Taint Analysis

Security Testing
Rahul Gopinath, Saarland University
Our Goal

• We want to cause the program to fail
• We have seen
  • random (unstructured) input
  • structured (grammar-based) input
  • generation based on grammar coverage
  • coverage guided fuzzing
  • search based fuzzing
  • symbolic execution
  • grammar inference from input
Mercurial > cpython

changeset 94638:02865d22a98d 2.7

<table>
<thead>
<tr>
<th>Issue #28885: Fixed arbitrary code execution vulnerability in the dumbdbm module. Original patch by Claudiu Popa. [#28885]</th>
</tr>
</thead>
<tbody>
<tr>
<td>author Serhiy Storchaka <a href="mailto:storchaka@gmail.com">storchaka@gmail.com</a></td>
</tr>
<tr>
<td>date Mon, 16 Feb 2015 00:29:52 +0200 (2015-02-15)</td>
</tr>
<tr>
<td>parents 7d2018774925</td>
</tr>
<tr>
<td>children 13f5e5a7fe8</td>
</tr>
<tr>
<td>files Lib/dumbdbm.py Lib/test/test_dumbdbm.py Misc/NEWS</td>
</tr>
<tr>
<td>diffstat 3 files changed, 13 insertions(+), 1 deletions(-) [-]</td>
</tr>
</tbody>
</table>

line diff

```
--- a/Lib/dumbdbm.py
+++ b/Lib/dumbdbm.py
@@ -21,6 +21,7 @@ is read when the database is opened, and
  
 +++
  
+import ast as _ast
+import os as _os
+import _builtin_
+import UserDict
@@ -85,7 +86,7 @@ class _Database(UserDict.DictMixin):
     with f:
         for line in f:
             line = line.rstrip()
-            key, pos_and_siz_pair = eval(line)
+            key, pos_and_siz_pair = _ast.literal_eval(line)
            self._index[key] = pos_and_siz_pair

 # Write the index dict to the directory file. The original directory
```
changeset 94638:02865d22a98d 2.7

Issue #22885: Fixed arbitrary code execution vulnerability in the dumbdbm module. Original patch by Claudiu Popa. [#22885]

author Serhiy Storchaka <storchaka@gmail.com>
date Mon, 16 Feb 2015 00:29:52 +0200 (2015-02-15)
parents 7d2018774925
children 13fc5ea7fe8
files Lib/dumbdbm.py Lib/test/test_dumbdbm.py Misc/NEWS
diffstat 3 files changed, 13 insertions(+), 1 deletions(-) [-]

line diff
--- a/Lib/dumbdbm.py
+++ b/Lib/dumbdbm.py
@@ -21,6 +21,7 @@ is read when the database is opened, and

 ***
@@ -45,7 +46,7 @@ class _Database(UserDict.DictMixin):
 with f:
     for line in f:
         line = line.rstrip()
-       key, pos_and_siz_pair = eval(line)
+       key, pos_and_siz_pair = _ast.literal_eval(line)
         self._index[key] = pos_and_siz_pair
 # Write the index dict to the directory file. The original directory
Defence

Ensure payload does not reach sensitive calls
HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.

OH, DEAR — DID HE BREAK SOMETHING? IN A WAY—

DID YOU REALLY NAME YOUR SON 'Robert'; DROP TABLE Students; --

OH, YES. LITTLE BOBBY TABLES, WE CALL HIM.

WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.

AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.
Sandboxes
Sandboxes?

Eval (with disable dangerous language features disabled)

`eval` (Expression, Globals, Locals)
Sandboxes?

eval('os.system("touch me")', globals(), locals())
Sandboxes?

Restricted Globals?

eval('os.system("touch me"'), globals(), locals())
eval('os.system("touch me")',{}, locals())

Sandboxes?

Restricted Globals?

ref: https://nedbatchelder.com/blog/201206/eval_really_is_dangerous.html
Restricted Globals?

Sandboxes?

```python
eval("__import__('os').system('touch me')", {},{})
```
Sandboxes?

Restricted Globals? ✗

eval("__import__('os').system('touch me')", {},{})
eval("__import__('os').system('touch me')", {},{})

__import__ is part of __builtins__

ref: https://nedbatchelder.com/blog/201206/eval_really_is_dangerous.html
Sandboxes?

Restricted Globals?  ×

Remove __builtins__?

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

__import__ is part of __builtins__

ref: https://nedbatchelder.com/blog/201206/eval_really_is_dangerous.html
Sandboxes?

Restricted Globals? ×

Remove __builtins__?

eval('__import__('os').system('touch me'),
     {'__builtins__':{},{}})

__import__ is part of __builtins__

ref: https://nedbatchelder.com/blog/201206/eval_really_is_dangerous.html
Sandboxes?

Restricted Globals?  

Remove __builtins__?

s = """
(lambda fc=(
    lambda n: [
        c for c in
            ()).__class__.__bases__[0].__subclasses__()
            if c.__name__ == n
    ][0]
):;
    fc("function")(
        fc("code")(0,0,0,0,0,b'\x01',(),(),"",",",0,b",",{};
    )())
)()
"""

eval(s, {'__builtins__':{}})

ref:https://nedbatchelder.com/blog/201206/eval_really_is_dangerous.html
Sandboxes?

Restricted Globals?  ✗

Remove __builtins__?

```
s = """
(lambda fc=(
    lambda n: [
        c for c in
        ().__class__.__bases__[0].__subclasses__()
        if c.__name__ == n
    ][0]
    ):)
    fc("function")(
        fc("code")((0,0,0,0,0,b'\x01',(),(),(),"",,",",0,b",",),{},
    )())

""

eval(s, {'__builtins__':{}})
```

ref:https://nedbatchelder.com/blog/201206/eval_really_is_dangerous.html
Python does not verify bytecode before executing.

We can get arbitrary bytecode to execute!

ref: https://nedbatchelder.com/blog/201206/eval_really_is_dangerous.html
30.1. rexec — Restricted execution framework

Deprecation warning: Since version 2.6, the rexec module has been removed in Python 3.

Changed in version 2.3: Disabled module.

Warning: The documentation has been left in place to help in reading old code that uses the module.

This module contains the ReExec class, which supports rexec.eval(), rexec.executefile(), rexec.execute(), and rexec.import() methods, which are restricted versions of the standard Python functions eval(), execfile(), and import() statements. Code executed in this restricted environment will only have access to modules and functions that are deemed safe; you can subclass ReExec to add or remove capabilities as desired.

Warning: While the rexec module is designed to perform as described below, it does have a few known vulnerabilities which could be exploited by carefully written code. Thus it should not be relied upon in situations requiring “production ready” security. In such situations, execution via sub-processes or very careful “cleaning” of both code and data to be processed may be necessary. Alternatively, help in patching known rexec vulnerabilities would be welcomed.

Note: The ReExec class can prevent code from performing unsafe operations like reading or writing disk files, or using TCP/IP sockets. However, it does not protect against code using extremely large amounts of memory or processor time.

class rexec.ReExec(hooks=None, verbose=False)

Returns an instance of the ReExec class.

hooks is an instance of the ReExec class or a subclass of it. If it is omitted or None, the default ReExec class is instantiated. Whenever the rexec module searches for a module (even a built-in one) or reads a module’s code, it doesn’t actually output to the file system itself. Rather, it calls methods of an ReExec instance that was passed to or created by its constructor. (Actually, the ReExec object doesn’t make these calls — they are made by a module loader object that’s part of the ReExec object. This allows another level of flexibility, which can be useful when changing the mechanics of import within the restricted environment.)
Sandboxes?

!!! WARNING WARNING WARNING WARNING WARNING WARNING WARNING WARNING WARNING !!!
!!!
!!! pysandbox is BROKEN BY DESIGN, please move to a new sandboxing !!!
!!! solution: run python in a sandbox, not the opposite! !!!
!!!
!!! Learn more about pysandbox failure:
!!! https://lwn.net/Articles/574215/
!!!
!!! WARNING WARNING WARNING WARNING WARNING WARNING WARNING !!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

On Linux, SECCOMP security feature looks nice a nice start to build a Python sandbox.

Other sandboxing projects for Python:

* PyPy project has a sandbox:
  http://pypy.org/
* http://chdir.org/~nico/seccomp-nurse/

The old README follows.
Sandbox with *eval* are hard to do correctly
Limited VM
PyPy Sandbox

Sandboxing

PyPy's sandboxing is a working prototype for the idea of running untrusted user programs. Unlike other sandboxing approaches for Python, PyPy's does not try to limit language features considered "unsafe". Instead we replace all calls to external libraries (C or platform) with a stub that communicates with an external process handling the policy.

Please be aware that it is a prototype only. It needs work to become more complete, and you are welcome to help. In particular, almost none of the extension modules work (not even time!), and pypy_interact is merely a demo. Also, a more complete system would include a way to do the same as pypy_interact from other languages than Python, to embed a sandboxed interpreter inside programs written in other languages.

To run the sandboxed process, you need to get the full sources and build pypy-sandbox from it (see Building from source). These instructions give you a pppy-c that you should rename to pppy-sandbox to avoid future confusion. Then run:

cd pppy/sandbox
pppy_interact.py path/to/pppy-sandbox
# don't confuse it with pppy/goal/pyinteractive.py!

You get a fully sandboxed interpreter, in its own filesystem hierarchy (try os.listdir('/')). For example, you would run an untrusted script as follows:

mkdir virtualtmp
cp untrusted.py virtualtmp/
pppy_interact.py --tmp=virtualtmp pppy-sandbox /tmp/untrusted.py

Note that the path /tmp/untrusted.py is a path inside the sandboxed filesystem. You don't have to put untrusted.py in the real /tmp directory at all.

To read more about its features, try pppy_interact.py --help or go to our documentation site.
Roll your own

VIRTUAL MACHINE
class Vm:
    def i_pop_top(self, i): self.stack.pop()
    def i_load_global(self, i): self.stack.append(self.code.co_names[i])
    def i_load_name(self, i): self.stack.append(self.code.co_names[i])
    def i_store_name(self, i):
        if len(self.local) < i + 1: self.local.append(self.stack.pop())
        else: self.local[i] = self.stack.pop()
    def i_load_const(self, i): self.stack.append(self.code.co_consts[i])
    def i_load_fast(self, i): self.stack.append(self.local[i])
    def i_store_fast(self, i):
        if len(self.local) < i + 1: self.local.append(self.stack.pop())
        else: self.local[i] = self.stack.pop()
    def i_return_value(self, i): return self.stack.pop()
    def i_call_function(self, i):
        nargs = i + 1
        fn, *args = self.stack[-nargs:]
        self.stack = self.stack[:len(self.stack)-nargs]
        if type(fn) == tuple:
            self.stack.append(Vm(args).bytes(fn[1]).i().result)
        elif fn in self.code.co_names:
            v = dict(zip(self.code.co_names, self.local))
            if fn not in v:
                l = {**globals(), **locals()}
                if fn not in l: raise Exception('Function[%s] not found' % str(fn))
                self.stack.append(Vm(args).bytes(l[fn].__code__).i().result)
            else:
                (name, myfn, p) = v[fn]
                self.stack.append(Vm(args).bytes(myfn).i().result)
def i(self):  # Vm
    ops = self.code.opcodes
    ins = 0
    while ins < len(ops):
        i = ops[ins]
        if i.opname in mathops:
            fn = mathops[i.opname]
            nargs = fn.__code__.co_argcount
            args = self.stack[-nargs:]
            self.stack = self.stack[:len(self.stack)-nargs]
            v = fn(*args)
            self.stack.append(v)
        elif i.opname in self.fnops:
            fn = self.fnops[i.opname]
            self.result = fn(i.arg)
        elif i.opname in self.jmpops:
            fn = self.jmpops[i.opname]
            ins = fn(i.arg, ins)
            assert ops[i.arg//2].is_jump_target
            continue
        elif i.opname in self.blockops:
            fn = self.blockops[i.opname]
            fn(i.arg)
        elif i.opname in self.otherops:
            fn = self.otherops[i.opname]
            fn(i.arg)
        else:
            assert False
        ins += 1
    return self

v = Vm()
v.statement('def x(a, b): return a+b').i()
v.expr('x(1,2)').i()
print(v.result)
v.expr('(lambda a, b: a+b)(2, 3)').i()
print(v.result)

VM Loop
def i_compare_op(self, opname):  # Vm
    op = dis.cmp_op[opname]
    fn = boolops[op]
    nargs = 2
    args = self.stack[-nargs:]
    self.stack = self.stack[:len(self.stack)-nargs]
    v = fn(*args)
    self.stack.append(v)

def i_pop_jump_if_true(self, i, ins):  # Vm
    v = self.stack.pop()
    if v: return i // 2
    return ins + 1

def i_make_function(self, i):  # Vm
    p = None
    if i == 0x01:
        # tuple of default args in positional order
        p = self.stack.pop()
    elif i == 0x02:
        # a dict of kw only args and vals
        p = self.stack.pop()
    elif i == 0x04:
        # an annotation dict
        p = self.stack.pop()
    elif i == 0x08:
        # closure
        p = self.stack.pop()
    qname = self.stack.pop()
    code = self.stack.pop()
    self.stack.append((qname, code, p))

Interesting Operators
Disabling sensitive APIs is too coarse
Restrict access to sensitive APIs if input comes directly from outside

Disabling sensitive APIs is too coarse
Taint Analysis

A Light Weight Approach
Taint Analysis
Taint Analysis

- Taint (label) data that comes into contact with insecure APIs
Taint Analysis

• Taint (label) data that comes into contact with insecure APIs

• Propagate the taint to any other data derived from the tainted data
Taint Analysis

• Taint (label) data that comes into contact with insecure APIs

• Propagate the taint to any other data derived from the tainted data

• Remove taint if the data is validated
Taint Analysis

- Taint (label) data that comes into contact with insecure APIs
- Propagate the taint to any other data derived from the tainted data
- Remove taint if the data is validated
- If tainted data reaches sensitive APIs RAISE ALARM!
Taint Policies
Taint Policies

• Taint Introduction
Taint Policies

• Taint Introduction

• Taint Propagation
Taint Policies

• Taint Introduction

• Taint Propagation

• Taint Checking
Taint Introduction
Taint Introduction

• User Input
Taint Introduction

- User Input
- Network protocols
Taint Introduction

- User Input
- Network protocols
- Files
Taint Introduction

- User Input
- Network protocols
- Files
- Any external data
Taint Propagation
Taint Propagation

- Which arguments influence the result?
• Which arguments influence the result?

• What is the granularity of tracking?
Taint Propagation

- Which arguments influence the result?

- What is the granularity of tracking?
  - Are different taints tracked?
  - Sub-object (byte level) tracking
Taint Propagation

• Which arguments influence the result?

• What is the granularity of tracking?
  • Are different taints tracked?
  • Sub-object (byte level) tracking

• Is control flow tracked?
Taint Checking
Taint Checking

- What are the sensitive APIs?
Taint Checking

• What are the sensitive APIs?

• What is the action taken?
Taint Analysis
Taint Analysis

• Reliability
Taint Analysis

- Reliability
  - Under-tainting
Taint Analysis

- Reliability
  - Under-tainting
  - Over-tainting
Taint Analysis

- Reliability
  - Under-tainting
  - Over-tainting
- Overhead
Taint Analysis

- Reliability
  - Under-tainting
  - Over-tainting

- Overhead

- Time of attack vs time of Detection
Taint Analysis

- Insecure API (Source) ➔
- Escaping Functions (Sanitizer) ↔
- Sensitive APIs (Sink) ×

Policy: No direct information flow between a tainted source and a sink
Taint Analysis

Policy: No direct information flow between a tainted source and a sink
Taint Analysis

\[
x = \text{input()}
\]

\[
\times
\]

Policy: No direct information flow between a tainted source and a sink
Taint Analysis

Policy: No direct information flow between a tainted source and a sink

$x = \text{input}()$
Taint Analysis

Policy: No direct information flow between a tainted source and a sink

\[ x = \text{input()}; \]
\[ \text{exec}(x); \]
Policy: No direct information flow between a tainted source and a sink

Taint Leak!

exec(x)

x = input()
Taint Analysis

Propagation Policy: transparent
Taint Analysis

\[ x = \text{input()} \]

\[ \times \]

Propagation Policy: transparent
Taint Analysis

$\text{x = x + y}$

$\text{x = input()}$

Propagation Policy: transparent
Taint Analysis

Propagation Policy: transparent

x = input()

x = x \times y

\times \quad \text{exec}(x)
Taint Analysis

Propagation Policy: transparent

x = input()

x = x + y

Taint Leak!

exec(x)
Policy: No direct information flow between a tainted source and a sink unless it passes through a sanitizer
Taint Analysis

\[ x = \text{input()} \]

Policy: No direct information flow between a tainted *source* and a *sink* unless it passes through a *sanitizer*.
Policy: No direct information flow between a tainted source and a sink unless it passes through a sanitizer.
Policy: No direct information flow between a tainted source and a sink unless it passes through a sanitizer
Policy: No direct information flow between a tainted source and a sink unless it passes through a sanitizer.
Taint Analysis

Propagation Policy: opaque
Taint Analysis

\[ x = \text{input()} \]

Propagation Policy: opaque
Taint Analysis

\[ x = \text{input()} \]

\[ x = y \text{ if } x \text{ else } z \]

Propagation Policy: opaque
Taint Analysis

Propagation Policy: opaque

x = input()

x = y if x else z

exec(x)

Propagation Policy: opaque
Taint Analysis

Propagation Policy: opaque

\[ x = \text{input()} \]

\[ x = y \text{ if } x \text{ else } z \]

exec(x)

\[ x = y \text{ if } x \text{ else } z \]
Static Taint Analysis
Static Taint Analysis

- Offline - Use the control flow graph
Static Taint Analysis

- Offline - Use the control flow graph
- No runtime overhead
- Significant over/under-tainting
- Scalability issues in large codebases
- Difficult in heavily interactive applications
Dynamic Taint Analysis
Dynamic Taint Analysis

- Runtime taint propagation
Dynamic Taint Analysis

- Runtime taint propagation
- Significant runtime overhead
- Precise tainting
Instrumenting Python VM
DEMO
def _call(fn, tupl): return Instrument.i(fn)(*tupl)

def _unary(a, op):
    v = unaryops[op](a)
    if Instrument.is_tainted(a): Instrument.mark(v)
    return v

def _bin(a, b, op):
    v = binops[op](a, b)
    if Instrument.is_tainted((a, b)): Instrument.mark(v)
    return v

class Instrument:
    @classmethod
    def mark(cls, i):
        global tainted
        tainted[id(i)] = True

    @classmethod
    def unmark(cls, i):
        global tainted
        if id(i) in tainted: del tainted[id(i)]

    @classmethod
    def is_tainted(cls, obj):
        global tainted
        if id(obj) in tainted: return True
        elif isinstance(obj, dict):
            for k, v in obj.items():
                if cls.is_tainted(k) or cls.is_tainted(v): return True
        elif isinstance(obj, tuple):
            for v in obj:
                if cls.is_tainted(v): return True
        elif isinstance(obj, list):
            for v in obj:
                if cls.is_tainted(v): return True
        else:
            return False
```python
def __init__(self, func):  # Instrument
    self._function = func
    self.fn = Function(func)
    lst = []
jump_displacement = 0
for i in self.fn.opcodes:
    op = i.opname
    if op in binops:
        self.fn.co_names.extend(['fn', '__bin'])
        self.fn.consts.append(op)
        ops = [self.i_load_global(), self.i_load_attr('__bin'),
               self.i_rot_three(), self.i_load_const(op), self.i_call_function(3)]
        lst.extend(ops)
        jump_displacement += len(ops) - 1
    elif op in unaryops:
        self.fn.co_names.extend(['fn', '__unary'])
        self.fn.consts.append(op)
        ops = [self.i_load_global(), self.i_load_attr('__unary'),
               self.i_rot_two(), self.i_load_const(op), self.i_call_function(2)]
        lst.extend(ops)
        jump_displacement += len(ops) - 1
    elif i.opname == 'CALL_FUNCTION':
        nargs = i.arg
        self.fn.co_names.extend(['fn', '__call'])
        self.fn.consts.append(op)
        ops = [self.i_build_tuple(nargs), self.i_load_global(),
               self.i_load_attr('__call'), self.i_rot_three(),
               self.i_call_function(2)]
        lst.extend(ops)
        jump_displacement += len(ops) - 1
    elif i.opname in jumpops:
        j = jumpops[i.opname](self, i, jump_displacement)
        lst.append(j)
    else:
        lst.append(i)
    self.fn.opcodes = lst
    self.fn.update_bytecode()
    self.function = self.fn.build()
```

Can input get to Eval?

https://hg.python.org/cpython/rev/02865d22a98d

Mercurial > cpython

changeset 94638:02865d22a98d 2.7

Issue #22885: Fixed arbitrary code execution vulnerability in the dumbdbm module. Original patch by Claudiu Popa. [#22885]

author Serhiy Storchaka <storchaka@gmail.com>
date Mon, 16 Feb 2015 00:29:52 +0200 (2015-02-15)
parents 7d2018774925
children 13fc5e7e8e
files Lib/dumbdbm.py Lib/test/test_dumbdbm.py Misc/NEWS
diffstat 3 files changed, 13 insertions(+), 1 deletions(-) [-]

line diff

```
--- a/Lib/dumbdbm.py
+++ b/Lib/dumbdbm.py
@@ -21,6 +21,7 @@ is read when the database is opened, and
     ***
@@ -17,9 +17,11 @@
+import _ast as _ast
+import os as _os
+import _builtin_
+import UserDict
@@ -57,7 +56,7 @@ class _Database(UserDict.DictMixin):
     with f:
         for line in f:
             line = line.rstrip()
-            key, pos_and_size_pair = eval(line)
+            key, pos_and_size_pair = _ast.literal_eval(line)
             self._index[key] = pos_and_size_pair
@@ -189,3 +188,6 @@ class _Database(UserDict.DictMixin):
             self._index[key] = pos_and_size_pair

 # Write the index dict to the directory file. The original directory
```
Track Objects!
Track Objects!

DEMO
Tainted String

class tstr(str):
    def __new__(cls, value):
        s = str.__new__(cls, value)
        s._s = value
        return s
    def __radd__(self, o):
        return tstr(str.__add__(o, self))
    def __repr__(self):
        return 'Tainted: ' + str.__repr__(self)
    def untaint(self):
        return self._s

def mark(module):
    for name in dir(module):
        obj = getattr(module, name)
        if isinstance(obj, (types.FunctionType, types.BuiltinFunctionType)):
            yield (module, name, obj)
class ProxyIter(collections.Iterator):
    def __init__(self, i): self.i = i
    def __iter__(self): return self
    def __enter__(self): return self
    def __exit__(self, exc_type, exc_val, exc_tb):
        return self.i.__exit__(exc_type, exc_val, exc_tb)
    def __next__(self): return tstr(self.i.__next__())
    def __hasattr__(self, name):
        return hasattr(self.i, name)
    def __getattr__(self, name):
        func = getattr(self.i, name)
        return source(func) if func else None

def source(func):
    def wrapper(*args, **kwargs):
        v = func(*args, **kwargs)
        if isinstance(v, list):
            return [tstr(l) for l in v]
        elif isinstance(v, tuple):
            return tuple(tstr(l) for l in v)
        elif isinstance(v, set):
            return set(tstr(l) for l in v)
        elif isinstance(v, dict):
            return {tstr(k): tstr(l) for k, l in v}
        elif isinstance(v, str):
            return tstr(v)
        elif isinstance(v, collections.Iterator):
            return ProxyIter(v)
        else:
            return v
    return wrapper

def mark_sources(module):
    for (module, name, obj) in mark(module):
        setattr(module, name, source(obj))
def sink(func):
    @functools.wraps(func)
    def wrapper(*args, **kwargs):
        if any(isinstance(e, tstr) for e in list(args) + list(kwargs.values())):
            raise Exception("tainted")
        return func(*args, **kwargs)
    return wrapper

def mark_sinks(module):
    for (module, name, obj) in mark(module):
        setattr(module, name, sink(obj))
def sanitizer(func):
    @functools.wraps(func)
    def wrapper(*args, **kwargs):
        my_args = [a.untaint() for a in args]
        my_kwargs = {k:v.untaint() for k,v in kwargs}
        return func(*my_args, **my_kwargs)
    return wrapper
def make_strtuple_wrapper(fun):
    return lambda *a, **kw: tuple(tstr(l) for l in fun(*a, **kw))

def make_strlst_wrapper(fun):
    return lambda *a, **kw: [tstr(l) for l in fun(*a, **kw)]

def make_str_wrapper(fun):
    return lambda *a, **kw: tstr(fun(*a, **kw))

for name, fn in inspect.getmembers(str, callable):
    tuple_names = ['partition', 'rpartition']
    bool_names = ['__eq__', '__lt__', '__gt__', '__contains__']
    list_names = ['rsplit', 'splitlines', 'split']
    repr_names = ['__repr__', '__str__', '__hash__']
    if name not in ['__class__', '__new__', '__init__', '__getattribute__',
                    '__init_subclass__', '__subclasshook__', '__setattr__',
                    '__len__', 'find', 'rfind', '__iter__']
    + tuple_names + list_names + bool_names + repr_names:
        setattr(tstr, name, make_str_wrapper(fn))
    elif name in list_names:
        setattr(tstr, name, make_strlst_wrapper(fn))
    elif name in tuple_names:
        setattr(tstr, name, make_strtuple_wrapper(fn))
Where Tainting Fails

```
i = input()
x = c_identity(i)
print(x)
```
Where Tainting Fails

```python
i = input()
x = c_identity(i)
print(x)
```
Where Tainting Fails

\[
i = \text{input}() \\
x = \text{c_identity}(i) \\
\text{print}(x)
\]

\[
\text{int c_identity(int i) }
\{
\text{return i}
\}
\]
Where Tainting Fails

```python
i = input()
x = c_identity(i)
print(x)
```

```python
int c_identity(int i) {
    return i
}
```
Where Tainting Fails

```python
i = input()
x = c_identity(i)
print(x)
```

```c
int c_identity(int i) {
    return i
}
```
Where Tainting Fails

```python
i = input()
x = c_identity(i)
print(x)

int c_identity(int i) {
    return i
}
```
Where Tainting Fails

i = input()
x = c_identity(i)
print(x)

int c_identity(int i) {
    return i
}

No source code for native functions
→ No taint propagation
Tainting Without Taints

```python
i = input()
x = c_identity(i)
print(x)
```
Tainting Without Taints

```python
i = input()
x = c_identity(i)
print(x)
```
Tainting Without Taints

```
i = input()
x = c_identity(i)
print(x)
```

Log Original

```
input = 42
```
Tainting Without Taints

```python
i = input()
x = c_identity(i)
print(x)
```

Log Original

```
input = 42
output = 42
```
## Tainting Without Taints

```python
i = input()
x = c_identity(i)
print(x)
```

### Log Original
- **input**: 42
- **output**: 42

### Log Mutation
- **input**: 21
Tainting Without Taints

```python
i = input()
x = c_identity(i)
print(x)
```

<table>
<thead>
<tr>
<th>Log Original</th>
<th>Log Mutation</th>
</tr>
</thead>
<tbody>
<tr>
<td>input = 42</td>
<td>input = 21</td>
</tr>
<tr>
<td>output = 42</td>
<td>output = 21</td>
</tr>
</tbody>
</table>
Tainting Without Taints

```python
i = input()
x = c_identity(i)
print(x)
```

<table>
<thead>
<tr>
<th>Log Original</th>
<th>Log Mutation</th>
</tr>
</thead>
<tbody>
<tr>
<td>input = 42</td>
<td>input = 21</td>
</tr>
<tr>
<td>output = 42</td>
<td>output = 21</td>
</tr>
</tbody>
</table>

Mutating the input changes the output
→ There must be a data dependency
Tainting Without Taints
Tainting Without Taints

- No explicit taint propagation
Tainting Without Taints

• No explicit taint propagation
• Instrument program to …
Tainting Without Taints

• No explicit taint propagation

• Instrument program to …
  • Observe sinks
Tainting Without Taints

- No explicit taint propagation
- Instrument program to …
  - Observe sinks
  - Mutate sources (i.e. create several mutated versions of the program)
Tainting Without Taints
Tainting Without Taints

- Execute program versions with the same input
Tainting Without Taints

- Execute program versions with the same input
- Analyze the logged values
Tainting Without Taints

- Execute program versions with the same input
- Analyze the logged values
  - Changes in the sinks indicate a data dependency to the mutated source
MutaFlow

- Successfully tested for Android applications
- Detecting Information Flow by Mutating Input Data: https://dl.acm.org/citation.cfm?id=3155598
- Code publicly available
  - https://github.com/bjrnmath/mutaflow
Our Goal

• We want to cause the program to fail

• We have seen
  • random (unstructured) input
  • structured (grammar-based) input
  • generation based on grammar coverage
  • coverage guided fuzzing
  • search based fuzzing
  • symbolic execution
  • grammar inference from input (AUTOGRAM)
Review

AUTOGRAM
parseURL(spec) -> setURL(protocol, host, port, authority,...)

input

http://admin:pass123@www.google.com:80
\texttt{parseURL}(\texttt{spec})
\rightarrow \texttt{setURL}(\texttt{protocol, host, port, authority,...})
\rightarrow \texttt{setUserInfo}(\texttt{user, password})
parseURL(spec)
-> setURL(protocol, host, port, authority,...)
   -> setuserInfo(user, password)
parseURL(spec)
-> setURL(protocol, host, port, authority,...)
    -> setUserinfo(user, password)
**parseURL**(spec)

-> **setURL**(protocol, host, port, authority,...)

-> **setUserInfo**(user, password)

**SPEC** ::= **PROTOCOL** `://` **AUTHORITY** `@` **HOST** [`:` **PORT**]

**AUTHORITY** ::= **USER** `:` **PASSWORD**

**USER** ::= r{[a-z]+}

**PASSWORD** ::= r{[a-z0-9]+}

**HOST** ::= r{[a-z]+}

**PORT** ::= r{[0-9]+}
def urlparse(url):
    scheme, netloc, url, query, fragment = urlsplit(url)
    if scheme in uses_params and ';' in url:
        url, params = _splitparams(url)
    else:
        params = ''
    return scheme, netloc, url, params, query, fragment

def _splitparams(url):
    if '/' in url:
        i = url.find(';', url.rfind('/'))
        if i < 0:
            return url, ''
    else:
        i = url.find('
')
    return url[:i], url[i+1:]

def urlsplit(url, scheme='', allow_fragments=1):
    key = url, scheme, allow_fragments
    netloc = query = fragment = ''
    i = url.find(':')
    if i > 0:
        if url[:i] == 'http':  # optimize the common case
            scheme = url[:i].lower()
            url = url[i+1:]
        if url[2] == '/':
            i = url.find('/', 2)
            if i < 0:
                i = len(url)
            netloc = url[2:i]
            url = url[i:]
    if allow_fragments and '#' in url:
        url, fragment = url.split('#', 1)
    if '?' in url:
        url, query = url.split('?', 1)
    return (scheme, netloc, url, query, fragment)

http://www.st.cs.uni-saarland.de/zeller#ref
https://www.cispa.saarland:80/bar
http://foo@google.com:8080/bar?q=r#ref2

$START ::= $SCHEME://$URL
    | $SCHEME://$URL#$FRAGMENT
    | $SCHEME://$URL?$QUERY#$FRAGMENT

$PATH ::= /bar
    | /zeller

$SCHEME ::= http
    | https

$FRAGMENT ::= ref
    | ref2

$NETLOC ::= foo@google.com:8080
    | www.cispa.saarland:80
    | www.st.cs.uni-saarland.de

$URL ::= $NETLOC$PATH

$QUERY ::= q=r
How can we make it better?
Track with Taints!
DEMO
class tstr(str):
    
    """Basic tainting class""
    def __new__(cls, value, parent=None):
        assert parent is not None
        s = str.__new__(cls, value)
        s._parent = parent
        return s

    def __radd__(self, other): return tstr(str.__add__(other, self), self)
    def __repr__(self): return self.__class__.__name__ + str.__repr__(self)
    def i(self): return id(self)
    def parent(self): return self._parent

    def is_child_of(self, other):
        s = self
        while type(s) == tstr:
            if other.i() == s.i(): return True
            s = s.parent()
        return False

    def __iter__(self):
        for i in str.__iter__(self): yield tstr(i, self)
def make_strtuple_wrapper(fun):
    def proxy(*a, **kw):
        return tuple(tstr(l, a[0]) for l in fun(*a, **kw))
    return proxy

def make_strlst_wrapper(fun):
    def proxy(*a, **kw):
        return [tstr(l, a[0]) for l in fun(*a, **kw)]
    return proxy

def make_str_wrapper(fun):
    def proxy(*a, **kw):
        return tstr(fun(*a, **kw), a[0])
    return proxy

for name, fn in inspect.getmembers(str, callable):
    tuple_names = ['partition', 'rpartition']
    bool_names = ['__eq__', '__lt__', '__gt__', '__contains__']
    list_names = ['rsplit', 'splitlines', 'split']
    repr_names = ['__repr__', '__str__', '__hash__']

    if name not in ['__class__', '__new__', '__init__', '__getattribute__',
                    '__init_subclass__', '__subclasshook__', '__setattr__',
                    '__len__', 'find', 'rfind', '__iter__',
                    '__repr__', '__str__', '__hash__'] + tuple_names + list_names + bool_names + repr_names:
        setattr(tstr, name, make_str_wrapper(fn))
    elif name in list_names: setattr(tstr, name, make_strlst_wrapper(fn))
    elif name in tuple_names: setattr(tstr, name, make_strtuple_wrapper(fn))
class InputStack:
    inputs = []

    def has(val):
        return (tainted(val) and
                any(val.is_child_of(var) for var in InputStack.inputs[-1].values()))

    def push(inputs):
        if InputStack.inputs:
            my_inputs = {k:v for k,v in inputs.items() if InputStack.has(v)}
            InputStack.inputs.append(my_inputs)
        else:
            my_inputs = {k:v for k,v in inputs.items() if type(v) is tstr}
            InputStack.inputs.append(my_inputs)

    def pop():
        return InputStack.inputs.pop()
Questions!