Interprocedural and Intraprocedural Alias Analysis Algorithms

Shaotao Li\textsuperscript{1,a}, Yong Cai\textsuperscript{2,b}

\textsuperscript{1}College of Computer Science and Electronic Engineering, Hunan University, Changsha 410082, China;
\textsuperscript{2}Key Laboratory for Micro-nano Optoelectronic Devices of Ministry of Education, Changsha 410082, China.

\textsuperscript{a}hnulst@foxmail.com, \textsuperscript{b}cy218@foxmail.com

\textbf{Keywords:} Vulnerabilities Detection, Static Analysis, Parse Tree, Control Flow Graph, Alias Analysis.

\textbf{Abstract.} The quantity and significance of web application increases quickly. Meanwhile, the influence of vulnerabilities in web application grows as well. Automated tools are urgently needed because manual code reviews are inefficient and fallible. However, previous static code detection tools lack of alias analysis between variables in codes, leading to possible false positives or false negatives. To solve this problem, we propose a set of sound and precise alias analysis algorithms which can conduct intraprocedural and interprocedural alias analysis. Then we apply them to a previous static detection system. Experiments on practical open source web applications and manually written test cases show that system with alias analysis can handle complex alias relationship accurately and detect vulnerabilities related to alias with greater precision. Moreover, alias analysis's impact on scanning speed of the system is negligible.

\textbf{Introduction}

This paper focus on sound and precise alias analysis algorithm in static detection tool of web application vulnerabilities. Detection of web application vulnerabilities can be divided into two categories depending on whether to run application or not. Dynamic detection, for example [1,2], is to detection attack when the application is running, but static detection[3,4,5] is trying to find vulnerabilities in code of web application before application is deployed to server. Predecessors have done abundant research on the technology of static detection. But they ignored thorough study of alias relationship between variables in code of web applications. For instance, paper [6] proposed a three-tiered static detection which performs well but lacks of analysis of alias relationship. Moreover, paper [7,8] etc. focus on analysis of pointers in C which differs from alias in the most popular language PHP used in web applications. The lacking of alias analysis may lead to false positives or false negatives in static detection. Our system is based on control ow graph [9] of web applications' source codes. First of all, system converts PHP source codes of web application into parse trees [10] using Java lexical analyzer generator JFlex. Then, it constructs a divided control ow graph for each function. At last, system conducts data ow analysis on the control flow graphs.

\textbf{Alias Analysis}

PHP is a scripting language which supports process-oriented programming style. According to the difference of objects being analyzed, we can divide alias analysis into two types. Intraprocedural analysis represents analysis of alias in a piece of code which doesn't contain function calls. Interprocedural analysis represents analysis of alias in a piece of code which encounters function calls. More precisely, interprocedural analysis is responsible for adjusting analysis information to be propagated into the callee when calling a function and responsible for adjusting analysis information returned to the caller when the callee is over. Obviously, the former is the foundation of the latter.

\textbf{Intraprocedural Alias Analysis.} The key point of intraprocedural alias analysis is to combine alias information from different execution paths which intersects. Fig. 1 is an example of how to combine alias information. We use “u” to represent must alias and “a” to represent may alias in code.
comments. It is worthy of mentioning that we use group to represent must alias information but pair to represent may alias for the convenience of dealing with may alias information which is more complicated. On line 3, $a$ is aliased to $b$ using reference assignment. Because of the if statement on line 4, the control flow graph forks, the alias relationship between $c$, $d$ and $e$ is valid only on one of the execution path, so when control flow graph intersects on line 8, the alias relationship of $c$, $d$ and $e$ belongs to may alias information. Fig. 2 is the pseudo code of combination operation.

![Fig. 1 Example of intraprocedural analysis](image1)

```
< ?
3 $a &=& $b; // u{(a,b)} a{}
4 if (...) {
  5 $c &=& $d; // u{(a,b) (c,d)} a{}
  6 $e &=& $d; // u{(a,b) (c,d,e)} a{}
  7 }
8 // u{(a,b)} a{(c,d) (c,e) (d,e)}
9 >
```

Fig. 1 Example of intraprocedural analysis

**Interprocedural Alias Analysis.** Because we should take the scope of variables into account, interprocedural alias analysis is far more complicated.

(1) Which alias information should be passed to the callee?
(2) Which alias information is available and should be returned to the caller when the callee is over?

The answers are listed below:

(1) From the perspective of the callee, it needs the information listed below:
   a. Alias relationship information of global variables.
   b. Alias relationship information of the callee's formal parameters.
   c. Alias relationship information of global variables and the formal parameters of callee.

(2) From the perspective of the caller, information below should be returned:
   a. Alias relationship information of global variables.
   b. Alias relationship information of local variables of the caller and global variables.

**Analysis between global variables.** Alias relationship information of global variables is significant. Firstly, the callee should be aware of the alias relationship information about global variables when called, secondly, the caller should know how the alias relationship information of global variables is changed in the callee. This situation can be handled in a relatively direct way similar to the classic analysis way [11].
As Fig. 3 shows, we use the function name which the variable belongs to as prefix in alias information. Specially, we consider global variables belong to “main” function which is abbreviated as “m”. Alias relationship information of local variables belongs function a is deleted before passed to function b because each function only tracks alias information of its local variables and global variables. Inside function b, the alias relationship between global variables is modified on line 13. Then, the information which is changed is returned to function a and alias relationship information of local variables belongs to function a is regained, too. May alias analysis is not presented in this example but it is handled in a similar way in fact.

**Analysis between formal parameters of the callee**. If there is alias relationship between actual parameters of a function, the callee's formal parameters are aliases. As is shown in Fig. 4, the function b has two formal parameters, $bp1$ and $bp2$. When function b is called, its corresponding actual parameters $a1$ and $a2$ is must alias, so $bp1$ and $bp2$ is must alias at the beginning of function b.
Analysis between formal parameters of the callee and global variables. We have to take the following cases into account to analyze alias relationship information between formal parameters of the callee and global variables:

1. The corresponding actual parameter is a global variable's must alias.
2. The corresponding actual parameter is a global variable's may alias.
3. The corresponding actual parameter is actually a global variable.

Fig. 4 is an example of the first case. When function b is called on line 5, the variable $a$ and the global variable $X1$ belong to one must alias group. Because variable $a$ is function b's actual parameter, formal parameter $bp1$ of function b is a must alias of global variable $X1$.

Fig. 6 Analysis between global variables

Analysis between local variables of the caller and global variables. Alias relationship of local variables belong to the caller and global variables can be changed by means listed below:

1. If the local variable is an alias of one global variable:
   a. Other local variable or global variables can become alias of the global variable, so the local variable can become other variables' alias.
   b. The global can be changed to an alias of other variables, so it is no longer an alias of the local variable.

2. If the local variable becomes an alias of a formal parameter belongs to the called function:
   a. Another global variable can be changed to an alias of the formal parameter, so it becomes an alias of the local variable, too.

Fig. 7 Analysis between global variables and local variables
Fig. 8 Algorithm for computing alias information after a function call about global shadow variables

Fig. 7 is an example of case 1b. Global variable $X_1$ and local variable $a_1$ belong to one must alias group on line 6. But the global variable $X_1$ is aliased to another global variable $X_2$. We cannot find that $a$ is no longer an alias of $X_1$ when control flow graph returns without using global shadow variables, but we can find this by using global shadow variables. In the alias information passed from function b, the global shadow variable $X_1_{gs}$ is no longer an alias of the global variable $X_1$, so we can deduce that $a_1$ is not an alias of the global variable $X_1$ any more. Fig. 8 shows the algorithm applied here.

**Experiment**

Table 2 is the result of scanning practical applications. It shows that a home computer is enough to run the system fast and far more efficient than manually code review. It is worth mentioning that most false positives are caused by impossible execution path. This is determined by the nature that static analysis doesn't run code. Result shows that the system includes alias analysis performs a little bit slower in scanning speed, but better at false positives due to alias relationship is analyzed.

**Table 1 Result of scanning practical Web applications**

<table>
<thead>
<tr>
<th>Application</th>
<th>Line of Code</th>
<th>Speed(s/file)</th>
<th>Vulnerabilities</th>
<th>False Positives (original system / our system)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MyBloggie 2.1.4</td>
<td>20326</td>
<td>58.3</td>
<td>14</td>
<td>5 / 3</td>
</tr>
<tr>
<td>TxtForum1.0.4v</td>
<td>4398</td>
<td>1.3</td>
<td>31</td>
<td>17 / 13</td>
</tr>
</tbody>
</table>

**Table 2 Result of scanning test cases**

<table>
<thead>
<tr>
<th>System</th>
<th>True Positives</th>
<th>False Positives</th>
<th>False Negatives</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original System</td>
<td>15</td>
<td>10</td>
<td>25</td>
</tr>
<tr>
<td>with alias analysis</td>
<td>40</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 3 is the result of scanning 50 manually written PHP files which include all cases discussed in this paper. 15 of them don't contain alias, others contains complicated alias relationships.

The result in table 3 is distinguishable. The original system without alias analysis can only detected vulnerabilities in files which do not contain alias relationships and didn't find out vulnerabilities introduced by alias, what more, it also reported 10 false positives which is sanitized by alias in fact. The system with alias analysis, by contrast, found out all vulnerabilities perfectly.
Summary

This paper proposes a sound and precise alias analysis algorithm. Experiment results show that it enables static analysis to handle sophisticated alias relationships accurately, so it reduces false positive rate caused by lack of alias analysis and improves the accuracy of the vulnerabilities detection greatly. The alias analysis algorithm cannot handle alias relationship in arrays and it doesn't support object oriented programming style, it’s the future direction of our work.

References


