Breadcrumbs: Efficient Context Sensitivity for Dynamic Bug Detection Analyses

Bond, Baker and Guyer
Basic Contribution

• Decoding a PCC value
  – Human readable sequence of calls

• Evaluation
  – Dynamic Race Detector
  – Origin Tracking – Null Pointer Exception Diagnosis
Basic PCC

- PCC

\[ p' = f(p, c) = (3p + c) \mod 2^{32} \]

- \( p_0 = 0 \) [main]
- \( p_1 = f(p_0, c_0) \)
- \( p_2 = f(p_1, c_1) \)
- \( p_3 = f(p_2, c_2) \)
- \( p_i = f(p_{i-1}, c_{i-1}) \)
- \( p_n = f(p_{n-1}, c_{n-1}) \) [return main]
Decoding PCC...

- **Meaning**

\[
\begin{align*}
p_0 &= 0 \\
p_1 &= f(p_0, c_0) \\
p_2 &= f(p_1, c_1) \\
p_3 &= f(p_2, c_2) \\
&\quad \vdots \\
p_i &= f(p_{i-1}, c_{i-1}) \\
&\quad \vdots \\
p_n &= f(p_{n-1}, c_{n-1})
\end{align*}
\]
Inverse, $f^{-1}()$

- Given $p'$ in
  
  \[ p' = f(p, c) = (3p + c) \mod 2^{32} \]

  - Find $p$ and $c$
  
  - for a given $c$ and $p'$ ... $p$ is unique.

  - err... we want to find $c$

  - but, in order to **track back**, $p$ is required
Inverse, $f^{-1}()$

- Given $p'$ in $p' = f(p, c) = (3p + c) \mod 2^{32}$
  - Find $p$ and $c$
  - Choose a $c$, then $p = f^{-1}(p', c)$

  » $p_n$
  » $p_{n-1} = f^{-1}(p_n, c_{n-1})$
  » $p_{n-2} = f^{-1}(p_{n-1}, c_{n-2})$
  » $p_0 = f^{-1}(p_1, c_0)$
Challenges?

• Difficult search problem
  – Many Call sites $c$ to choose from (1000s)
    • $p_{n-1} = f^{-1}(p_n, c_{n-1})$
      – Accurately choosing the right $c_{n-1}$ will be difficult.
      – Compounds the problem of deriving the right sequence.

$1000^{10} = 10^{30}$ possible calling sequences. (this is minimum)
Reducing the Search Space

• == reducing the probable Call sites
• Static
• Dynamic
Static

M1()
M2()
M3()
M4()
M5()
M6()
M7()
M8()
Issues with Static

class A {
    static { methodA(); }
}

public methodA() {
    System.out.println("helloworld");
}

public static void main(String[] args) {
    A objecta = new A();
}

JVM

registerKeyPressEvent(e);

void HandleKeyPressEvent(e, arg) {
    Display("hello!");
}

void Display() {
}

JAVA/SWING

The possible call sites are incomplete.
Dynamic Analysis is then used to find the missing links.
Dynamic

- Calculate **and store** all PCC values at specific call sites.
- \( p = f^{-1}(p', c) \)

a. 3 out of 1000 call sites (static)
b. Find all Per call site PCC values. See where \( p' \) is. (dynamic)
Issues with Dynamic

- As always, too expensive.
- Solution -
  - `hotThreshold`
    - Stop recording the PCC values after the threshold.
- Issues with Solution
  - You can’t guess accurately anymore.
- As always, the Accuracy – Performance Tradeoff

```
M1()

M2(){
  }

M3(){
  }

M4(){
  }
```

\[ p', p^{*}, p'' \]

\[ p'^*, p' \]
PCC Values are Client sites - Extensibility

• The PCC values are generally calculated at callsites.

• Thus, you can’t look at the program flow at all points.

• So, you start storing the information at the client sites (sites which are of interest to the client, like suspicious bug locations, or memory operations).
Evaluations

• No client
  – PCC only
  – $T= 100; 1,000; 10,000; 100,000; \text{inf.}$

• Origin Tracking
  – OT only
  – $T= 100; 1,000; 10,000; 100,000; \text{inf.}$

• Race Detection
  – RD only
  – $T= 100; 1,000; 10,000; 100,000; \text{inf.}$
contd.

• PCC only – No Client
  – “No threshold” adds as high as 90% overhead.
  – T = 100 to 1000, adds about 10 to 20%. Still too high for production.

• Origin Tracking
  – Direct application of PCC.
  – Propagation of null values.
  – The overheads are very similar to PCC only.
contd.

• Race Detector - Pacer
  – FastTrack Algorithm
    • Significant Runtime and Space
  – Calling contexts of all memory operations
  – Overhead of PCC Decoding is very small compared to the overhead of Pacer.
Take Away

• Add-on to the original PCC work
• Significant runtime overhead
  – 10 to 20 % at the minimum. (if you want accurate reconstruction of graphs.)
• Reconstruction not easy even at t = 10,000 sometimes.
Observations

• **Space overhead was not talked about.**

• Did not specify what call-depth is practically useful
  – Do you need 10+ levels of depth to debug?
  – Would give a more practical picture.

• Can we use an arithmetic encoding function instead, like in compression techniques?