Static Code Analysis on Networking Code:

Identifying the capabilities of finding implementation flaws using Abstract Syntax Trees

RP1 *4th of July, 2019*

Presenter: **Ivar Slotboom**, SNE/UvA

Supervisor: **Wouter van Dongen**, DongIT

Static code analysis

- Find bugs and performance issues.
- Produce a report providing feedback and improvement points.
- Often powered by machine learning.

Abstract syntax trees (AST)

- **Break down** static code into nodes.
- AST output is a **structure** on how the code is read by the interpreter.
- Nodes tree where you can **traverse** through its child and parent nodes.

```
39    Assign(targets=[<_ast.Name object at 0x7fb03dbfcc18>], value=Call(func=Attribute(value
40      Name(id='s', ctx=Store())
41        Store()
42      Call(func=Attribute(value=Name(id='socket', ctx=Load()), attr='socket', ctx=Load()),
43        Attribute(value=Name(id='socket', ctx=Load()), attr='socket', ctx=Load())
44          Name(id='socket', ctx=Load())
45            Load()
46          Load()
47        Attribute(value=Name(id='socket', ctx=Load()), attr='AF_INET', ctx=Load())
48          Name(id='socket', ctx=Load())
49            Load()
50          Load()
```

Research question

Is it possible to create a tool to **analyze static Python code** to detect potential network implementation flaws?

How can network implementation flaws be detected using Abstract Syntax Trees?

What are the limitations of identifying network implementation issues using Abstract Syntax Trees?

Related work

Al Bessey et al.

- Static Code Analysis done preferably:
  - Minimal manual setup.
  - Maximum serious issues.
  - Minimum false positives.
- Making an analyzer is an iterative process.
- Best reports come when all context is available.
- No code equals to no error.

Tasnim and Rahman

- ASTs do not describe every detail of the syntax, but enough to identify patterns and flaws.

Goseva-Popstojanova et al.

- Researched the capabilities of static code analysis.
- Not very effective in detecting security vulnerabilities.
- Sees opportunity to be more effective than manual inspection.

Methodology

Iterative process to create an analyzer, as well as test projects to test the analyzer on.

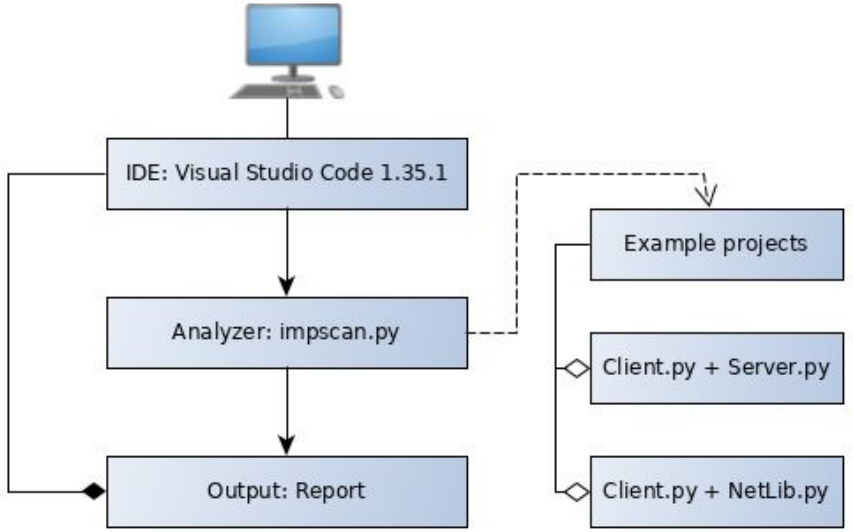
Analyzer:

- Uses AST to parse the test project in question.
- Uses predefined rulesets to spot implementation flaws.

Test projects:

- Purposefully implement network flaws.
- Simulate real-world scenarios.

All code publically available on GitHub.



Results

AST parsing is an effective method

Network implementation flaws are usually implemented on a higher level. This makes it easier to discover for the analyzer.

- It is important that the rules are well defined.
- It is possible to traverse the node tree backwards to find out what happened.

```
### Report ###
Errors: 1
        Socket "s" could be sending infinite amount of bytes because of its latest buffer assignment: sys.stdin.readline
Warnings: 1
        Socket connectivity is not configured for IPv6 connections
```

```
42                sys.stdout.write("Say: ")
43                sys.stdout.flush()
44                str_send = aText
45                str_send = sys.stdin.readline()
46                str_send = str_send.encode('utf-8')
47                s.send(str_send)
```

Multi-file projects

AST parsing does not mind merging two files into one. The analytical results stay the same.

```
1  import sys
2  import netlib
3
4  def OnReceive(aMessage):
5      sys.stdout.write("\r{}".format(aMessage))
6      sys.stdout.write("Say: ")
7      sys.stdout.flush()
8
9  netlib.Initialize(True, OnReceive)
10
11 try:
12     while True:
13         sys.stdout.write("Say: ")
14         sys.stdout.flush()
15         str_send = sys.stdin.readline()
16         netlib.Send(str_send)
17 except:
18     netlib.Exit()
```

+

```
1  import socket
2  import sys
3  from threading import Thread, Lock
4  import time
5  import os
6
7  # <Some variables>
8
9  def Initialize(aIsClient, aRecvCallbackFunc):...
27
28 def NetlibMain():...
32
33 # Funcs
34 def Connect():...
41
42 def Exit(aSocket = None):...
50
51 def Send(aMessage):...
61
62 def Receive(aRecvCallbackFunc):...
```

Limitation 1: Threading

- Causes unique, unpredictable behaviour.
- Can only be checked on run time.
- May alter context that is required for analysis.
- Some rules cannot be checked because of run time requirements, e.g. socket dtors.

```
# Semi-Python pseudo code - assume imports and sockets were handled
stringToSend = ""

def ThreadFunc1():
    global stringToSend
    while (True):
        time.sleep(3)
        stringToSend = "NewTextAssignment"

def ThreadFunc2():
    global stringToSend
    while (True):
        stringToSend += "1234567890!"

t1 = Thread(target=ThreadFunc1)
t2 = Thread(target=ThreadFunc2)

t1.start()
t2.start()

time.sleep(5) # Let the threads run for a while
socket.send(stringToSend) # How big is this string?
```

Limitation 2: Imports

Imports can be confused due to the nature of the Python language. How can we separate installed libraries from files?

1. Use heuristics, check if file exists in the directory.
2. Parse installed libraries to match alias.

Either way, context is lost.

```
1  # Import using existing library
2  import sys # imports the installed sys library
3  # AST:
4  #    Import(names=[<_ast.alias object at 0
       x0000020C212CFAC8>])
5  #    alias(name='sys', asname=None)
6
7  # Import using file in the same directory
8  import netlib # imports the netlib.py in the same
       directory
9  # AST:
10 #    Import(names=[<_ast.alias object at 0
       x0000020C212CFB70>])
11 #    alias(name='netlib', asname=None)
```

Limitation 3: Implementing rule definitions

- Every rule needs to traverse the node tree.
- Larger code bases have millions of lines of code.
- Alias names can be changed when used as arguments in functions.

Overall: Very costly per rule definitions. May not scale well with larger codebases.

Limitation 4: Dead code is still parsed

- “No code = no error”, but dead code could also lead to false reports.
- Could alter context wrongly as code may not always be called.
- Functions can be called based on runtime scenarios.

```
5  def DeadCode(aStuff):
6      v6sock = socket.socket(socket.AF_INET6, socket.SOCK_STREAM)
7      v6sock.send(aStuff)
8
```

```
342  FunctionDef(name='DeadCode', args=arguments(args=[],
343  arguments(args=[], vararg=None, kwonlyargs=[], kw_de
344  Assign(targets=[<_ast.Name object at 0x7fb41d733a20>
345  Name(id='v6sock', ctx=Store())
346  Store()
347  Call(func=Attribute(value=Name(id='socket', ctx=Load
348  Attribute(value=Name(id='socket', ctx=Load()), attr=
349  Name(id='socket', ctx=Load())
```

```
Sockets are implemented for both IPv4 and IPv6.

### Report ###
```

Conclusion

It is **possible** to detect network implementation flaws using an **AST**, but **limitations** make it difficult to make it scalable and confident.

How can network implementation flaws be detected using ASTs?

- Network implementation issues commonly are implemented on a high level.
- Node traversal can give context on the implementation in question.
- ASTs are not hindered by moved code.
- Iterative process as solutions to one bug could allow others to be found.

What are the limitations of using ASTs to identify network implementation flaws?

- Static code versus run time code could hinder context during analysis.
- Imports are difficult to identify, which also affects the context of the analysis.
- Rule definitions are difficult to implement.
- Dead code could be altering context, or is hard to analyze itself.

Future work

Machine learning?

- Commonly used in static code analysis for bugs and performance issues.
- Could potentially find patterns and behaviour in network implementation flaws.

Solution to dead code?

- How can you identify dead code in runtime environments?
- Is it possible to simulate runtime environments when analyzing static code?

Lower level languages?

- Require more detail to function, e.g. C/C++.
- Usually have projects with larger code bases.
- Could improve context from the output of AST, causing less confusion such as imports.

Thank you for your time.

Questions?