Metrics-based Evaluation of Slicing Obfuscations

Anirban Majumdar, Stephen Drape and Clark Thomborson

The University of Auckland
Obfuscation

- An obfuscation is a functionality-preserving transformation.
- The goal of obfuscation is to keep something secret within a program.
- Barak et al proved that perfect obfuscation is impossible for their model.
- We aim for transformations that are difficult but not impossible to reverse engineer.
Program Slicing

- Slicing is a reverse engineering technique often used to aid program comprehension.
- A slice consists of the program parts that potentially affect the values computed at a particular point.
- We will restrict ourselves just to backwards slices and output statements.
Experimental Design

- We would like to restrict the usefulness of slicing for program comprehension.
- Use CodeSurfer to slice our programs.
- CodeSurfer uses system dependence graphs (SDGs) to compute slices.
- We slice our unobfuscated program and use this information to create obfuscations that are targeted to restrict the effectiveness of slicing.
Adding dependencies

- Consider the nodes from the SDG that are left behind after slicing – called *orphans*.
- Add in obfuscations that create dependencies between the slicing variable and the variables contained within the orphans.
- Forthcoming PhD will explore dependencies in more detail.
Slicing Metrics

**Tightness** measures the number of statements common to every slice: \( T(M) = \frac{|SL_{int}|}{|M|} \)

**Minimum Coverage** is the ratio of the smallest slice in a method to its length: \( Min(M) = \frac{1}{|M|} \min_i |SL_i| \)

**Coverage** compares the length of slices to the length of the entire method: \( C(M) = \frac{1}{|V_O|} \sum_{i=1}^{|V_O|} \frac{|SL_i|}{|M|} \)

**Maximum Coverage** is the ratio of the largest slice in a method to its length: \( Max(M) = \frac{1}{|M|} \max_i |SL_i| \)

**Overlap** is a measure of how many statements in a slice are found in all the other slices: \( O(M) = \frac{1}{|V_O|} \sum_{i=1}^{|V_O|} \frac{|SL_{int}|}{|SL_i|} \)
Aims

We are going to add simple obfuscations to our programs which:
- add dependencies between variables
- increase the size of the slices
- increase the metric values
- do not increase the code size significantly
Suitable Obfuscations

We will restrict ourselves to only using the following obfuscations:

- Adding a bogus predicate
- Variable encoding
- Adding to the guard of a while loop

These transformations have been chosen because they enable us to add dependencies between variables.
As an example, consider the program \texttt{wc} which counts the number of lines (\textit{nl}), words (\textit{nw}) and characters (\textit{nc}) in a file.

\begin{verbatim}
wc() {
    int c, nl = 0, nw = 0, nc = 0, in;
    in = F;
    while ((c = getchar())! = EOF) {
        nc ++;
        if (c == ' ' || c == '
' || c == '	')
            in = F;
        else if (in == F)
            {in = T; nw ++;}
        if (c == '
') nl ++;
        out(nl, nw, nc);
    }
}
\end{verbatim}
A Particular Example

As an example, consider the program \texttt{wc} which counts the number of lines \((nl)\), words \((nw)\) and characters \((nc)\) in a file.

The backwards slice from \(nl\).

\begin{verbatim}
wc() {
    int c, nl = 0, nw = 0, nc = 0, in;
    in = F;
    while ((c = getchar())! = EOF) {
        nc ++;
        if (c == ' ' || c == 'n' || c == 't')
            in = F;
        else if (in == F)
            {in = T; nw ++;}
        if (c == 'n') nl ++;
    }
    out(nl, nw, nc);
}
\end{verbatim}
A Particular Example

As an example, consider the program $wc$ which counts the number of lines ($nl$), words ($nw$) and characters ($nc$) in a file.

The backwards slice from $nl$ ...

Our goal is to include these orphans in the slice.

```c
wc() {
    int c, nl = 0, nw = 0, nc = 0, in;
    in = F;
    while (((c = getchar())! = EOF)) {
        nc ++;
        if (c == ' ' || c == '
' || c == ' \
')
            in = F;
        else if (in == F)
            { in = T; nw ++; }
        if (c == '
') nl ++;
        out(nl, nw, nc);
    }
}
```
An Example Obfuscation

As an obfuscation, we add bogus predicates to create dependencies.

```c
wc-obf1() {
    int c, nl = 0, nw = 0, nc = 0, in;
in = F;
while ((c = getchar())! = EOF) {
    nc ++;
    if (c == ' ' || c == '
' || c == ' \/t')
in = F;
else if (in == F)
    {in = T; nw ++;}
if (c == ' \
') nl ++;
out(nl, nw, nc); }
}```
An Example Obfuscation

As an obfuscation, we add bogus predicates to create dependencies.

These predicates use the invariant:

\[ nc \geq nw \land nc \geq nl \]

```c
wc-obf1() {
    int c, nl = 0, nw = 0, nc = 0, in;
    in = F;
    while ((c = getchar()) != EOF) {
        nc ++;
        if (c == ' ' || c == '
' || c == '	')
            in = F;
        else if (in == F)
            { in = T; nw ++; }
        if (c == '
')
            if (nw <= nc) nl ++;
        if (nl > nc) nw = nc + nl;
        else
            { if (nw > nc) nc = nw - nl; }
    out(nl, nw, nc); }
}
An Example Obfuscation

As an obfuscation, we add bogus predicates to create dependencies.

The backwards slice from \textit{nl}.

Now we’ve included all of the orphans in the slice for \textit{nl}.

\begin{verbatim}
wc-obj1() {
  int c, nl = 0, nw = 0, nc = 0, in;
  in = F;
  while \((c = \text{getchar}())! = \text{EOF}\) {
    nc ++;
    \textbf{if} \((c == ' ' \mid c == '\n' \mid c == 't') \)
    \quad \text{in} = F;
    \textbf{else if} \((\text{in} == F)\)
    \quad \{ in = T; nw ++; \}
    \textbf{if} \((c == '\n') \)
    \quad \{ \textbf{if} \((\text{nw} <= \text{nc}) \) \text{nl} ++; \}
    \textbf{if} \((\text{nl} > \text{nc}) \)
    \quad \text{nw} = \text{nc} + \text{nl};
    \textbf{else} \{ \textbf{if} \((\text{nw} > \text{nc}) \) \text{nc} = \text{nw} - \text{nl}; \}
  } \}
  \text{out}(nl, nw, nc);
}\end{verbatim}
An Example Obfuscation

As an obfuscation, we add bogus predicates to create dependencies.

We have also included the orphans of the slices for the other two output variables.

```c
wc-obj1() {
    int c, nl = 0, nw = 0, nc = 0, in;
    in = F;
    while ((c = getchar()) != EOF) {
        nc ++;
        if (c == ' ' || c == '\n' || c == '\t')
            in = F;
        else if (in == F)
            { in = T; nw ++; }
        if (c == '\n')
            { if (nw <= nc) nl ++; }
        if (nl > nc) nw = nc + nl;
        else { if (nw > nc) nc = nw - nl; }
        out(nl, nw, nc); }
```
## Results for wordcount

<table>
<thead>
<tr>
<th>Method M</th>
<th></th>
<th></th>
<th>Size of nl</th>
<th>Size of nw</th>
<th>Size of nc</th>
<th>SLint</th>
</tr>
</thead>
<tbody>
<tr>
<td>wc</td>
<td>36</td>
<td>3</td>
<td>15</td>
<td>20</td>
<td>10</td>
<td>7</td>
</tr>
<tr>
<td>wc-obf1</td>
<td>42</td>
<td>3</td>
<td>30</td>
<td>30</td>
<td>30</td>
<td>28</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>T(M)</th>
<th>Min(M)</th>
<th>C(M)</th>
<th>Max(M)</th>
<th>O(M)</th>
</tr>
</thead>
<tbody>
<tr>
<td>wc</td>
<td>19.4%</td>
<td>27.8%</td>
<td>41.7%</td>
<td>55.6%</td>
<td>50.6%</td>
</tr>
<tr>
<td>wc-obf1</td>
<td>66.7%</td>
<td>71.4%</td>
<td>71.4%</td>
<td>71.4%</td>
<td>93.3%</td>
</tr>
</tbody>
</table>

Measurements obtained from CodeSurfer
## Table of Results

| Method \( M \) | \(|M|\) | \(|V_O|\) | For each \( v_i \) the slice size | \( |SL_i|\) | \( |SL_{int}|\) | \( T(M) \) | \( Min(M) \) | \( C(M) \) | \( Max(M) \) | \( O(M) \) |
|----------------|------|------|-------------------------------|------|------|--------|--------|--------|--------|--------|--------|
| \( wc \)       | 36   | 3    | \( nl \) 15 \( nw \) 20 \( nc \) 10 | 7    | 19.4%| 27.8%  | 41.7%  | 55.6%  | 50.6%  |
| \( wc-obf1 \)  | 42   | 3    | \( nl \) 30 \( nw \) 30 \( nc \) 30 | 28   | 66.7%| 71.4%  | 71.4%  | 71.4%  | 93.3%  |
| \( ps \)       | 21   | 2    | \( prod \) 12 \( sum \) 12 | 7    | 33.3%| 57.1%  | 57.1%  | 57.1%  | 58.3%  |
| \( ps-obf1 \)  | 22   | 2    | \( prod \) 16 \( sum \) 13 | 11   | 50.0%| 59.1%  | 65.9%  | 72.7%  | 76.7%  |
| \( ps-obf2 \)  | 26   | 2    | \( prod \) 19 \( sum \) 19 | 17   | 65.4%| 73.1%  | 73.1%  | 73.1%  | 89.5%  |
| \( search \)   | 107  | 2    | \( n \) 9 \( secs \) 11 | 2    | 1.9% | 8.4%   | 9.3%   | 10.3%  | 20.2%  |
| \( search-obf1 \) | 120 | 2    | \( n \) 45 \( secs \) 11 | 10   | 8.3% | 9.2%   | 23.3%  | 37.5%  | 56.6%  |
| \( search-obf2 \) | 127 | 2    | \( n \) 49 \( secs \) 48 | 46   | 36.2%| 37.8%  | 38.2%  | 38.6%  | 94.9%  |
| \( r0v \)      | 124  | 2    | \( fuel \) 23 \( dist \) 46 | 19   | 15.3%| 18.5%  | 27.8%  | 37.1%  | 62.0%  |
| \( r0v-obf1 \) | 129  | 2    | \( fuel \) 60 \( dist \) 46 | 45   | 34.9%| 35.7%  | 41.1%  | 46.5%  | 86.4%  |
| \( r0v-obf2 \) | 132  | 2    | \( fuel \) 62 \( dist \) 60 | 59   | 44.7%| 45.5%  | 46.2%  | 47.0%  | 96.7%  |
| \( scatter \)  | 143  | 3    | \( si \) 116 \( ru \) 111 \( i \) 9 | 8    | 5.6% | 6.3%   | 55.0%  | 81.1%  | 34.3%  |
| \( scatter-obf1 \) | 148 | 3    | \( si \) 132 \( ru \) 132 \( i \) 132 | 131  | 88.5%| 89.2%  | 89.2%  | 89.2%  | 99.2%  |
| \( scatter-obf2 \) | 150 | 3    | \( si \) 139 \( ru \) 139 \( i \) 139 | 138  | 92.0%| 92.7%  | 92.7%  | 92.7%  | 99.3%  |
Graph of Results
Discussion

- Managed to increase the slice sizes for all our obfuscations and so we increased the metric values.
- Slightly increase the code size.
- Important to increase the slice intersection size.
- Should aim to add obfuscations for all of the variables.
Future Work

- Automation – need to consider:
  - where to place the obfuscations
  - what order to apply the obfuscations
  - composition of obfuscations

- Additional Concerns:
  - methods
  - pointers
  - arrays
Conclusion

- Proposed a new approach to designing obfuscations by attacking a program first then defending against further attacks.
- Create obfuscations that have false data dependencies.
- Aim is to include statements that are orphaned.