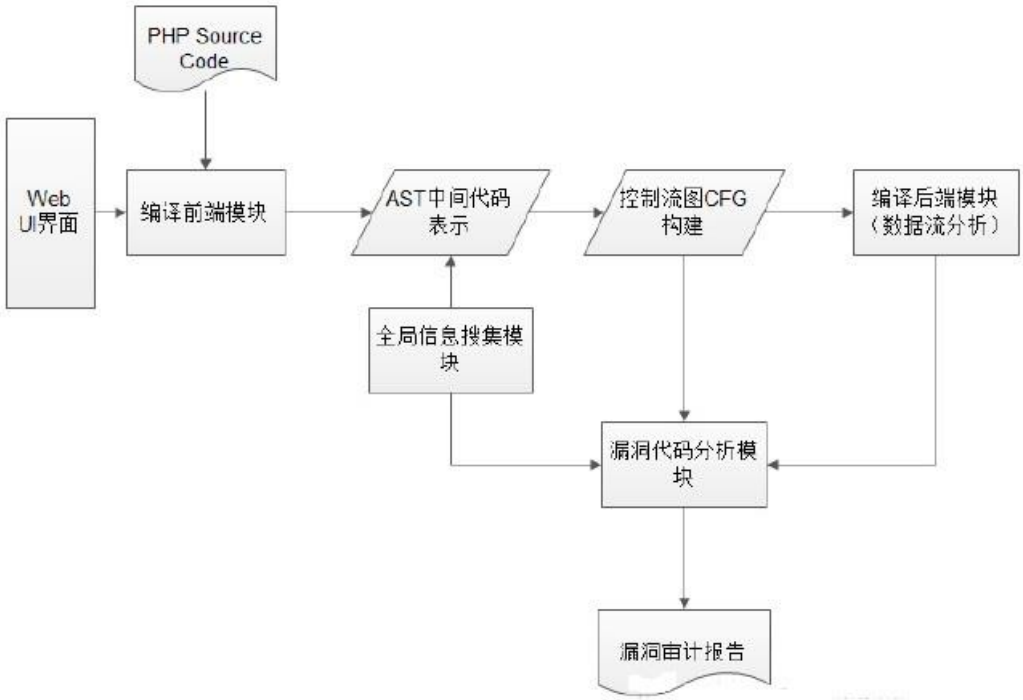
浅谈 PHP 自动化代码审计技术

目前市面上有不少 PHP 的自动 化审计工具，开源的有 RIPS、Pixy，商业版本的有 Fortify。RIPS 现在只有第一版，由于不支持 PHP 面向对象分析，所以现在来看效果不是太 理想。Pixy 是基于数据流分析的工具 ,但是只支持 PHP4。而 Fortify 是商业版本 ,由于这个限制 ,对它的研究也就无从谈起。国内对于 PHP 自动审 计的研究一般都是公司在做，目前有些工具大多数使用简单的 token 流分析或者直接粗暴一些，使用正则表达式来匹配，效果会很一般。

今天所要谈的技术是基于静态分析的一种 PHP 自动化审计的实现思路，也是我的项目中的思路。为了进行更加有效的变量根据和污点分析，以及很好的应对 PHP 脚本中的各种灵活的语法表示，正则表达式效果肯定是不理想的，我所介绍的思路是基于代码静态分析技术和数据流分析技术的审计。

首先，我认为一个有效审计工具至少包含如下的模块：



1、编译前端模块

编译前端模块主要运用编译技术中的抽象语法树构建、控制流图构建方法，将源码文件转为适合后端静态分析的形式。

2、全局信息搜集模块

该模块主要用于对分析的源码文件进行统一的信息搜集，比如搜集该审计工程中有多少类的定义，并对类中的方法名、参数、以及方法定义代码块的起始和终止的行号进行搜集，用于加快后续的静态分析的速度。

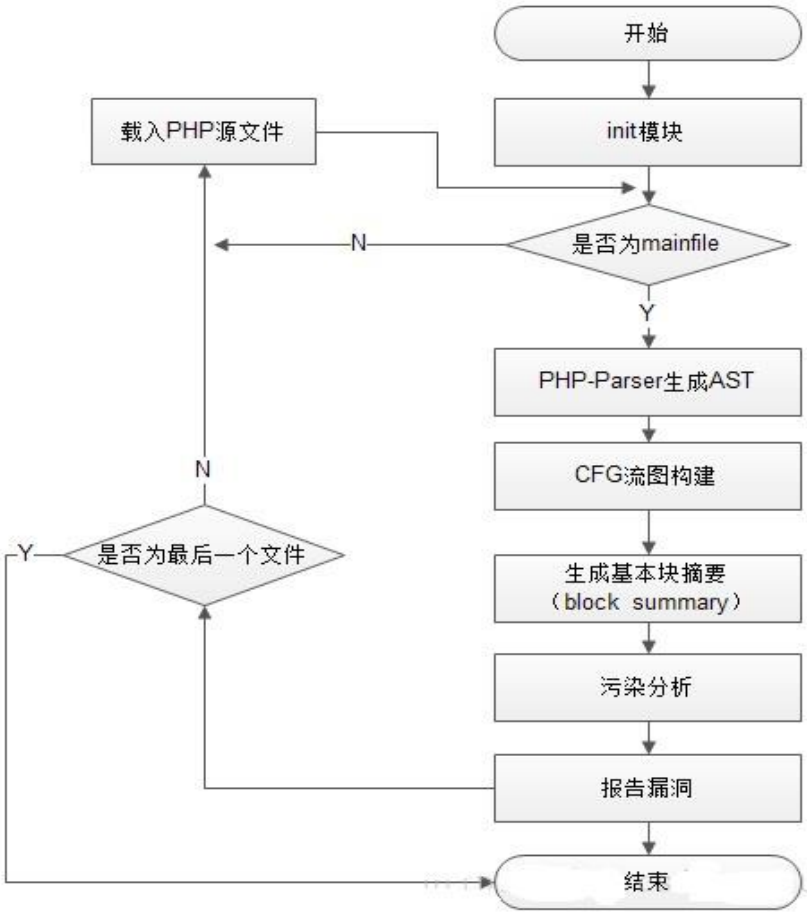
3、数据流分析模块

该模块不同于编译技术中的数据流分析算法，在项目中更注重对 PHP 语言本身特性的处理。当系统的过程间和过程内分析过程中发现了敏感函数的调用，则对该函数中敏感的参数进行数据流分析，即跟踪该变量的具体变化，为后续污点分析做准备。

4、漏洞代码分析模块

该模块基于数据流分析模块收集的全局变量、赋值语句等信息，进行污点数据分析。主要针对敏感 sink 中的危险参数，如 mysql_query 函数中的第一 个参数，经过回溯获取到相应的数据流信息，如果在回溯过程中发现该参数有用户控制的迹象，就进行记录。如果该危险参数有相应的编码、净化操作也要进行记 录。通过对危险参数的数据进行跟踪和分析，完成污点分析。

有了模块，那么如何进行有效的流程来实施自动化审计，我使用了如下的流程：



分析系统经过的大致流程如下：

1、框架初始化

首先进行分析框架的初始化工作，主要是搜集待分析源码工程中的所有用户自定义类的信息，包括类名，类属性，类方法名，类所在的文件路径。

这些Record存放在全局上下文类Context中，该类使用单例模式进行设计，并且常驻内存，便于后续的分析使用。

2、判断 Main File

其次判断每个PHP文件是否是Main file。在PHP语言中，没有所谓的main函数，大部分 Web 中的 PHP 文件分为调用和定义两种类型，定义类型的 PHP 文件是用来定义一些业务类、工 具类、工具函数等，不提供给用户进行访问，而是提供给调用类型的 PHP 文件进行调用。而真正处理用户请求的则是调用类

型的 PHP 文件，比如全局 index.php 文件。静态分析主要是针对处理用户请求的调用类型的 PHP 文件，即 Main File。判断依据为：

在 AST 解析完成的基础上，判断一个 PHP 文件中的类定义、方法定义的代码行数占该文件所有代码行数是否超过一个范围，如果是，则视为定义类型的 PHP 文件，否则为 Main File，添加到待分析的文件名列表中。

3、AST 抽象语法树的构建

本项目基于 PHP 语言本身进行开发，对于其 AST 的构建，我们参考目前比较优秀的 PHP AST 构建的实现————PHP Parser。

该开源项目基于 PHP 语言本身进行开发，可以对 PHP 的大多数结构如 if、while、switch、数组声明、方法调用、全局变量等语法结构进行解析。可以很好的完成本项目的编译前端处理的一部分工作。

4、CFG 流图构建

使用 CFGGenerator 类中的 CFGBuilder 方法。方法定义如下：

```
/**
 * 由AST节点创建相应的CFG，用于后续分析
 *
 * @param $nodes  传入的PHP file的所有nodes
 * @param $condition   构建CFGNode时的跳转信息
 * @param $pEntryBlock   入口基本块
 * @param $pNextBlock   下一个基本块
 */
public function CFGBuilder($nodes,$condition,$pEntryBlock,$pNextBlock){
```

具体思路是采用递归构建 CFG。首先输入遍历 AST 获取的 nodes 集合，遍历中对集合中的元素（node）进行类型判断，如判断是否是分支、跳转、结束等语句，并按照 node 的类型进行 CFG 的构建。

这里对于分支语句、循环语句的跳转条件（conditions）要存储至 CFG 中的边（Edge）上，方便数据流分析。

5、数据流信息的收集

对于一段代码块，最有效的并且值得收集的信息是赋值语句、函数调用、常量（const define）、注册的变量（extract parse_str）。

赋值语句的作用就是为了后续进行变量跟踪，在实现中，我使用了一种结构来表示赋值的 value 以及 location。而其他的数据信息是基于 AST 来判别 和获取的。比如函数调用中，判断变量是否受到转义、编码等操作，或者调用的函数是否是 sink（如 mysql_query）。

6、变量净化、编码信息处理

```php
<?php
$id = $_GET['id'] ; //'id'=>_GET
$sql = "select * from $id" ;  //'sql' => "".$id
$clearsql = addslashes($sql) ; //'clearsql'=> func_call
mysql_query($clearsql) ;  //sink

?>
```

$clearsql = addslashes($sql) ;

赋值语句 ,当右边是过滤函数时(用户自定义过滤函数或者内置过滤函数），则调用函数的返回值被净化，即$clearsql 的净化标签加上 addslashes。

发现函数调用，判断函数名是否是配置文件中进行配置的安全函数。

如果是，则将净化标签添加至 location 的 symbol 中。

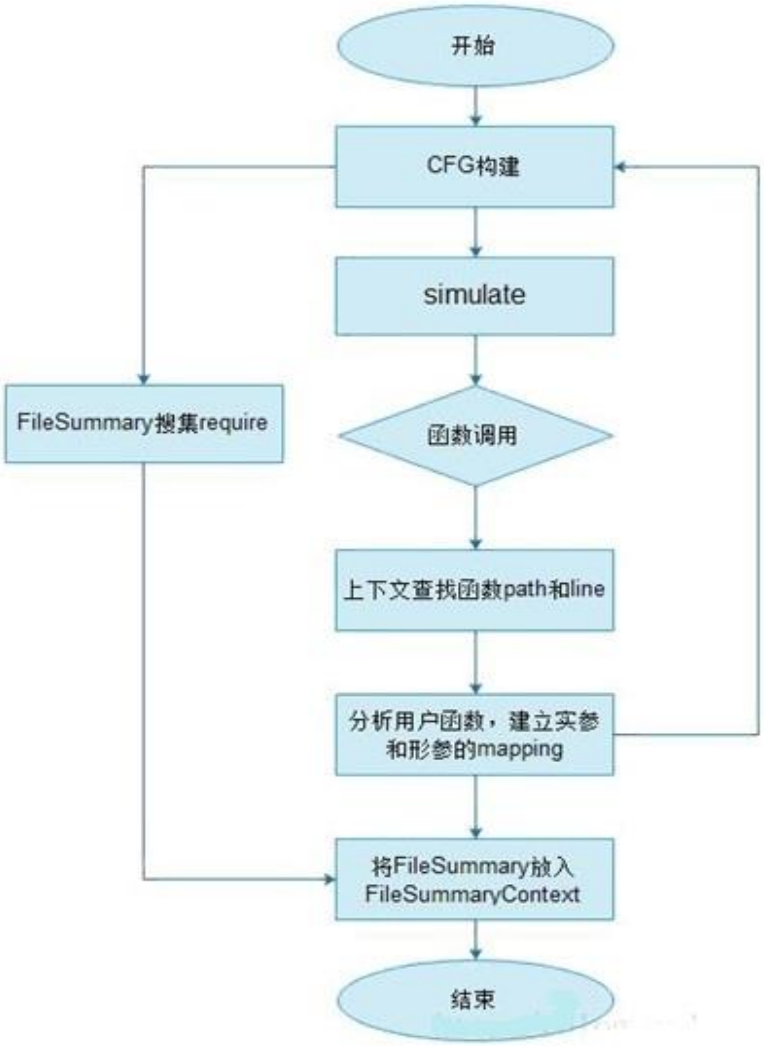
7、过程间分析

如果在审计中，发现用户函数的调用，这时候必须要进行过程间的分析，在分析的工程中定位到具体方法的代码块，带入变量进行分析。

难点在于，如何进行变量回溯、如何应对不同文件中的相同名称的方法、如何支持类方法的调用分析、如何保存用户自定义的 sink（比如在 myexec 中调

用 exec 函数，如果没有经过有效的净化，那么 myexec 也要视为危险函数）、如何对用户自定义的 sink 进行分类（如 SQLI XSS XPATH 等）。

处理流程如下：



8、污点分析

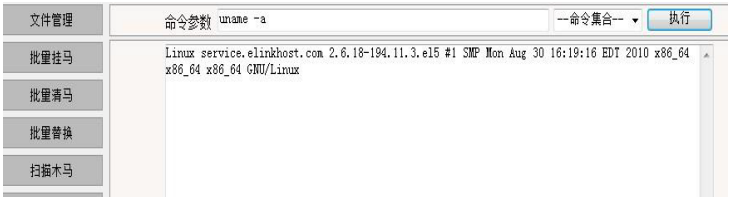
有了上面的过程，最后要进行的就是污点分析，主要针对系统中内置的一些风险函数，比如可能导致 xss 的 echo。并且要对危险函数中的危险参数做有 效的分析，这些分析包括判断是否进行了有效的净化（比如转义、正则匹配等），以及制定算法来回溯前面该变量的赋值或者其他变换。这无疑对安全研究人员工

程 能力的一个考验，也是自动化审计最重要的阶段。

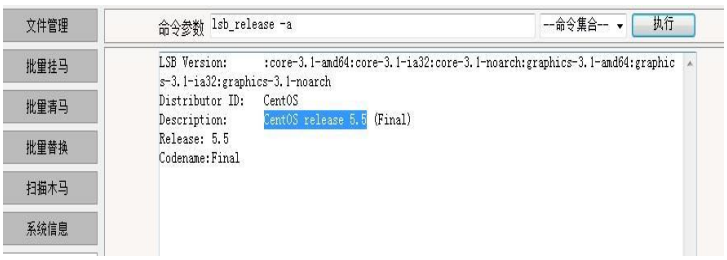
通过上面的介绍 ,你可以看到要实现一款自己的自动化审计工具所要趟的坑是很多的。我的尝试中也是遇到了 N 多的困难，并且静态分析确实带有一定的局限 性，比如动态分析中轻易可以获得的字符串变换的过程，在静态分析中就难以实现，这不是技术上能够突破的，而是静态分析本身的局限性导致的，所以单纯的静态 分析如果想要做到误报和漏报很低，毕竟引入一些动态的思想，比如对 eval 中的代码进行模拟，对字符串变化函数以及正则表达式进行处理等。还有就是对于一 些基于 MVC 框架的，比如 CI 框架，代码很分散，比如数据净化的代码放在 input 类的扩展中，像这种 PHP 应用，我认为很难做到一个通用的审计框架，应 该要单独对待。

以上只是粗略的把我当前的尝试（目前没有完全实现）拿来 share，毕竟大学狗不是专业人员，希望可以抛砖引玉，使得越来越多的安全研究人员关注这一领域。

首先先看下 linux 内核（uname -a）。

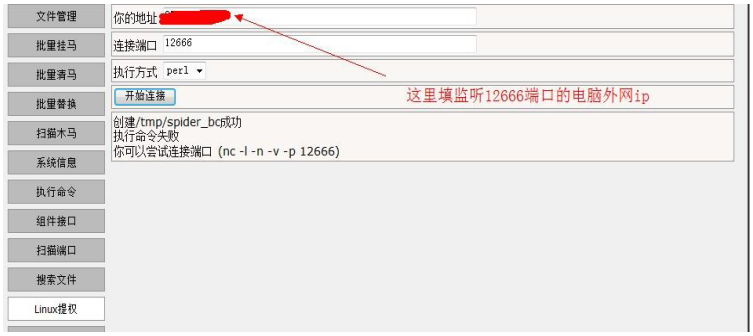


其次看下 linux 版本信息（lsb_release -a）。

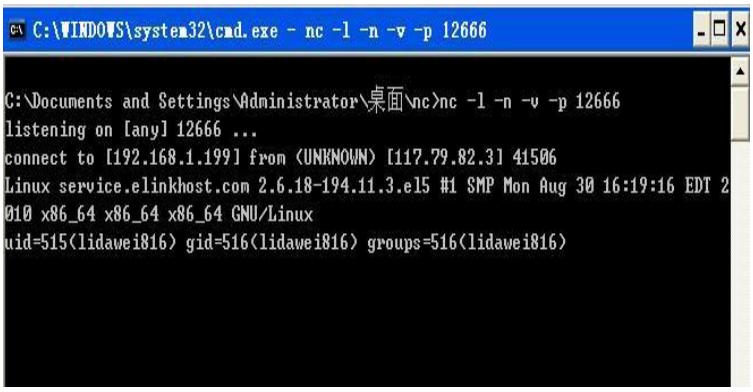


本地监听端口，上传特定的 exp 到/tmp 目录下（tmp 目录可写可执行）。

然后配置 webshell，以便将 shell 反弹到本机。



反弹成功，进入/tmp 目录编译 exp。



然后执行溢出（成功提权）。