Fine Grained Dataflow Tracking with Proximal Gradients

Gabriel Ryan†, Abhishek Shah†, Dongdong She†, Koustdubha Bhat‡, Suman Jana†

†Columbia University ‡Vrije Universiteit Amsterdam
Dataflow Analysis

Summarize dataflows for all possible inputs

```c
//input x

int x1 = 2 * x;
int x2 = x1 % 4;

return x2;
```

Dataflow Summary:

- even inputs $\rightarrow$ return 0
- odd inputs $\rightarrow$ return 2
Dynamic Taint Analysis (DTA)

Uses boolean **Taint Labels**

Applies **Propagation Rules** during Execution

Used in many security applications:

- Exploit Detection
- Greybox Fuzzing
- Malware Analysis
- Information Leak Identification
Dynamic Taint Analysis (DTA)

Uses boolean Taint Labels

Applies Propagation Rules during Execution

```c
//input x

int x1 = 2 * x;
int x2 = x1 % 4;

return x2;
```
Dynamic Taint Analysis (DTA)

Uses boolean Taint Labels

Applies Propagation Rules during Execution

```c
//input x

int x1 = 2 * x;
int x2 = x1 % 4;

return x2;
```
Dynamic Taint Analysis (DTA)

Uses boolean Taint Labels

Applies Propagation Rules during Execution

```c
//input x

int x1 = 2 * x;
int x2 = x1 % 4;

return x2;
```
Dynamic Taint Analysis (DTA)

Uses boolean **Taint Labels**

Applies **Propagation Rules** during Execution

```java
//input x
int x1 = 2 * x;
int x2 = x1 % 4;
return x2;
```

Dataflow Summary:
- `input x` can change `x2`
Dynamic Taint Analysis (DTA)

**Boolean** Taint labels *cannot represent* all possible behavior

Causes taint **propagation errors**
Dynamic Taint Analysis (DTA)

Boolean Taint labels cannot represent all possible behavior

Causes taint propagation errors

//input x, k

int x1 = x - k;
int x2 = x - x1;

return x2;
Dynamic Taint Analysis (DTA)

**Boolean** Taint labels cannot represent all possible behavior

Causes taint propagation errors

```c
//input x, k

int x1 = x - k;
int x2 = x - x1;  // x - x + k

return x2;
```
Dynamic Taint Analysis (DTA)

**Boolean** Taint labels **cannot represent** all possible behavior

Causes taint **propagation errors**

```c
//input x, k
int x1 = x - k;
int x2 = x - x1;
return x2;  ← taint error!
```
Dynamic Taint Analysis (DTA)

**Boolean** Taint labels cannot represent all possible behavior

Causes taint **propagation errors**

**Sound** Rules

→ **OverTaint** Errors, False Alerts

**Precise** Rules

→ **UnderTaint** Errors, Missed Violations

Slowinska, Asia, and Herbert Bos. "Pointer tainting still pointless: (but we all see the point of tainting)." ACM SIGOPS Operating Systems Review 44.3 (2010): 88-92.
Richer form of Dataflow?
Richer form of Dataflow?

**Gradient** is a popular **Dataflow** measure in machine learning:
Richer form of Dataflow?

**Gradient** is a popular **Dataflow** measure in machine learning:

\[
\frac{df(x)}{dx} = \frac{\text{change in output}}{\text{change in input}}
\]

\[
f(x) = x \times x
\]

\[
\frac{df(-1)}{dx} = -2
\]
Richer form of Dataflow?

**Gradient** used for similar applications to Dataflow:

- Guiding adversarial testing

Vulnerability search:

```java
int y = x1 + x2 % 2;

if (y > THRESHOLD) {
    // vulnerability
}
```
Gradient as Dataflow Measure

**Gradient** can be composed with **chain rule**

\[
\frac{df(g(x))}{dx} = \frac{df}{dg} \star \frac{dg}{dx}
\]
Gradient as Dataflow Measure

Gradient can be composed with chain rule

\[
\frac{df(g(x))}{dx} = \frac{df}{dg} \ast \frac{dg}{dx}
\]

Key Idea: Propagate Gradient directly over Program
Gradient as Dataflow Measure

**Neutaint:** Measures Gradient of Neural Network


**TaintInduce:** Learns taint rules from I/O samples

\[
\frac{df}{dg} \times \frac{dg}{dx}
\]
Gradient Reduces Errors

Propagates dataflow correctly

```c
//input x, k

int x1 = x - k;
int x2 = x - x1;  // x - x + k

return x2;
```
Gradient Reduces Errors

Propagates dataflow correctly

//input x, k
int x1 = x - k;  \frac{dx1}{dx} = 1
int x2 = x - x1;  \frac{dx2}{dx} = 1 - \frac{dx1}{dx} = 0

return x2;
Gradient Reduces Errors

Propagates dataflow correctly

//input x, k

```
int x1 = x - k;
int x2 = x - x1;
return x2;
```

\[
\frac{dx1}{dx} = 1
\]

\[
\frac{dx2}{dx} = 1 - \frac{dx1}{dx} = 0
\]

gradient to x is 0!
Problem: Programs are not differentiable!

\[ f(x) = x \times x \]

\[ \frac{df(-1)}{dx} = -2 \]
Problem: Programs are not differentiable!
Problem: Programs are not differentiable!
Proximal Gradient

\[ y = x \& 4 \]
Proximal Gradient

\[ y = x \& 4 \]
Proximal Gradient

$y = x \& 4$

grad?

largest change
Proximal Gradient

\[ y = x \& 4 \]
Proximal Gradient Analysis (PGA)

Implement as **LLVM Sanitizer Pass (grsan)**

Evaluate **Dataflow Accuracy**:

**Ground Truth Dataflow:**
For each input byte:
  - flip each bit
  - set to 0 and 255
  - record changed sink variables
Proximal Gradient Analysis (PGA)

Implement as **LLVM Sanitizer Pass** (**grsan**)

Evaluate **Dataflow Accuracy**:

**Ground Truth Dataflow:**
For each input byte:
- flip each bit
- set to 0 and 255
- record changed sink variables
## Proximal Gradient Analysis (PGA)

<table>
<thead>
<tr>
<th></th>
<th>libdft</th>
<th></th>
<th></th>
<th>dfsan</th>
<th></th>
<th></th>
<th>grsan (floats)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Prec.</td>
<td>Rec.</td>
<td>F1</td>
<td>Prec.</td>
<td>Rec.</td>
<td>F1</td>
<td>Prec.</td>
</tr>
<tr>
<td>minigzip</td>
<td>0.42</td>
<td>0.29</td>
<td>0.17</td>
<td>0.29</td>
<td>0.60</td>
<td>0.39</td>
<td>0.63</td>
</tr>
<tr>
<td>djpeg</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.22</td>
<td>1.00</td>
<td>0.37</td>
<td>0.60</td>
</tr>
<tr>
<td>mutool</td>
<td>0.70</td>
<td>0.32</td>
<td>0.22</td>
<td>0.63</td>
<td>0.61</td>
<td>0.62</td>
<td>0.86</td>
</tr>
<tr>
<td>xmllint</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.62</td>
<td>0.99</td>
<td>0.76</td>
<td>0.94</td>
</tr>
<tr>
<td>objdump</td>
<td>0.47</td>
<td>0.67</td>
<td>0.28</td>
<td>0.37</td>
<td>0.93</td>
<td>0.52</td>
<td>0.66</td>
</tr>
<tr>
<td>strip</td>
<td>0.26</td>
<td>0.59</td>
<td>0.18</td>
<td>0.20</td>
<td>0.96</td>
<td>0.33</td>
<td>0.50</td>
</tr>
<tr>
<td>size</td>
<td>0.20</td>
<td>0.59</td>
<td>0.30</td>
<td>0.37</td>
<td>0.95</td>
<td>0.53</td>
<td>0.62</td>
</tr>
</tbody>
</table>

**F1 accuracy** improvement: Up to 33% over SOTA (20% better on average)
Proximal Gradient Analysis

Introduce **proximal gradients (PGA)** as new formulation of dataflow problem

Show PGA **improves dataflow accuracy**, and gradients are useful in dataflow applications.

To learn more about PGA please see our paper:
