3.9.5. eval(): Security

Fig. 3.9.5.1 Image by Tumisu from Pixabay

Note

Outline

1. eval()
    a. Security
        i. Ex1a: Delete File
        ii. Ex1b: __import__()
        iii. Ex2a: Limit: globals+locals
        iv. Ex2b: Limit: Build-in Names
        v. Ex3a: Secure Steps
        vi. Ex3b: Secure Steps 2
    b. Ex10: Math Evaluator

Note

Roadmap

1. This topic: TryExcept

> myBlock
>
> | Exception Handling | | | | |
> |---|---|---|---|---|
> | try Statement | except Statement | finally Statement | Error Type | with Statement |

2. Course: Python 1
3. Subject: Programming
4. Field
    a. Software Engineering (SE)
    b. Computer Science and Information Engineering (CSIE)
    c. Electrical/Electronics Engineering (EE)

3.9.5.1. eval()

1. The eval() function in Python is used to evaluate a dynamically created Python expression, which can be a string containing Python code.
2. However, be cautious when using eval(), as it can execute arbitrary code and potentially introduce security risks if used with untrusted input.
3. Syntax

```
eval(expression[, globals[, locals]])
```

4. To evaluate a string-based expression, Python's eval() runs the following steps:
    a. Parse expression
    b. Compile it to bytecode
    c. Evaluate it as a Python expression
    d. Return the result of the evaluation

Important

Security

1. eval() is considered insecure because it allows you (or your users) to dynamically execute arbitrary Python code.
2. For this reason, good programming practices generally recommend against using eval().

3.9.5.1.1. Security

3.9.5.1.1.1. Ex1a: Delete File

1. Code+Output

Code | Output

Listing 3.9.5.1.1.1.1 /src/Eval+Repr/Security/EvalDelFile/__init__.py

```
'''
author: cph
since:  20230827
'''
import subprocess

print(f'Step 1: List of files BEFORE delete operation...')
sCmd = 'dir'
print(eval("subprocess.getoutput(sCmd)"))   # List of files

print(f'Step 2: Delete a file, [ToDeleteFile.txt]...')
sCmd = 'del ToDeleteFile.txt'
print(eval("subprocess.getoutput(sCmd)"))   # Delete a file

print(f'Step 3: List of files AFTER delete operation...')
sCmd = 'dir'
print(eval("subprocess.getoutput(sCmd)"))   # List of files
```

3.9.5.1.1.2. Ex1b: __import__()

1. Code+Output

Code | Output

Listing 3.9.5.1.1.2.1 /src/Eval+Repr/Security/EvalImport/__init__.py

```
'''
author: cph
since:  20230827
'''
sImport = '__import__("subprocess")'

print(f'Step 1: List of files BEFORE delete operation...')
sCmd = 'dir'
print(eval(f"{sImport}.getoutput(sCmd)"))   # List of files

print(f'Step 2: Delete a file, [ToDeleteFile.txt]...')
sCmd = 'del ToDeleteFile.txt'
print(eval(f"{sImport}.getoutput(sCmd)"))   # Delete a file

print(f'Step 3: List of files AFTER delete operation...')
sCmd = 'dir'
print(eval(f"{sImport}.getoutput(sCmd)"))   # List of files
```

3.9.5.1.1.3. Ex2a: Limit: globals+locals

1. Code+Output

Code | Output

Listing 3.9.5.1.1.3.1 /src/Eval+Repr/Security/EvalLimitGlobalsLocals/__init__.py

```
'''
author: cph
since:  20190102
'''
import math

x = 4
print(eval('x * x', {'x': x}))

x = 5
print(eval('x * x', {}, {}))
```

3.9.5.1.1.4. Ex2b: Limit: Built-in Names

1. Code+Output

Code Output

Listing 3.9.5.1.1.4.1 /src/Eval+Repr/Security/EvalLimitBuiltinNames/__init__.py

```
'''
author: cph
since:  20190102
'''
print(eval('__import__("math").sqrt(5)'))
print(eval('__import__("math").sqrt(5)',
           {"__builtins__": {}}, {}))
```

3.9.5.1.1.5. Ex3a: Secure Steps

1. A possible solution is listed as below.

    a. Step 1: Create a dictionary containing the names that we want to use with eval().

    b. Step 2: Compile the input string to bytecode using compile() in mode “eval”.

    c. Step 3: Check .co_names on the bytecode object to make sure it contains only allowed names.

    d. Step 4: Raise a NameError if the user tries to enter a name that is not allowed.

2. Code+Output

Code Output

Listing 3.9.5.1.1.5.1 /src/Eval+Repr/Security/EvalMoreSecure/__init__.py

```
'''
author: cph
since:  20190102
'''
def evalExpression(sIn):
    # Step 1: Put all allowed variable names in the dictionary
    dAllowNames = {"x": x}

```

```
    print(f'dAllowNames: {dAllowNames}')

    # Step 2: Use compile() to eval()
    oCode = compile(sIn, "<string>", "eval")
    print(f'oCode: {oCode}')
    print(f'oCode.co_names: {oCode.co_names}')

    # Step 3: Check .co_names
    for n in oCode.co_names:        # iterate all names
        if n not in dAllowNames:    # NOT in dictionary
            # Step 4: Raise error and found illegal variable name
            raise NameError(f'Use of "{n}" not allowed')
    return eval(oCode, {"__builtins__": {}}, dAllowNames)

if __name__ == '__main__':
    x = 4
    sIn = 'x * x'
    print(evalExpression(sIn))

    print()
    sIn = 'x * y'
    print(evalExpression(sIn))
```

3.9.5.1.1.6. Ex3b: Secure Steps 2

1. Use the same steps for different example.
2. Code+Output

Code | **Output**

Listing 3.9.5.1.1.6.1 /src/Eval+Repr/Security/EvalMoreSecure2/__init__.py

```
'''
author: cph
since:  20190102
'''
def evalExpression(sIn):
    # Step 1: Put all allowed variable names in the dictionary
    dAllowNames = {'sum': sum,
                   'max': max}
    print(f'dAllowNames: {dAllowNames}')

    # Step 2: Use compile() to eval()
    oCode = compile(sIn, "<string>", "eval")
    print(f'oCode.co_names: {oCode.co_names}')

    # Step 3: Check .co_names
    for n in oCode.co_names:        # iterate all names
        if n not in dAllowNames:    # NOT in dictionary
            # Step 4: Raise error and found illegal variable name
            raise NameError(f'Use of "{n}" not allowed')
    return eval(oCode, {"__builtins__": {}}, dAllowNames)

if __name__ == '__main__':
    sIn = ['sum([4, 3, 2, 1])',
           'max([4, 3, 2, 1])',
           'sorted([4, 3, 2, 1])',
           ]

    for s in sIn:
        print(evalExpression(s))
        print()
```

3.9.5.1.2. Ex10: Math Evaluator

1. Code+Output

Code: __init__.py | Code: mathrepl.py | Output

Listing 3.9.5.1.2.1 /src/Eval+Repr/MathEvaluator/__init__.py

```
'''
author: cph
since:  20230827
ref:    https://realpython.com/python-eval-function/
'''
import mathrepl as mr

def evaluate(expression):
    # Compile the expression
    code = compile(expression, "<string>", "eval")

    # Validate allowed names
    for name in code.co_names:
        if name not in mr.ALLOWED_NAMES:
            raise NameError(f"The use of '{name}' is not allowed")

    return eval(code, {"__builtins__": {}}, mr.ALLOWED_NAMES)

if __name__ == "__main__":
    # Step 1: Show messages
    print(mr.WELCOME)
    mr.USAGE = mr.USAGE_HEADER + '\n' + mr.usageKeys(list(mr.ALLOWED_NAMES.keys()))

```

```
24
25      while True:
26          # Step 2: Read input
27          try:
28              expression = input(f"{mr.PS1} ")
29          except (KeyboardInterrupt, EOFError):
30              raise SystemExit()
31
32          # Step 3: Handle special commands
33          if expression.lower() == "help":
34              print(mr.USAGE)
35              continue
36          if expression.lower() == "version":
37              print(mr.__version__)
38              continue
39          if expression.lower() == "exit":
40              raise SystemExit()
41
42          # Step 4: Evaluate expression and handle errors
43          try:
44              result = evaluate(expression)
45          except SyntaxError:        # Invalid expression
46              print("Invalid input expression syntax")
47              continue
48          except NameError as err:   # NOT allowed command
49              print(err)
50              continue
51          except ValueError as err:  # Invalid value
52              print(err)
53              continue
54
55          # Step 5: Print result
            print(f"The result is: {result}")
```

Important

Summary: eval()

1. Remember that using eval() with untrusted or unsanitized input can be dangerous, as it can execute arbitrary code.
2. Be cautious and avoid using it with input from untrusted sources.

See also

Reference

1. Leodanis Pozo Ramos, Python eval(): Evaluate Expressions Dynamically, RealPython, 20230801. [Web]

Note

1. Start: 20170719
2. System Environment:

Listing 3.9.5.1.2.4 requirements.txt

```
sphinx==7.1.2                           # Sphinx
graphviz>=0.20.1                        # Graphviz
sphinxbootstrap4theme>=0.6.0            # Theme: Bootstrap
sphinx-material>=0.0.35                 # Theme: Material
sphinxcontrib-plantuml>=0.25            # PlantUML
sphinxcontrib.bibtex>=2.5.0             # Bibliography
sphinx-autorun>=1.1.1                   # ExecCode: pycon
sphinx-execute-code-python3>=0.3        # ExecCode
btd.sphinx.inheritance-diagram>=2.3.1   # Diagram
sphinx-copybutton>=0.5.1                # Copy button
sphinx_code_tabs>=0.5.3                 # Tabs
sphinx-immaterial>=0.11.3               # Tabs

#-----------------------------------------------------
#-- Library Upgrade Error by Library Itself
#   >> It needs to fix by library owner
#   >> After fixed, we need to try it later
#-----------------------------------------------------
pydantic==1.10.10                       # 2.0: sphinx compiler error, 20230701

#-----------------------------------------------------
#-- Minor Extension
#-----------------------------------------------------
sphinxcontrib.httpdomain>=1.8.1         # HTTP API

#sphinxcontrib-blockdiag>=3.0.0         # Diagram: block
#sphinxcontrib-actdiag>=3.0.0           # Diagram: activity
#sphinxcontrib-nwdiag>=2.0.0            # Diagram: network
#sphinxcontrib-seqdiag>=3.0.0           # Diagram: sequence

#-----------------------------------------------------
#-- Still Wait For Upgrading Version
#-----------------------------------------------------

#-----------------------------------------------------
#-- Still Under Testing
#-----------------------------------------------------
#numpy>=1.24.2                          # Figure: numpy

#-----------------------------------------------------
#-- NOT Workable
#-----------------------------------------------------
#sphinxcontrib.jsdemo==0.1.4    # ExecCode: Need replace add_js_file()
#jupyter-sphinx==0.4.0          # ExecCode: Need gcc compiler
#sphinxcontrib.slide==1.0.0     # Slide: Slideshare
#hieroglyph==2.1.0              # Slide: make slides
#matplotlib>=3.7.1              # Plot: Need Python >= v3.8
#manim==0.17.2                  # Diagram: scipy, numpy need gcc
#sphinx_diagrams==0.4.0         # Diagram: Need GKE access
#sphinx-tabs>=3.4.1                     # Tabs: Conflict w/ sphinx-material
```