Precise Dynamic Data Flow Tracking with Proximal Gradients

Gabriel Ryan[1], Abhishek Shah[1], Dongdong She[1], Koustubha Bhat[2], Suman Jana[1]

[1]Columbia University [2]Vrije Universiteit

gabe@cs.columia.edu, as5258@columbia.edu, ds3619@columbia.edu, k.bhat@vu.nl, suman@cs.columia.edu

```
taint source: x
taint sink: y
// x is a 4 byte int
int x = 0x12345678;
for (int i=0; i<6; i++){
    y[i] = x;
    x = x<<8;
}
```

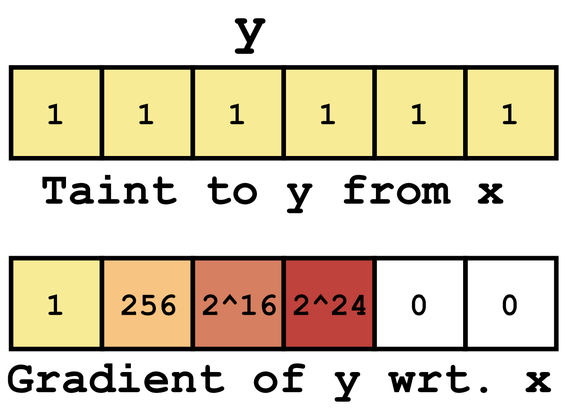
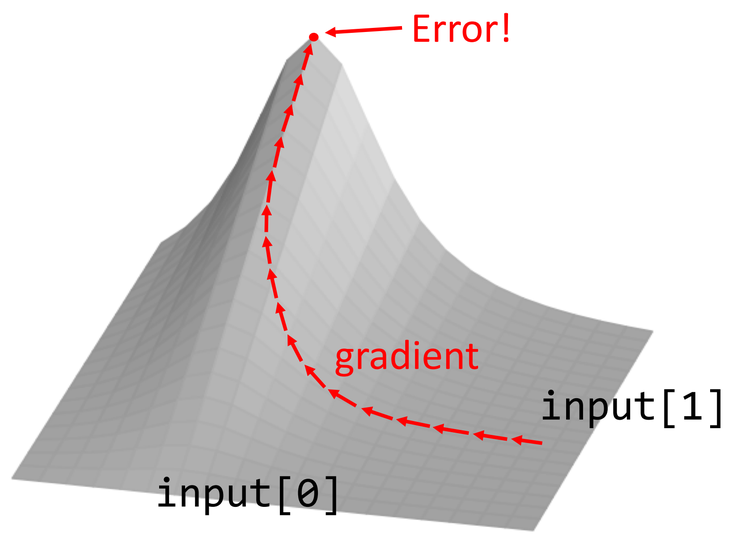


Figure 1. Gradient avoids approximation errors and identifies where **x** has the greatest effect on **y**.

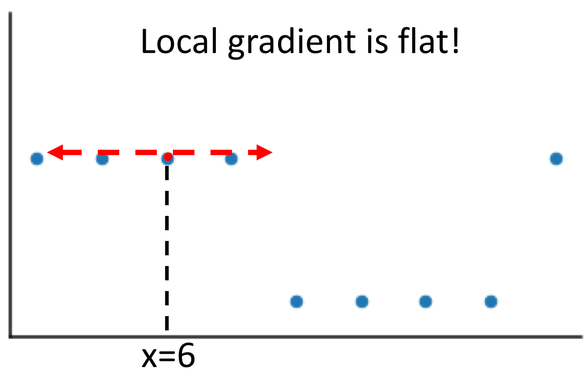


Gradient can be used to guide searches over large input spaces.

Gradient provides more precise and meaningful information about program behavior than taint based dataflow tracking.

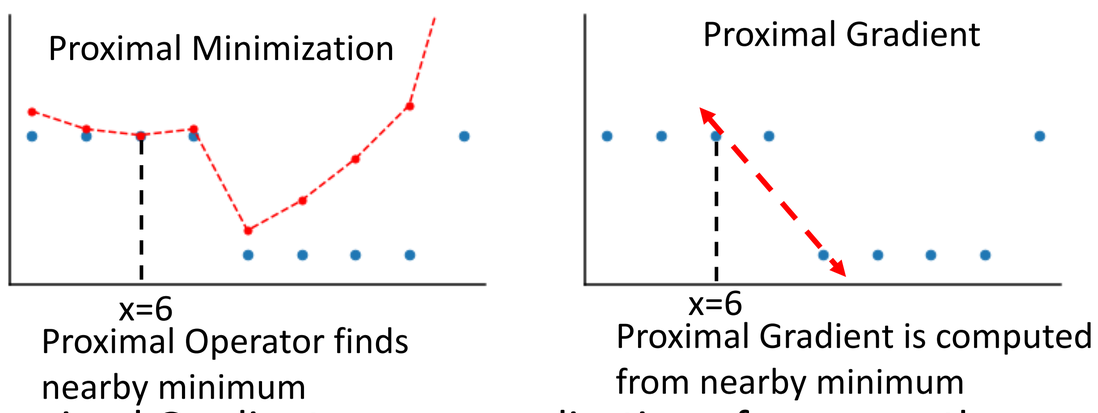
Problem: Programs are not Differentiable

```
int f(int x) {
    return x & 4;
}
```



Programs contain nonsmooth operations like Bitwise And, making it impossible to compute gradients normally.

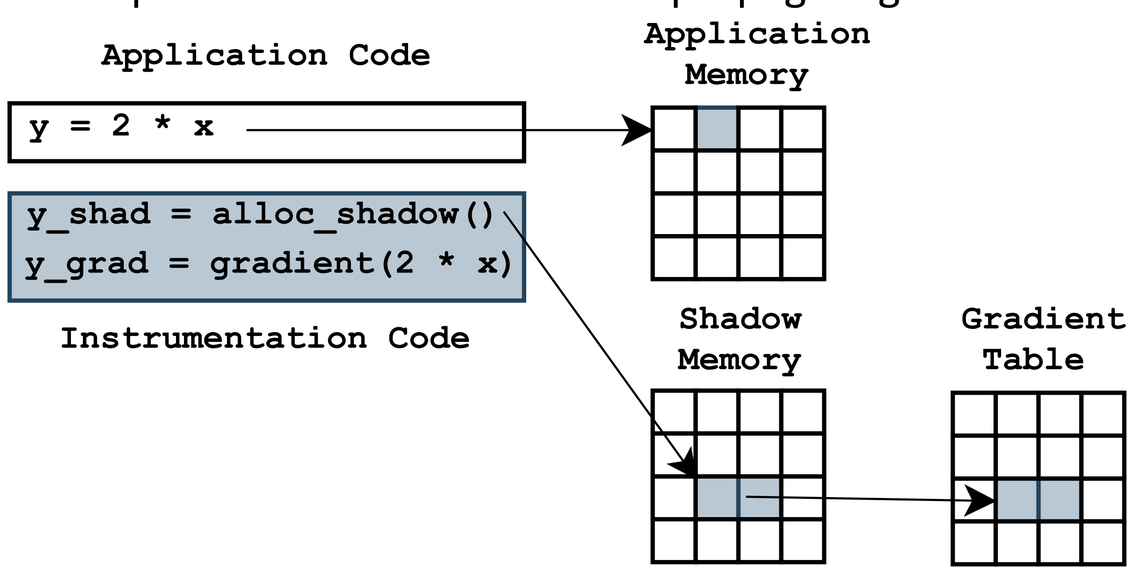
Solution: Nonsmooth Optimization



Proximal Gradients are an application of nonsmooth optimization methods that base the gradient on the local minimum of a function.[3]

Implementation:

- Implemented as LLVM Sanitizer, based on DataFlowSanitizer
- Each address has associated shadow memory with label
- Each operation is instrumented to propagate gradient

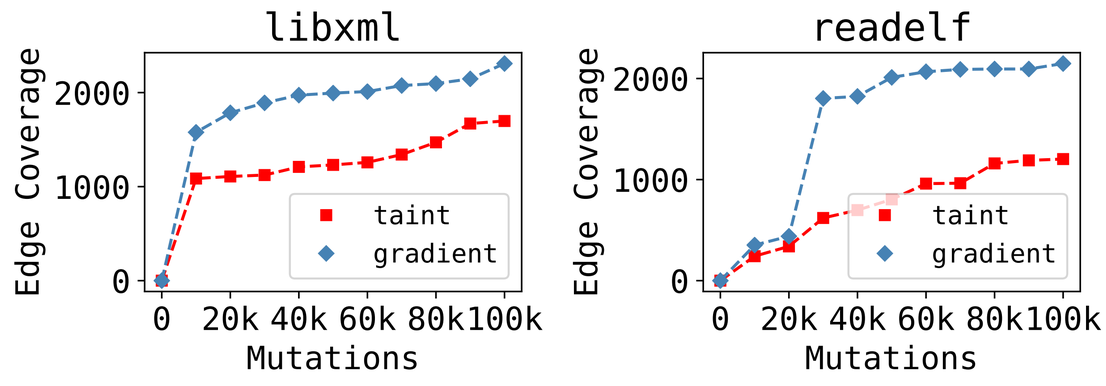


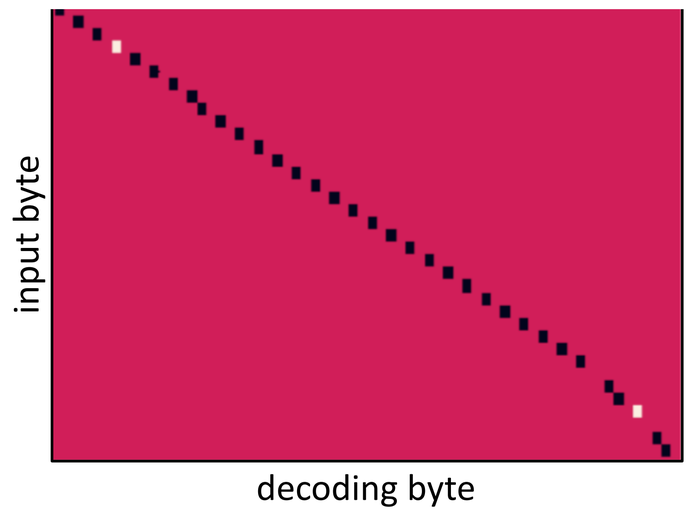
Evaluation:

- Achieved up to 39% better dataflow precision than state of the art method (Taint Analysis) on 7 programs.

	Taint			Gradient		
Program	Precision	Recall	F1	Precision	Recall	F1
minigzip	0.55	0.86	0.68	0.94	0.71	**0.81**
djpeg	0.63	0.73	0.68	0.96	0.61	**0.74**
mutool	1.0	0.01	**0.02**	1.0	0.01	**0.02**
xmllint	0.97	0.39	**0.56**	0.97	0.39	**0.56**
readelf	0.17	0.95	0.28	0.18	0.93	**0.30**
objdump	0.77	0.80	0.78	0.94	0.79	**0.85**
strip	0.60	0.83	0.70	0.88	0.79	**0.84**

- Overhead is comparable to Taint Analysis on average (<5%).
- Example application: Dataflow guided fuzzing.
  - Increase of 57% edge coverage on 2 programs





Gradient Analysis identifies which input bytes are used to look up each Huffman decoding index in gzip decompression.

References:

[1] LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton. "Deep learning." *nature* 521.7553 (2015): 436.
[2] Selvaraju, Ramprasaath R., et al. "Grad-cam: Visual explanations from deep networks via gradient-based localization." *Proceedings of the IEEE International Conference on Computer Vision*. 2017.
[3] Parikh, Neal, and Stephen Boyd. "Proximal algorithms." *Foundations and Trends® in Optimization* 1.3 (2014): 127-239.