Interprocedural and Intraprocedural Alias Analysis Algorithms

Shaotao Li[1, a], Yong Cai[2, b]

[1]College of Computer Science and Electronic Engineering, Hunan University, Changsha 410082, China;

[2]Key Laboratory for Micro-nano Optoeletronic Devices of Ministry of Education, Changsha 410082, China.

[a]hnulst@foxmail.com, [b]cy218@foxmail.com

Keywords: Vulnerabilities Detection,Static Analysis,Parse Tree,Control Flow Graph,Alias Analysis.

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.

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.

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. Inter procedural 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.

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.

```
1  <?
2          // u{} a{}
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

```
procedure COMBINE(input-1,input-2)
  output.MayAliases ← input-1.may-alias ∪ input-2.may-alias
  for each MustAliasesGroup ∈ input-1.MustAliases:
    create a complete component consisting
    ∀ members ∈ MustAliasesGroup in a graph
  end for
  for each MustAliasesGroup ∈ input-2.MustAliases:
    add ∀ members ∈ MustAliasesGroup to graph
    if ∃ edge between v1 and v2 in the graph then
      graph.singleEdges.remove(v1,v2)
      graph.doubleEdges.add(v1,v2)
    end if
  end for
  for each single edge ∈ graph between v1 and v2:
    output.MayAliases.add(v1,v2)
  end for
  for each complete component ∄ single edge:
    output.MustAliases.add(v1,v2,⋯ ,vn)
  end for
  return output
end procedure
```

Fig. 2 Pseudo code of combination

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.

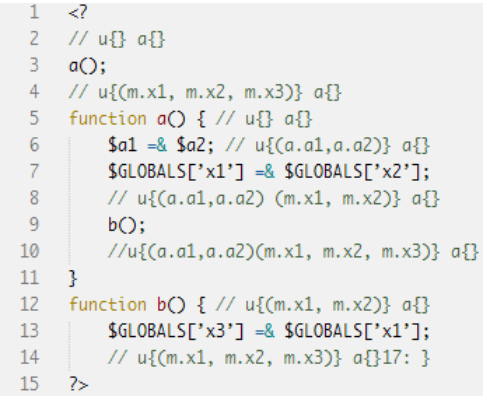
c. Alias relationship information of global variables and the formal parameters of callee.

(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].



```
1  <?
2  // u{} a{}
3  a();
4  // u{(m.x1, m.x2, m.x3)} a{}
5  function a() { // u{} a{}
6      $a1 =& $a2; // u{(a.a1,a.a2)} a{}
7      $GLOBALS['x1'] =& $GLOBALS['x2'];
8      // u{(a.a1,a.a2) (m.x1, m.x2)} a{}
9      b();
10     //u{(a.a1,a.a2)(m.x1, m.x2, m.x3)} a{}
11 }
12 function b() { // u{(m.x1, m.x2)} a{}
13     $GLOBALS['x3'] =& $GLOBALS['x1'];
14     // u{(m.x1, m.x2, m.x3)} a{}17: }
15 ?>
```

Fig. 3 Example of analysis between global variables

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 it's 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 $bp1and $bp2 is must alias at the beginning of function b.

```
1  <?
2  a();
3  function a() {  // u{} a{}
4      $a1 =& $a2; // u{(a.a1, a.a2)} a{}
5      b(&$a1, &$a2);
6  }
7  function b(&$bp1, &$bp2) {
8      // u{(b.bp1, b.bp2)} a{}
9  }
10 ?>
```

Fig. 4 Example of analysis between formal parameters

As to may alias, it is relatively sophisticated to handle may alias information. As to may alias information, firstly, we copy the may alias pairs which contains corresponding actual parameters, then, replace the actual parameter with the formal parameter, last, remove local variables of the caller and alias information contains only one parameter. Fig. 5 shows the pseudo code of the algorithm.

```
for each call-by-reference pair:
  fp ← the formal parameter ∈ the pair
  add fp to the actual parameter's MustAliasGroup
  for each MayAliasPair ⊇ the actual parameter ∈ the pair:
    copy MayAliasPair, replace the actual parameter with fp,
    add the copy to the may alias set
  end for
  remove all local variables ∈ caller
  remove all aliases contains one element
  replace fp by the formal parameter ∈ the pair
end for
```

Fig. 5 Pseudo code for adjusting alias information that is propagated into a callee

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 belongs 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.

```
1  <?
2  a();
3  function a() {  // u{} a{}
4      $a1 =& $GLOBALS['x1']; // u{(a.a1,m.x1)} a{}
5      b(&$a1);
6  }
7  function b(&$bp1) {
8      // u{(m.x1,b.bp1)} a{}
9  }
10 ?>
```

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.

```
1  <?
2  a();
3  // u{(m.x1, m.x2)} a{}
4  function a() {
5      // u{(m.x1, a.x1_gs)(m.x2, a.x2_gs)} a{}
6      $a1 =& $GLOBALS['x1'];
7      // u{(m.x1, a.x1_gs, a.a1) (m.x2, a.x2_gs)} a{}
8      b();
9      // u{(m.x1, m.x2, a.x2_gs) (a.a1, a.x1_gs)} a{}
10 }
11 function b() {
12     //u{(m.x1, b.x1_gs) (m.x2, b.x2_gs)} a{}
13     $GLOBALS['x1'] =& $GLOBALS['x2'];
14     // u{(m.x2, b.x2_gs, m.x1)} a{}
15 }
16 ?>
```

Fig. 7 Analysis between global variables and local variables

```
origInfo ← input information
localInfo ← alais information of local variables
interInfo ← alais information of global variables
outputInfo ← output information
//Must-Aliases
for each MustAliasGroup ∈ origInfo:
    if ∃ global variable g and local variable v ∈ MustAliasGroup then
    set it to visited
    if ∃ global must-alias g_u of global shadow of g at last then
      merge MustAliasGroup ⊇ v with MustAliasGroup ⊇ g_u
    end if
    for each global may alias g_a of global shadow at last:
      add MayAliasPair(v,g_a) and ∀ MayAliasPair(v_u,g_a) to outputInfo
      (v_u represents all local may alias of vairable v)
    end for
  end if
end for
//May-Aliases
for each MayAliasPair ⊇ a global variable g and a local variable l∈origInfo:
 for each global variable aliased to the global shadow at last:
   add MayAliasPair (l, g) to outputInfo
  end for
end for
```

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 $X1 and local variable $a1 belong to one must alias group on line 6. But the global variable $X1 is aliased to another global variable $X2.We cannot find that $a is no longer an alias of $X1 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 $X1_gs is no longer an alias of the global variable $X1, so we can deduce that $a1 is not an alias of the global variable $X1 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 performed 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

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

Table 2 Result of scanning test cases

System	True Positives	False Positives	False Negatives
Original System	15	10	25
with alias analysis	40	0	0

Table 3 is the result of scanning 50 manually written PHP files which include all cases discussed in this paper. 15 of them doesn'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

[1] Nguyen-Tuong A, Guarnieri S, Greene D, et al. Automatically hardening web applications using precise tainting[M]. Springer US, 2005.

[2] Pietraszek T, Berghe C V. Defending against injectio-n attacks through context-sensitive string valuation[C].Recent Advances in Intrusion Detection. Springer Berlin Heidelberg, 2006: 124-145.

[3] Livshits V B, Lam M S. Finding security errors in Java program with static analysis[C].Proc. 14th Usenix Security Symposium, Baltimore, MD, USA. 2005.

[4] Minamide Y. Static approximation of dynamically enerated web pages[C].Proceedings of the 14th international conference on World Wide Web. ACM, 2005: 432-441.

[5] Huang Y W, Yu F, Hang C, et al. Securing web application code by static analysis and runtime protection[C].Proceedings of the 13th international conference on World Wide Web. ACM, 2004: 40-52.

[6] Xie Y, Aiken A. Static Detection of Security Vulnerabilities in Scripting Languages[C].USENIX Security. 2006, 6: 179-192.

[7] Chase D,Wegman M, Zadeck F K. Analysis of pointers and structures [J]. ACM SIGPLAN Notices,2004, 39(4): 343-359.

[8] Wang Tiantian, Su Xiaohong , Ma Peijun . Research on pointer analysis algorithm for program standardizationJ .Acta Electronica Sinica,2009,37(5):1104-1108

[9] Arlt S, Rmmer P, Schf M. A theory for control-ow graph exploration [M].Automated Technology for Verication and Analysis. Springer International Publishing, 2013: 506-515.

[10] Parse treeEB/OL.https://en.wikipedia.org/wiki/Parse tree.

[11] Sharir M, Pnueli A. Two approaches to interprocedural data ow analysis [M]. New York University. Courant Institute of Mathematical Sciences. ComputerScience Department, 1978.