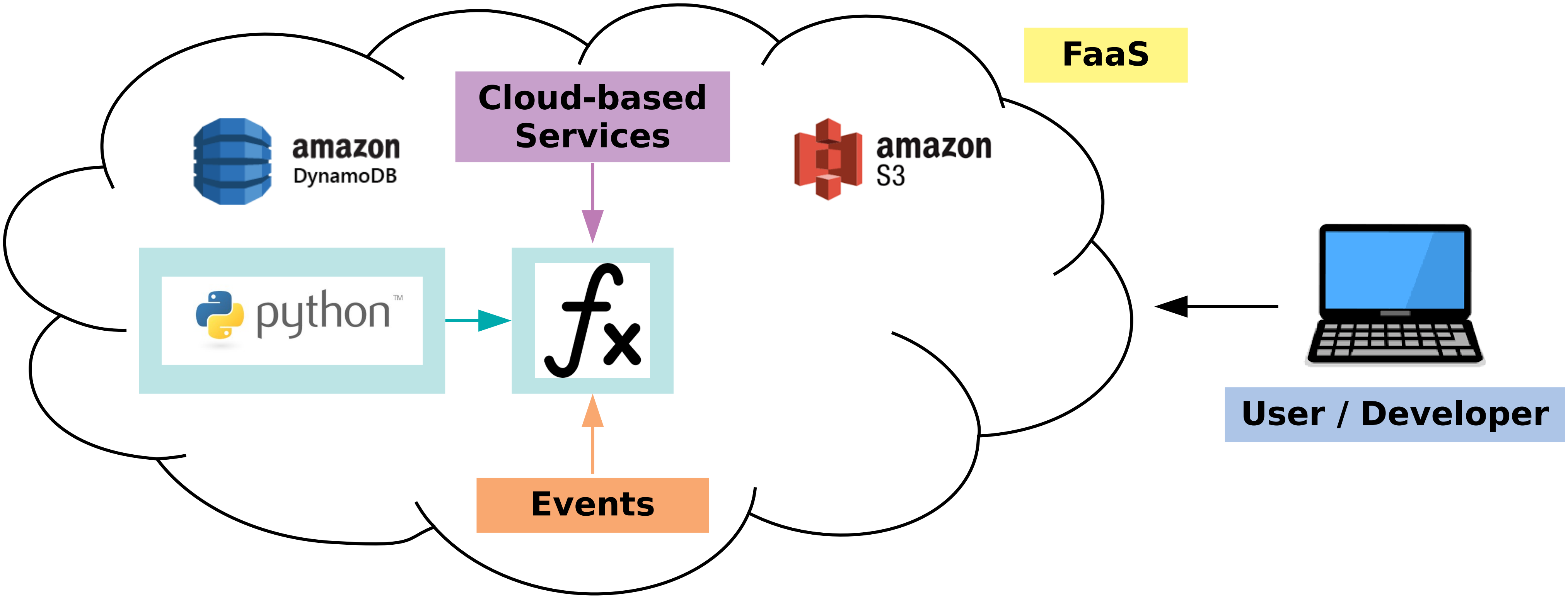
Co-authors: J. Blasco, D. O’Keeffe, S. K. Dash

PyCon Wroclaw Conference, 30th November 2024

Email: giuseppe.raffa.2018@live.rhul.ac.uk

ROYAL
HOLLOWAY
UNIVERSITY
Of LONDON

Serverless Computing Model



- **Advantage**
  - No infrastructure management
- **Challenge**
  - Security

Critical Risks for Serverless

- Risks identified by the Cloud Security Alliance

Function Event Data Injection

Broken Authentication

Insecure Serverless Deployment Config.

Over-Privileged Functions & Roles

Inadequate Function Monitoring

Insecure Third-Party Dependencies

Insecure Application Secrets Storage

DOS & Financial Resource Exhaustion

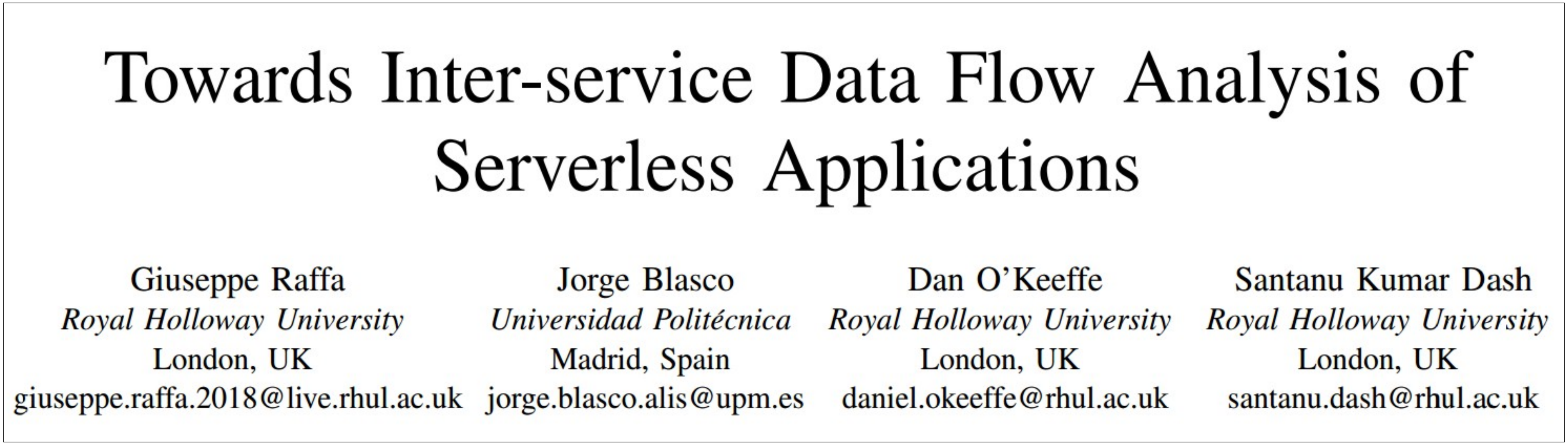
Business Logic Manipulation

Improper Exception Handling

Obsolete Functions, Resources & Events

Cross-Execution Data Persistency

SANER 2024 Paper



Towards Inter-service Data Flow Analysis of Serverless Applications

Giuseppe Raffa
Royal Holloway University
London, UK
giuseppe.raffa.2018@live.rhul.ac.uk

Jorge Blasco
Universidad Politécnica
Madrid, Spain
jorge.blasco.alis@upm.es

Dan O'Keeffe
Royal Holloway University
London, UK
daniel.okeeffe@rhul.ac.uk

Santanu Kumar Dash
Royal Holloway University
London, UK
santanu.dash@rhul.ac.uk

SANER 2024 Early Research Achievement (ERA) Track

https://github.com/giusepperaffa/serverless-security-microbenchmarks

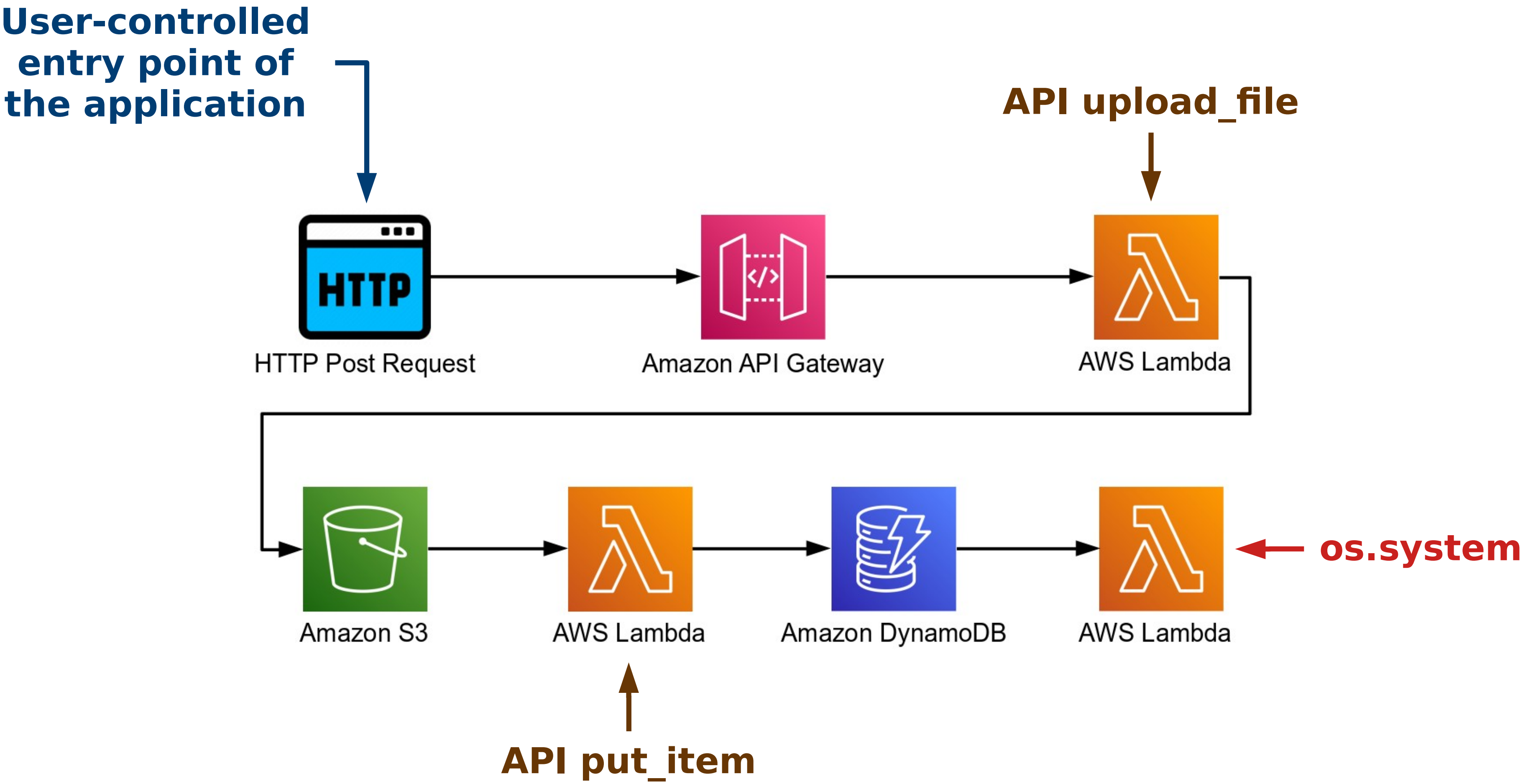
Motivation & Challenges

- **Why static data flow analysis?**
  - Most of serverless security tools rely on dynamic analysis
  - Static analysis is an effective supplement
- **What are the challenges?**
  - Information from infrastructure and application code
  - Variety of sources and events
  - Black-box nature of platform services
- **Our work**

Suite of security-oriented microbenchmarks

Approach to detecting security-sensitive data flows

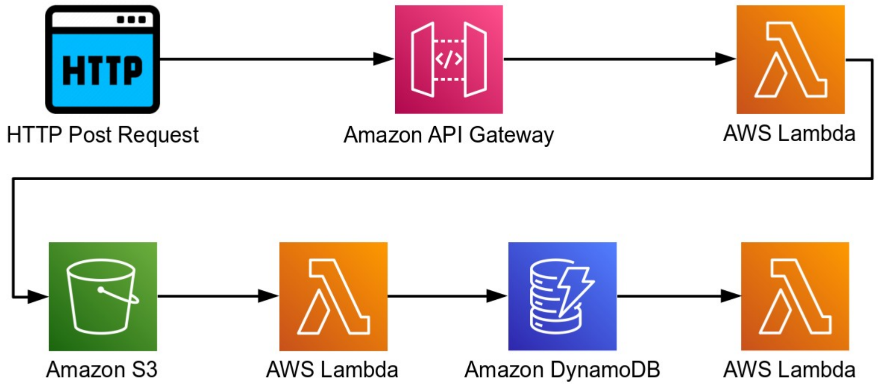
Motivating Example



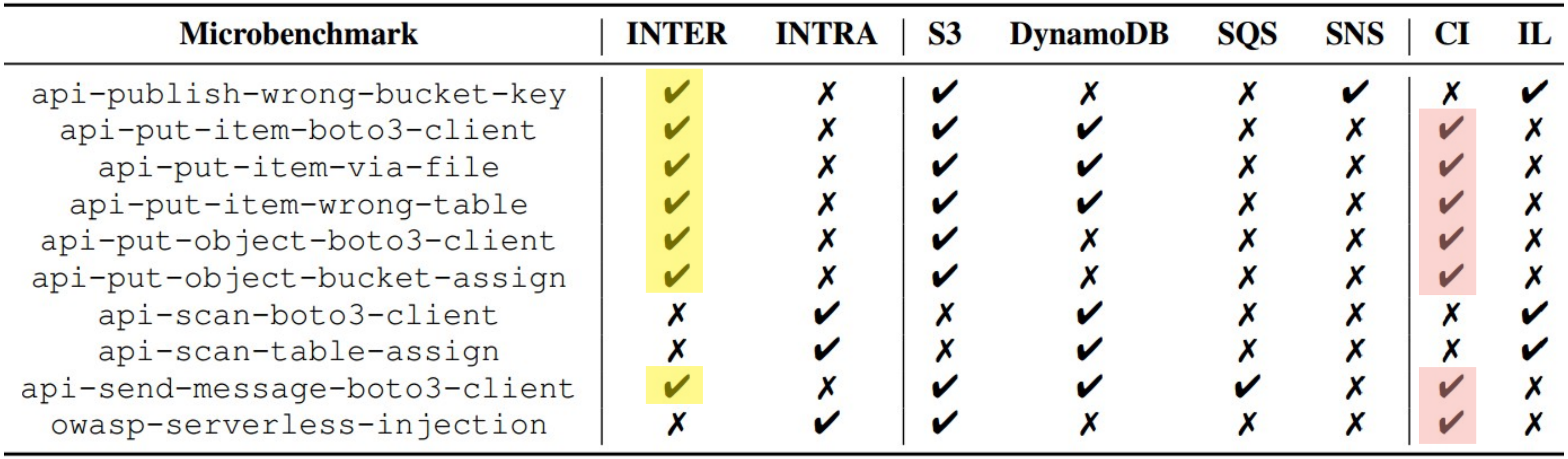
A general-purpose static analysis tool would ignore the triggered events

Microbenchmarks Suite

- **Design approach**
  - Code injection and information leakage vulnerabilities
  - AWSomePy dataset characterization

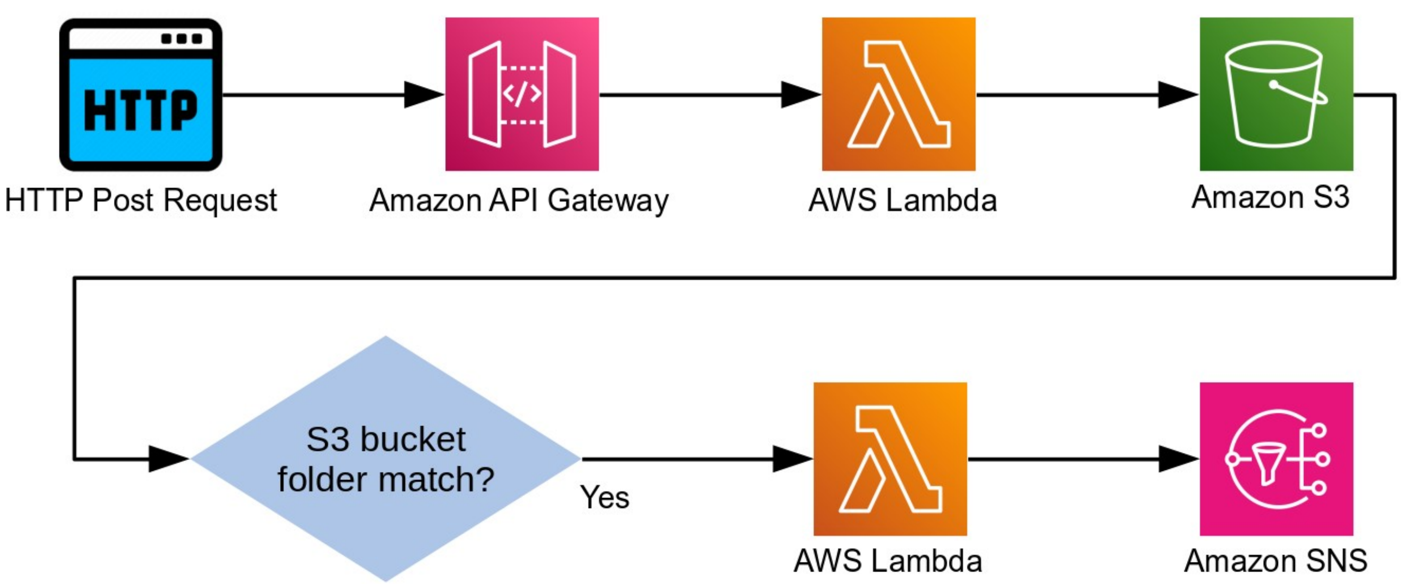


- **Summary**



	Flow		Services				Vuln.	
Microbenchmark	INTER	INTRA	S3	DynamoDB	SQS	SNS	CI	IL
api-publish-wrong-bucket-key	✔	✗	✔	✗	✗	✔	✗	✔
api-put-item-boto3-client	✔	✗	✔	✔	✗	✗	✔	✗
api-put-item-via-file	✔	✗	✔	✔	✗	✗	✔	✗
api-put-item-wrong-table	✔	✗	✔	✔	✗	✗	✔	✗
api-put-object-boto3-client	✔	✗	✔	✗	✗	✗	✔	✗
api-put-object-bucket-assign	✔	✗	✔	✗	✗	✗	✔	✗
api-scan-boto3-client	✗	✔	✗	✔	✗	✗	✗	✔
api-scan-table-assign	✗	✔	✗	✔	✗	✗	✗	✔
api-send-message-boto3-client	✔	✗	✔	✔	✔	✗	✔	✗
owasp-serverless-injection	✗	✔	✔	✗	✗	✗	✔	✗

Information Leakage Example



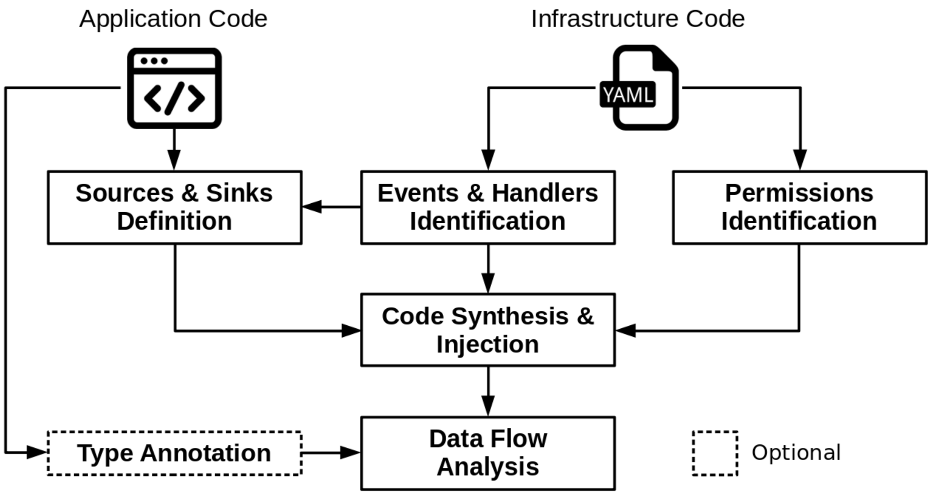
User-controlled entry point of the application

Request inspection and file uploaded to a S3 bucket

Email notification with user-provided information

Prototype Analysis Framework

- **Analysis approach**
  - Infrastructure and application code processed
  - Code instrumented to obtain synchronous equivalent



Application Code
Infrastructure Code
YAML
Sources & Sinks Definition
Events & Handlers Identification
Permissions Identification
Code Synthesis & Injection
Type Annotation
Data Flow Analysis
Optional

- **Implementation**
  - Code modified semi-automatically
  - Data flow analysis with Pysa
- **Evaluation**

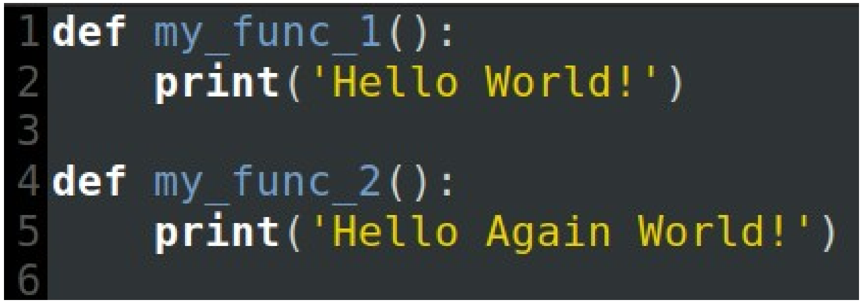
7 true positives

2 false positives

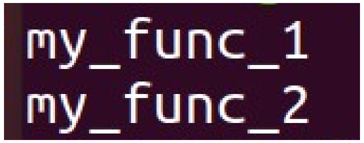
1 false negative

AST-based Processing (1)

- **Extraction of function names**

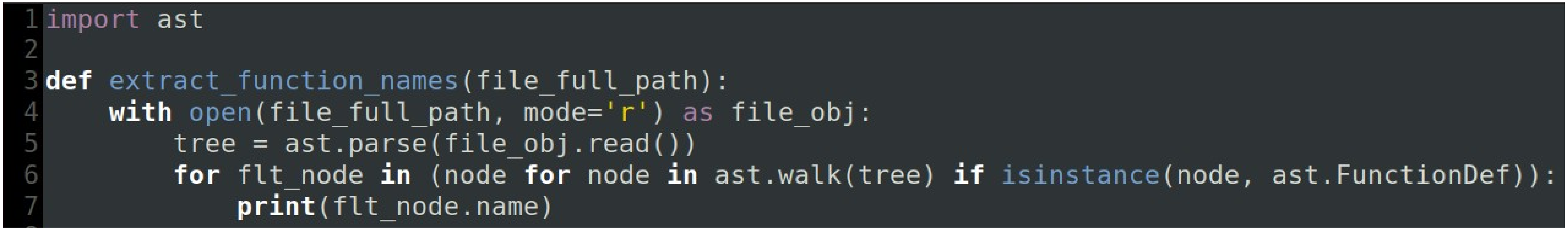


```python
def my_func_1():
    print('Hello World!')

def my_func_2():
    print('Hello Again World!')

```



```
my_func_1
my_func_2
```

- **Implementation**



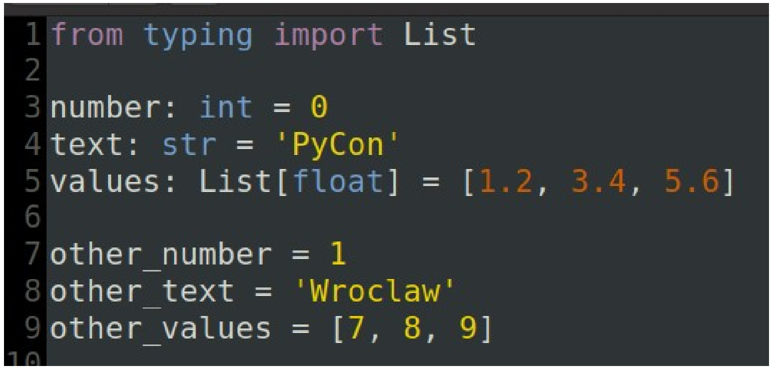
```python
import ast

def extract_function_names(file_full_path):
    with open(file_full_path, mode='r') as file_obj:
        tree = ast.parse(file_obj.read())
        for flt_node in (node for node in ast.walk(tree) if isinstance(node, ast.FunctionDef)):
            print(flt_node.name)
```

Create in-memory data structure with AST

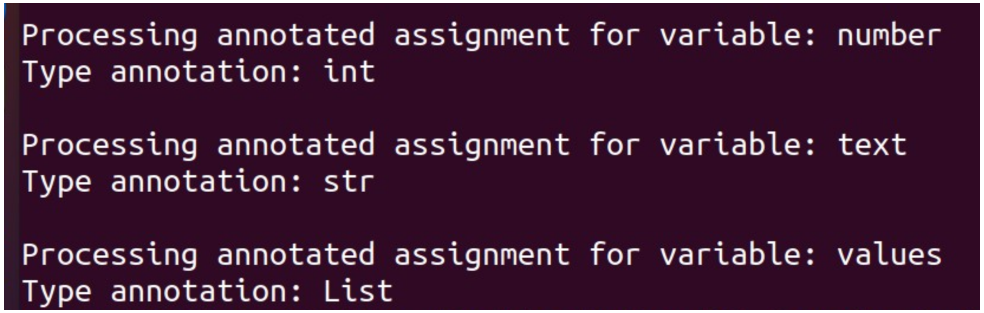
AST nodes inspection (ast.FunctionDef)

AST-based Processing (2)

- **Extraction of type annotations**

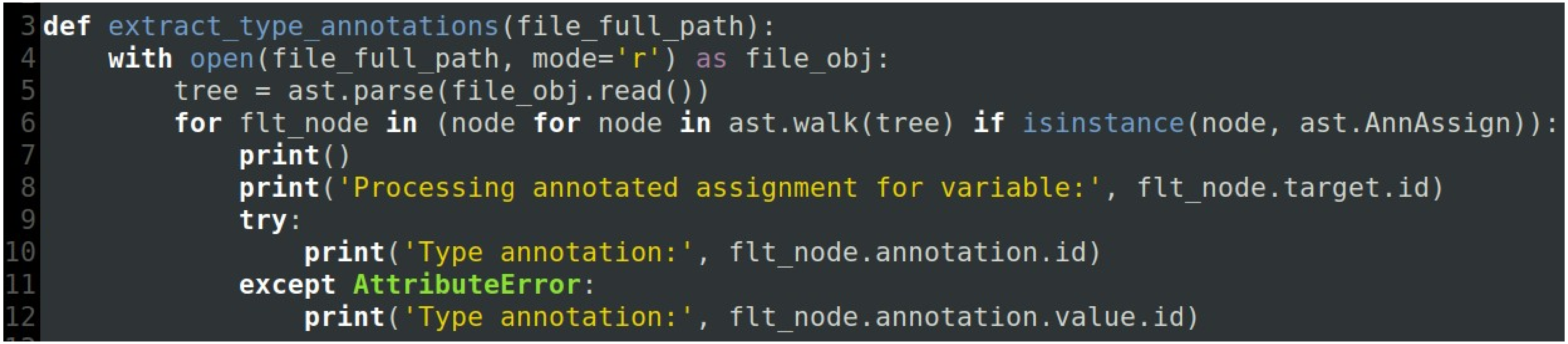


```python
from typing import List

number: int = 0
text: str = 'PyCon'
values: List[float] = [1.2, 3.4, 5.6]

other_number = 1
other_text = 'Wroclaw'
other_values = [7, 8, 9]
```



```
Processing annotated assignment for variable: number
Type annotation: int

Processing annotated assignment for variable: text
Type annotation: str

Processing annotated assignment for variable: values
Type annotation: List
```

- **Implementation**

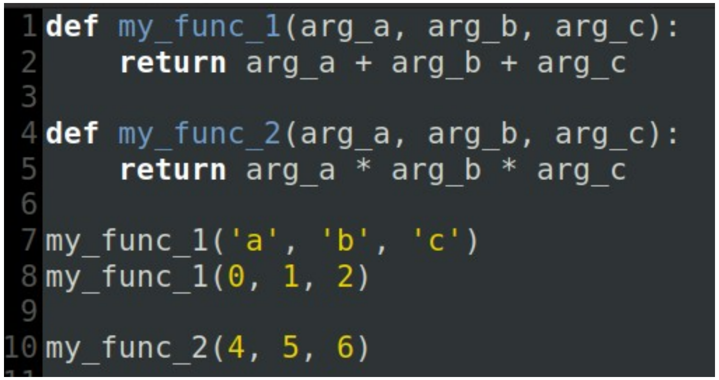


```python
def extract_type_annotations(file_full_path):
    with open(file_full_path, mode='r') as file_obj:
        tree = ast.parse(file_obj.read())
        for flt_node in (node for node in ast.walk(tree) if isinstance(node, ast.AnnAssign)):
            print()
            print('Processing annotated assignment for variable:', flt_node.target.id)
            try:
                print('Type annotation:', flt_node.annotation.id)
            except AttributeError:
                print('Type annotation:', flt_node.annotation.value.id)
```

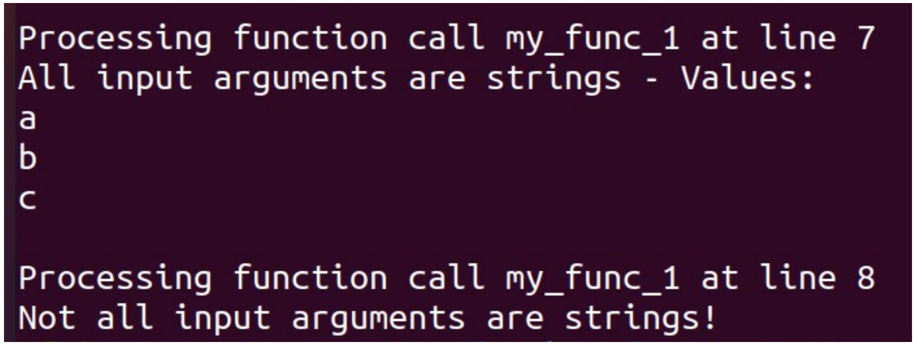
AST nodes inspection (ast.AnnAssign)

AST-based Processing (3)

- **Processing of function call arguments**

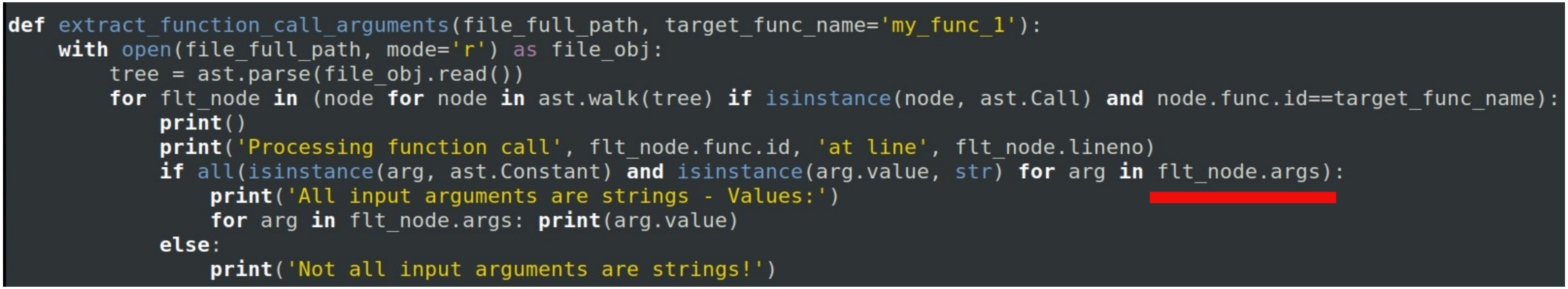


```
 1 def my_func_1(arg_a, arg_b, arg_c):
 2     return arg_a + arg_b + arg_c
 3
 4 def my_func_2(arg_a, arg_b, arg_c):
 5     return arg_a * arg_b * arg_c
 6
 7 my_func_1('a', 'b', 'c')
 8 my_func_1(0, 1, 2)
 9
10 my_func_2(4, 5, 6)
```



```
Processing function call my_func_1 at line 7
All input arguments are strings - Values:
a
b
c

Processing function call my_func_1 at line 8
Not all input arguments are strings!
```

- **Implementation**

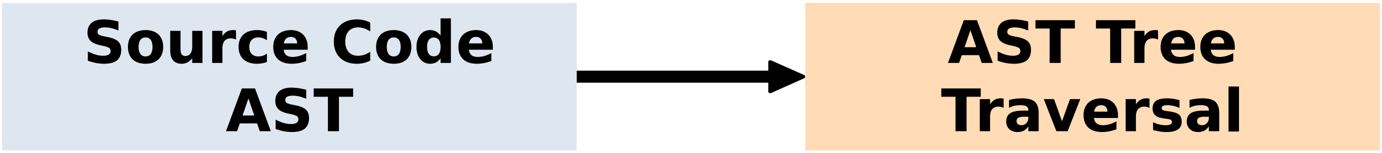


```
def extract_function_call_arguments(file_full_path, target_func_name='my_func_1'):
    with open(file_full_path, mode='r') as file_obj:
        tree = ast.parse(file_obj.read())
        for flt_node in (node for node in ast.walk(tree) if isinstance(node, ast.Call) and node.func.id==target_func_name):
            print()
            print('Processing function call', flt_node.func.id, 'at line', flt_node.lineno)
            if all(isinstance(arg, ast.Constant) and isinstance(arg.value, str) for arg in flt_node.args):
                print('All input arguments are strings - Values:')
                for arg in flt_node.args: print(arg.value)
            else:
                print('Not all input arguments are strings!')
```

AST nodes inspection (ast.Call)

AST-based Processing (4)

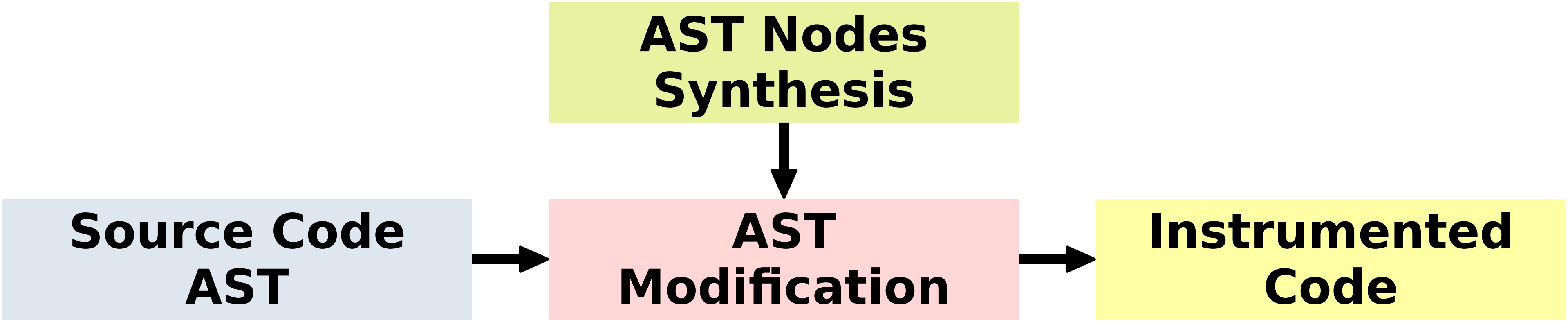
- Extraction of information



Source Code AST

AST Tree Traversal

- Source code modification



AST Nodes Synthesis

Source Code AST

AST Modification

Instrumented Code

Conclusion

- **Key takeaways**

Security-sensitive data flows

New suite of microbenchmarks

Studied approach is feasible

- **Stay tuned**
  - This is ongoing research
  - Scan the QR code to know more

giuseppe.raffa.2018@live.rhul.ac.uk