北京邮电大学

网络空间安全学院



**软件详细设计报告**

**项目：基于源代码检测和动态执行的二阶SQL注入漏洞检测**

**组员： 王硕、彭致远、李懿飞、王晨旭**

**2020年11月6日**

目录

[1 引言 4](#_Toc58503578)

[1.1 目的 4](#_Toc58503579)

[1.2 背景及范围 4](#_Toc58503580)

[1.3 定义、术语或缩略语 4](#_Toc58503581)

[2 软件系统结构 5](#_Toc58503582)

[2.1 需求概述 5](#_Toc58503583)

[2.2 子模块划分 5](#_Toc58503584)

[2.2.1 系统模型图 5](#_Toc58503585)

[2.2.2 系统流程描述 5](#_Toc58503586)

[2.2.3 子模块划分 6](#_Toc58503587)

[3 各模块详细设计 6](#_Toc58503588)

[3.1 PHP文件解析模块 6](#_Toc58503589)

[3.1.1 模块功能描述 6](#_Toc58503590)

[3.1.2 获取PHP项目文件 7](#_Toc58503591)

[3.1.2.1 思路 7](#_Toc58503592)

[3.1.3.2 算法 7](#_Toc58503593)

[3.1.3 PHP文件生成语法分析树 8](#_Toc58503594)

[3.1.3.1 思路 8](#_Toc58503595)

[3.1.3.2 算法 8](#_Toc58503596)

[3.1.3.3 输出效果 8](#_Toc58503597)

[3.1.4 AST结点遍历的算法 9](#_Toc58503598)

[3.2变量控制流图生成模块 11](#_Toc58503599)

[3.2.1 模块功能描述 11](#_Toc58503600)

[3.2.2 CFG结点的数据结构 12](#_Toc58503601)

[3.2.3 通过AST提取所有变量名 12](#_Toc58503602)

[3.2.3.1 VariableVisitor类 12](#_Toc58503603)

[3.2.3.2 提取变量名 13](#_Toc58503604)

[3.2.4 根据变量流动确定变量的来龙去脉 13](#_Toc58503605)

[3.2.4.1 表达式 14](#_Toc58503606)

[3.2.4.2 数组 14](#_Toc58503607)

[3.2.4.3 函数调用 17](#_Toc58503608)

[3.2.4.4 方法调用 19](#_Toc58503609)

[3.2.4.5 超全局变量的传递 20](#_Toc58503610)

[3.3 SQL语言解析模块 22](#_Toc58503611)

[3.3.1 模块功能描述 22](#_Toc58503612)

[3.3.2 Sql语句的获取 22](#_Toc58503613)

[3.3.3 Sql语句的解析 23](#_Toc58503614)

[3.3.3 输出示例 26](#_Toc58503615)

[3.4回溯扫描查找模块 28](#_Toc58503616)

[3.4.1 模块功能描述 28](#_Toc58503617)

[3.4.2 CFG的最终确定 28](#_Toc58503618)

[3.4.3 回溯，确定去向 29](#_Toc58503619)

[3.4.4 查找注入点和触发点 30](#_Toc58503620)

[3.5动态注入测试模块 31](#_Toc58503621)

[3.5.1 模块功能描述 31](#_Toc58503622)

[3.5.2 网站注入示例 31](#_Toc58503623)

[3.5.2.1 漏洞分析 31](#_Toc58503624)

[3.5.2.2 利用过程 32](#_Toc58503625)

[3.5.2.3 算法实现 32](#_Toc58503626)

[3.5.3 输出 34](#_Toc58503627)

# 引言

### 1.1 目的

编写该详细设计说明书是为说明该项目——基于源代码检测和动态执行的二阶SQL注入漏洞检测工具的设计考虑，包括程序描述、输入/输出、算法和流程逻辑等，为软件编程和系统维护提供基础。本说明书的预期读者为系统设计人员、软件开发人员、软件测试人员和项目评审人员。其中系统设计人员、软件开发人员、软件测试人员为小组内部成员组成，软件评审人员为课程老师或助教。

### 1.2 背景及范围

* + 项目名称：基于源代码检测和动态执行的二阶SQL注入漏洞检测工具
  + 项目成员：北京邮电大学网络空间安全学院“网络安全分析实践”课程开发小组
    - 王硕（组长）：2018213641
    - 彭致远：2018213646
    - 李懿飞：2018213632
    - 王晨旭：2018213636
  + 系统范围：具有PHP7.4环境的Windows系统计算机
  + 用户：无限制
  + 实现项目的计算机网络：校园网

### 1.3 定义、术语或缩略语

|  |  |  |
| --- | --- | --- |
| 序号 | 术语或缩写 | 解释 |
| 1 | SQL | 结构化查询语言(Structured Query Language)简称SQL，是一种特殊目的的编程语言，是一种数据库查询和程序设计语言，用于存取数据以及查询、更新和管理关系数据库系统。 |
| 2 | AST | 抽象语法树（Abstract Syntax Tree，AST），是源代码语法结构的一种抽象表示。它以树状的形式表现编程语言的语法结构，树上的每个节点都表示源代码中的一种结构。 |
| 3 | CFG | 控制流程图，是一个过程或程序的抽象表现，是用在编译器中的一个抽象数据结构，由编译器在内部维护，代表了一个程序执行过程中会遍历到的所有路径。它用图的形式表示一个过程内所有基本块执行的可能流向, 也能反映一个过程的实时执行过程。本项目用于描述变量在流动过程的执行过程 |
| 4 | PHP-Parser | 开源工具，用于生成PHP文件的语法分析树 |
| 5 | SQL-Parser | 开源工具，用于生成SQL语句的语法分析树 |

# 2 软件系统结构

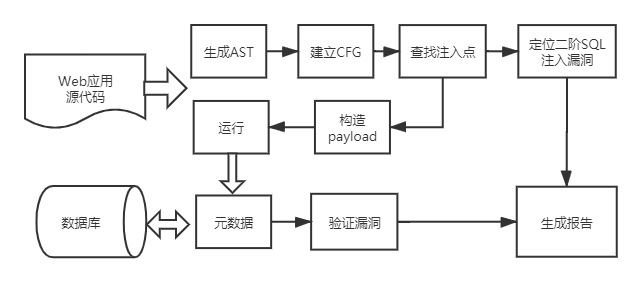
### 2.1 需求概述

开发一个基于源代码检测和动态执行的二阶SQL注入漏洞检测工具。源代码检测是系统中的主要部分，输入主要是PHP网站项目的源代码，包括前端生成应用程序源代码的CFG和后端的查找注入点、触发点，以及通过污点分析定位二阶SQL的注入漏洞，并生成漏洞报告。动态分析起到辅助作用，包括前期信息搜集获取的元数据和后期漏洞的验证。

### 2.2 子模块划分

#### 2.2.1 系统模型图

本工具的系统模型图如下所示：



#### 2.2.2 系统流程描述

本工具实现的过程总结如下：

* 以PHP项目源码作为输入，对每个PHP文件生成语法分析树，进而对每个PHP文件内的变量生成变量的控制流图，通过“超全局变量”“引用”“函数调用”等建立各个PHP文件之间的关系，从而生成所有变量的控制流图。
* 分析变量在流动过程中的来源、去向、变化。发现敏感数据即发现二阶SQL注入的注入点和触发点，保存为静态检测的结果
* 通过动态的执行，注入点输入payload，观察数据库内数据的变化，如果符合二阶SQL注入漏洞的特点，即生成报告

#### 2.2.3 子模块划分

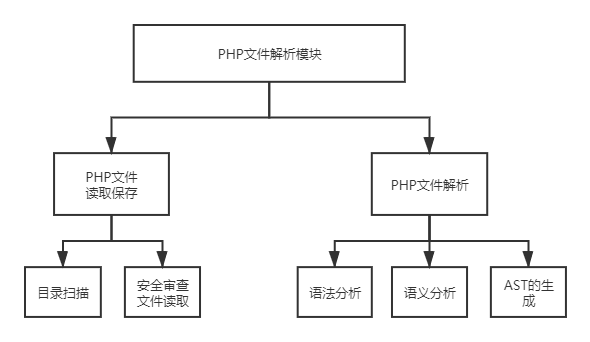
根据系统模型，可以将本模块划分为五个模块，分别是：

1. PHP文件解析模块
2. 变量控制流图生成模块
3. SQL语言解析模块
4. 回溯扫描查找模块
5. 动态注入测试模块

# 3 各模块详细设计

### 3.1 PHP文件解析模块

#### 3.1.1 模块功能描述

、

本模块的主要功能为：扫描用户PHP网站项目目录，从中提取PHP文件，对每个PHP文件，进行词法分析、语法分析、语义分析等，最终生成每个PHP文件对应的语法分析树，并提供一套递归的遍历方法，从而便于下一个模块从中递归的提取变量，并判断变量的深度（全局、局部、循环内等）

#### 3.1.2 获取PHP项目文件

###### 3.1.2.1 思路

如果考虑图形化界面，直接拖拽项目即可。这里采用控制台交互，故引导用户将web项目源代码放入指定文件夹内，输入项目名称，即可开始扫描。并判断文件是否存在，对用户的输入做检查。

###### 3.1.3.2 算法

1. echo "请将php项目放入桌面\n请输入项目名称: ";
3. $name = fgets(STDIN);
4. **while** (true){
5. $name = substr($name, 0, strpos($name, "\n"));
6. $dir = "C:\\Users\\23959\\Desktop\\" . $name;
7. **if**(is\_dir($dir)){
8. **break**;
9. }
10. **else**{
11. echo "项目文件不存在，请重新输入";
12. $name = fgets(STDIN);
13. }
14. }
15. echo "\n正在进行扫描项目".$name."...\n\n";
17. fetchPhpFile('C:/Users/23959/Desktop/'.$name, $file\_name\_list, $file\_contents\_list);
18. // 输出待扫描的php文件
19. echo "Scan Files:\n";
20. **foreach** ($file\_name\_list **as** $item){
21. echo $item."\n";
22. }
23. //获取所有php文件，文件名+文件内容
24. function fetchPhpFile($dir, &$array\_name, &$array\_contents) {
25. foreach(glob($dir.'/\*') as $file) {
26. **if**(is\_dir($file)) {
27. fetchPhpFile($file, $array\_name, $array\_contents);
28. }**else**{
29. $file\_name = basename($file);
30. **if**(substr($file\_name, -4) == '.php'){
31. $handle = fopen($file, "r");
32. $contents = fread($handle, filesize ($file));
33. array\_push($array\_contents, $contents);
34. array\_push($array\_name, $file\_name);
35. fclose($handle);
36. }
37. }
38. }
39. }

#### 3.1.3 PHP文件生成语法分析树

###### 3.1.3.1 思路

我们的检测模型主要针对PHP语言，选用开源工具PHP-Parser，通过语法分析，语义分析等步骤生成抽象语法树即可。

###### 3.1.3.2 算法

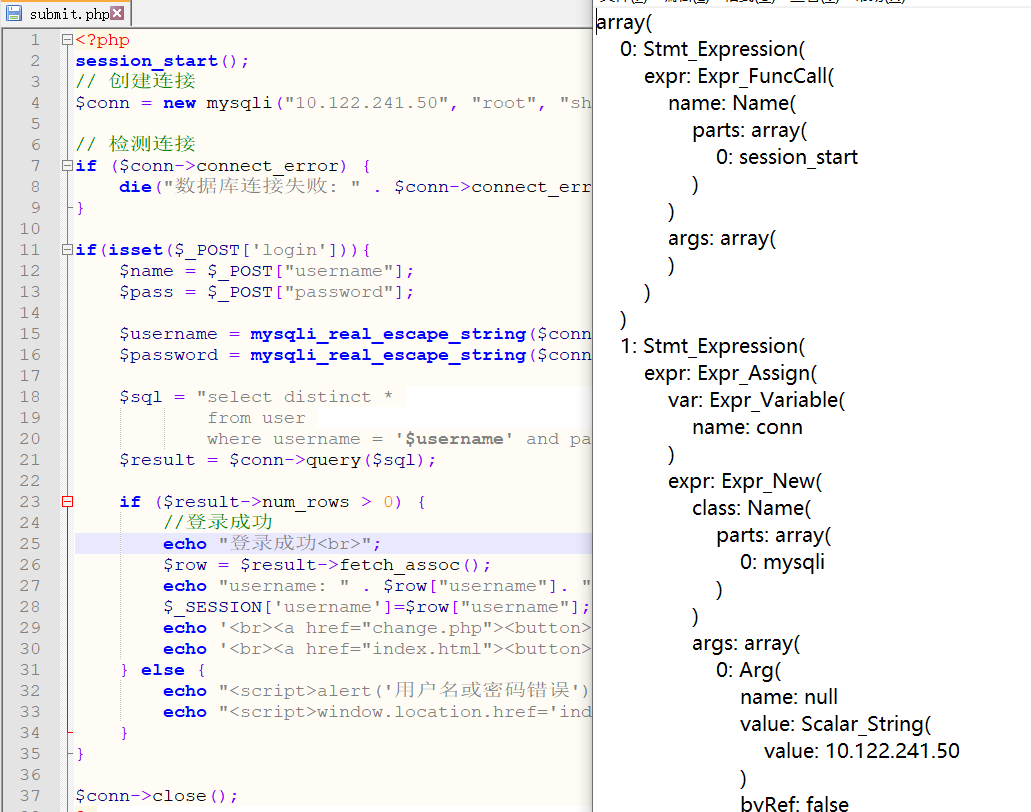
有了PHP-Parser工具，生成AST的算法较为简单，如下所示：

1. // 生成语法树
2. $parser = (**new** ParserFactory)->create(ParserFactory::PREFER\_PHP7);
3. try {
4. **for**($i = 0; $i < count($file\_name\_list); $i++){
5. $ast = $parser->parse($file\_contents\_list[$i]);
6. array\_push($ast\_list, $ast);
7. $dumper = **new** NodeDumper;
8. array\_push($dumper\_list, $dumper->dump($ast));
9. //echo $dumper->dump($ast)."\n";
10. }
11. } catch (Error $error) {
12. echo "Parse error: {$error->getMessage()}\n";
13. **return**;
14. }

###### 3.1.3.3 输出效果

生成的语法分析树实际上是树状数据结构，根节点即当前PHP文件，子节点可以是类、函数、表达式、数组、变量等等，利用print\_r（）函数将语法分析树打印出来效果如下：

源文件为左边的submit.php文件，输出后为右边的树状结构。



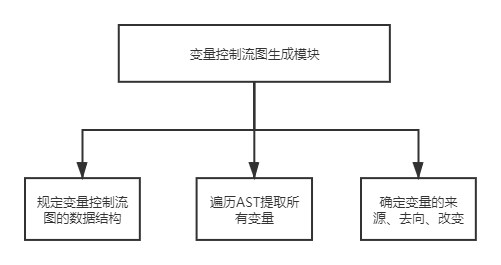
#### 3.1.4 AST结点遍历的算法

由于是树状结构，且每个结点的类型都是Node类型，递归的进行遍历即可，部分算法如下：

1. /\*\*
2. \* 递归遍历当前节点
3. \* @param Node $node 待遍历的节点
4. \* @return Node 遍历的结果 (可能是新节点、也可能是旧节点)
5. \*/
6. rotected **function** traverseNode(Node $node) : Node {
7. **foreach** ($node->getSubNodeNames() **as** $name) {
8. $subNode =& $node->$name;
9. // 子节点还是数组，继续递归遍历
10. **if** (\is\_array($subNode)) {
11. $subNode = $this->traverseArray($subNode);
12. **if** ($this->stopTraversal) {
13. **break**;
14. }
15. }
16. //不是的话，断掉
17. **elseif** ($subNode instanceof Node) {
18. $traverseChildren = true;
19. $breakVisitorIndex = null;
20. // 对节点的每一个属性，进行赋值（例如节点类型、左右节点、节点名称等）
21. **foreach** ($this->visitors **as** $visitorIndex => $visitor) {
22. $return = $visitor->enterNode($subNode);
23. **if** (null !== $return) {
24. **if** ($return instanceof Node) {
25. $this->ensureReplacementReasonable($subNode, $return);
26. $subNode = $return;
27. } **elseif** (self::DONT\_TRAVERSE\_CHILDREN === $return) {
28. $traverseChildren = false;
29. } **elseif** (self::DONT\_TRAVERSE\_CURRENT\_AND\_CHILDREN === $return) {
30. $traverseChildren = false;
31. $breakVisitorIndex = $visitorIndex;
32. **break**;
33. } **elseif** (self::STOP\_TRAVERSAL === $return) {
34. $this->stopTraversal = true;
35. **break** 2;
36. } **else** {
37. **throw** **new** \LogicException(
38. 'enterNode() returned invalid value of type ' . gettype($return)
39. );
40. }
41. }
42. }
43. **if** ($traverseChildren) {
44. $subNode = $this->traverseNode($subNode);
45. **if** ($this->stopTraversal) {
46. **break**;
47. }
48. }
49. **foreach** ($this->visitors **as** $visitorIndex => $visitor) {
50. $return = $visitor->leaveNode($subNode);
52. **if** (null !== $return) {
53. **if** ($return instanceof Node) {
54. $this->ensureReplacementReasonable($subNode, $return);
55. $subNode = $return;
56. } **elseif** (self::STOP\_TRAVERSAL === $return) {
57. $this->stopTraversal = true;
58. **break** 2;
59. } **elseif** (\is\_array($return)) {
60. **throw** **new** \LogicException(
61. 'leaveNode() may only return an array ' .
62. 'if the parent structure is an array'
63. );
64. } **else** {
65. **throw** **new** \LogicException(
66. 'leaveNode() returned invalid value of type ' . gettype($return)
67. );
68. }
69. }
70. **if** ($breakVisitorIndex === $visitorIndex) {
71. **break**;
72. }
73. }
74. }
75. }
76. **return** $node;

### 3.2变量控制流图生成模块

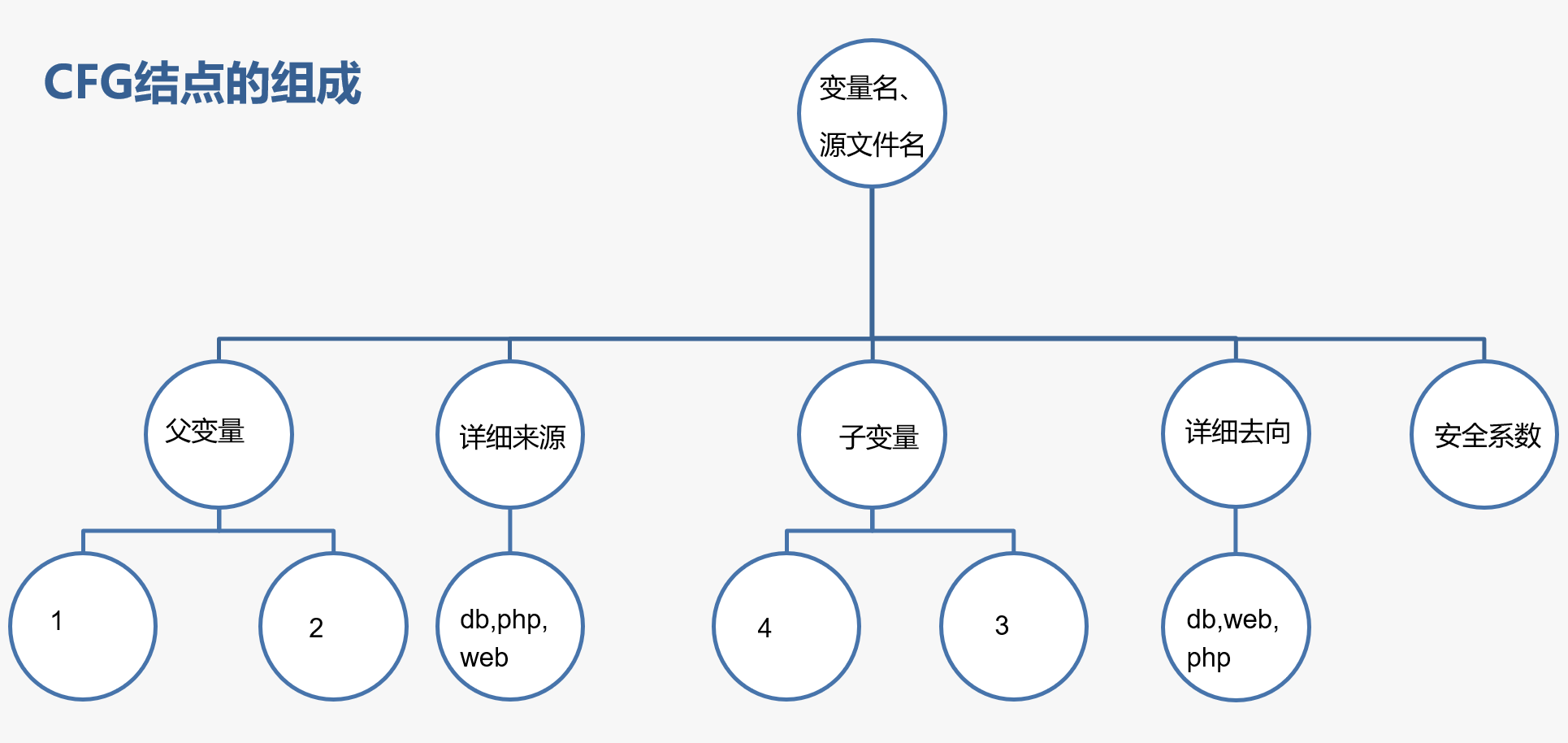
#### 3.2.1 模块功能描述



本模块的功能主要是通过PHP文件生成的语法分析树以及上一个模块提供的遍历AST的方法，提取PHP文件内部的变量，并通过变量之间的传递，建立变量的控制流图。通过“超全局变量”“引用”“函数调用”等建立各个PHP文件之间的关系，从而生成所有变量的控制流图。

#### 3.2.2 CFG结点的数据结构

要想确定变量的来源、去向，以及它在传递过程中发生了哪些变化，首先要确定变量的位置，即文件名+变量名，为了确定来源，就要有父变量的数组；为了确定去向，就要有自变量的数组。为了能检测出来二阶SQL注入漏洞的危险变量，就要确定它的最终来源和去向，即是否来源于/去向数据库、是否来源于/去向web网页，是否来源于/去向于PHP文件内部。所以CFG结点的结构图如下：



#### 3.2.3 通过AST提取所有变量名

###### 3.2.3.1 VariableVisitor类

利用上一个模块提供的遍历方法，编写VariableVisitor类，遇到节点类型为PhpParser\Node\Expr\Variable即返回，代码如下：

1. **class** VariableVisitor **extends** NodeVisitorAbstract {
2. **public** **array** $v\_name;
3. **public** **function** \_\_construct() {
4. $this->v\_name = **array**();
5. }
6. **public** **function** enterNode(Node $node)
7. {
8. **if** ($node instanceof PhpParser\Node\Expr\Variable){
9. array\_push($this->v\_name, $node->name);
10. //echo $node->name . "\n";
11. }
12. }
13. **public** **function** getVName(): **array**
14. {
15. **return** $this->v\_name;
16. }
17. }

###### 3.2.3.2 提取变量名

循环遍历每一棵语法分析树即可，代码如下：

1. //提取变量
2. $variable\_list = **array**();
3. **foreach** ($file\_name\_list **as** $item=>$value){
4. $traverser = **new** NodeTraverser();
5. $var\_visitor = **new** VariableVisitor();
6. $traverser->addVisitor($var\_visitor);
7. $modifiedStmts = $traverser->traverse($ast\_list[$item]);
8. fetchVariableList($value, $var\_visitor->getVName(),$variable\_list);
9. }
11. **function** fetchVariableList($file\_name, $var\_list\_tmp, &$var\_list){
12. **foreach** ($var\_list\_tmp **as** $var){
13. //[var\_name, var\_dir, src(db,php,web), src\_var, dst(db,php,web), dst\_var, safe\_level]
14. $arr = **array**();
15. $var\_info = [$var, $file\_name, "", $arr, "", $arr, 0];
16. **if**($var == "\_POST" || $var == "\_GET" || $var == "\_FILE")
17. $var\_info[2] = "web";
18. **elseif** ($var == "\_SESSION")
19. $var\_info[2] = "db";
20. $flag = false;
21. **for**($i = 0; $i < count($var\_list); $i++){
22. **if**($var\_list[$i][0] == $var && $var\_list[$i][1] == $file\_name){
23. $flag = true;
24. }
25. }
26. **if**($flag == false)
27. array\_push($var\_list, $var\_info);
28. }
29. }

#### 3.2.4 根据变量流动确定变量的来龙去脉

变量的来源/去向主要分类如下几类：

* 表达式 x=y
* 数组获取 x=y[“index”]
* 函数调用 x=f(y, z)
* 方法调用 x=y->f(y)
* 超全局变量的传递
* Web表单
* 数据库

下面进行逐个实现：

###### 3.2.4.1 表达式

普通的表达式即x=y+z之类的其实在实现下一块“数组”一节后，实际上就是“零维数组”的运算，只需要查看下面数组一节如何实现的就可以。

###### 3.2.4.2 数组

对于多维数组，其实也是一棵树，需要重新写一个遍历方法来遍历数组内的每一个元素，思路同上述遍历节点的算法，遍历数组的算法如下：

1. **function** traverseArray(**array** $nodes) : **array** {
2. $doNodes = [];
3. **foreach** ($nodes **as** $i => &$node) {
4. **if** ($node instanceof Node) {
5. $traverseChildren = true;
6. $breakVisitorIndex = null;
7. **foreach** ($this->visitors **as** $visitorIndex => $visitor) {
8. $return = $visitor->enterNode($node);
9. **if** (null !== $return) {
10. **if** ($return instanceof Node) {
11. $this->ensureReplacementReasonable($node, $return);
12. $node = $return;
13. } **elseif** (self::DONT\_TRAVERSE\_CHILDREN === $return) {
14. $traverseChildren = false;
15. } **elseif** (self::DONT\_TRAVERSE\_CURRENT\_AND\_CHILDREN === $return) {
16. $traverseChildren = false;
17. $breakVisitorIndex = $visitorIndex;
18. **break**;
19. } **elseif** (self::STOP\_TRAVERSAL === $return) {
20. $this->stopTraversal = true;
21. **break** 2;
22. } **else** {
23. **throw** **new** \LogicException(
24. 'enterNode() returned invalid value of type ' . gettype($return)
25. );
26. }
27. }
28. }
29. **if** ($traverseChildren) {
30. $node = $this->traverseNode($node);
31. **if** ($this->stopTraversal) {
32. **break**;
33. }
34. }
35. **foreach** ($this->visitors **as** $visitorIndex => $visitor) {
36. $return = $visitor->leaveNode($node);
38. **if** (null !== $return) {
39. **if** ($return instanceof Node) {
40. $this->ensureReplacementReasonable($node, $return);
41. $node = $return;
42. } **elseif** (\is\_array($return)) {
43. $doNodes[] = [$i, $return];
44. **break**;
45. } **elseif** (self::REMOVE\_NODE === $return) {
46. $doNodes[] = [$i, []];
47. **break**;
48. } **elseif** (self::STOP\_TRAVERSAL === $return) {
49. $this->stopTraversal = true;
50. **break** 2;
51. } **elseif** (false === $return) {
52. **throw** **new** \LogicException(
53. 'bool(false) return from leaveNode() no longer supported. ' .
54. 'Return NodeTraverser::REMOVE\_NODE instead'
55. );
56. } **else** {
57. **throw** **new** \LogicException(
58. 'leaveNode() returned invalid value of type ' . gettype($return)
59. );
60. }
61. }
62. **if** ($breakVisitorIndex === $visitorIndex) {
63. **break**;
64. }
65. }
66. } **elseif** (\is\_array($node)) {
67. **throw** **new** \LogicException('Invalid node structure: Contains nested arrays');
68. }
69. }
70. **if** (!empty**empty**($doNodes)) {
71. **while** (list($i, $replace) = array\_pop($doNodes)) {
72. array\_splice($nodes, $i, 1, $replace);
73. }
74. }
75. **return** $nodes;
76. }

实现遍历数组的方法后，就可以对数组的元素进行获取了，下面列出对一维数组和二维数组的遍历提取方法，数组的来源和去向相当于对遍历路径的每一个结点，都要进行分析。如果有某个结点的参数为某个遍历，如果该变量存在风险，那么表达式左边的变量也存在风险。只有当数组某个结点的遍历路径均没有危险时，表达式左边的遍历才不存在危险。

1. **if** ($node instanceof PhpParser\Node\Stmt\Expression){
2. **if**($node->expr->getType() == "Expr\_Assign"){
3. **if**($node->expr->**var**->getType() == "Expr\_ArrayDimFetch")
4. $variable\_left = $node->expr->**var**->**var**->name;
5. **else**
6. $variable\_left = $node->expr->**var**->name;
7. $index\_left = findIndex($variable\_left, $this->dir, $this->variable\_list);
9. **if**($node->expr->expr->getType() == "Expr\_ArrayDimFetch"){
10. **if**($node->expr->expr->**var**->getType() == "Expr\_ArrayDimFetch")
11. $variable\_right = $node->expr->expr->**var**->**var**->name;
12. **else**
13. $variable\_right = $node->expr->expr->**var**->name;
14. $index\_right = findIndex($variable\_right, $this->dir, $this->variable\_list);
16. **if**($this->variable\_list[$index\_right][2] != ""){
17. $this->variable\_list[$index\_left][2] = $this->variable\_list[$index\_right][2];
18. }
20. **if**($variable\_right == "\_POST"){
21. $dim\_value\_right = $node->expr->expr->dim->value;
22. $this->variable\_list[$index\_left][2] = "web\_form(".$dim\_value\_right.")";
23. }**elseif**($variable\_right == "\_GET"){
24. $dim\_value\_right = $node->expr->expr->dim->value;
25. **if**($variable\_left == "fName" && $this->variable\_list[$index\_left][1] == "download.php"){
26. $this->variable\_list[$index\_left][2] = "db(file(".$dim\_value\_right."))";
27. }
28. **else**
29. $this->variable\_list[$index\_left][2] = "web\_form(".$dim\_value\_right.")";
30. }**elseif**($variable\_right == "\_FILES"){
31. $dim\_value\_right = $node->expr->expr->dim->value;
32. $this->variable\_list[$index\_left][2] = "web\_file(".$dim\_value\_right.")";
33. } **elseif** ($variable\_right == "\_SESSION"){
34. $dim\_value\_right = $node->expr->expr->dim->value;
35. $this->variable\_list[$index\_left][2] = "db(user(".$dim\_value\_right."))";
36. }
37. array\_push($this->variable\_list[$index\_left][3], $index\_right);
38. array\_push($this->variable\_list[$index\_right][5], $index\_left);
40. }
41. }

###### 3.2.4.3 函数调用

函数调用，主要分析函数内部的参数，只要函数的某一个参数来源于web表单或者数据库，那么表达式左边的变量即存在风险，标记为对应的来源于web还是来源于数据库，算法相比数组较为简单，代码如下所示：

接上面的if，下面为else if，表示在遇到变量流动时候的一些条件：

1. **else** **if**($node->expr->expr->getType() == "Expr\_FuncCall" || $node->expr->expr->getType() == "Expr\_New"){
2. $flag = true;
3. **foreach**($node->expr->expr->args **as** $item){
4. **if**($item->value->getType() == "Expr\_Variable"){
5. $flag = false;
6. $variable\_right = $item->value->name;
7. $index\_right = findIndex($variable\_right, $this->dir, $this->variable\_list);
8. array\_push($this->variable\_list[$index\_left][3], $index\_right);
9. array\_push($this->variable\_list[$index\_right][5], $index\_left);
10. **if**($this->variable\_list[$index\_left][2] == ""){
11. **if**($this->variable\_list[$index\_right][2] != "php"){
12. $this->variable\_list[$index\_left][2] = $this->variable\_list[$index\_right][2];
13. }
14. }
15. }
16. }
17. **if**($flag){
18. $this->variable\_list[$index\_left][2] = "php";
19. }
20. }
21. **else** **if**($node->expr->expr->getType() == "Expr\_FuncCall"){
22. $flag = true;
23. **foreach**($node->expr->expr->args **as** $item){
24. **if**($item->value->getType() == "Expr\_Variable"){
25. $flag = false;
26. $variable\_right = $item->value->name;
27. $index\_right = findIndex($variable\_right, $this->dir, $this->variable\_list);
28. array\_push($this->variable\_list[$index\_left][3], $index\_right);
29. array\_push($this->variable\_list[$index\_right][5], $index\_left);
30. **if**($this->variable\_list[$index\_left][2] == ""){
31. **if**($this->variable\_list[$index\_right][2] != "php"){
32. $this->variable\_list[$index\_left][2] = $this->variable\_list[$index\_right][2];
33. }
34. }
35. }
36. }
37. **if**($flag){
38. $this->variable\_list[$index\_left][2] = "php";
39. }
41. $white\_list = [
42. "mysql\_real\_escape\_string", "mysqli\_real\_escape\_string",
43. "addslashes", "htmlspecialchars", "strip\_tags",
44. "magic\_quotes\_gpc", "register\_globals",
45. ];
46. $func\_name = $node->expr->expr->name->parts[0];
47. $flag = true;
48. **foreach** ($white\_list **as** $safe\_func){
49. **if**($func\_name == $safe\_func){
50. $this->variable\_list[$index\_left][6] = 1;
51. $flag = false;
52. **break**;
53. }
54. }
55. **if** ($flag){
56. $rtn = analyseFunction($func\_name);
57. $this->variable\_list[$index\_left][6] = $rtn;
58. }
59. }

###### 3.2.4.4 方法调用

方法调用相比于函数调用，它没有等于号，即不算是一个表达式，但是方法调用的时候，也会对变量进行操作，也有可能存在方法参数，而参数的来源不安全，也会导致调用方法的变量存在风险。所以也是存在变量的流动的。只有当方法内部的变量、参数都安全时，被调用方法的变量才安全，算法如下：

1. **else** **if**($node->expr->expr->getType() == "Expr\_MethodCall") {
2. $flag = true;
3. $variable\_right = $node->expr->expr->**var**->name;
4. $index\_right = findIndex($variable\_right, $this->dir, $this->variable\_list);
5. array\_push($this->variable\_list[$index\_left][3], $index\_right);
6. array\_push($this->variable\_list[$index\_right][5], $index\_left);
7. **if** ($this->variable\_list[$index\_right][2] != "" && $this->variable\_list[$index\_right][2] != "php") {
8. $this->variable\_list[$index\_left][2] = $this->variable\_list[$index\_right][2];
9. } **else** {
10. **foreach** ($node->expr->expr->args **as** $item) {
11. **if** ($item->value->getType() == "Expr\_Variable") {
12. $flag = false;
13. $variable\_right = $item->value->name;
14. $index\_right = findIndex($variable\_right, $this->dir, $this->variable\_list);
15. array\_push($this->variable\_list[$index\_left][3], $index\_right);
16. array\_push($this->variable\_list[$index\_right][5], $index\_left);
17. **if** ($this->variable\_list[$index\_left][2] == "") {
18. **if** ($this->variable\_list[$index\_right][2] != "php") {
19. $this->variable\_list[$index\_left][2] = $this->variable\_list[$index\_right][2];
20. }
21. }
22. }
23. }
24. **if** ($flag) {
25. $this->variable\_list[$index\_left][2] = "php";
26. }
27. }

###### 3.2.4.5 超全局变量的传递

（1）PHP的超全局变量

PHP 中的许多预定义变量都是“超全局的”，这意味着它们在一个脚本的全部作用域中都可用。在函数或方法中无需执行 global $variable; 就可以访问它们。

* $GLOBALS 一个包含了全部变量的全局组合数组。变量的名字就是数组的键。
* $\_SERVER 是一个包含了诸如头信息(header)、路径(path)、以及脚本位置(script locations)等等信息的数组。这个数组中的项目由 Web 服务器创建。
* $\_GET 通过 URL 参数（又叫 query string）传递给当前脚本的变量的数组。
* $\_POST 当 HTTP POST 请求的 Content-Type 是 form-data 时，会将变量以关联数组形式传入当前脚本。
* $\_FILES 通过 HTTP POST 方式上传到当前脚本的项目的数组。
* $\_COOKIE 通过 HTTP Cookies 方式传递给当前脚本的变量的数组。
* $\_SESSION 当前脚本可用 SESSION 变量的数组。
* $\_REQUEST  — HTTP Request 变量，默认情况下包含了 $\_GET，$\_POST 和 $\_COOKIE 的数组。
* $\_ENV 通过环境方式传递给当前脚本的变量的数组。

（2）利用超全局变量可以建立PHP文件间的联系

实际上超全局变量为基本上是数组的形式，如GET[“username”]等等，所以在数组里加入超全局变量的解析即可，部分算法如下：

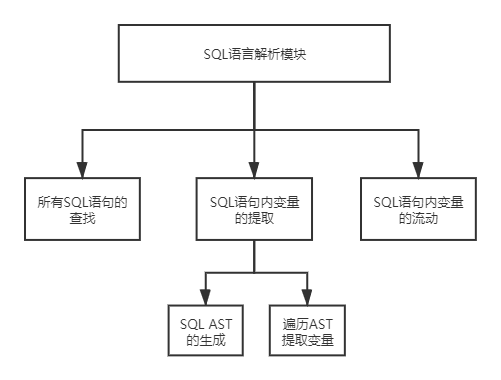
1. **if**($node->expr->expr->getType() == "Expr\_ArrayDimFetch"){
2. **if**($node->expr->expr->**var**->getType() == "Expr\_ArrayDimFetch")
3. $variable\_right = $node->expr->expr->**var**->**var**->name;
4. **else**
5. $variable\_right = $node->expr->expr->**var**->name;
6. $index\_right = findIndex($variable\_right, $this->dir, $this->variable\_list);
8. **if**($this->variable\_list[$index\_right][2] != ""){
9. $this->variable\_list[$index\_left][2] = $this->variable\_list[$index\_right][2];
10. }
12. $dim\_value\_right = $node->expr->expr->dim->value;
13. **if**($variable\_right == "\_POST"){
14. $var = "\_POST[".$dim\_value\_right."]";
15. $file\_name = $this->variable\_list[$index\_left][1];
16. $arr = **array**();
17. $var\_info = [$var, $file\_name, "", $arr, "", $arr, 0];
18. array\_push($var\_list, $var\_info);
19. $this->variable\_list[$index\_left][2] = "web\_form(".$dim\_value\_right.")";
20. }**elseif**($variable\_right == "\_GET"){
21. $var = "\_POST[".$dim\_value\_right."]";
22. $file\_name = $this->variable\_list[$index\_left][1];
23. $arr = **array**();
24. $var\_info = [$var, $file\_name, "", $arr, "", $arr, 0];
25. array\_push($var\_list, $var\_info);
26. analyseGlobalVariable("\_GET", $dim\_value\_right,$this->dir, $this->variable\_list);
27. }**elseif**($variable\_right == "\_FILES"){
28. $this->variable\_list[$index\_left][2] = "web\_file(".$dim\_value\_right.")";
29. } **elseif** ($variable\_right == "\_SESSION"){
30. $var = "\_SESSION[".$dim\_value\_right."]";
31. $file\_name = $this->variable\_list[$index\_left][1];
32. $arr = **array**();
33. $var\_info = [$var, $file\_name, "", $arr, "", $arr, 0];
34. array\_push($var\_list, $var\_info);
35. analyseGlobalVariable("\_SESSION", $dim\_value\_right, $this->dir, $this->variable\_list);
36. }

（3）其中analyseGlobalVariable() 函数为分析超全局变量的流向，更新变量的控制流图，其代码如下：

1. **function** analyseGlobalVariable($name, $dim, $dir, $var\_list){
2. $left\_index = findIndex($name.$dim, $dir, $var\_list);
3. $list = searchGlobalVariableLocation($dim);
4. $var\_right\_name = $list[0];
5. $var\_right\_dir = $list[1];
6. $var\_right\_index = findIndex($var\_right\_name, $var\_right\_dir, $var\_list);
7. **if**($var\_list[$var\_right\_index][2] != "" && ($var\_list[$var\_right\_index][2][1] == 'w' ||  $var\_list[$var\_right\_index][2][1] == 'd')){
8. $var\_list[$left\_index][2] = $var\_list[$var\_right\_index][2];
9. }**elseif**($var\_list[$var\_right\_index][2] != "" && $var\_list[$var\_right\_index][2][1] == 'p'){
10. $var\_list[$left\_index][2] = "php";
11. }
12. }

### 3.3 SQL语言解析模块

#### 3.3.1 模块功能描述



本模块主要是用来获取和解析php源代码中的SQL语句，首先通过php-parser解析树的解析结果获取sql语句的信息，然后再利用sql-parser解析获取的sql语句，将sql语句中的关键信息（如变量名、表项名、数据库名、变量流动）提取和整理，保存在数组中，供其他php分析模块使用。

#### 3.3.2 Sql语句的获取

Sql语句的获取主要通过开源的sql-parser来实现，sql-parser会根据php源码生成php解析树，并根据解析的结果为每一个元素定义类型。在这里，sql语句主体的类型为**EncapsedStringPart，**而sql语句中的变量类型为**Variable**，这些变量类型都是sql-parser中自己定义的。通过Visitor访问各个节点，并判断当前访问的节点是否为该类型的数据，就可以将php源代码中的sql语句进行识别和拼接，方便我们的分析。

具体代码如下所示：

1. **if** ($node instanceof Encapsed)
2. {
3. $sql\_array = $node->parts;
4. $sql\_string = "";
5. **foreach** ($sql\_array **as** $sql){
6. **if** ($sql instanceof EncapsedStringPart){
7. $sql\_string = $sql\_string . $sql->value;
8. }
9. **if** ($sql instanceof Variable){
10. $sql\_string = $sql\_string . $sql->name;
11. }
12. }
13. echo $sql\_string . "\n";

#### 3.3.3 Sql语句的解析

对于每一个获得的sql语句，我们都可以将其放入开源库sql-parser中进行解析。我们首先定义一个sql-parser的解析器，然后对我们刚刚得到的sql语句进行解析，并输出解析树验证正确性，同时分析sql语句的解析结果，从而可以对每一个我们需要的数据进行分析。

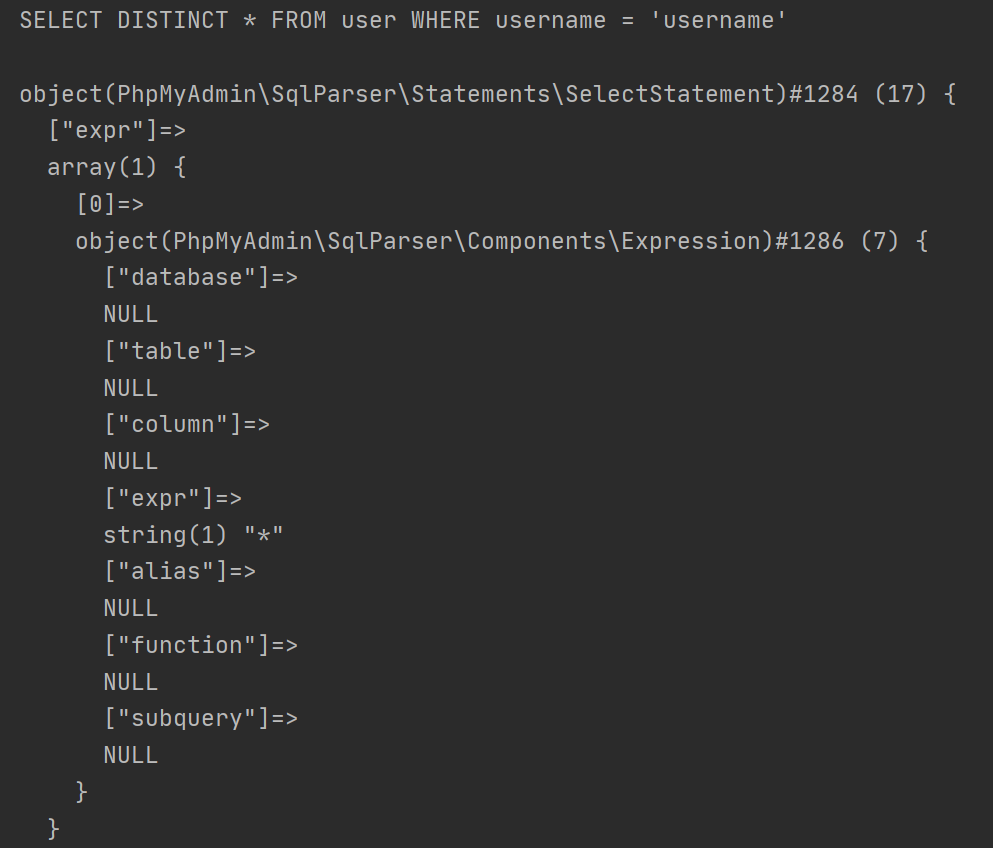
由于sql语句有不同的类型，每种类型的sql语句有着不同结构，因此对于不用类型的sql语句有着不同得解析规则，但是解析得结果都是统一的，即获取sql语句中得变量名、表项名、数据库名、并分析变量的流动方向，然后将这些信息存入variable\_list数组中。

具体实现过程如下所示：

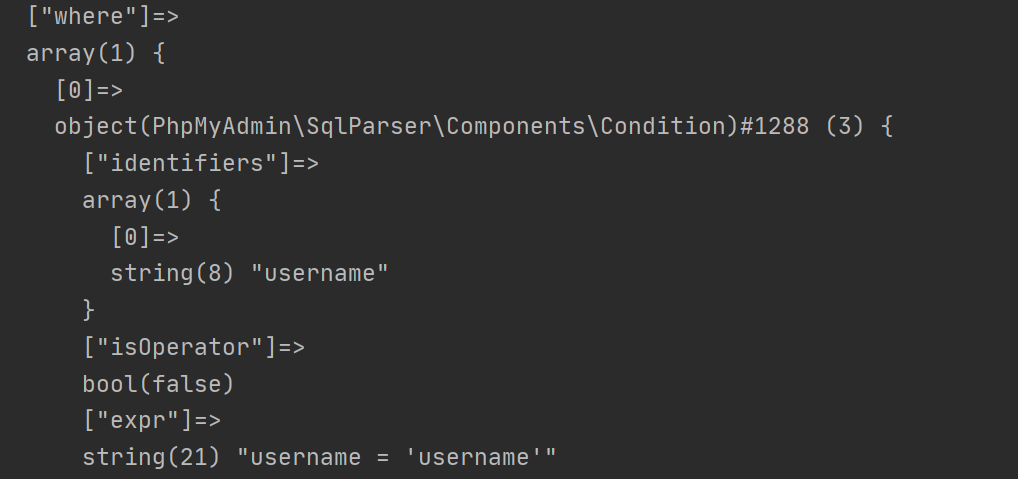
1. echo $sql\_string . "\n";
2. $parser = **new** PHPSQLParser($sql\_string, true);
3. $sql\_ast = $parser->parsed;
4. print\_r($sql\_ast);
5. $table\_name = "";
6. $attributes\_list = **array**();
7. $var\_list = **array**();
8. **if** (strtolower(array\_key\_first($sql\_ast)) == "select"){
9. **foreach** ($sql\_ast **as** $item=>$value){
10. **if** ($item == "SELECT"){
11. **continue**;
12. }
13. **elseif**($item == "FROM"){
14. **if** ($value[0]["expr\_type"] == "table")
15. $table\_name = $value[0]["table"];
16. }
17. **elseif**($item == "WHERE"){
18. **foreach** ($value **as** $part){
19. **if** ($part["expr\_type"] == "colref")
20. array\_push($attributes\_list, $part["base\_expr"]);
21. **elseif** ($part["expr\_type"] == "const"){
22. $tmp\_expr = str\_replace("%","",$part["base\_expr"]);
23. array\_push($var\_list, $tmp\_expr);
24. }
25. }
26. }
27. }
28. $this->sql\_variable =
29. addToSqlVariable($this->sql\_variable, $var\_list, $table\_name, $attributes\_list, "from\_db");
30. }
31. **elseif** (strtolower(array\_key\_first($sql\_ast)) == "insert"){
32. **foreach** ($sql\_ast **as** $item => $value){
33. **if** ($item == "INSERT"){
34. **foreach** ($value **as** $part){
35. **if** ($part["expr\_type"] == "table")
36. $table\_name = $part["table"];
37. **else** **if** ($part["expr\_type"] == "column-list"){
38. **foreach** ($part["sub\_tree"] **as** $attr){
39. **if** ($attr["expr\_type"] == "colref")
40. array\_push($attributes\_list, $attr["base\_expr"]);
41. }
42. }
43. }
44. }
45. **elseif** ($item == "VALUES"){
46. **foreach** ($value **as** $part){
47. **if** ($part["expr\_type"] == "record"){
48. **foreach** ($part["data"] **as** $var){
49. **if** ($var["expr\_type"] == "const")
50. array\_push($var\_list, $var["base\_expr"]);
51. }
52. }
53. }
54. }
55. }
56. $this->sql\_variable =
57. addToSqlVariable($this->sql\_variable, $var\_list, $table\_name, $attributes\_list, "to\_db");
58. }
59. **elseif** (strtolower(array\_key\_first($sql\_ast)) == "update"){
60. **foreach** ($sql\_ast **as** $item=>$value){
61. **if** ($item == "UPDATE"){
62. **foreach** ($value **as** $part){
63. **if** ($part["expr\_type"] == "table")
64. $table\_name = $part["table"];
65. }
66. }
67. **elseif** ($item == "SET"){
68. **foreach** ($value **as** $set\_part){
69. **if** ($set\_part["expr\_type"] == "expression"){
70. **foreach** ($set\_part["sub\_tree"] **as** $part){
71. **if** ($part["expr\_type"] == "colref")
72. array\_push($attributes\_list, $part["base\_expr"]);
73. **elseif** ($part["expr\_type"] == "const")
74. array\_push($var\_list, $part["base\_expr"]);
75. }
76. }
77. }
78. $this->sql\_variable =
79. addToSqlVariable($this->sql\_variable, $var\_list, $table\_name, $attributes\_list, "to\_db");
80. $var\_list = **array**();
81. $attributes\_list = **array**();
82. }
83. **elseif** ($item == "WHERE"){
84. **foreach** ($value **as** $where\_part){
85. **if** ($where\_part["expr\_type"] == "colref")
86. array\_push($attributes\_list, $where\_part["base\_expr"]);
87. **if** ($where\_part["expr\_type"] == "const"){
88. $temp\_expr = str\_replace("%","", $where\_part["base\_expr"]);
89. array\_push($var\_list, $temp\_expr);
90. }
91. }
92. $this->sql\_variable =
93. addToSqlVariable($this->sql\_variable, $var\_list, $table\_name, $attributes\_list, "from\_db");
94. }
95. }
96. }
97. echo "\n";
98. print\_r($this->sql\_variable);
99. echo "\n";
100. print\_r($this->sql\_variable);

#### 3.3.3 输出示例

对于如下的SQL语句，输出的语法分析树如下图所示：

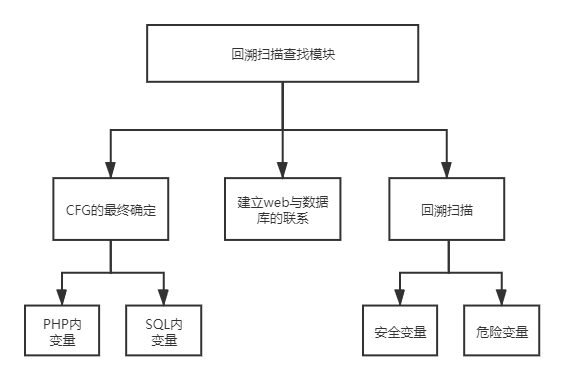






### 3.4回溯扫描查找模块

#### 3.4.1 模块功能描述



本模块的主要功能为：建立PHP和SQL的变量关系，生成最终的变量控制流图，并回溯扫描将变量分类，分成安全变量和危险变量，并记录二阶SQL注入漏洞的注入点和触发点，作为静态扫描的结果。

#### 3.4.2 CFG的最终确定

在遍历每一条SQL语句的时候，通过SQL-Parser以及上一个模块生成了变量在数据库内的流动关系，更新变量的控制流图，算法较为简单，代码如下：

1. // 对每一条SQL语句进行分析并调用下面的函数更新CFG
2. **foreach** ($file\_name\_list **as** $item=>$value){
3. $traverser3 = **new** NodeTraverser();
4. $sql\_visitor = **new** SqlVisitor();
5. $traverser3->addVisitor($sql\_visitor);
6. $modifiedStmts = $traverser3->traverse($ast\_list[$item]);
7. $sql\_variable = $sql\_visitor->getSqlVariable();
8. improveDatabaseVariable($value, $sql\_variable, $variable\_list);
9. // print\_r($sql\_variable);
10. }
12. // 更新CFG
13. **function** improveDatabaseVariable($file\_name, $sql\_var\_list, &$variable\_list){
14. **if**(count($sql\_var\_list) != 0){
15. **foreach** ($sql\_var\_list **as** $value){
17. $var\_name = str\_replace("%", "", substr($value[0], 1, strlen($value[0])-2));
19. $index = findIndex($var\_name, $file\_name, $variable\_list);
20. **if**($value[3] == "from\_db"){
21. $variable\_list[$index][4] = "db(".$value[1]."(".$value[2]."))";
22. }**elseif**($value[3] == "to\_db"){
23. $variable\_list[$index][4] = "db(".$value[1]."(".$value[2]."))";
24. }
25. }
26. }
27. }

#### 3.4.3 回溯，确定去向

知道了来源，回溯即可得到每个变量的去向，这是一个将单项链表变为双向链表的过程，代码如下：

1. //回溯，确定去向
2. $flag = true;
3. **while**($flag){
4. $flag = false;
5. **foreach** ($variable\_list **as** $k => $v){
6. **if**($v[4] != ""){
7. **foreach** ($v[3] **as** $item){
8. **if**($variable\_list[$item][4] == ""){
9. $variable\_list[$item][4] = $variable\_list[$k][4];
10. $flag = true;
11. }
12. }
13. }
14. }
15. }
16. // 空的填充php
17. **foreach** ($variable\_list **as** $value){
18. **if**($value[2] == ""){
19. $value[2] = "php";
20. }
21. **if**($value[4] == ""){
22. $value[4] = "php";
23. }
24. }

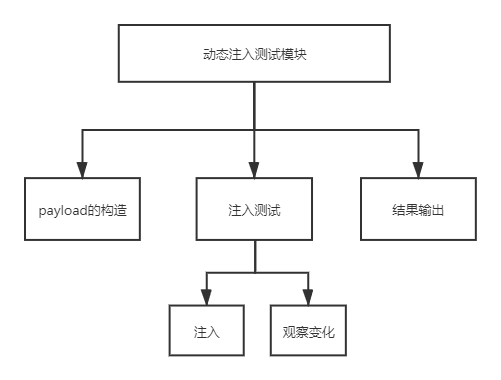
#### 3.4.4 查找注入点和触发点

根据二阶SQL注入产生的条件，如果有变量来源于web，在某个地方进入了数据库的某个table里，而在另一处又从数据库内拿出来，并最终进入某条SQL语句执行，并对其他变量产生的影响，那么它就存在二阶SQL注入的风险。代码如下：

1. //比较，输出结果
2. $count = 0;             //注入点数
3. $result = **array**();      //注入点、触发点等信息
4. **foreach** ($variable\_list **as** $sk=>$sv){
5. **if**($sv[2] != "" && $sv[4]!="" && $sv[2][0] == 'd' && $sv[4][0] == 'd'){
6. **foreach** ($variable\_list **as** $fk=>$fv){
7. **if**($fv[2] != "" && $fv[4]!="" && $fv[2][0] == 'w' && $fv[4][0] == 'd' && $fv[2] != "web"){
8. **if**($fv[4] != $sv[2]){ **continue**; }
9. $flag = true;
10. **foreach** ($result **as** $value){
11. **if**($value[0] == $fv[1] && $value[1] == $fv[2] && $value[2] == $sv[1] && $value[3] == $sv[2]){
12. $flag = false;
13. **break**;
14. }
15. }
16. **if**($flag){
17. $result[$count] = [$fv[1], $fv[2], $sv[1], $sv[2]];
18. $count++;
19. }
20. }
21. }
22. }
23. }

### 3.5动态注入测试模块

#### 3.5.1 模块功能描述



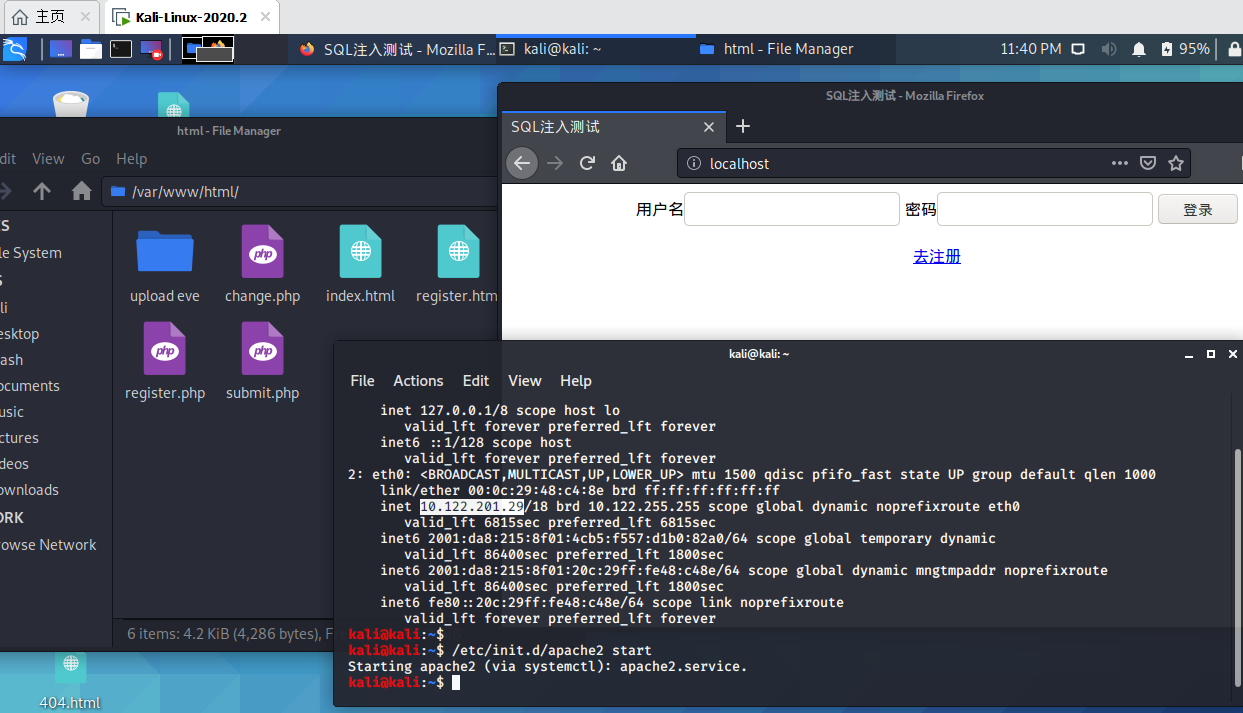
本模块的主要功能为：由于静态的扫描误报率很高，所以动态的去验证是必要的。通过构造payload，进行数据库的注入，并观察网站和数据库内的变化，如果构造的payload与正常的数据产生的变化不同，那么就存在二阶SQL注入漏洞。并将结果保存，输入。

#### 3.5.2 网站注入示例

二阶SQL注入的payload构造与一阶SQL盲注不一样，这里以我们搭建的网站demo进行说明其过程：

###### 3.5.2.1 漏洞分析

服务器及运行效果如下：



网站过滤了一阶SQL注入漏洞，即使用PHP安全函数进行了字符的转义再执行SQL语句，但是存储在数据库内还是原数据，再次从数据库拿出来的时候没有进行转义，所以存在二阶SQL注入漏洞。

###### 3.5.2.2 利用过程

这种二阶SQL注入漏洞的检测构造成功的payload以及操作过程如下：

* 对注入点构造http请求，模拟用户输入（test）
* 类似1，模拟输入（test‘#）
* 寻找触发点，构造http请求
* 修改攻击（test’#）相关的信息、
* 查看（test）相关的信息是否被修改

###### 3.5.2.3 算法实现

由于可以从源码扫描中知道数据库内表格的信息，所以这里直接进行第一次正常操作如下和第二次的注入操作，将test和test’#写入数据库。

1. $regist\_url = "http://localhost:63342/SQL注入环境/register.php?\_ijt=m255cl3hk9n8behc1n0p4f9fg";
2. $cookie = dirname(**\_\_FILE\_\_**) . '/cookie\_ydma.txt'; //设置cookie保存的路径
3. $regist = **array**(
4. 'username' => 'test3',
5. 'password' => 123456
6. );
8. $regist1 = **array**(
9. 'username' => 'test3#',
10. 'password' => 123456
11. );
12. $ch = curl\_init();
13. curl\_setopt($ch, CURLOPT\_URL, $regist\_url);
14. curl\_setopt($ch, CURLOPT\_HEADER, 0); //是否显示头信息
15. curl\_setopt($ch, CURLOPT\_RETURNTRANSFER, 0); //是否自动显示返回的信息
16. //curl\_setopt($ch, CURLOPT\_COOKIEJAR, $cookie); //设置cookie信息保存在指定的文件夹中
17. curl\_setopt($ch, CURLOPT\_POST, 1); //以POST方式提交
18. curl\_setopt($ch, CURLOPT\_POSTFIELDS, $regist);//要执行的信息
19. curl\_exec($ch); //执行CURL
20. curl\_close($ch);
22. $ch = curl\_init();
23. curl\_setopt($ch, CURLOPT\_URL, $regist\_url);
24. curl\_setopt($ch, CURLOPT\_HEADER, 0); //是否显示头信息
25. curl\_setopt($ch, CURLOPT\_RETURNTRANSFER, 0); //是否自动显示返回的信息
26. //curl\_setopt($ch, CURLOPT\_COOKIEJAR, $cookie); //设置cookie信息保存在指定的文件夹中
27. curl\_setopt($ch, CURLOPT\_POST, 1); //以POST方式提交
28. curl\_setopt($ch, CURLOPT\_POSTFIELDS, $regist1);//要执行的信息
29. curl\_exec($ch); //执行CURL
30. curl\_close($ch);

接下来是触发点（本demo在修改密码界面）：模拟用户登录后修改密码的过程如下：

1. $login\_url = "http://localhost:63342/SQL%E6%B3%A8%E5%85%A5%E7%8E%AF%E5%A2%83/submit.php";
2. $cookie = dirname(**\_\_FILE\_\_**) . '/cookie\_ydma.txt'; //设置cookie保存的路径
3. $first\_login = **array**(
4. 'username' => "test",
5. 'password' => 123456
6. );
8. $ch = curl\_init();
9. curl\_setopt($ch, CURLOPT\_URL, $login\_url); //登录提交的地址
10. curl\_setopt($ch, CURLOPT\_HEADER, 0); //是否显示头信息
11. curl\_setopt($ch, CURLOPT\_RETURNTRANSFER, 0); //是否自动显示返回的信息
12. curl\_setopt($ch, CURLOPT\_COOKIEJAR, $cookie); //设置cookie信息保存在指定的文件夹中
13. curl\_setopt($ch, CURLOPT\_POST, 1); //以POST方式提交
14. curl\_setopt($ch, CURLOPT\_POSTFIELDS, $first\_login);//要执行的信息
15. curl\_exec($ch); //执行CURL
16. curl\_close($ch);
17. $cookie = getCookie('test\'#');
18. $change\_url = "localhost:63342/SQL%E6%B3%A8%E5%85%A5%E7%8E%AF%E5%A2%83/change.php";
19. $change = **array**(
20. 'old\_password' => '123456',
21. 'new\_password' => '123456change'
22. );
24. $ch = curl\_init();
25. curl\_setopt($ch, CURLOPT\_URL, $change\_url);
26. curl\_setopt($ch, CURLOPT\_HEADER, 0); //是否显示头信息
27. curl\_setopt($ch, CURLOPT\_RETURNTRANSFER, 0); //是否自动显示返回的信息
28. curl\_setopt($ch, CURLOPT\_COOKIEFILE, $cookie); //设置cookie信息保存在指定的文件夹中
29. curl\_setopt($ch, CURLOPT\_POST, 1); //以POST方式提交
30. curl\_setopt($ch, CURLOPT\_POSTFIELDS, $change);//要执行的信息
31. curl\_exec($ch); //执行CURL
32. curl\_close($ch);

#### 3.5.3 输出

采用命令行与用户交互，代码如下：

1. // 输出variable\_list内容的函数
2. function echoScanArray($array){
3. foreach ($array as $k => $v) {
4. echo $k.": ".$v[0]." ".$v[1]." ".$v[2]." [";
5. foreach ($v[3] as $vv){
6. echo $vv.", ";
7. }
8. echo "] ".$v[4]." [";
9. foreach ($v[5] as $vv){
10. echo $vv.", ";
11. }
12. echo "] ".$v[6]."\n";
13. }
14. }
16. // 用户与控制台的交互
17. **while** (**true**){
18. echo "[cmd]\n[1]输出扫描结果\n[2]输出变量流\n[3]退出系统\n";
19. $cmd = fgets(STDIN);
20. **if**($cmd[0] == '1'){
21. echo "\n".$count." result(s) found\n";
22. **if** ($count != 0){
23. **for**($i = 0; $i < $count; $i++){
24. **for**($j = 0; $j < 4; $j++){
25. echo $result[$i][$j];
26. **if** ($j < 3){
27. echo " -> ";
28. }
29. }
30. echo "\n";
31. }
32. }
33. }elseif ($cmd[0] == '2'){
34. //输出结果
35. echo "\nVariable Info:\n".count($variable\_list)."\n";
36. echoScanArray($variable\_list);
37. }elseif ($cmd[0] == '3'){
38. echo "\n欢迎下次使用，bye-bye！";
39. exit(0);
40. }
41. }

输出的示例如下：

