Testing python security

by

Jose Manuel Ortega

PYCONES 2018 MÁLAGA

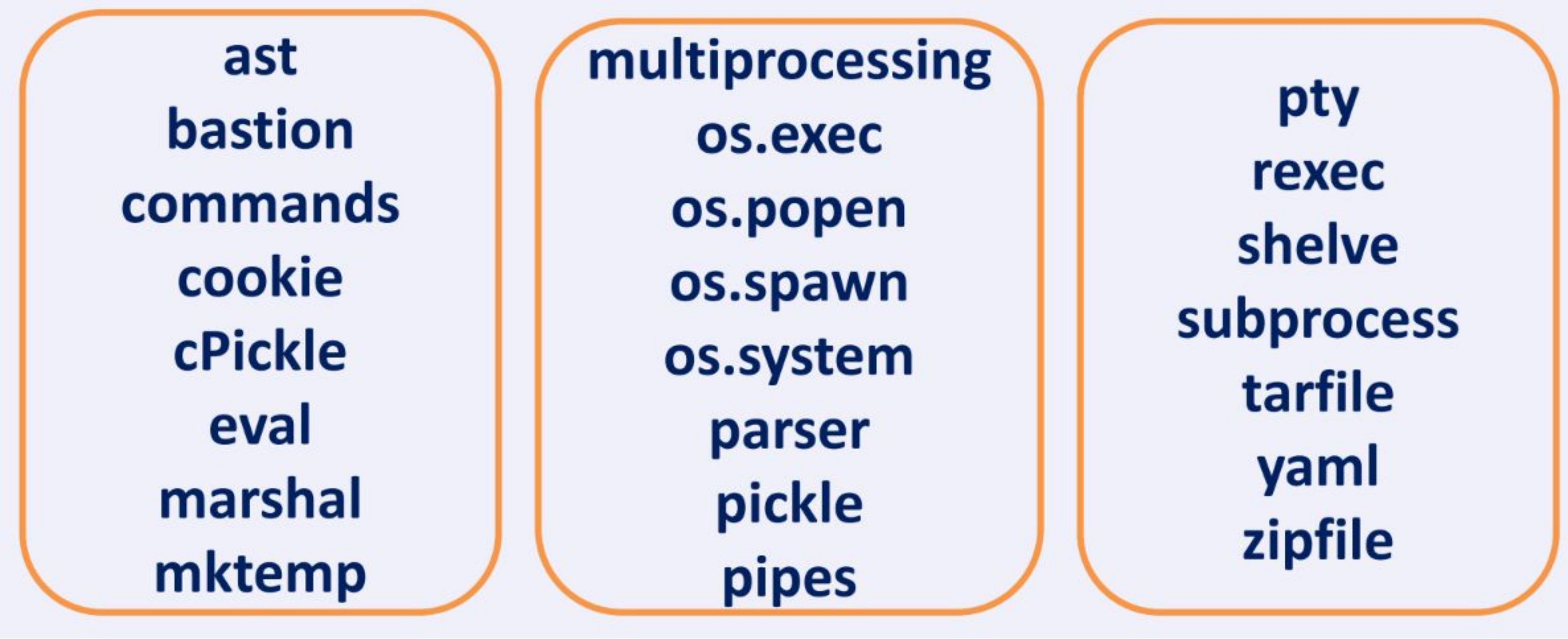
Agenda

1. Secure coding
2. Dangerous functions
3. Common attack vectors
4. Static analisys tools
5. Other security issues

Secure coding

1. Analysis of architectures involved
2. Review of implementation details
3. Verification of code logic and syntax
4. Operational testing (unit testing, white-box)
5. Functional testing (black-box)

Unsafe python components

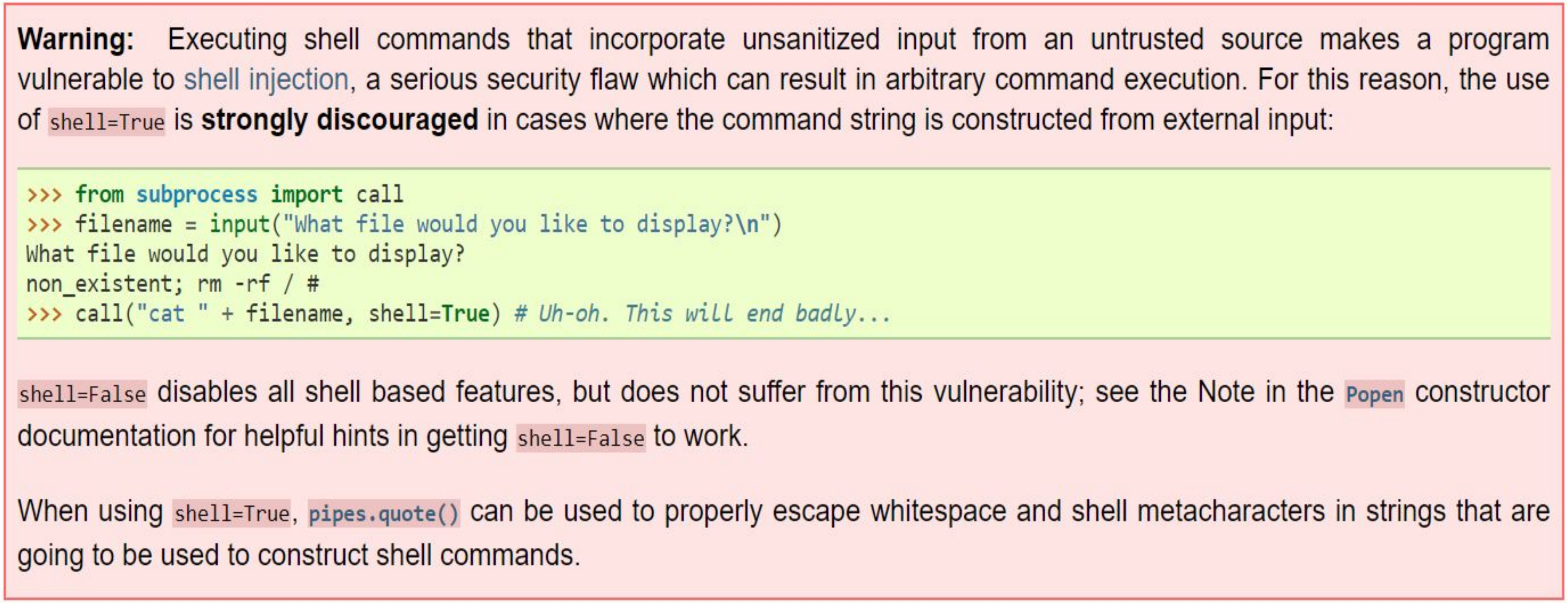


ast
bastion
commands
cookie
cPickle
eval
marshal
mktemp

multiprocessing
os.exec
os.popen
os.spawn
os.system
parser
pickle
pipes

pty
rexec
shelve
subprocess
tarfile
yaml
zipfile

Dangerous Python Functions



Warning: Executing shell commands that incorporate unsanitized input from an untrusted source makes a program vulnerable to shell injection, a serious security flaw which can result in arbitrary command execution. For this reason, the use of `shell=True` is **strongly discouraged** in cases where the command string is constructed from external input:

```
>>> from subprocess import call
>>> filename = input("What file would you like to display?\n")
What file would you like to display?
non_existent; rm -rf / #
>>> call("cat " + filename, shell=True) # Uh-oh. This will end badly...
```

`shell=False` disables all shell based features, but does not suffer from this vulnerability; see the Note in the `Popen` constructor documentation for helpful hints in getting `shell=False` to work.

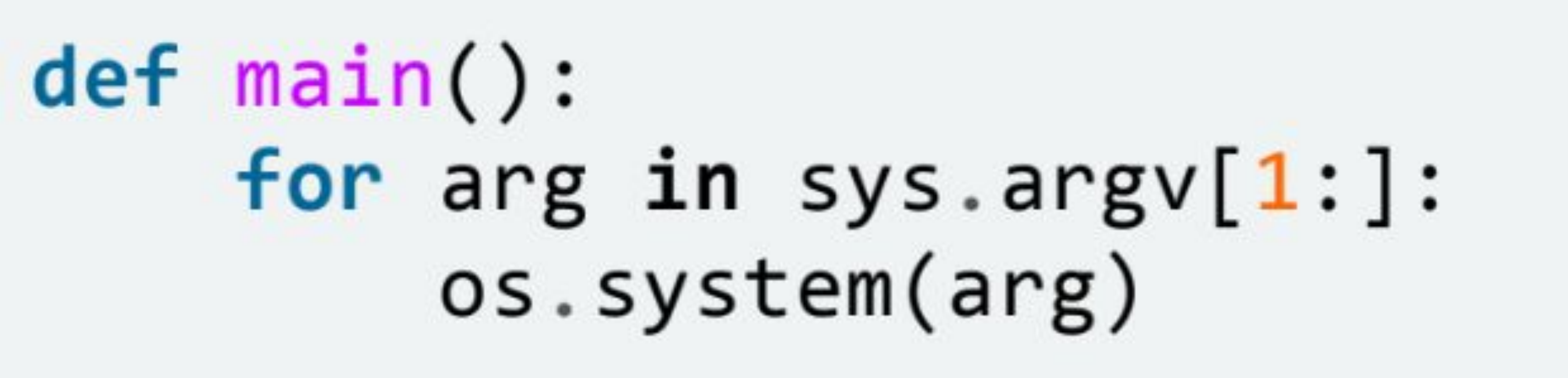
When using `shell=True`, `pipes.quote()` can be used to properly escape whitespace and shell metacharacters in strings that are going to be used to construct shell commands.

Security issues

Here’s a list of handful of other potential issues to watch for:

- **Dangerous python functions like eval()**
- **Serialization and deserialization objects with pickle**
- **SQL and JavaScript snippets**
- API keys included in source code
- HTTP calls to internal or external web services

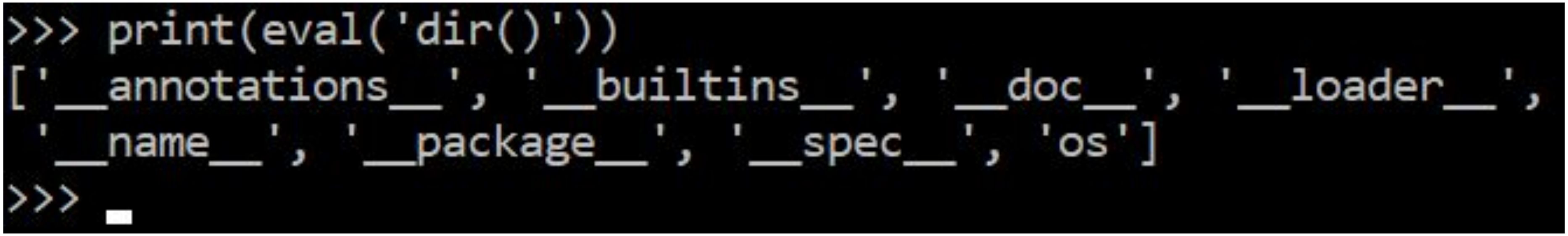
Improper input/output validation



```
def main():
    for arg in sys.argv[1:]:
        os.system(arg)
```

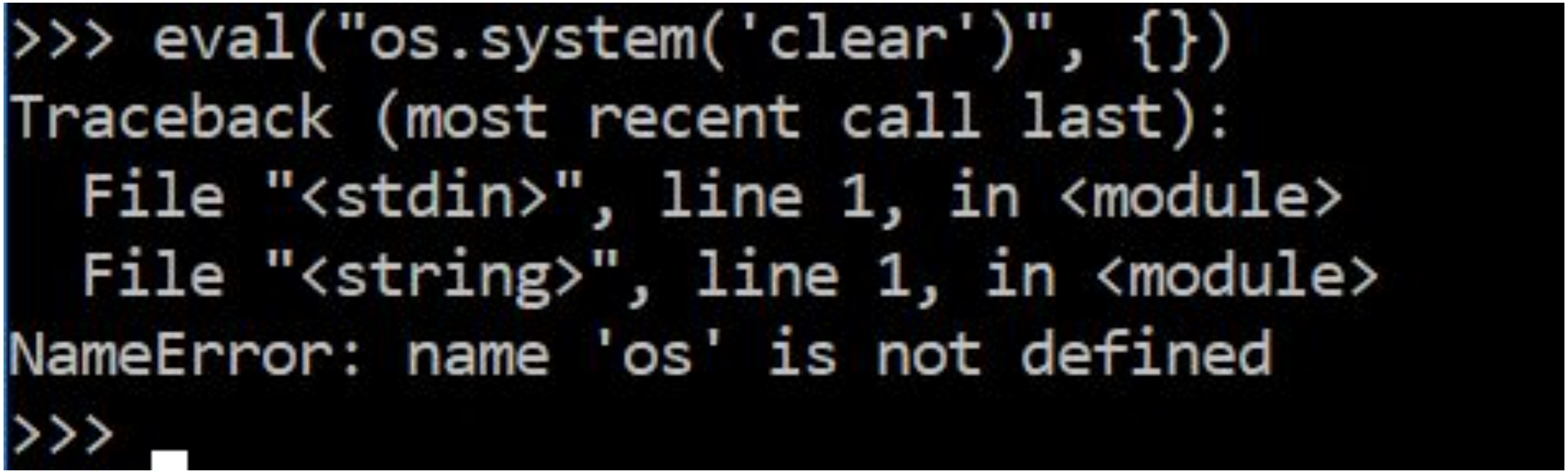
eval()

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



```
>>> print(eval('dir()'))
['__annotations__', '__builtins__', '__doc__', '__loader__',
 '__name__', '__package__', '__spec__', 'os']
>>> _
```

No globals



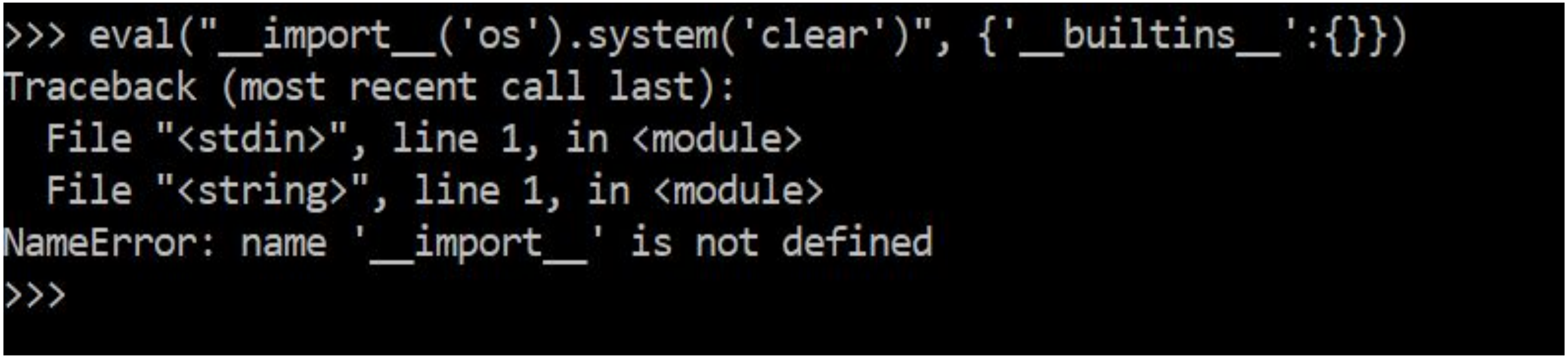
```
>>> eval("os.system('clear')", {})
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<string>", line 1, in <module>
NameError: name 'os' is not defined
>>>
```

eval()

```
eval("__import__('os').system('clear')", {})
```

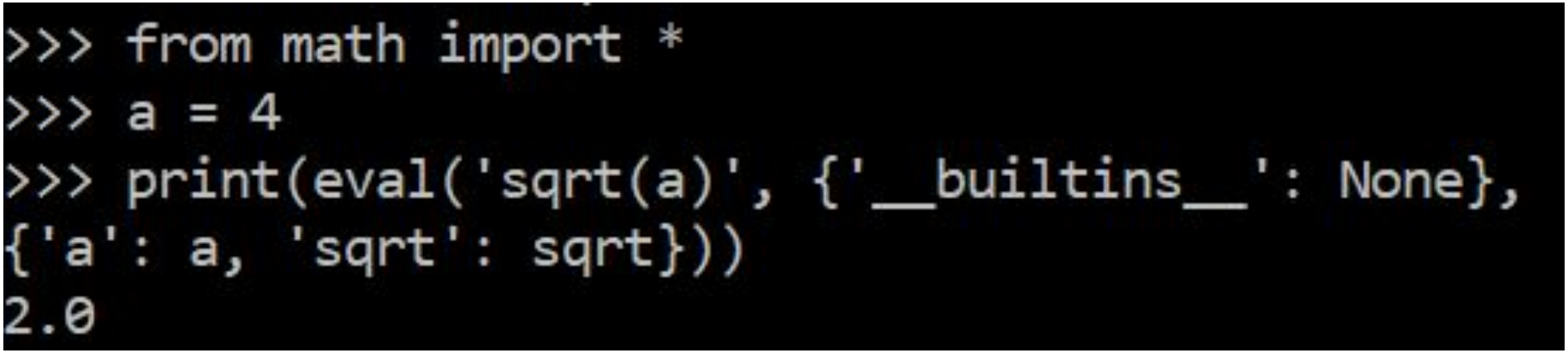
```
eval("__import__('os').system('rm -rf')", {})
```

Refuse access to the builtins



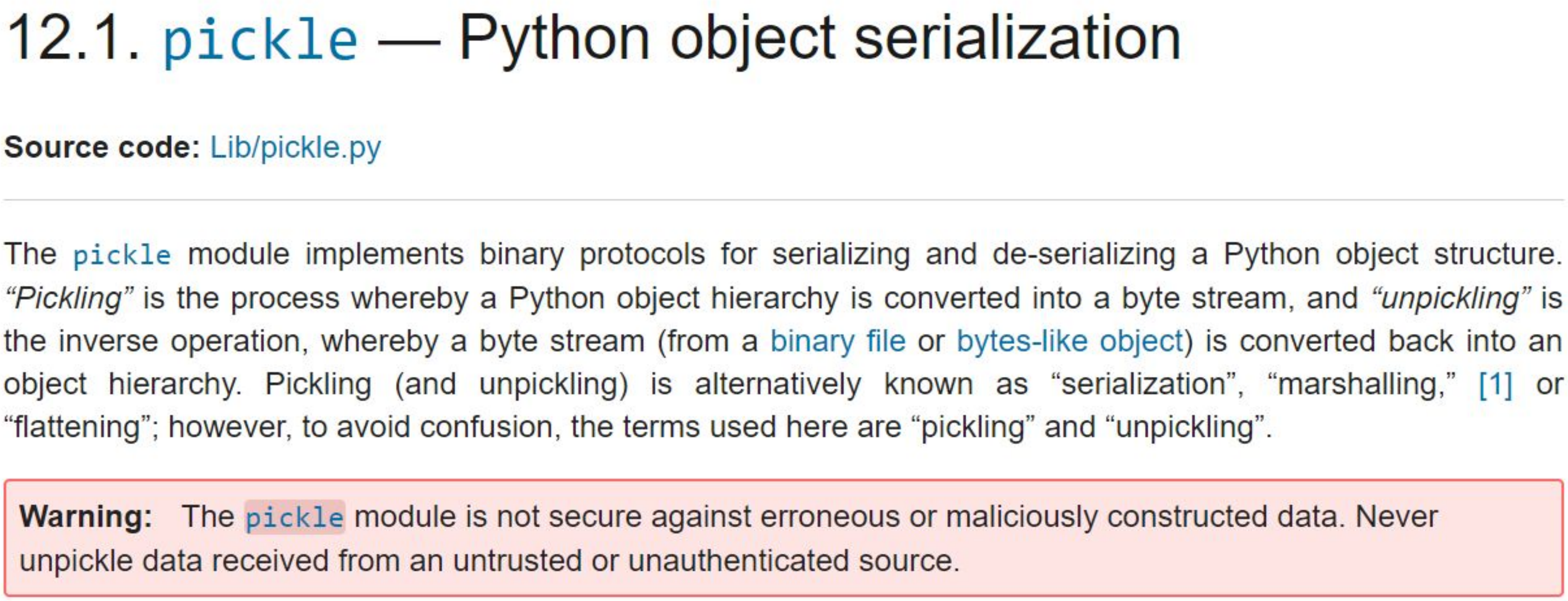
```
>>> eval("__import__('os').system('clear')", {'__builtins__':{}})
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<string>", line 1, in <module>
NameError: name '__import__' is not defined
>>>
```

eval()



```
>>> from math import *
>>> a = 4
>>> print(eval('sqrt(a)', {'__builtins__': None},
{'a': a, 'sqrt': sqrt}))
2.0
```

Serialization and Deserialization with Pickle



12.1. `pickle` — Python object serialization

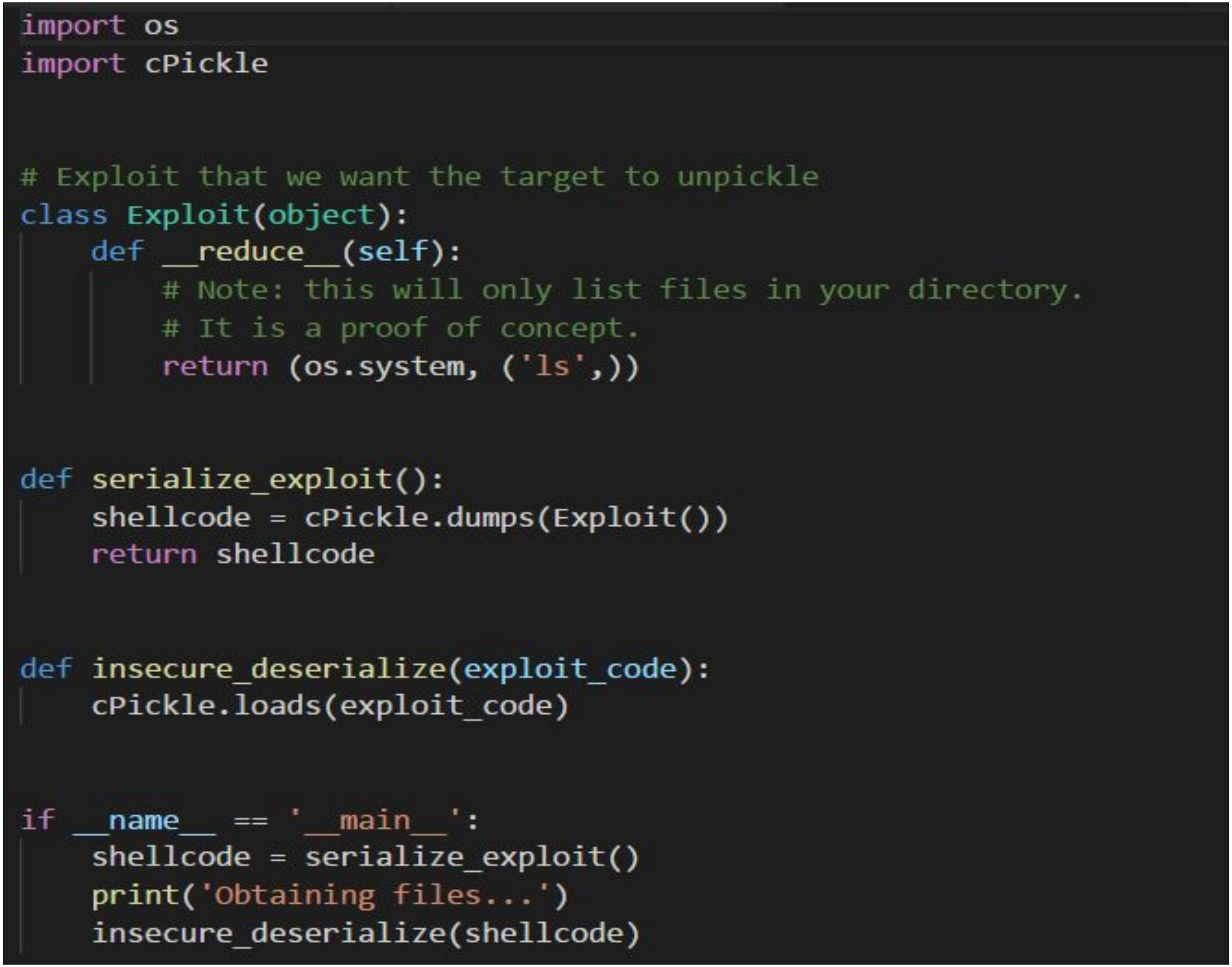
Source code: Lib/pickle.py

The `pickle` module implements binary protocols for serializing and de-serializing a Python object structure. *"Pickling"* is the process whereby a Python object hierarchy is converted into a byte stream, and *"unpickling"* is the inverse operation, whereby a byte stream (from a binary file or bytes-like object) is converted back into an object hierarchy. Pickling (and unpickling) is alternatively known as "serialization", "marshalling," [1] or "flattening"; however, to avoid confusion, the terms used here are "pickling" and "unpickling".

Warning: The `pickle` module is not secure against erroneous or maliciously constructed data. Never unpickle data received from an untrusted or unauthenticated source.

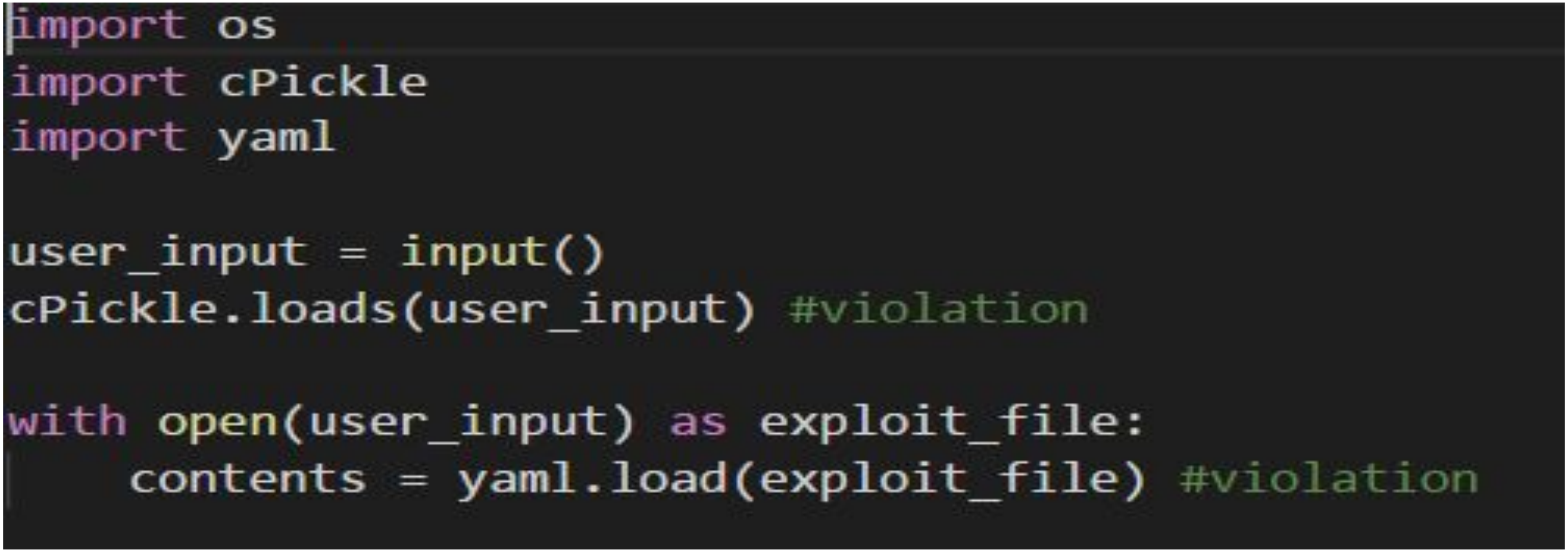
WARNING: pickle or cPickle are NOT designed as safe/secure solution for serialization

Serialization and Deserialization with Pickle

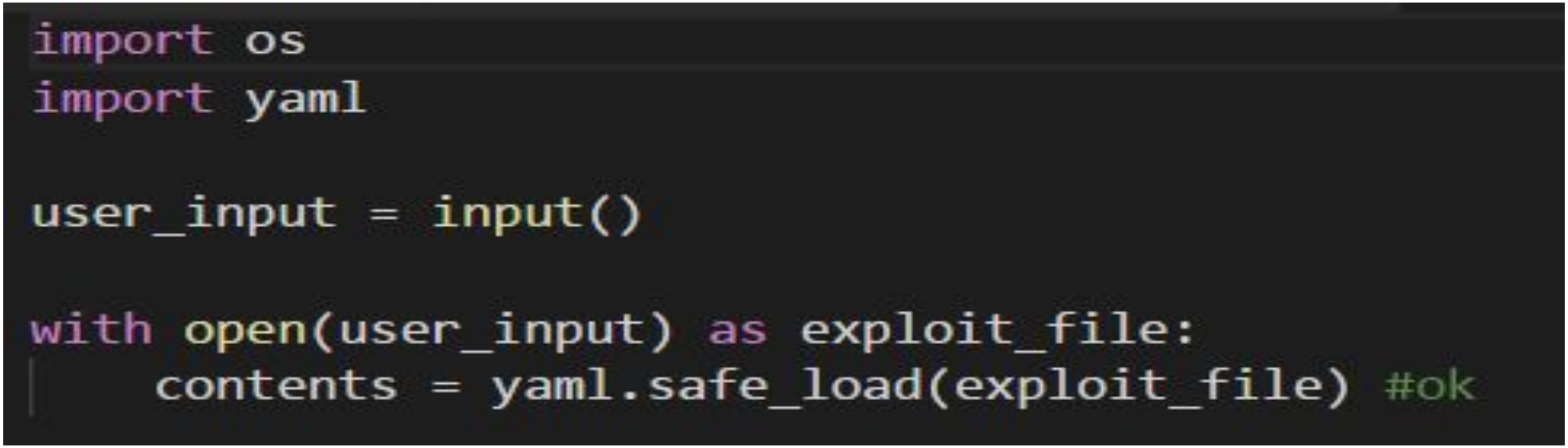


```
import os
import cPickle


# Exploit that we want the target to unpickle
class Exploit(object):
    def __reduce__(self):
        # Note: this will only list files in your directory.
        # It is a proof of concept.
        return (os.system, ('ls',))


def serialize_exploit():
    shellcode = cPickle.dumps(Exploit())
    return shellcode


def insecure_deserialize(exploit_code):
    cPickle.loads(exploit_code)


if __name__ == '__main__':
    shellcode = serialize_exploit()
    print('Obtaining files...')
    insecure_deserialize(shellcode)
```

Serialization and Deserialization with Pickle

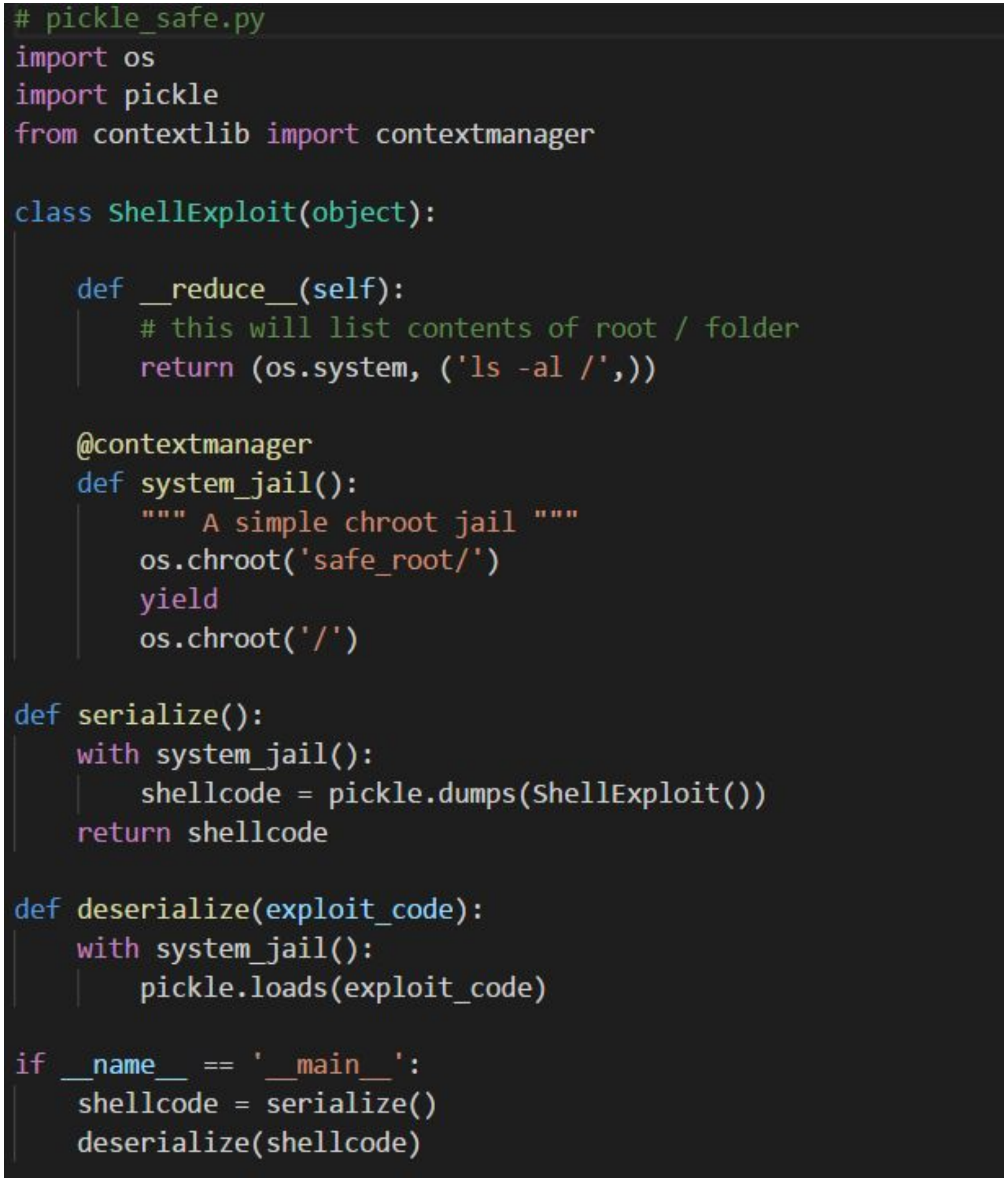


```
import os
import cPickle
import yaml

user_input = input()
cPickle.loads(user_input) #violation

with open(user_input) as exploit_file:
    contents = yaml.load(exploit_file) #violation
```



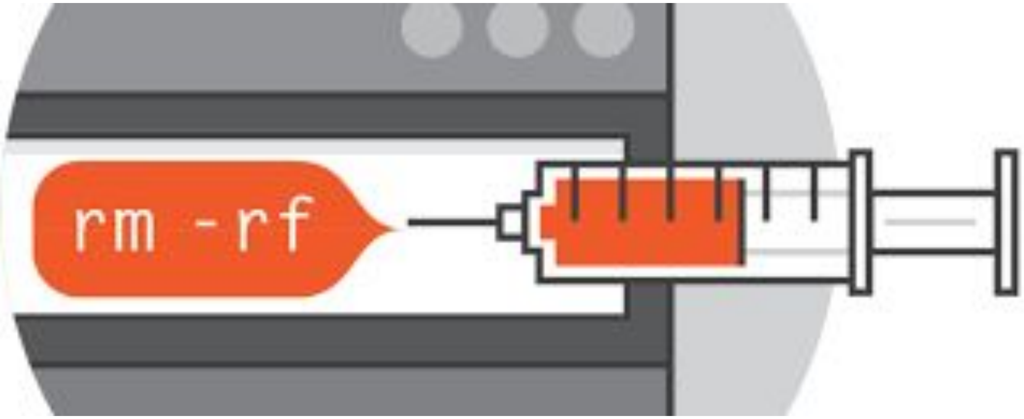
```
import os
import yaml

user_input = input()

with open(user_input) as exploit_file:
    contents = yaml.safe_load(exploit_file) #ok
```

Serialization and Deserialization with Pickle



```python
# pickle_safe.py
import os
import pickle
from contextlib import contextmanager

class ShellExploit(object):

    def __reduce__(self):
        # this will list contents of root / folder
        return (os.system, ('ls -al /',))

    @contextmanager
    def system_jail():
        """ A simple chroot jail """
        os.chroot('safe_root/')
        yield
        os.chroot('/')

def serialize():
    with system_jail():
        shellcode = pickle.dumps(ShellExploit())
    return shellcode

def deserialize(exploit_code):
    with system_jail():
        pickle.loads(exploit_code)

if __name__ == '__main__':
    shellcode = serialize()
    deserialize(shellcode)
```

Input injection attacks



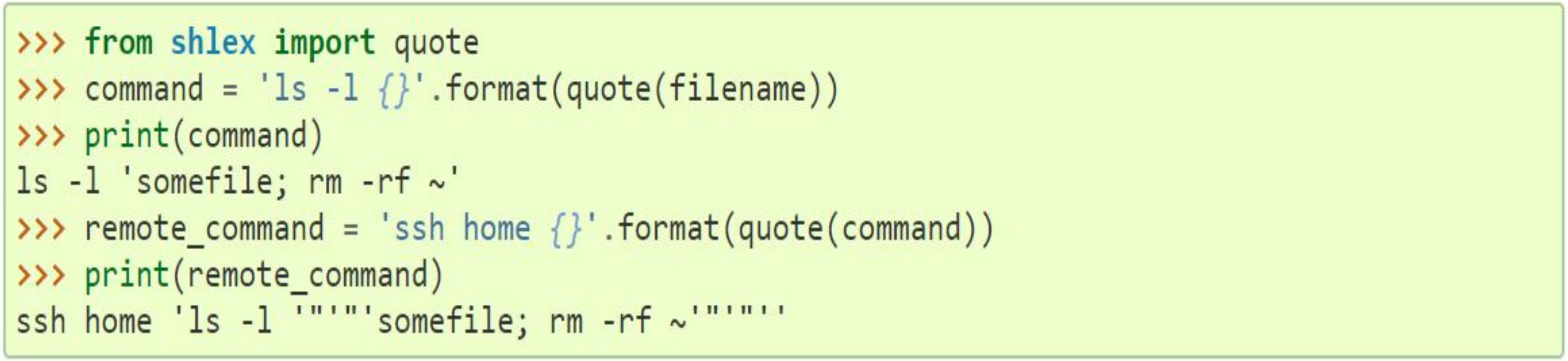
Command Injection

```
@app.route('/menu',methods =['POST'])
def menu():
    param = request.form [ ' suggestion ']
    command = ' echo ' + param + ' >> ' + ' menu.txt '
    subprocess.call(command,shell = True)
    with open('menu.txt','r') as f:
        menu = f.read()
    return render_template('command_injection.html',
menu = menu)
```

Command Injection

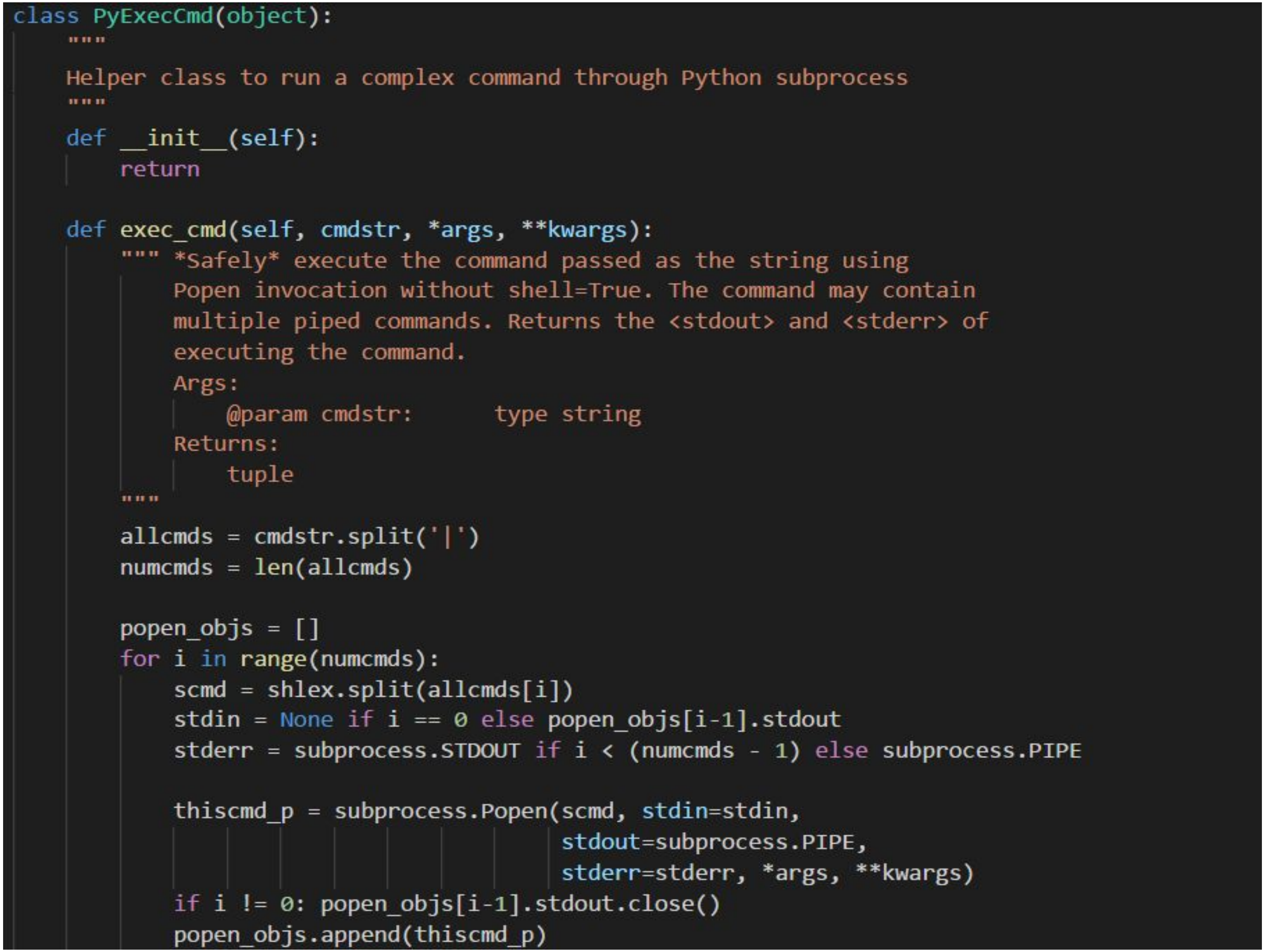
```
@app.route('/menu',methods =['POST'])
def menu():
    param = request.form [ ' suggestion ']
    command = ' echo ' + param + ' >> ' + ' menu.txt '
    subprocess.call(command,shell = False)
    with open('menu.txt','r') as f:
        menu = f.read()
    return render_template('command_injection.html',
menu = menu)
```

shlex module



```
>>> from shlex import quote
>>> command = 'ls -l {}'.format(quote(filename))
>>> print(command)
ls -l 'somefile; rm -rf ~'
>>> remote_command = 'ssh home {}'.format(quote(command))
>>> print(remote_command)
ssh home 'ls -l '"'"'somefile; rm -rf ~'"'"''
```

PyExecCmd



```
class PyExecCmd(object):
    """
    Helper class to run a complex command through Python subprocess
    """
    def __init__(self):
        return

    def exec_cmd(self, cmdstr, *args, **kwargs):
        """ *Safely* execute the command passed as the string using
            Popen invocation without shell=True. The command may contain
            multiple piped commands. Returns the <stdout> and <stderr> of
            executing the command.
            Args:
                @param cmdstr:      type string
            Returns:
                tuple
        """
        allcmds = cmdstr.split('|')
        numcmds = len(allcmds)

        popen_objs = []
        for i in range(numcmds):
            scmd = shlex.split(allcmds[i])
            stdin = None if i == 0 else popen_objs[i-1].stdout
            stderr = subprocess.STDOUT if i < (numcmds - 1) else subprocess.PIPE

            thiscmd_p = subprocess.Popen(scmd, stdin=stdin,
                                         stdout=subprocess.PIPE,
                                         stderr=stderr, *args, **kwargs)
            if i != 0: popen_objs[i-1].stdout.close()
            popen_objs.append(thiscmd_p)
```

Common attack vectors on web applications

OWASP TOP 10:

A1 Injection
A2 Broken Authentication and Session Management
A3 Cross-Site Scripting (XSS)
A4 Insecure Direct Object References
A5 Security Misconfiguration
A6 Sensitive Data Exposure
A7 Missing Function Level Access Control
A8 Cross-Site Request Forgery (CSRF)
A9 Using Components with Known Vulnerabilities
A10 Unvalidated Redirects and Forwards

SQL Injection

```
@app.route('/filtering')
def filtering():
    param = request.args.get('param', 'not set')
    Session = sessionmaker(bind = db.engine)
    session = Session()
    result = session.query(User).filter(" username ={}
".format(param))
    for value in result:
        print(value.username , value.email)
        return ' Result is displayed in console.'
```

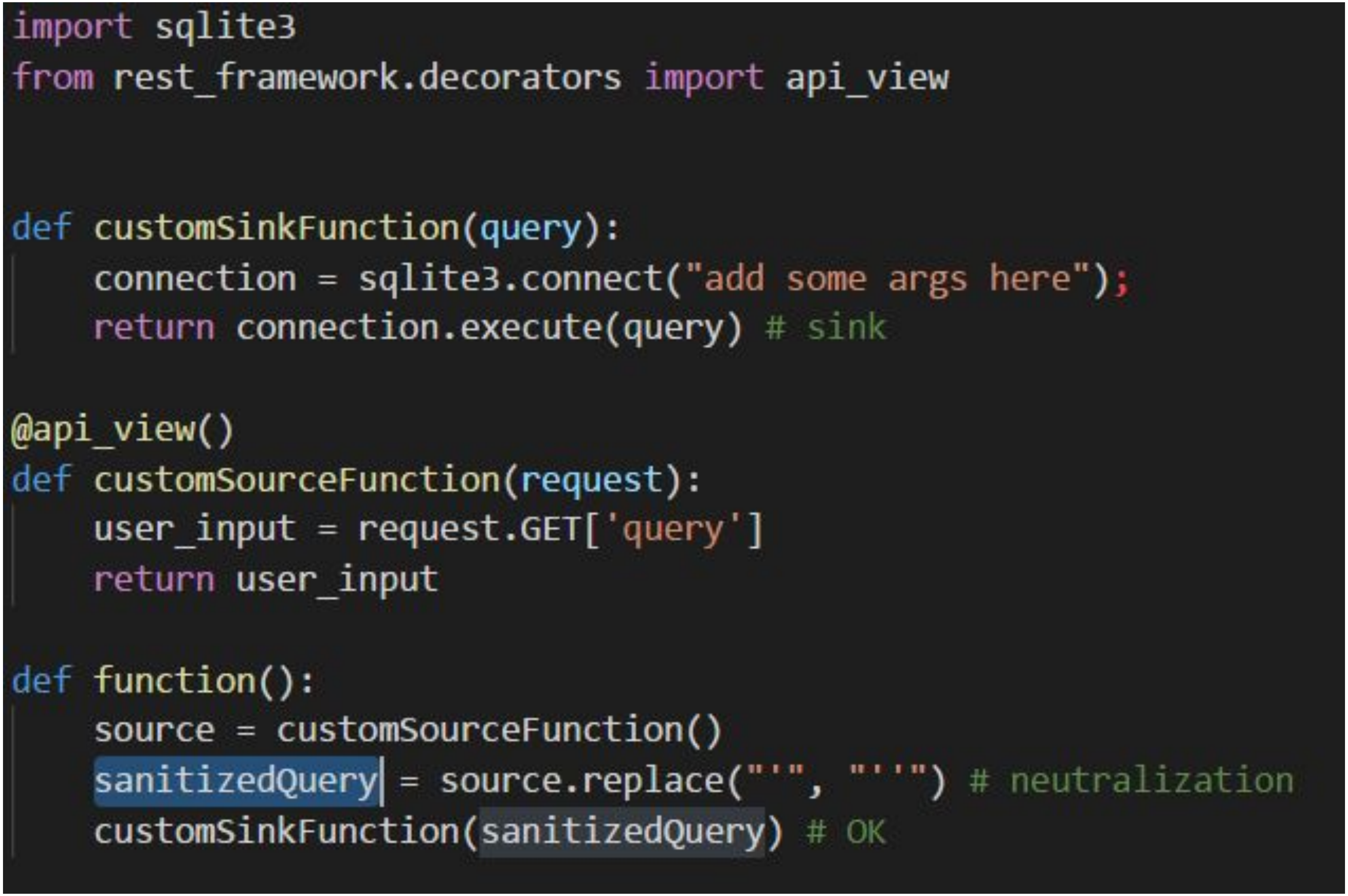
Prevent SQL injection attacks

- NEVER concatenate untrusted inputs in SQL code.

- Concatenate constant fragments of SQL (literals) with parameter placeholders.

- cur.execute("SELECT * FROM students WHERE name= '%s';" % name)
- c.execute("SELECT * from students WHERE name=(?)" , name)

Prevent SQL injection attacks

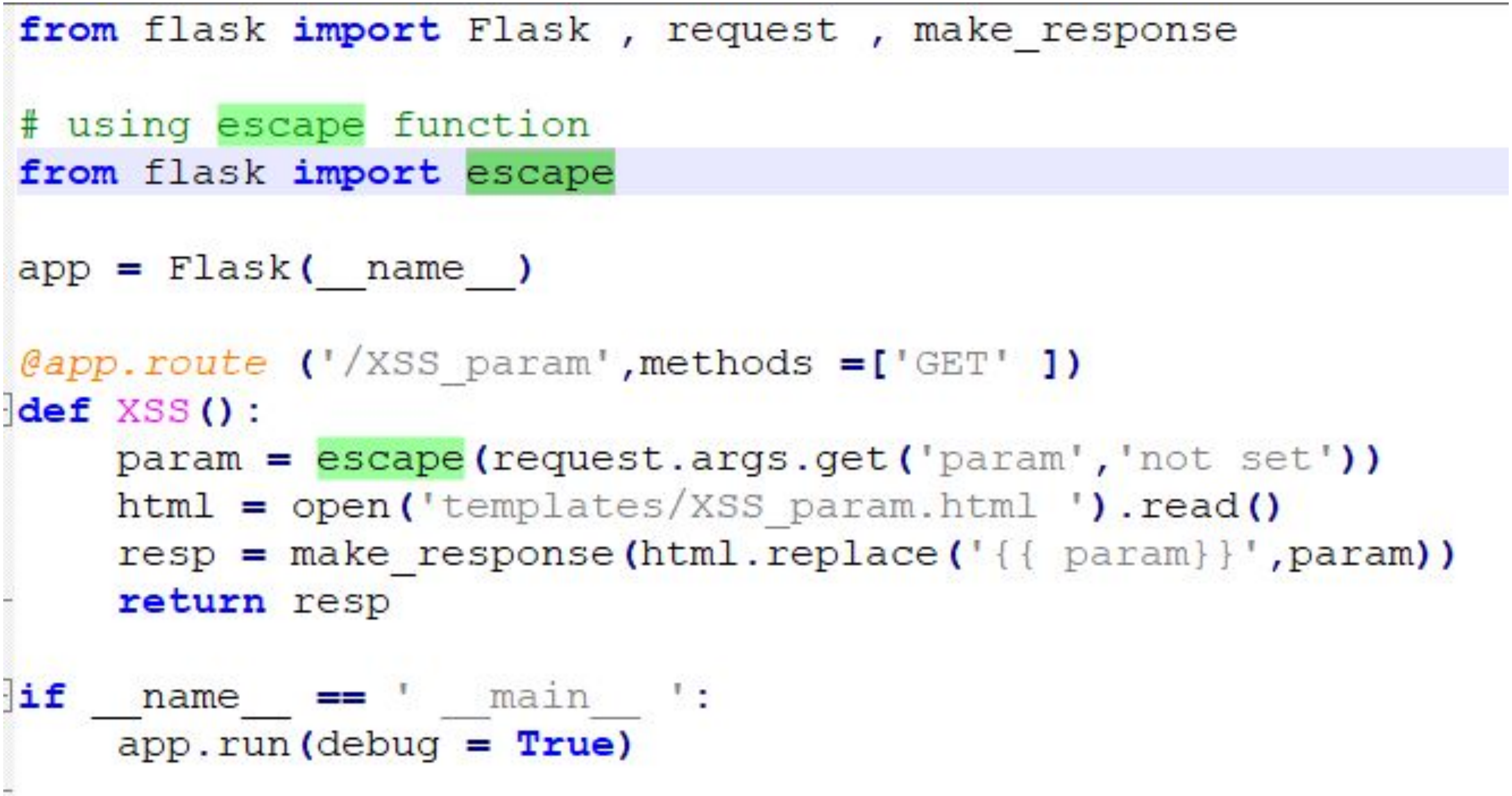


```python
import sqlite3
from rest_framework.decorators import api_view


def customSinkFunction(query):
    connection = sqlite3.connect("add some args here");
    return connection.execute(query) # sink

@api_view()
def customSourceFunction(request):
    user_input = request.GET['query']
    return user_input

def function():
    source = customSourceFunction()
    sanitizedQuery = source.replace("'", "''") # neutralization
    customSinkFunction(sanitizedQuery) # OK
```

XSS

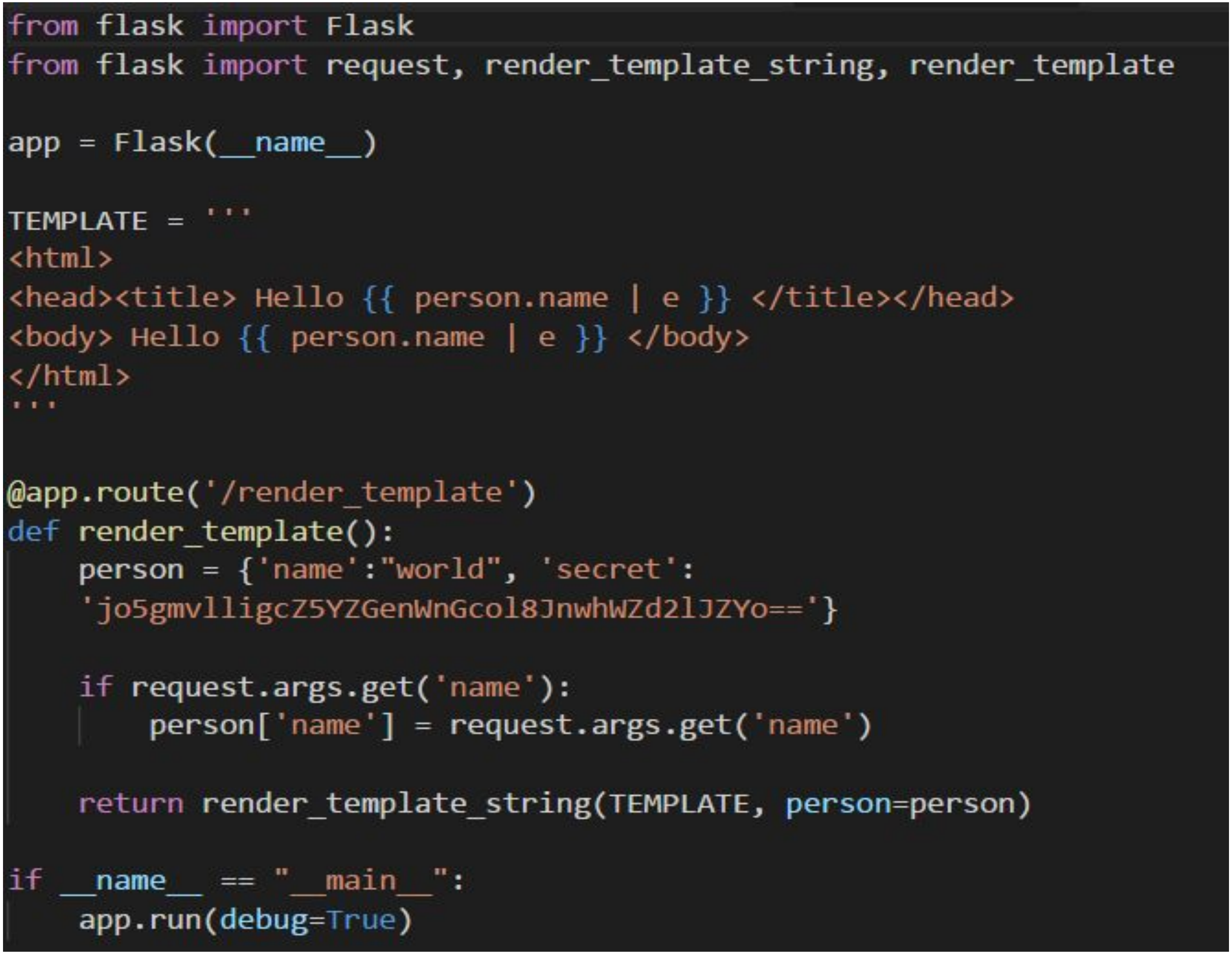
```
from flask import Flask , request , make_response
app = Flask(__name__)

@app.route ('/XSS_param',methods =['GET ])
def XSS():
    param = request.args.get('param','not set')
    html = open('templates/XSS_param.html ').read()
    resp = make_response(html.replace('{{ param}}',param))
    return resp

if __name__ == ' __main__ ':
    app.run(debug = True)
```

XSS



```python
from flask import Flask , request , make_response

# using escape function
from flask import escape

app = Flask(__name__)

@app.route ('/XSS_param',methods =['GET' ])
def XSS():
    param = escape(request.args.get('param','not set'))
    html = open('templates/XSS_param.html ').read()
    resp = make_response(html.replace('{{ param}}',param))
    return resp

if __name__ == '__main__':
    app.run(debug = True)
```

XSS



```
from flask import Flask
from flask import request, render_template_string, render_template

app = Flask(__name__)

TEMPLATE = '''
<html>
<head><title> Hello {{ person.name | e }} </title></head>
<body> Hello {{ person.name | e }} </body>
</html>
'''


@app.route('/render_template')
def render_template():
    person = {'name':"world", 'secret':
    'jo5gmvlligcZ5YZGenWnGcol8JnwhWZd2lJZYo=='}

    if request.args.get('name'):
        person['name'] = request.args.get('name')

    return render_template_string(TEMPLATE, person=person)

if __name__ == "__main__":
    app.run(debug=True)
```

Automated security testing

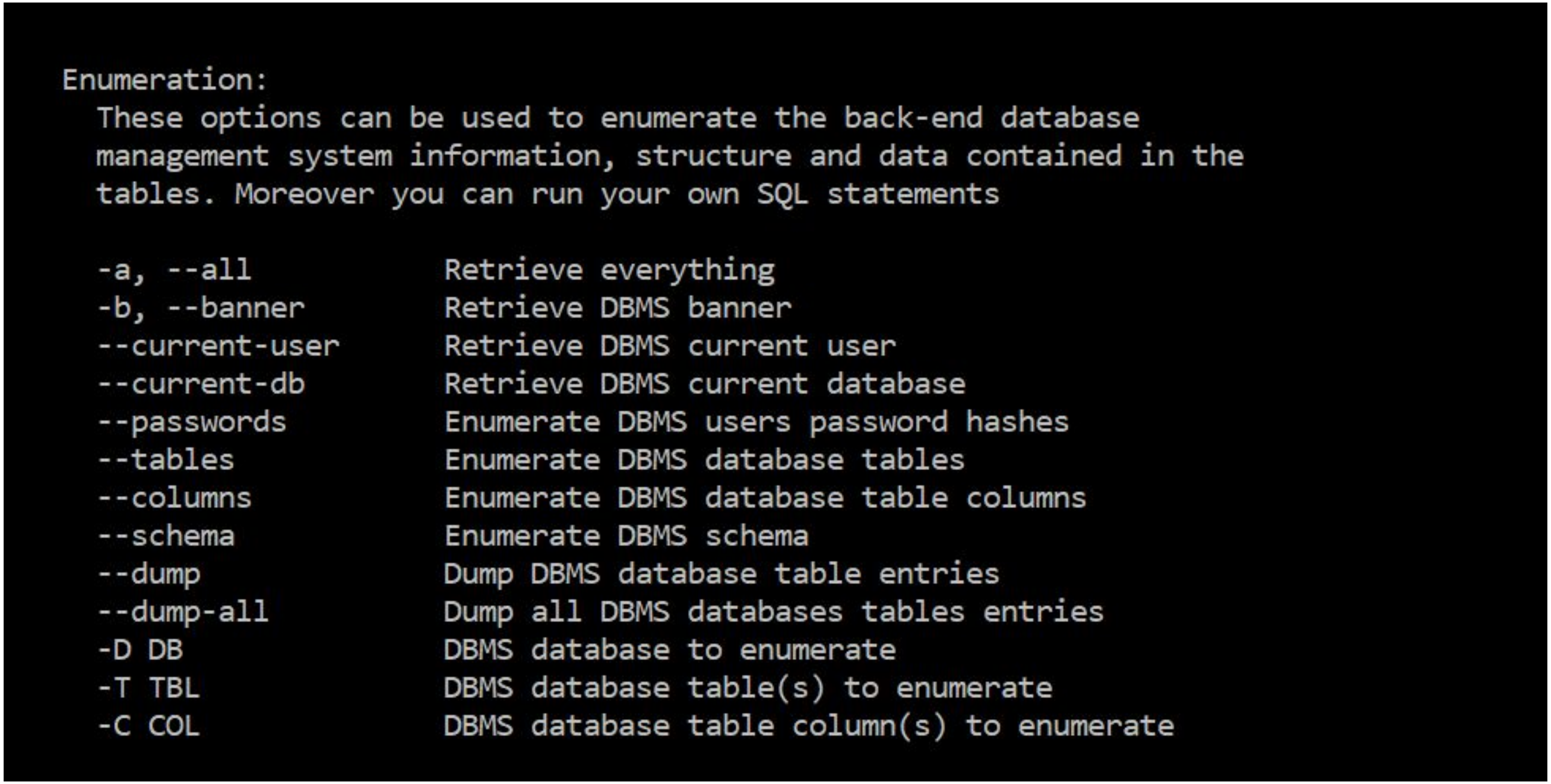
Automatic Scanning tools:

- **SQLMap**: Sql injection
- **XssScrapy:** Sql injection and XSS

Source Code Analysis tools:

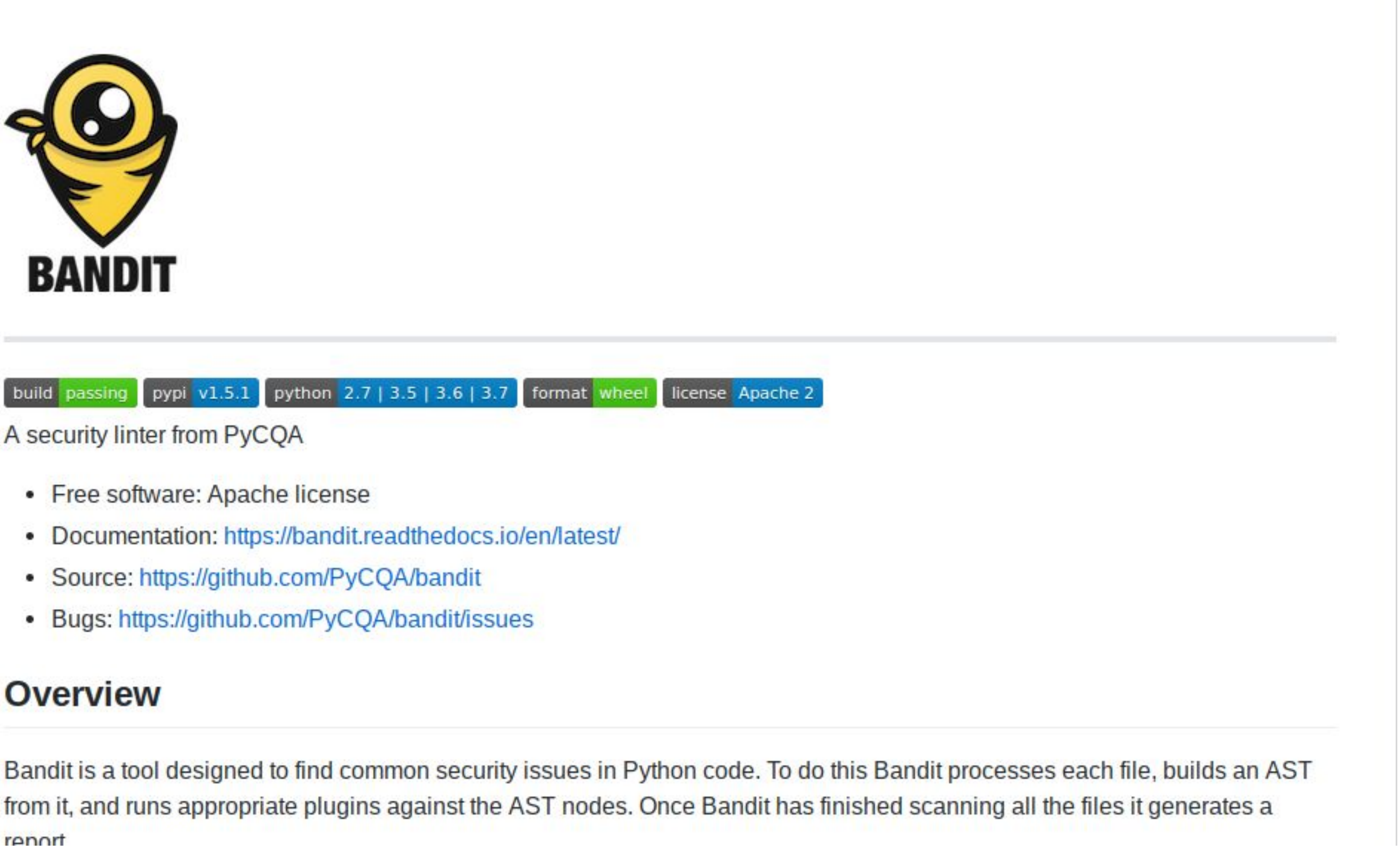
- **Bandit**: Open Source and can be easily integrated with Jenkins CI/CD

SQLMap



```
Enumeration:
  These options can be used to enumerate the back-end database
  management system information, structure and data contained in the
  tables. Moreover you can run your own SQL statements

  -a, --all             Retrieve everything
  -b, --banner          Retrieve DBMS banner
  --current-user        Retrieve DBMS current user
  --current-db          Retrieve DBMS current database
  --passwords           Enumerate DBMS users password hashes
  --tables              Enumerate DBMS database tables
  --columns             Enumerate DBMS database table columns
  --schema              Enumerate DBMS schema
  --dump                Dump DBMS database table entries
  --dump-all            Dump all DBMS databases tables entries
  -D DB                 DBMS database to enumerate
  -T TBL                DBMS database table(s) to enumerate
  -C COL                DBMS database table column(s) to enumerate
```

Bandit



BANDIT

build passing | pypi v1.5.1 | python 2.7 | 3.5 | 3.6 | 3.7 | format wheel | license Apache 2

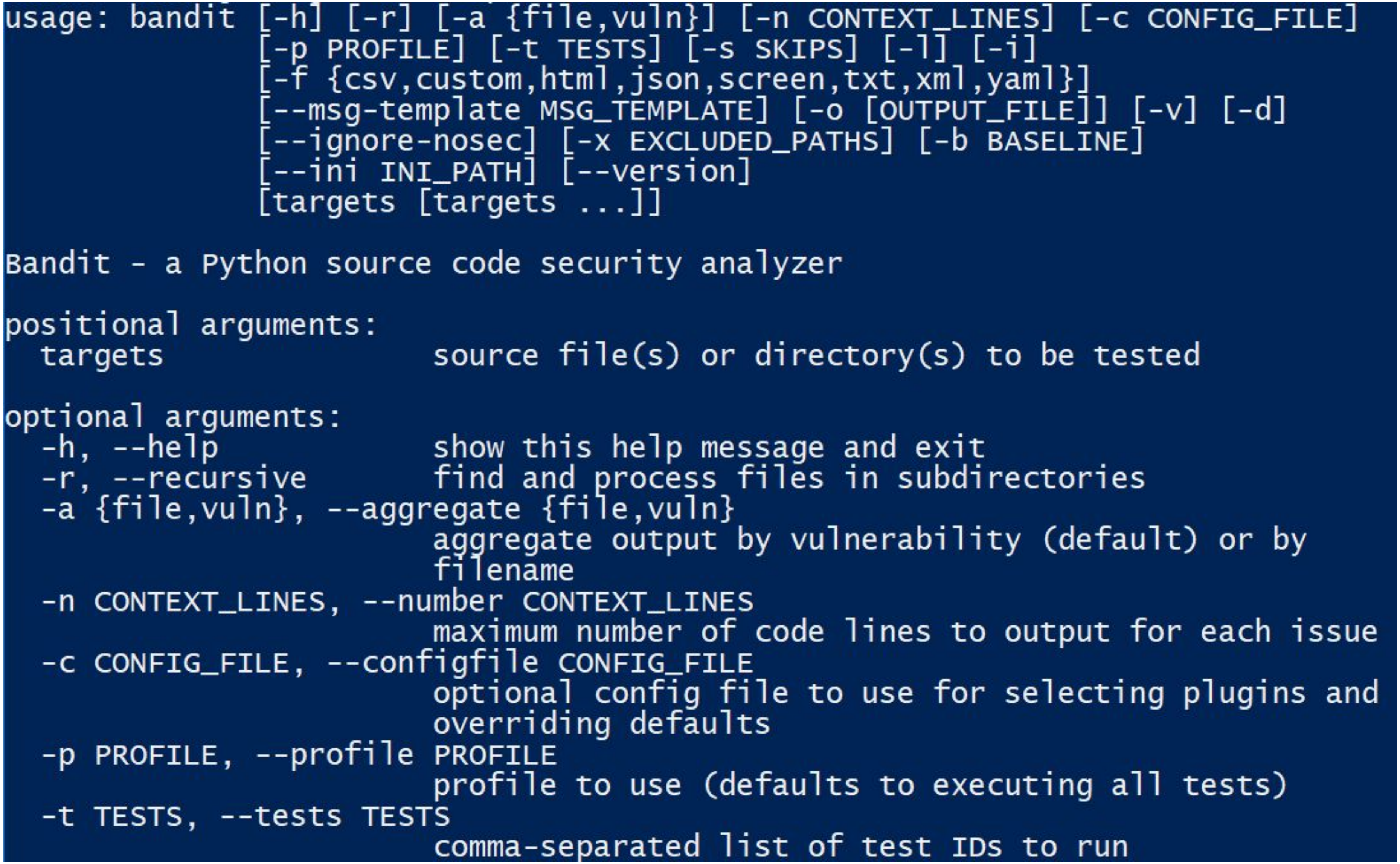
A security linter from PyCQA

- Free software: Apache license
- Documentation: https://bandit.readthedocs.io/en/latest/
- Source: https://github.com/PyCQA/bandit
- Bugs: https://github.com/PyCQA/bandit/issues

Overview

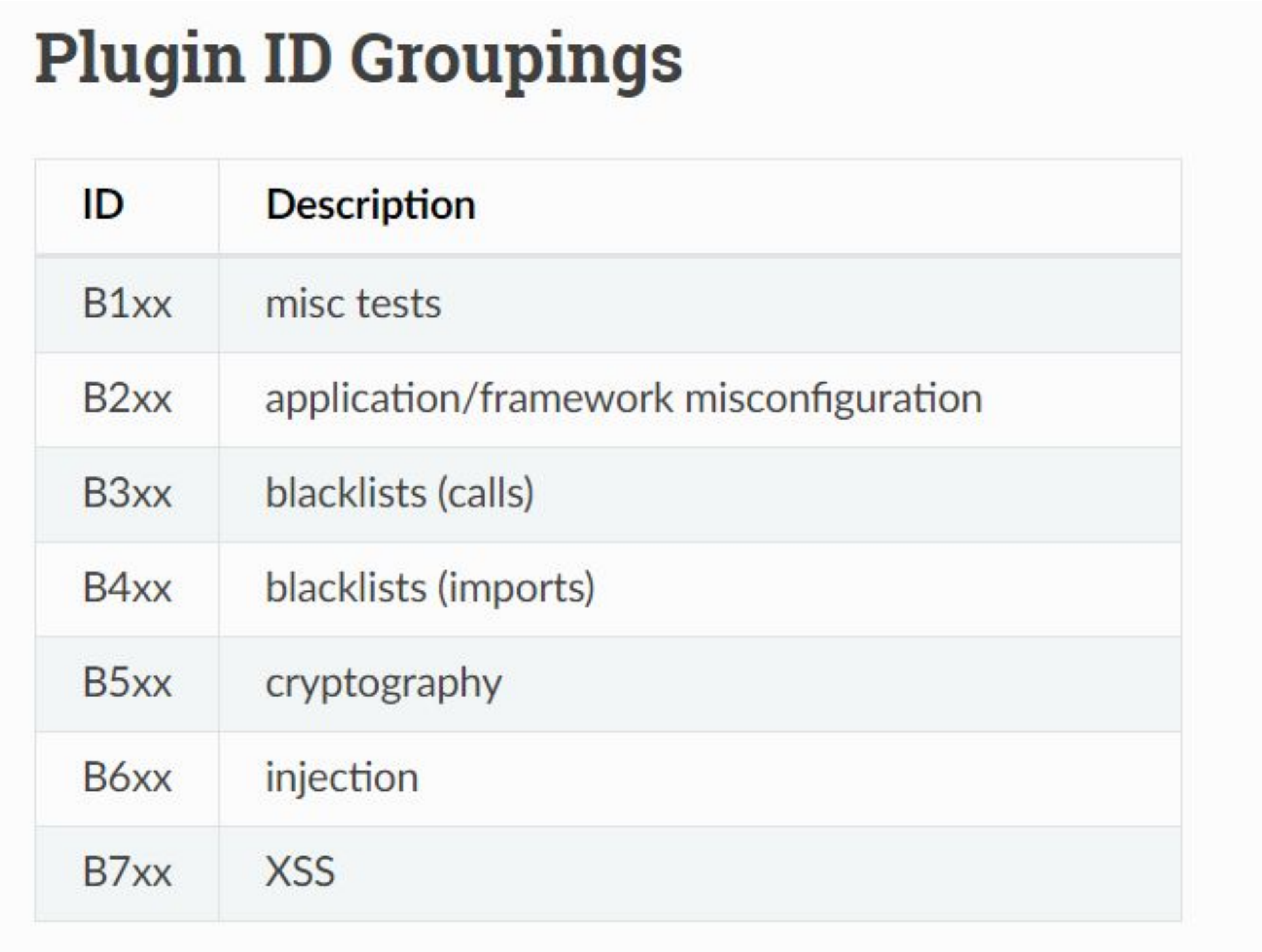
Bandit is a tool designed to find common security issues in Python code. To do this Bandit processes each file, builds an AST from it, and runs appropriate plugins against the AST nodes. Once Bandit has finished scanning all the files it generates a report

Bandit



```
usage: bandit [-h] [-r] [-a {file,vuln}] [-n CONTEXT_LINES] [-c CONFIG_FILE]
              [-p PROFILE] [-t TESTS] [-s SKIPS] [-l] [-i]
              [-f {csv,custom,html,json,screen,txt,xml,yaml}]
              [--msg-template MSG_TEMPLATE] [-o [OUTPUT_FILE]] [-v] [-d]
              [--ignore-nosec] [-x EXCLUDED_PATHS] [-b BASELINE]
              [--ini INI_PATH] [--version]
              [targets [targets ...]]

Bandit - a Python source code security analyzer

positional arguments:
  targets               source file(s) or directory(s) to be tested

optional arguments:
  -h, --help            show this help message and exit
  -r, --recursive       find and process files in subdirectories
  -a {file,vuln}, --aggregate {file,vuln}
                        aggregate output by vulnerability (default) or by
                        filename
  -n CONTEXT_LINES, --number CONTEXT_LINES
                        maximum number of code lines to output for each issue
  -c CONFIG_FILE, --configfile CONFIG_FILE
                        optional config file to use for selecting plugins and
                        overriding defaults
  -p PROFILE, --profile PROFILE
                        profile to use (defaults to executing all tests)
  -t TESTS, --tests TESTS
                        comma-separated list of test IDs to run
```

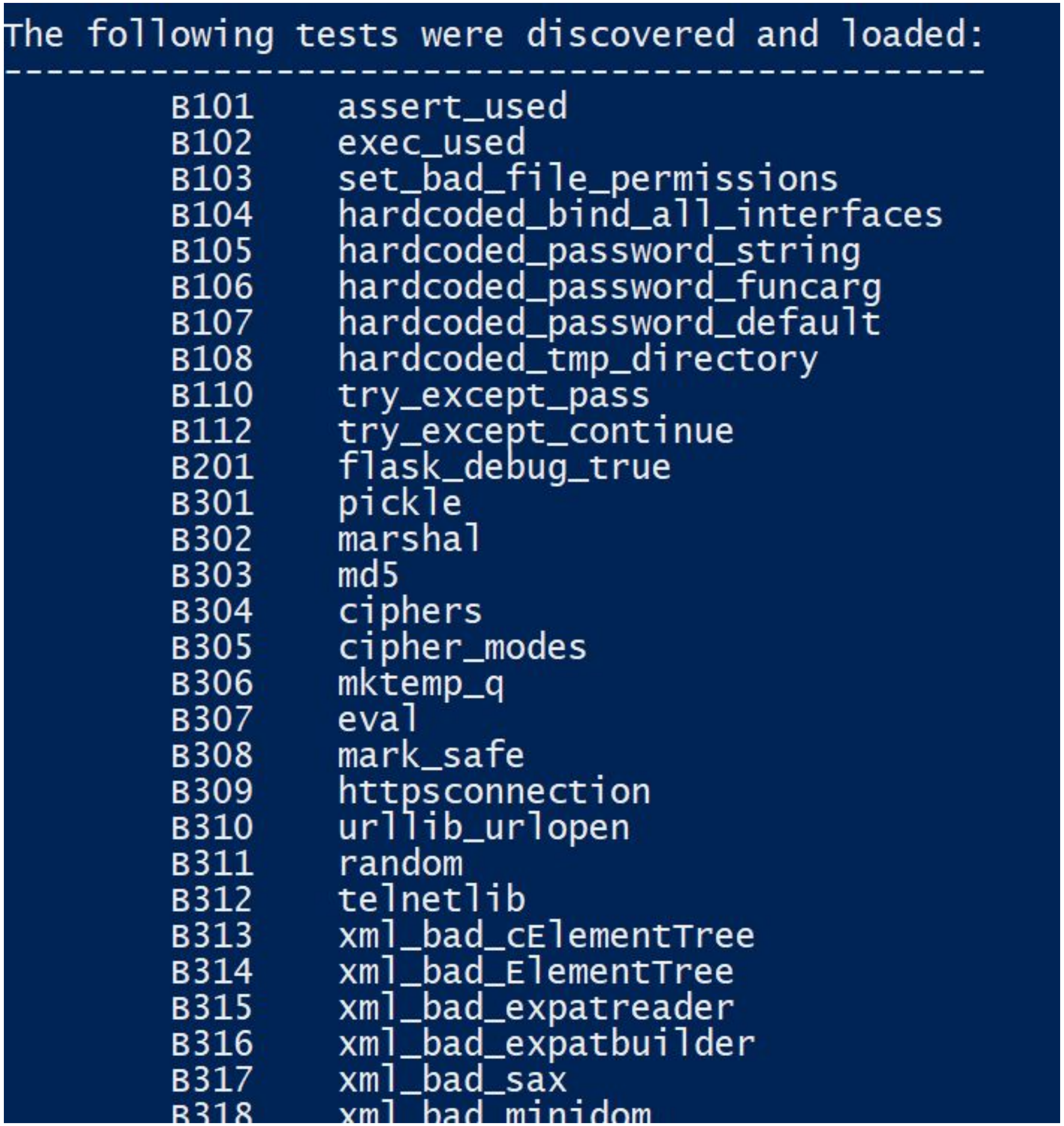
Bandit Test plugins



Plugin ID Groupings

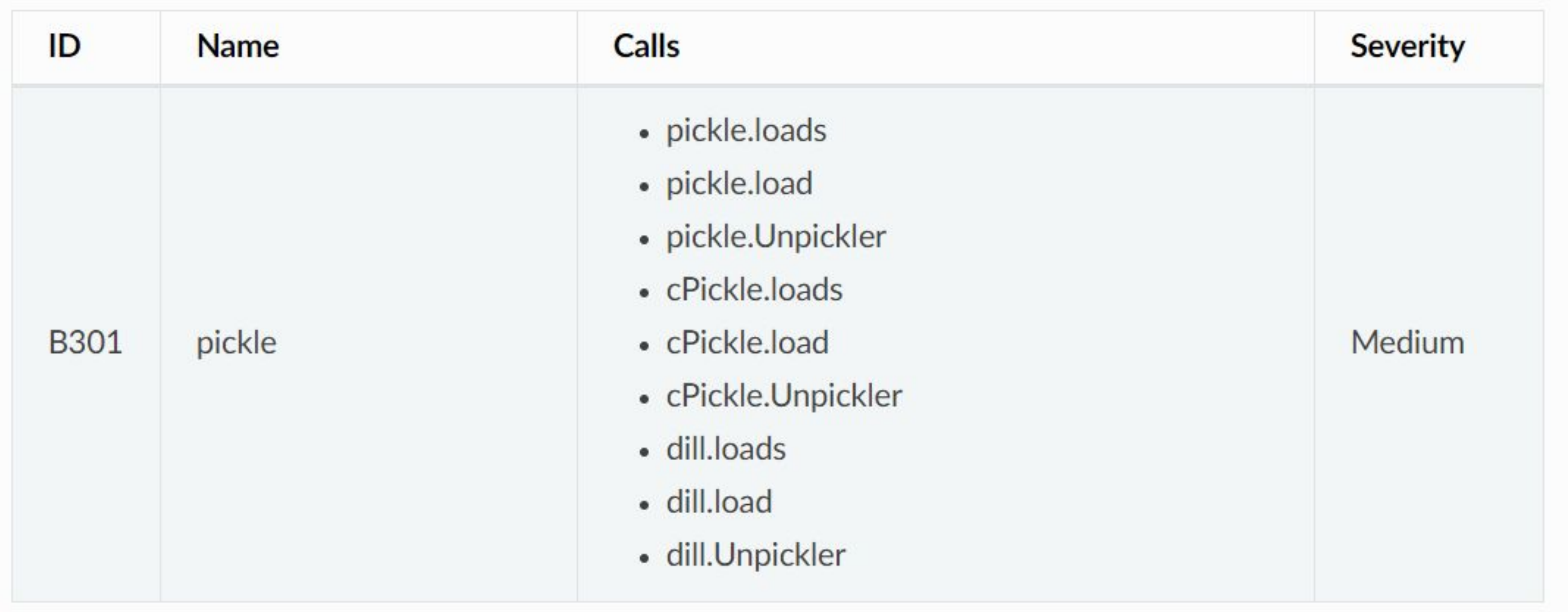
ID	Description
B1xx	misc tests
B2xx	application/framework misconfiguration
B3xx	blacklists (calls)
B4xx	blacklists (imports)
B5xx	cryptography
B6xx	injection
B7xx	XSS

Bandit Test plugins



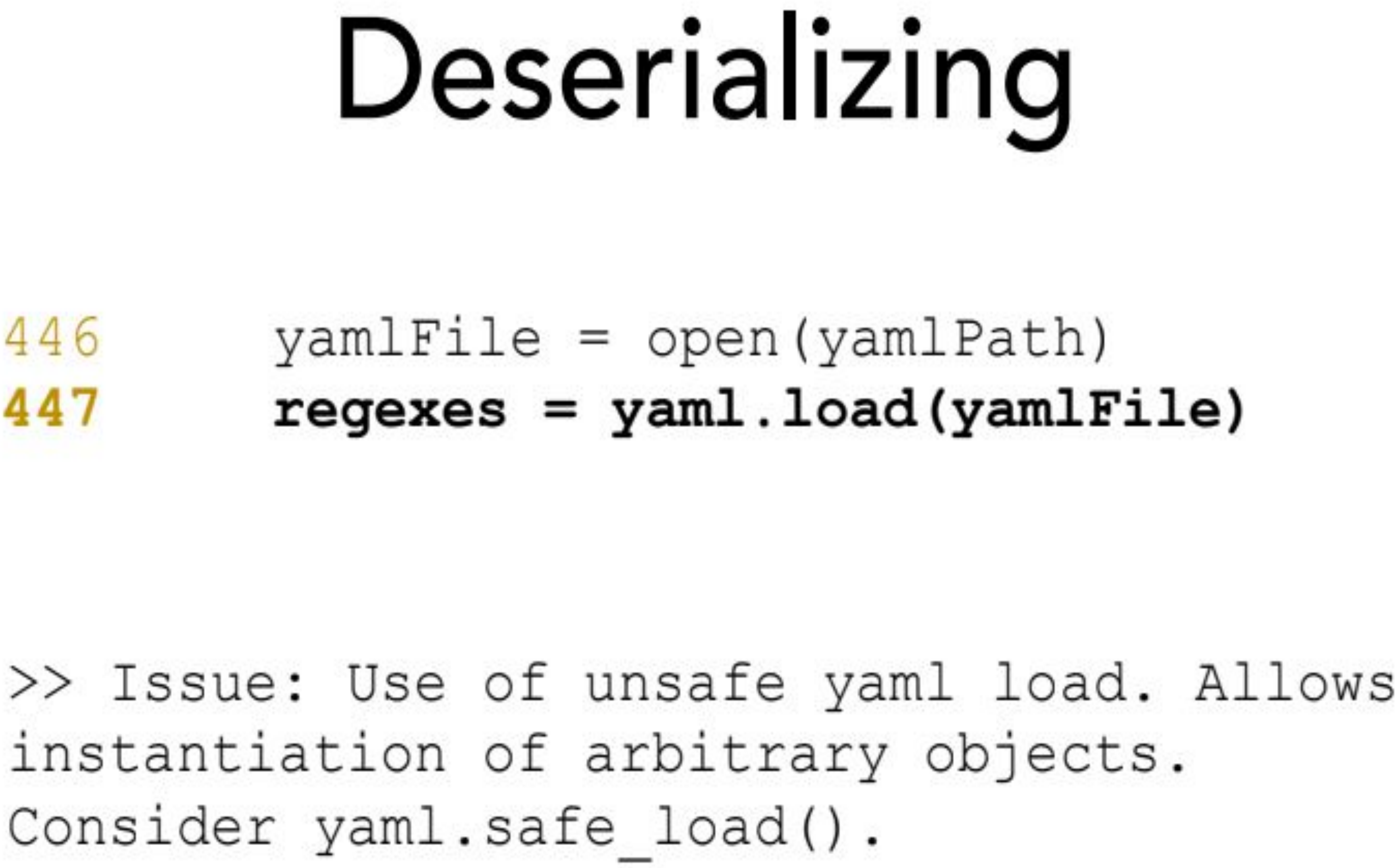
```
The following tests were discovered and loaded:
-----------------------------------------------
        B101    assert_used
        B102    exec_used
        B103    set_bad_file_permissions
        B104    hardcoded_bind_all_interfaces
        B105    hardcoded_password_string
        B106    hardcoded_password_funcarg
        B107    hardcoded_password_default
        B108    hardcoded_tmp_directory
        B110    try_except_pass
        B112    try_except_continue
        B201    flask_debug_true
        B301    pickle
        B302    marshal
        B303    md5
        B304    ciphers
        B305    cipher_modes
        B306    mktemp_q
        B307    eval
        B308    mark_safe
        B309    httpsconnection
        B310    urllib_urlopen
        B311    random
        B312    telnetlib
        B313    xml_bad_cElementTree
        B314    xml_bad_ElementTree
        B315    xml_bad_expatreader
        B316    xml_bad_expatbuilder
        B317    xml_bad_sax
        B318    xml_bad_minidom
```

Bandit Test plugins



ID	Name	Calls	Severity
B301	pickle	• pickle.loads • pickle.load • pickle.Unpickler • cPickle.loads • cPickle.load • cPickle.Unpickler • dill.loads • dill.load • dill.Unpickler	Medium

Bandit Test plugins

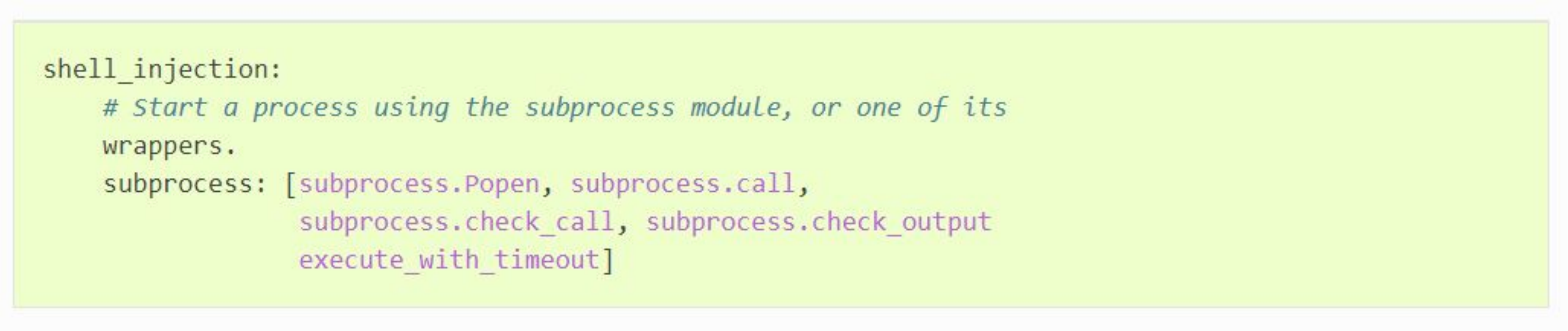


Deserializing

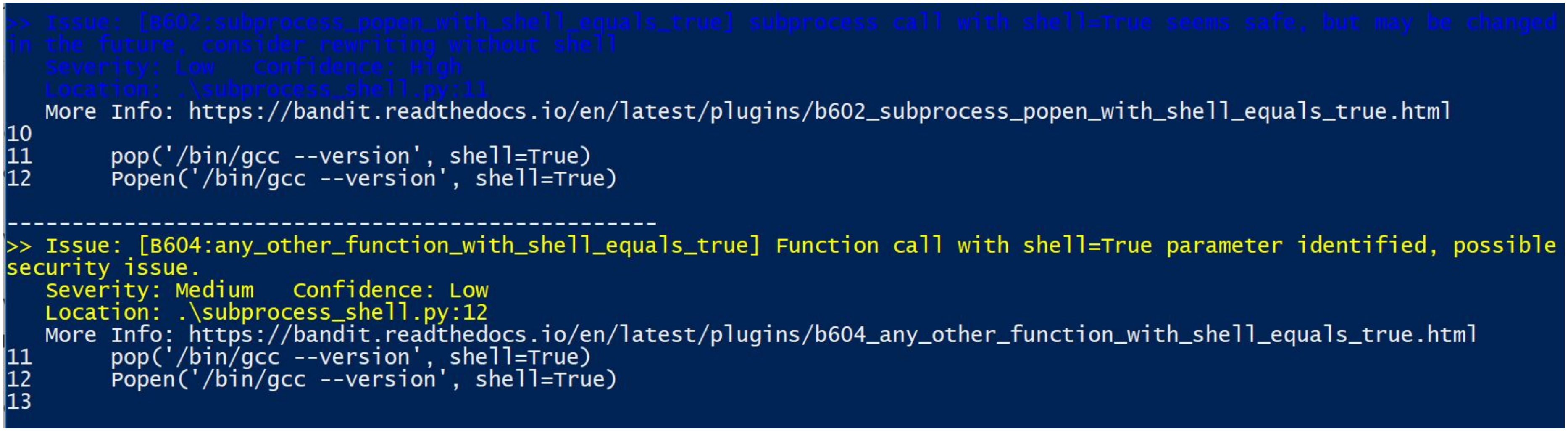
```
yamlFile = open(yamlPath)
regexes = yaml.load(yamlFile)
```

```
>> Issue: Use of unsafe yaml load. Allows
instantiation of arbitrary objects.
Consider yaml.safe_load().
```

Bandit Test plugins



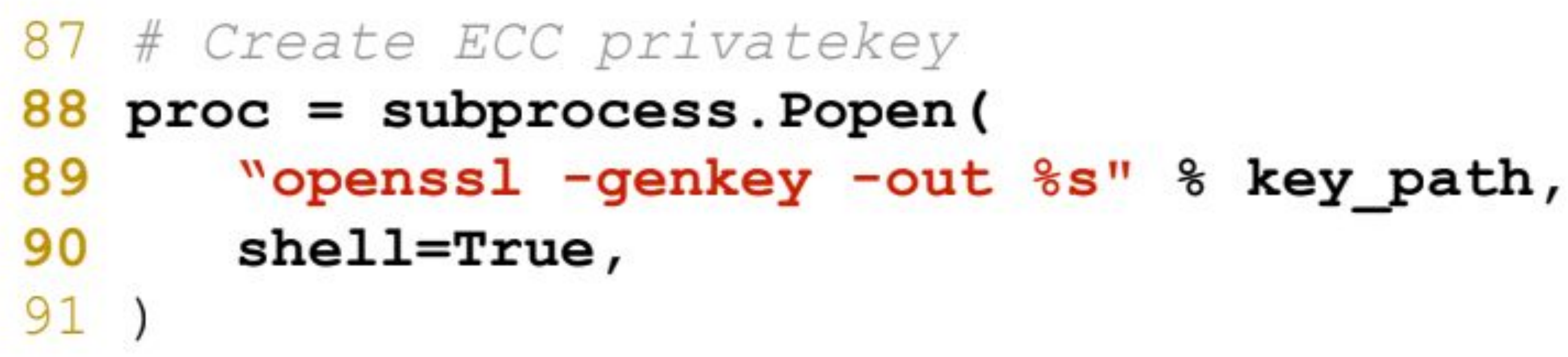
```
shell_injection:
    # Start a process using the subprocess module, or one of its
    wrappers.
    subprocess: [subprocess.Popen, subprocess.call,
                 subprocess.check_call, subprocess.check_output
                 execute_with_timeout]
```



```
>> Issue: [B602:subprocess_popen_with_shell_equals_true] subprocess call with shell=True seems safe, but may be changed
in the future, consider rewriting without shell
   Severity: Low   Confidence: High
   Location: .\subprocess_shell.py:11
   More Info: https://bandit.readthedocs.io/en/latest/plugins/b602_subprocess_popen_with_shell_equals_true.html
10
11      pop('/bin/gcc --version', shell=True)
12      Popen('/bin/gcc --version', shell=True)

--------------------------------------------------
>> Issue: [B604:any_other_function_with_shell_equals_true] Function call with shell=True parameter identified, possible
security issue.
   Severity: Medium   Confidence: Low
   Location: .\subprocess_shell.py:12
   More Info: https://bandit.readthedocs.io/en/latest/plugins/b604_any_other_function_with_shell_equals_true.html
11      pop('/bin/gcc --version', shell=True)
12      Popen('/bin/gcc --version', shell=True)
13
```

Bandit Test plugins

Shell Commands



```
# Create ECC privatekey
proc = subprocess.Popen(
    "openssl -genkey -out %s" % key_path,
    shell=True,
)
```

```
>> Issue: subprocess call with shell=True
identified, security issue.
```

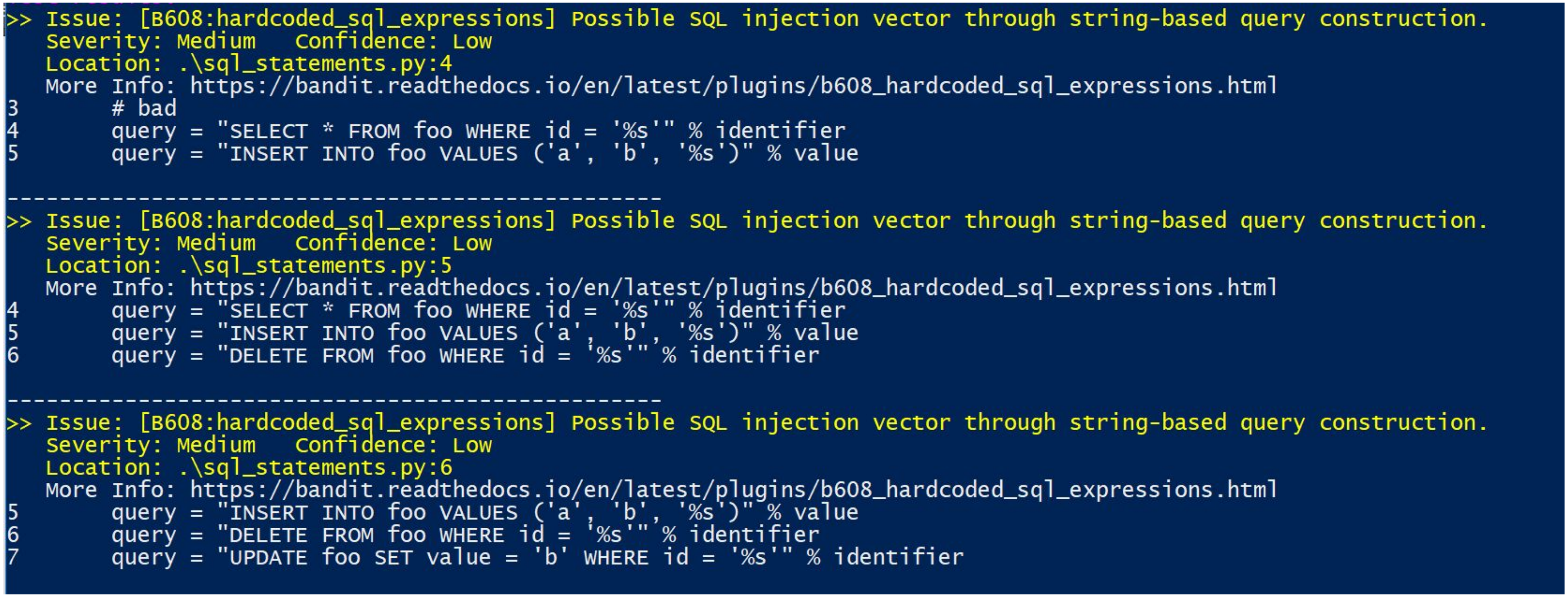
Bandit Test plugins

SELECT %s FROM derp;” % var

“SELECT thing FROM ” + tab

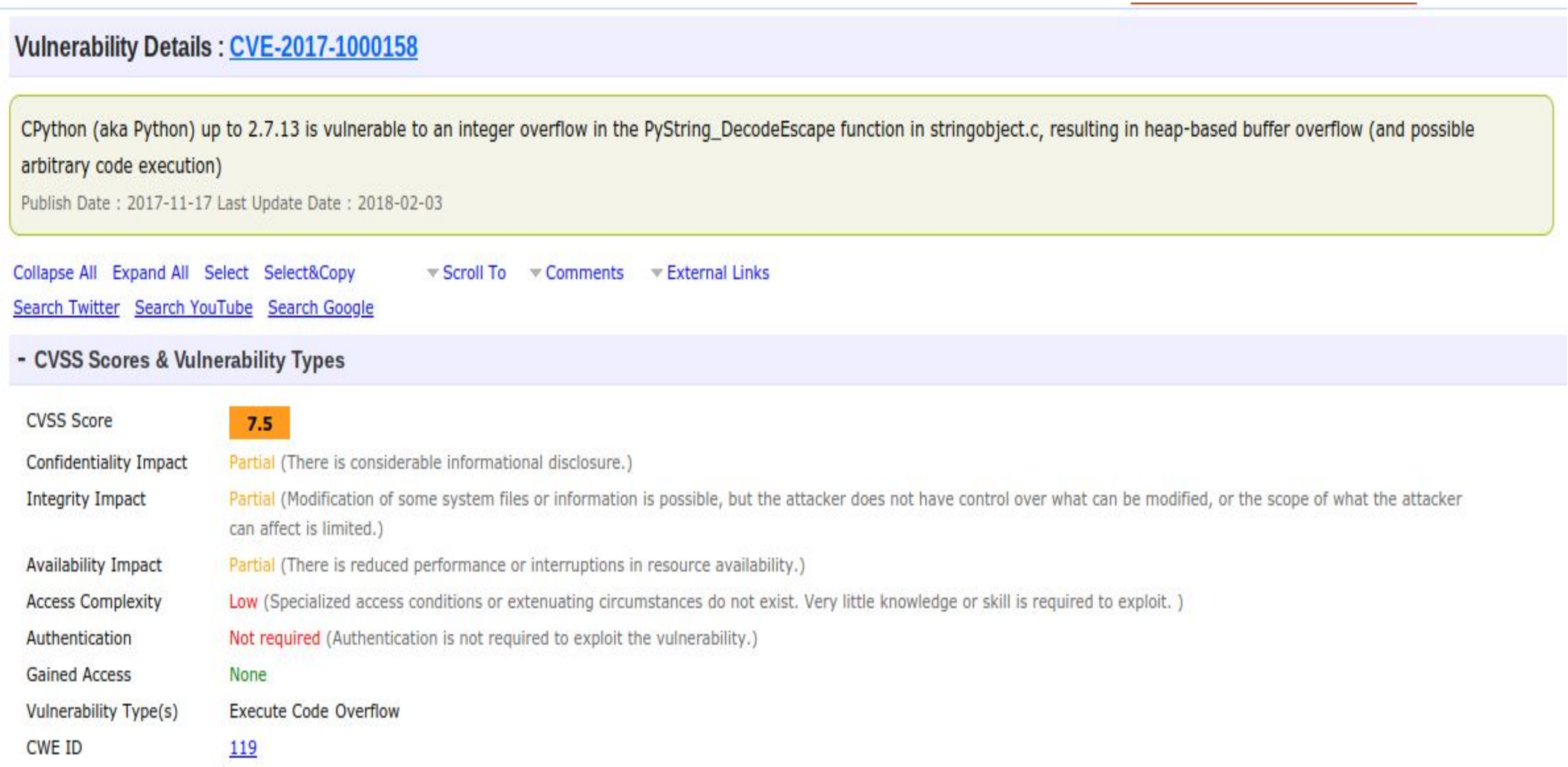
“SELECT ” + val + ” FROM ” + tab + …

“SELECT {} FROM derp;”.format(var)



```
>> Issue: [B608:hardcoded_sql_expressions] Possible SQL injection vector through string-based query construction.
   Severity: Medium   Confidence: Low
   Location: .\sql_statements.py:4
   More Info: https://bandit.readthedocs.io/en/latest/plugins/b608_hardcoded_sql_expressions.html
3       # bad
4       query = "SELECT * FROM foo WHERE id = '%s'" % identifier
5       query = "INSERT INTO foo VALUES ('a', 'b', '%s')" % value

--------------------------------------------------
>> Issue: [B608:hardcoded_sql_expressions] Possible SQL injection vector through string-based query construction.
   Severity: Medium   Confidence: Low
   Location: .\sql_statements.py:5
   More Info: https://bandit.readthedocs.io/en/latest/plugins/b608_hardcoded_sql_expressions.html
4       query = "SELECT * FROM foo WHERE id = '%s'" % identifier
5       query = "INSERT INTO foo VALUES ('a', 'b', '%s')" % value
6       query = "DELETE FROM foo WHERE id = '%s'" % identifier

--------------------------------------------------
>> Issue: [B608:hardcoded_sql_expressions] Possible SQL injection vector through string-based query construction.
   Severity: Medium   Confidence: Low
   Location: .\sql_statements.py:6
   More Info: https://bandit.readthedocs.io/en/latest/plugins/b608_hardcoded_sql_expressions.html
5       query = "INSERT INTO foo VALUES ('a', 'b', '%s')" % value
6       query = "DELETE FROM foo WHERE id = '%s'" % identifier
7       query = "UPDATE foo SET value = 'b' WHERE id = '%s'" % identifier
```

CPython vulnerabilities



Vulnerability Details : CVE-2017-1000158

CPython (aka Python) up to 2.7.13 is vulnerable to an integer overflow in the PyString_DecodeEscape function in stringobject.c, resulting in heap-based buffer overflow (and possible arbitrary code execution)

Publish Date : 2017-11-17 Last Update Date : 2018-02-03

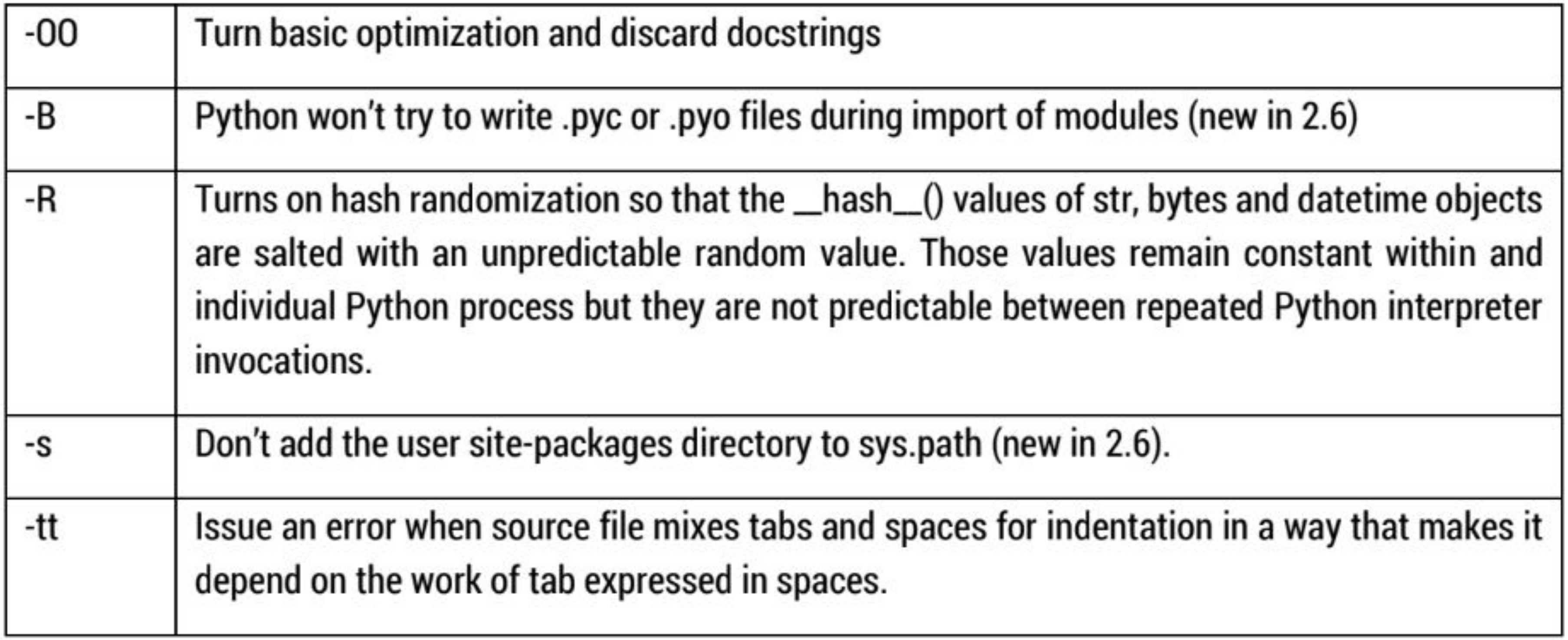
- CVSS Scores & Vulnerability Types

CVSS Score	7.5
Confidentiality Impact	Partial (There is considerable informational disclosure.)
Integrity Impact	Partial (Modification of some system files or information is possible, but the attacker does not have control over what can be modified, or the scope of what the attacker can affect is limited.)
Availability Impact	Partial (There is reduced performance or interruptions in resource availability.)
Access Complexity	Low (Specialized access conditions or extenuating circumstances do not exist. Very little knowledge or skill is required to exploit.)
Authentication	Not required (Authentication is not required to exploit the vulnerability.)
Gained Access	None
Vulnerability Type(s)	Execute Code Overflow
CWE ID	119

Insecure packages

– acqusition (uploaded 2017-06-03 01:58:01, impersonates acquisition)
– apidev-coop (uploaded 2017-06-03 05:16:08, impersonates apidev-coop_cms)
– bzip (uploaded 2017-06-04 07:08:05, impersonates bz2file)
– crypt (uploaded 2017-06-03 08:03:14, impersonates crypto)
– django-server (uploaded 2017-06-02 08:22:23, impersonates django-server-guardian-api)
– pwd (uploaded 2017-06-02 13:12:33, impersonates pwdhash)
– setup-tools (uploaded 2017-06-02 08:54:44, impersonates setuptools)
– telnet (uploaded 2017-06-02 15:35:05, impersonates telnetsrvlib)
– urlib3 (uploaded 2017-06-02 07:09:29, impersonates urllib3)
– urllib (uploaded 2017-06-02 07:03:37, impersonates urllib3)

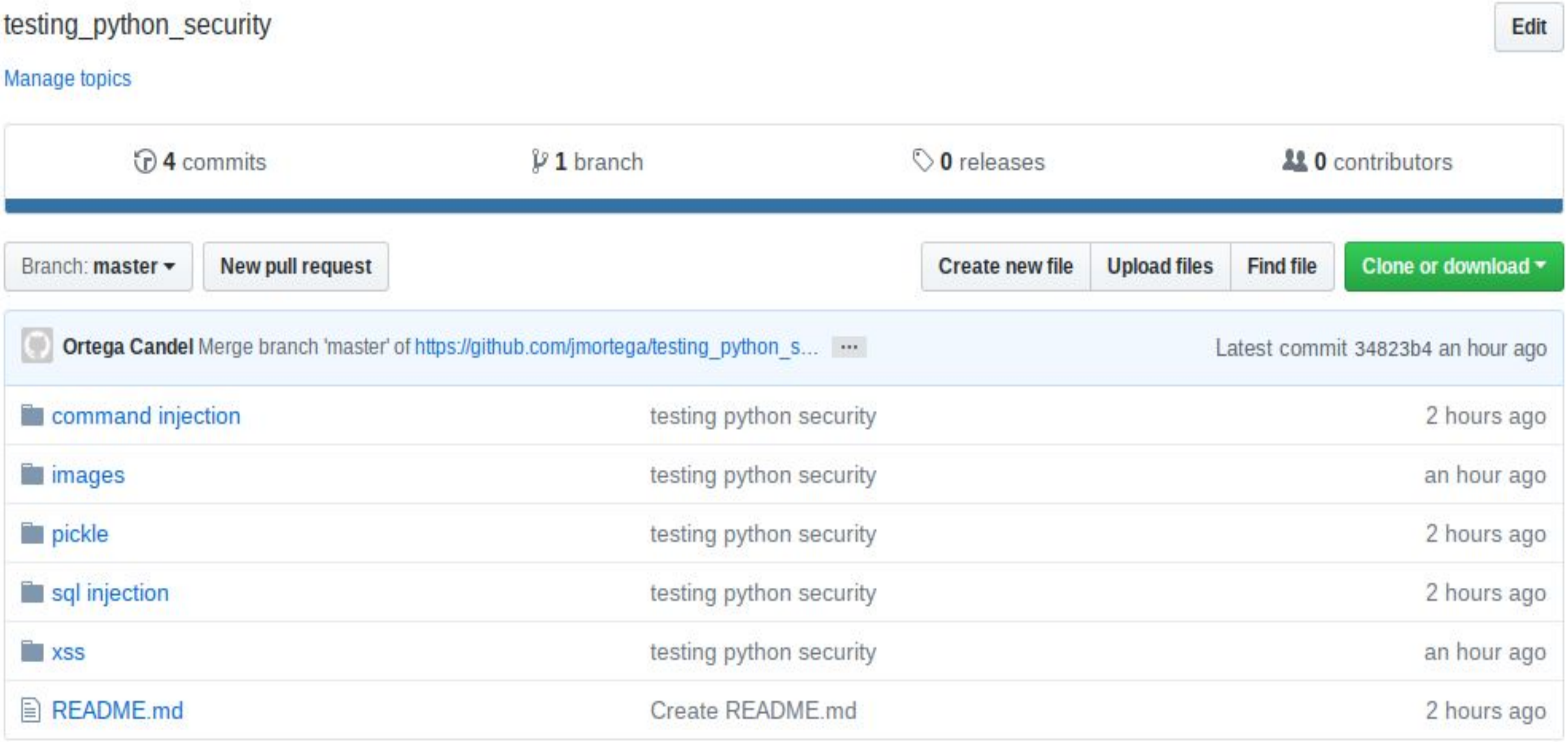
Code optimization



-OO	Turn basic optimization and discard docstrings
-B	Python won't try to write .pyc or .pyo files during import of modules (new in 2.6)
-R	Turns on hash randomization so that the __hash__() values of str, bytes and datetime objects are salted with an unpredictable random value. Those values remain constant within and individual Python process but they are not predictable between repeated Python interpreter invocations.
-s	Don't add the user site-packages directory to sys.path (new in 2.6).
-tt	Issue an error when source file mixes tabs and spaces for indentation in a way that makes it depend on the work of tab expressed in spaces.

Interesting links

https://github.com/jmortega/testing_python_security



testing_python_security

Ortega Candel Merge branch 'master' of https://github.com/jmortega/testing_python_s... Latest commit 34823b4 an hour ago

command injection	testing python security	2 hours ago
images	testing python security	an hour ago
pickle	testing python security	2 hours ago
sql injection	testing python security	2 hours ago
xss	testing python security	an hour ago
README.md	Create README.md	2 hours ago

Interesting links

https://security.openstack.org/guidelines/dg_use-subprocess-securely.html

https://security.openstack.org/guidelines/dg_avoid-shell-true.html

https://security.openstack.org/guidelines/dg_parameterize-database-queries.html

https://security.openstack.org/guidelines/dg_cross-site-scripting-xss.html

https://security.openstack.org/guidelines/dg_avoid-dangerous-input-parsing-libraries.html

Q & A

PYCONES
2018
MÁLAGA