fuckitllvm

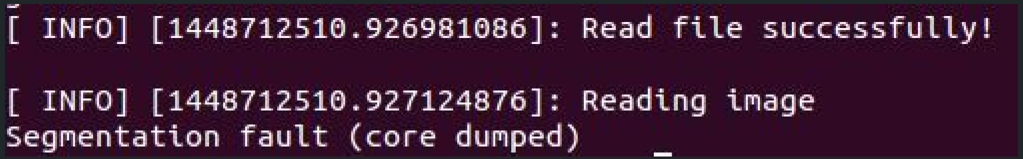
An experiment in improving program reliability

LLVM COMPILER INFRASTRUCTURE

Blue-Eyes LLVM Dragon
[DRAGON]
This legendary dragon is a powerful engine of destruction. Virtually invincible, very few have faced this awesome creature and lived to tell the tale.
CT13-EN008
89631139 LIMITED EDITION
ATK/3000 DEF

Motivation

Modern day C/C++ programming is hard.



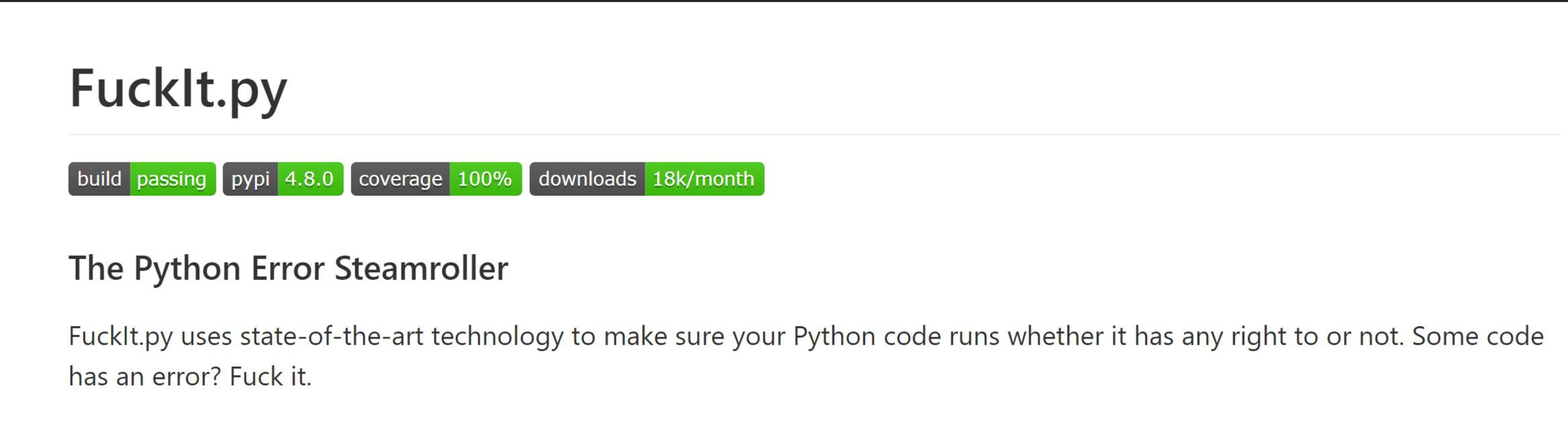
```
[ INFO] [1448712510.926981086]: Read file successfully!

[ INFO] [1448712510.927124876]: Reading image
Segmentation fault (core dumped)
```

Motivation

Javascript and Python developers have *advanced tooling* at their disposal

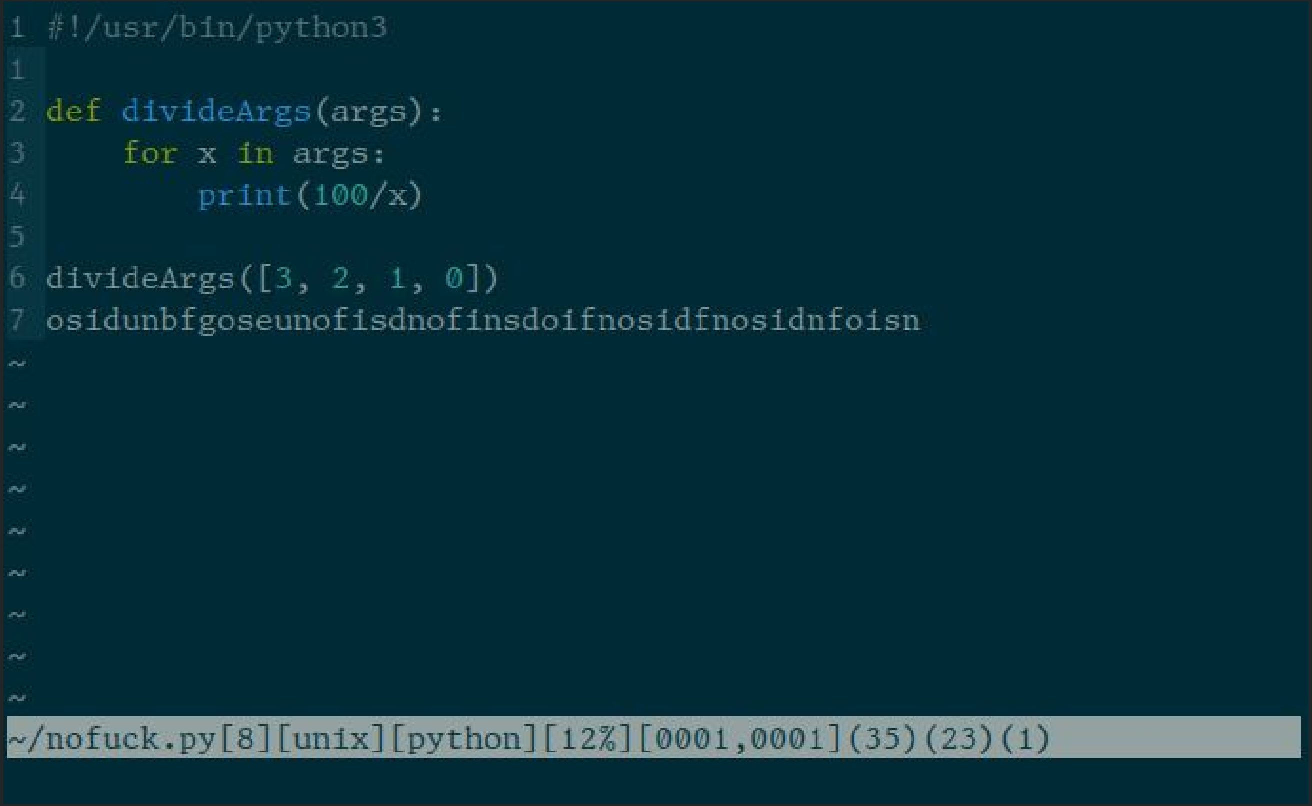
- Fuckitjs
- Fuckitpy



FuckIt.py

The Python Error Steamroller

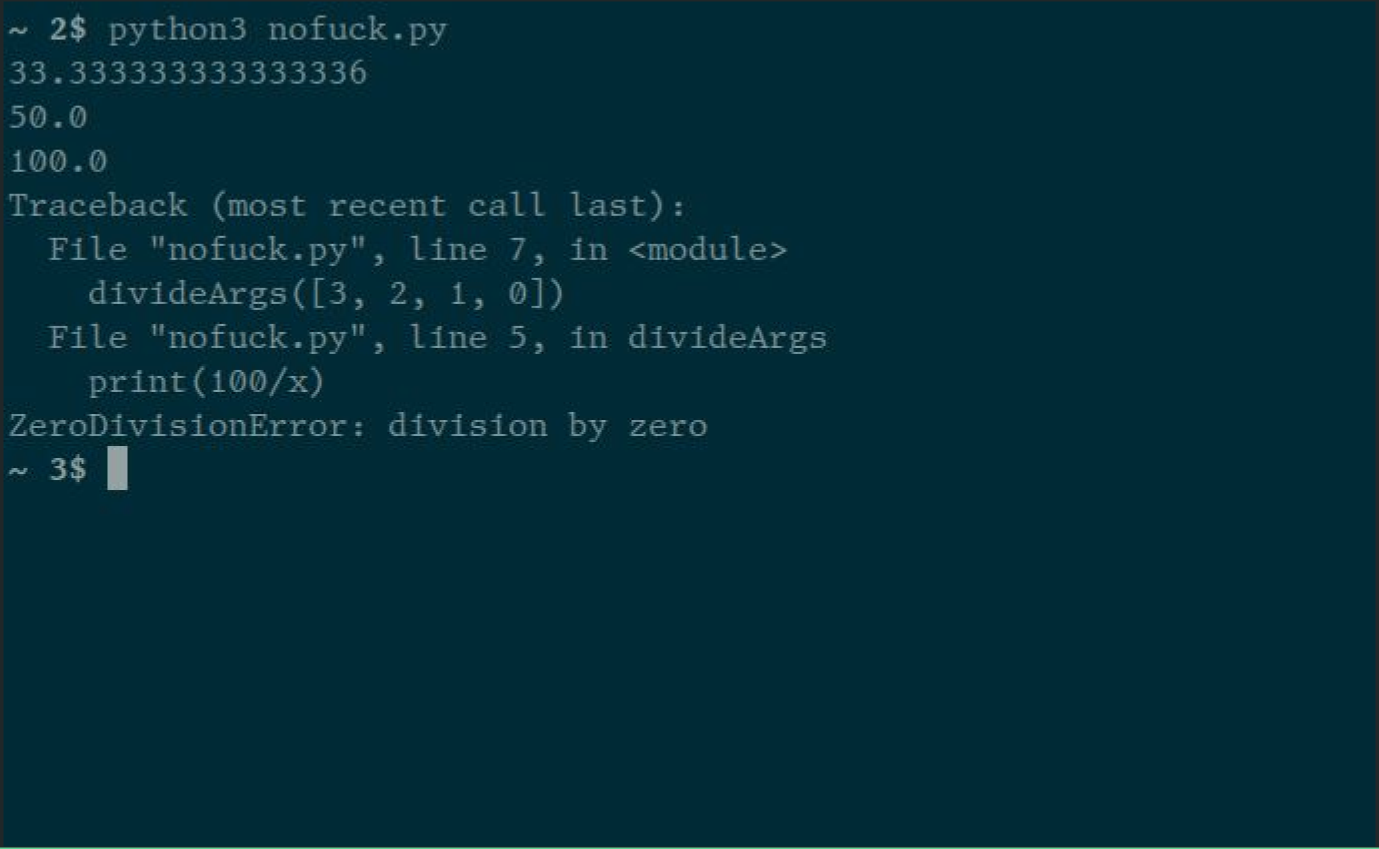
FuckIt.py uses state-of-the-art technology to make sure your Python code runs whether it has any right to or not. Some code has an error? Fuck it.



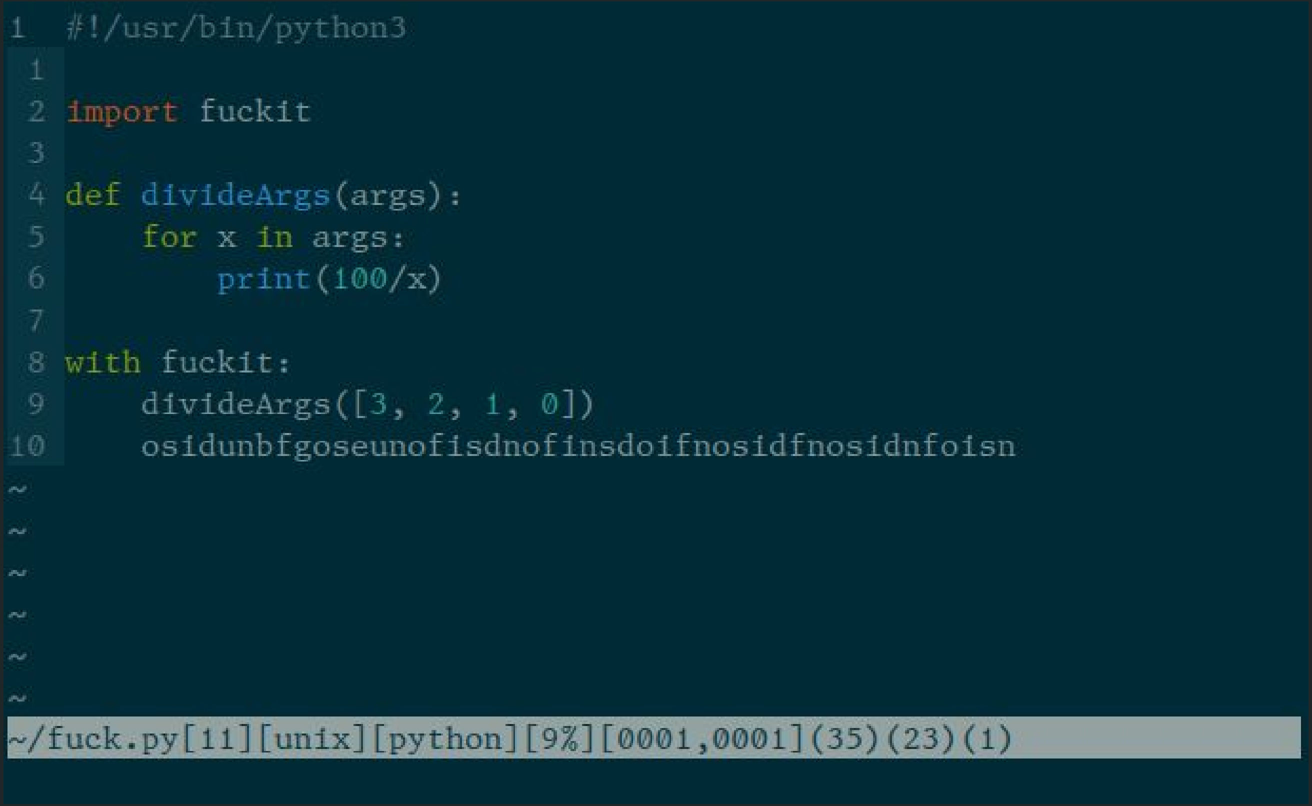
```
#!/usr/bin/python3

def divideArgs(args):
    for x in args:
        print(100/x)

divideArgs([3, 2, 1, 0])
osidunbfgoseunofisdnofinsdoifnosidfnosidnfoisn
~
~
~
~
~
~
~
~
~
~/nofuck.py[8][unix][python][12%][0001,0001](35)(23)(1)
```



```
~ 2$ python3 nofuck.py
33.333333333333336
50.0
100.0
Traceback (most recent call last):
  File "nofuck.py", line 7, in <module>
    divideArgs([3, 2, 1, 0])
  File "nofuck.py", line 5, in divideArgs
    print(100/x)
ZeroDivisionError: division by zero
~ 3$ 
```

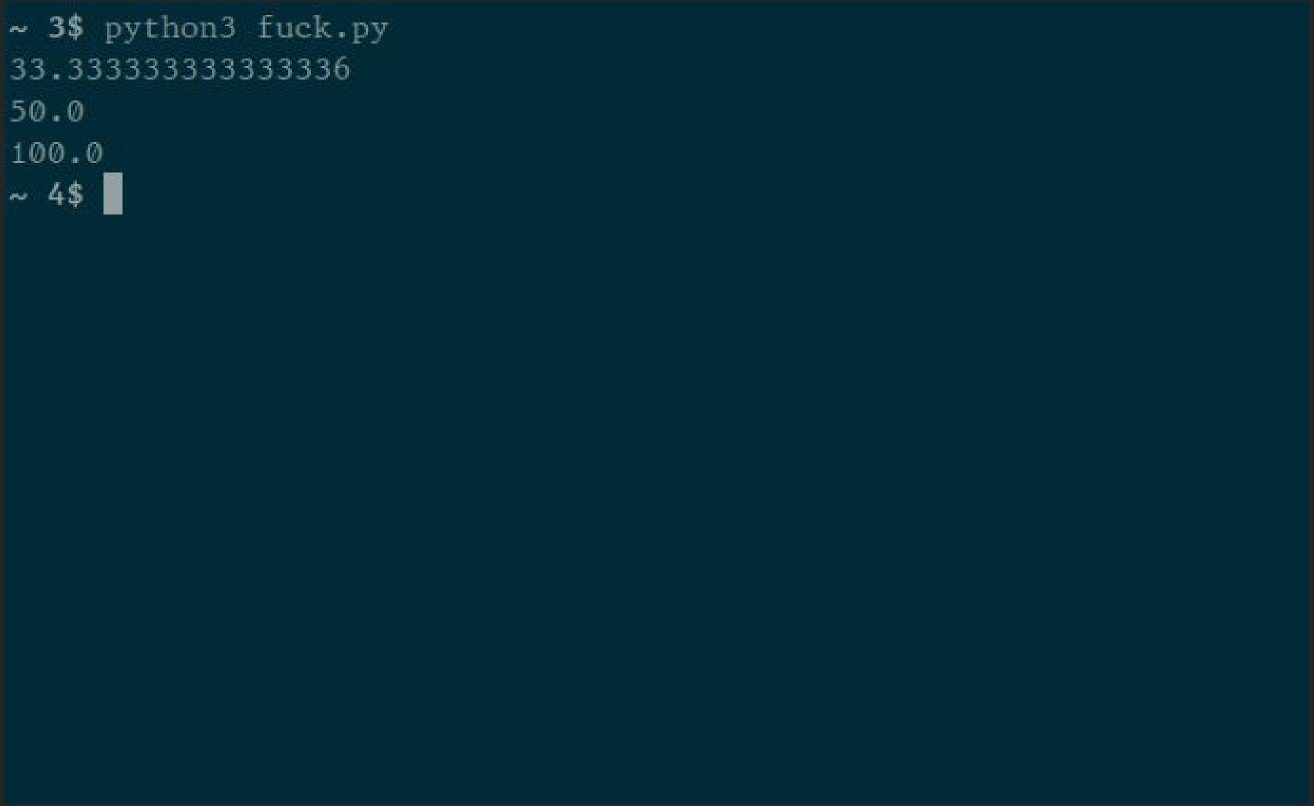


```
#!/usr/bin/python3

import fuckit

def divideArgs(args):
    for x in args:
        print(100/x)

with fuckit:
    divideArgs([3, 2, 1, 0])
    osidunbfgoseunofisdnofinsdoifnosidfnosidnfoisn
~
~
~
~
~
~
~/fuck.py[11][unix][python][9%][0001,0001](35)(23)(1)
```



```
~ 3$ python3 fuck.py
33.333333333333336
50.0
100.0
~ 4$
```

Motivation

We want to make compiled binaries that refuse to give up.

Motivation

We want to make compiled binaries that refuse to give up.

Or at least *look like* they might still be working.

Architectural Overview

Fuckitllvm is made up of a series of passes which improve reliability and resiliency.

2015-16 Theme For the Year: “Resilience”

Architectural Overview

Fuckitllvm is made up of a series of passes which improve reliability and resiliency.

These passes implement those values into target programs.

- Error Redirection and Mitigation
- Dynamic Program Correction
- “Survival of the Fittest”
- Control Flow Recovery

Error Redirection and Mitigation

Programs have two primary/standard ways of inconveniencing the user when they encounter potential problems.

1. Printing to Standard Error (stderr)
2. Returning a non-zero exit-code

Error Redirection and Mitigation

Programs have two primary/standard ways of inconveniencing the user when they encounter potential problems.

1. Printing to Standard Error (stderr)
2. Returning a non-zero exit-code

Our Solution: Mitigate the risk of unintended consequences by employing advanced redirection techniques.

Error Redirection and Mitigation

Programs have two primary/standard ways of inconveniencing the user when they encounter potential problems.

1. Printing to Standard Error (stderr)
   a. Close Stderr. Open /dev/null. ***Errors now go where they belong.***
2. Returning a non-zero exit-code
   a. Identify calls to exit. Make them return 0.
   b. Identify return from main. Make it return 0.
   c. No More Errors!!!

Dynamic Program Correction

Programs are not perfect. Sometimes, there are unavoidable mistakes.

```
int i = 0x41414141;
*(int *)i = 0x10;
printf("I'm still alive!\n");
```

Dynamic Program Correction

Programs are not perfect. Sometimes, there are unavoidable mistakes.

```
int i = 0x41414141;
*(int *)i = 0x10;
printf("I'm still alive!\n");
```

Advanced Program Analysis Techniques allow us to dynamically correct such subtle errors.

Dynamic Program Correction

Programs are not perfect. Sometimes, there are unavoidable mistakes.

```
int i = 0x41414141;
*(int *)i = 0x10;
printf("I'm still alive!\n");
```

1. Inject Signal Handlers at program start
2. If we get a SIGSEGV or SIGILL
    a. Increment instruction pointer

Survival of the Fittest

Unfortunately, our program has to share resources with all other userspace programs.

Survival of the Fittest

Unfortunately, our program has to share resources with all other userspace programs.

Normal programs are prone to being terminated by other programs, especially if another program thinks it is malfunctioning.

Survival of the Fittest

Unfortunately, our program has to share resources with all other userspace programs.

Our Solution: Create a genetically superior program.

```
if ( (siginfo->si_signo == SIGTERM) ) {
    if ( (my_pid != siginfo->si_pid)) {
        // wanna fite m8?
        kill(siginfo->si_pid, SIGKILL);
        return;
    }
}
```

Control Flow Recovery

Sometimes, things go *really* bad.

Like, end up at a point where something is calling SIGABRT bad.

Control Flow Recovery

Just catch it and try to find a suitable place to resume execution!

General Strategy:

- Walk up the stack until we see something that looks like it might be a suitable return address.
- Return there and throw the stack back around that spot.

Control Flow Recovery

Just catch it and try to find a suitable place to resume execution!

Problems:

- Lots of pointers to places we don’t want to return to
- Library Functions handle stack frames differently.

Control Flow Recovery

Lots of pointers to places we don’t want to return to

- Add “Marker” instructions after all calls that we think are an “okay” place to return to (.text is a good candidate)
- Check for these markers in SIGABRT handler
- Stick 0 into RAX before jumping back for good measure.

Control Flow Recovery

Return there and throw the stack back around that spot.

Accidentally discovered an interesting problem!

Stack Frames for library calls are often somewhat “non-standard”.

Control Flow Recovery

Return there and throw the stack back around that spot.

Accidentally discovered an interesting problem!

Stack Frames for library calls are often somewhat “non-standard”.

Makes it difficult to “put the stack back”, so it matches the frame we return to.

Future Work

Add hooks before calling functions to record frame boundaries in some dedicated memory region.

Check if potential base pointers fall within these boundaries

Add try-catch around every basic block

Questions/Outrage?