МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ КАЗАХСТАН
Некоммерческое акционерное общество
«АЛМАТИНСКИЙ УНИВЕРСИТЕТ ЭНЕРГЕТИКИ И СВЯЗИ»
Кафедра систем информационной безопасности

«ДОПУЩЕН К ЗАЩИТЕ»
Зав. кафедрой к.п.н., доцент Р. Ш. Бердибаев
_____ «_____» _____2019 г.
(подпись)

# ДИПЛОМНЫЙ ПРОЕКТ

На тему: Разработка инструмента проверки на уязвимость кода программного обеспечения
Специальность 5В100200 - "Системы информационной безопасности"
Выполнил Раисов Мәди Мұратұлы Группа СИБ-15-3
Научный руководитель Дмитриева Маргарита Валерьевна

Консультант:
по экономической части:
к.э.н., профессор Арендаева Н.Г.
(ученая степень, звание, Ф.И.О)
_____ «17» мая 2019 г.
(подпись)

по безопасности жизнедеятельности:
д.т.н старший преподователь Бекбасаров М.М.
(ученая степень, звание, Ф.И.О)
_____ «17» мая. 2019 г.
(подпись)

по применению вычислительной техники:
ст. преподаватель Дмитриева М.В.
(ученая степень, звание, Ф.И.О)
_____ «07» 06 2019 г.
(подпись)

Нормоконтролер:
Ст. преподаватель, м.т.н. Аскарова А.Т
(ученая степень, звание, Ф.И.О)
_____ «03» 06 2019 г.
(подпись)

Рецензент:
КТН, ассиц-проф Ашаижанова С.Т
(ученая степень, звание, Ф.И.О)
_____ «06» 06 2019 г.
(подпись)

Алматы 2019

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ КАЗАХСТАН
Некоммерческое акционерное общество
«АЛМАТИНСКИЙ УНИВЕРСИТЕТ ЭНЕРГЕТИКИ И СВЯЗИ»

Институт систем управления и информационных технологий

Кафедра систем информационной безопасности

Специальность 5В100200 - “Системы информационной безопасности”

**ЗАДАНИЕ**
на выполнение дипломного проекта студенту Раисову М.М.

Тема проекта: Разработка инструмента проверки на уязвимость кода программного обеспечения

Утверждена приказом по университету № 124 от « 26 » 10 2019 г.

Срок сдачи законченного проекта « 07 » июня 2019 г.

Исходные данные к проекту (требуемые параметры результатов исследования (проектирования) и исходные данные объекта): проект подразумевает разработку инструмента проверки на уязвимость программного обеспечения на языке PHP посредством статического анализа.

Спроектировать логическую модель этого приложения таким образом, чтобы оно отвечало требованиям по простоте, удобству, потреблению ресурсов и времени отклика.

Продумать организацию базовых функций для работы с файлом, настройки самой структуры.

Проанализировать средства для реализации данного продукта. Проработать вопрос взаимодействия с пользователем.

Реализовать программу, в соответствии с моделью, полученной в процессе проектирования. Описать методы ее реализации, и решение возникающих по ходу вопросов.

Перечень вопросов, подлежащих разработке в дипломном проекте, или краткое содержание дипломного проекта: дипломный проект включает в себя 5 глав, разделенных на подглавы, каждая из которых освещает определенную тематику.

В первой главе дипломного проекта представлена общая теоритическая информация о безопасности в веб приложениях.

Во второй главе дипломного проекта представлена концепция статического анализа

В третей главе подробно описывается разработка статического анализатора проверки на уязвимость программного обеспеченя на языке php.

В четвертой главе приводится технико-экономическое обоснование проекта.

В пятой главе рассматриваются необходимые условия для комфортной разработки программного обеспечения.

Перечень графического материала (с точным указанием обязательных чертежей):

1) блок-схема программного обеспечения;
2) скриншоты модулей программы;
3) скриншоты с результатами работы программы;
4) скриншоты исходного кода программы;
5) скриншоты с примером эксплуатации бота;
6) рисунки общих схем.

Основная рекомендуемая литература:

1) Кузнецов М., Симдянов И. Самоучитель PHP 5/6. - 3-е изд., перераб. и доп. - СПб.: «БХВ-Петербург», 2009. - С. 672.

2) Костарев А. Ф. PHP 5. - СПб.: «БХВ-Петербург», 2008. - С. 1104.

Конструкции по проекту с указанием относящихся к ним разделов проекта

| Раздел | Консультант | Сроки | Подпись |
|---|---|---|---|
| | | | |
| Технико-экономическая часть | Арембаева Ж.Г. | 04.03-17.05.19 | |
| Безопасность жизнедеятельности | Бекбасаров Ш.Ш. | 04.03-17.05.19 | |
| Статический анализ исх. кода | Дмитриева М.В. | 26.01-07.06.19 | |
| Реализация концепции статического анализа | Дмитриева М.В. | 21.02-07.06.19 | |
| | | | |
| | | | |
| | | | |
| | | | |

График
подготовки дипломного проекта

| Наименование разделов, перечень разрабатываемых вопросов | Сроки представления научному руководителю | Примечание |
|---|---|---|
| Из. Концепции статического анализа исходного кода. | 26.11.2018 – 10.12.2018 | |
| Твор. безопасность веб-приложений | 10.12.2018 – 27.12.2018 | |
| Анализ потенциально уязвимых функций | 27.12.2018 – 10.01.2019 | |
| Внутрипроцедурный и межпроцедурный анализ. | 10.01.2019 – 27.02.2019 | |
| Реализация статического анализатора на основе лексера | 27.02.2019 – 27.03.2019 | |
| Построение модели исходного кода на основе лексера. | 27.03.2019 – 11.04.2019 | |
| Лексико – семантический анализ. | 11.04.2019 – 25.04.2019 | |
| Анализ потока управления в исходном коде | 25.04.2019 – 11.05.2019 | |
| Анализ массива токенов веб-интерфейс. | 11.05.2019 – 27.05.2019. | |
| | | |
| | | |
| | | |
| | | |
| | | |

Дата выдачи задания «29» октября 2018 г.

Заведующий кафедрой _____ (Бердибаев Р. Ш.)
(Подпись) (Ф.И.О)

Научный руководитель проекта _____ (Дмитриева Ю. В.)
(Подпись) (Ф.И.О)

Задание принял к исполнению студент _____ (Рамиев М. М.)
(Подпись) (Ф.И.О)

## АННОТАЦИЯ

В данном дипломном проекте представлен инструмент, который может сократить время, необходимое тестеру на проникновение, за счет автоматизации процесса выявления потенциальных уязвимостей в исходном коде PHP с помощью статического анализа исходного кода. Находки могут быть легко проверены тестером на проникновение в его контексте без повторного просмотра всего исходного кода. Учитывая ограничения статического анализа исходного кода, уязвимость должна быть подтверждена рецензентом кода.

## АҢДАТПА

Бұл нұсқада статистикалық бастапқы кодты талдау арқылы PHP бастапқы кодында әлеуетті қауіпсіздік кемшіліктерін анықтау процесін автоматтандыру арқылы ену тестерлерін қажет ететін уақытты қысқартатын құрал енгізіледі. Содан кейін табыстарды бүкіл мәтіннің бастапқы кодын қайта қарап шықпай, контекстен ену тестерімен оңай қарап шығуға болады. Статикалық кодты талдаудың шектеулерін ескере отырып, осалдықты код тексеруші растауы керек.

## ANNOTATION

In this diploma thesis a tool is introduced that can reduce the time a penetration tester needs by automating the process of identifying potential security flaws in PHP source code by using static source code analysis. The finds can then be easily reviewed by the penetration tester in its context without reviewing the whole source code again. Given the limitations of static source code analysis a vulnerability needs to be confirmed by the code reviewer.

## Содержание

Введение ..... 6
1 Статический анализ исходного кода ..... 8
1.1 Безопасность веб-приложений ..... 8
1.2 Статический анализ исходного кода ..... 14
1.2.1 Анализ потенциально уязвимых функций ..... 15
1.2.2 Внутрипроцедурный и межпроцедурный анализ ..... 16
1.3 Обработка результатов анализа ..... 17
2 Реализация концепции статического анализа на основе лексера ..... 18
2.1 Обзор программы ..... 18
2.2 Построение модели исходного кода для последующего анализа ..... 20
2.2.1 Лексико-семантический анализ ..... 20
2.2.2 Разбор исходного кода ..... 21
2.2.3 Анализ потока управления в исходном коде ..... 22
2.3 Анализ массива токенов ..... 22
2.4 Веб-интерфейс ..... 25
3 Технико-экономическое обоснование ..... 36
3.1 Расчет трудоемкости разработки программного продукта ..... 36
3.2 Расчет материальных затрат на разработку программного продукта ..... 37
3.3 Расчет затрат на электроэнергию ..... 38
3.4 Расчет затрат на оплату труда ..... 39
3.5 Расчет затрат по социальному налогу ..... 40
3.6 Амортизация основных фондов и прочие затраты ..... 41
3.7 Определение возможной (договорной) цены программного продукта ..... 42
4 Безопасность жизнедеятельности ..... 44
4.1 Анализ условий труда ..... 44
4.1.1 Расчет естественного освещения ..... 46
4.1.2 Расчет искусственного освещения ..... 47
4.2 Расчет воздухообмена в рабочем помещении ..... 48
Заключение ..... 54
Список литературы ..... 55

## Введение

Проблема безопасности программ является одной из первостепенных в области информационной безопасности (ИБ), так как наличие уязвимостей в программных ресурсах информационных систем обуславливает возможность реализации промышленных компьютерных атак и вирусных эпидемий, а также причину разного рода непреднамеренных отказов и потери ресурсов. С учетом динамичности и сложности программ и развития технологий информационного противоборства решение указанной проблемы требует непрерывного совершенствования методов контроля, тестирования и испытаний, а именно: статического и динамического анализа, функционального тестирования, экспертиз бюллетеней безопасности, сканирования уязвимостей, антивирусного контроляидр.Синтез и комплексирование указанных подходов подразумевают систематизацию базовых понятий безопасности программного обеспечения (ПО). Но, несмотря на то что подобные работы регулярно проводятся, в практике ИБ остается различное толкование понятия уязвимости ПО, в частности,наиболее часто смешивают определения ошибок безопасного программирования и известных уязвимостей сетевых сервисов, недекларированных возможностей и программных закладок, скрытых каналов и скрытых криптографических каналов и т. п. В результате возникает не только путаница со средствами выявления уязвимостей и базами уязвимостей, но и с целями и результатами оценки соответствия.

Высокая сложность современного программного обеспечения (ПО), обусловленная большим объемом (до нескольких миллионов строк) исходного кода, интегрирование методов обфускации (запутывания), деградация производительности процессов, наличие дефектов (уязвимостей и ошибок) является фундаментальной проблемой, вызванной текущим состоянием развития информационных технологий [1, 2]. Уязвимости ПО – критические ошибки, не выявленные в ходе тестирования и не декларированные спецификацией разработчика или заложенные преднамеренно, предоставляющие злоумышленникам исключительные возможности по разглашению информации, ее модификации, блокированию использования и безостаточному уничтожению без возможности восстановления. Возможность изменения основных свойств защищенности (доступность, целостность, конфиденциальность) информационных ресурсов и дестабилизации процессов функционирования информационно-вычислительных систем различного назначения посредством применения злоумышленниками несанкционированных воздействий деструктивного характера (атак) на уязвимости ПО предопределяют острую потребность в своевременном обнаружении дефектов (уязвимостей и ошибок) на этапах разработки и проектирования ПО, проверки соответствия их заявленной политики безопасности и реализации механизмов защиты является актуальным из-за того, что все системы анализа не гарантируют полного нахождения

имеющихся в ПО уязвимостей. Используемые методы данного подхода позволяют существенно затруднить атакующему успешную реализацию своих целей, исключить возможность эксплуатации даже известной уязвимости и сформировать универсальный метод взлома конкретной программы.

PHP - самый популярный скриптовый язык во всемирной сети сегодня. Этот язык является одним из самых легких в освоении языков, и даже программист с очень ограниченными навыками программирования может создавать сложные веб-приложения в короткие сроки. Но очень часто безопасность является второстепенным или не реализован вообще, подвергая риску весь веб-сервер. Широкое распространение PHP и многочисленные уязвимости, связанные с PHP, приводят к высокой заинтересованности в поиске защиты. уязвимости в исходном коде PHP быстро[1].

В этой работе представлена концепция уязвимостей веб-приложений и как они могут быть обнаружены статическим анализом исходного кода автоматически.

# 1 Статический анализ исходного кода

## 1.1 Безопасность веб-приложений

Количество сайтов быстро увеличилось за последние годы. Хотя сайты состояли в основном из статических файлов HTML за последнее десятилетие, все больше и больше веб-приложений с динамическим контентом появились в результате легкого изучения языков, таких как PHP и растущей доступности и скорости интернета. Почти все веб-серверы поддерживают скриптовые среды для развертывания динамических веб-приложений. Помимо огромного количества новых возможностей, новый веб 2.0 также привносит много новых рисков для безопасности, когда данные, предоставляемые пользователем, обрабатываются приложением недостаточно деликатно. Различные типы уязвимостей могут привести к утечке данных, изменению или даже взлому сервера. Часто один единственный неправильный символ может оказать огромное влияние на безопасность. Из-за частых случаев ограниченных навыков программирования, недостаточная осведомленности о безопасности и временных ограничений могут поставить под угрозу весь веб-сервер.

Чтобы сдерживать риски уязвимых веб-приложений, исходный код должен быть проанализирован разработчиком или тестом на проникновения. Учитывая тот факт, что большие приложения могут иметь тысячи строк кода и ограниченными времязатратами, ручная проверка исходного кода может быть недостаточной. Этот интсрумент может помочь пен-тестерам минимизировать затраты времени за счет автоматизации процессов, требующих значительных затрат времени при просмотре исходного кода.

Цель работы это создание инструмента, который автоматизирует процесс поиска уязвимостей безопасности, может помочь лучше защитить исходный код, найти потенциально уязвимые места в PHP приложениях.

Веб-приложение - это компьютерное приложение, которое развернуто на веб-сервере и доступно через веб-интерфейс клиента. HTTP-сервер (например, Apache или Microsoft IIS) принимает HTTP-запросы, отправленные клиентом (обычно веб-браузером, таким как Mozilla) Firefox или Microsoft Internet Explorer) и возвращает ответ HTTP с результатом запрос.

Когда клиент отправляет HTTP-запрос, он анализируется HTTP-сервером (шаг 1 на рисунке 1.1).HTTP-сервер извлекает запрашиваемое имя файла и параметры отправки (для пример GET или POST параметров). Когда он обнаруживает запрос на динамический файл (например, сценарий PHP) он выбирает исходный код из файловой системы (шаг 2 на рисунке 1.1) и передает это вместе с параметрами для интерпретатора языка (шаг 3 на рисунке 1.1). Интерпретатор выполняет исходный код и обрабатывает внешние ресурсы, такие как базы данных (шаг 4 и 5 на рисунке 1.1) или файлы. Затем он создает результат (обычно файл HTML) и отправляет его обратно клиент, где результат отображается в веб-браузере.

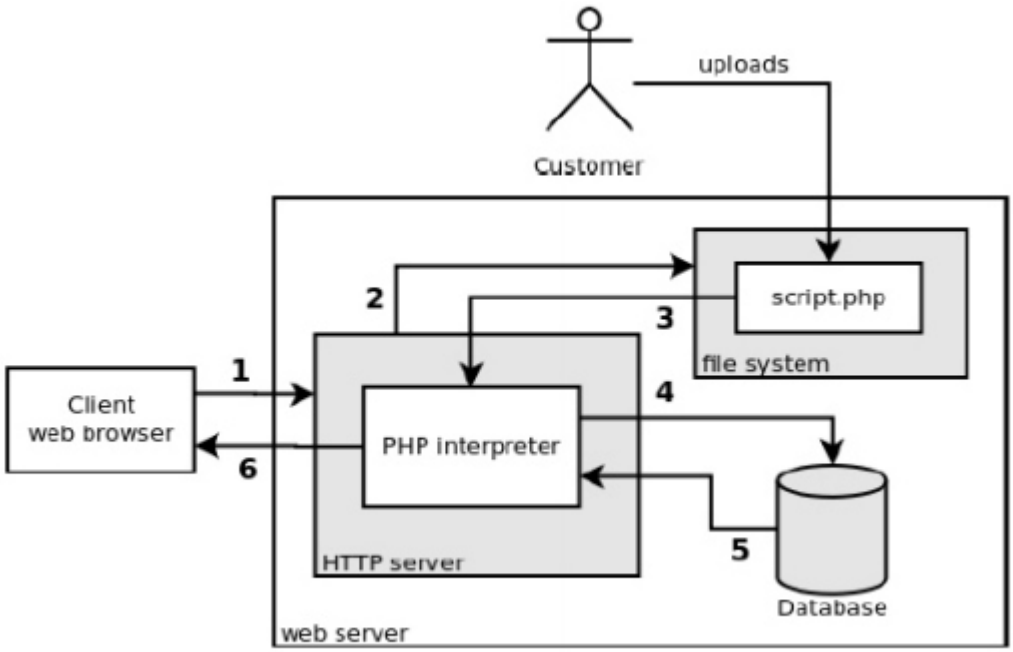


Рисунок 1.1 – Схема клиент-сервера

Уязвимость в безопасности веб-приложения может возникать, когда данные предоставляются пользователем (например, GET или параметры POST) не очищается правильно и используется в критических операциях динамического скрипт. Тогда злоумышленник сможет внедрить код, который изменяет поведение и результат операция при выполнении скрипта неожиданным образом. Такого рода уязвимости так называемые уязвимости в стиле taint, поскольку обрабатываются ненадежные источники, такие как предоставленные пользователем данные как испорченные данные, попадающие в уязвимые части программы, называемые чувствительными приемниками(sensitive sinks) .

Любые данные, которые могут быть изменены пользователем, такие как параметры GET или POST так же как и файлов cookie, пользовательский агент или даже записи или файлы базы данных должны рассматриваться как испорченные. Для каждого отдельного типа уязвимости существуют разные чувствительные приемники и процедуры очистки.

Чувствительные приемники – это потенциально уязвимые функции (potentially vulnerable functions), которые выполняют критические операции и должны вызываться только с проверенными или очищенными данными. В противном случае злоумышленник может повлиять на данные которые передаются в PVF и читать, изменять и удалять данные или атаковать базовый веб-сервер или клиент, в зависимости от PVF. На рисунке 1.2 показана концепция уязвимостей в стиле taint в PHP[1].

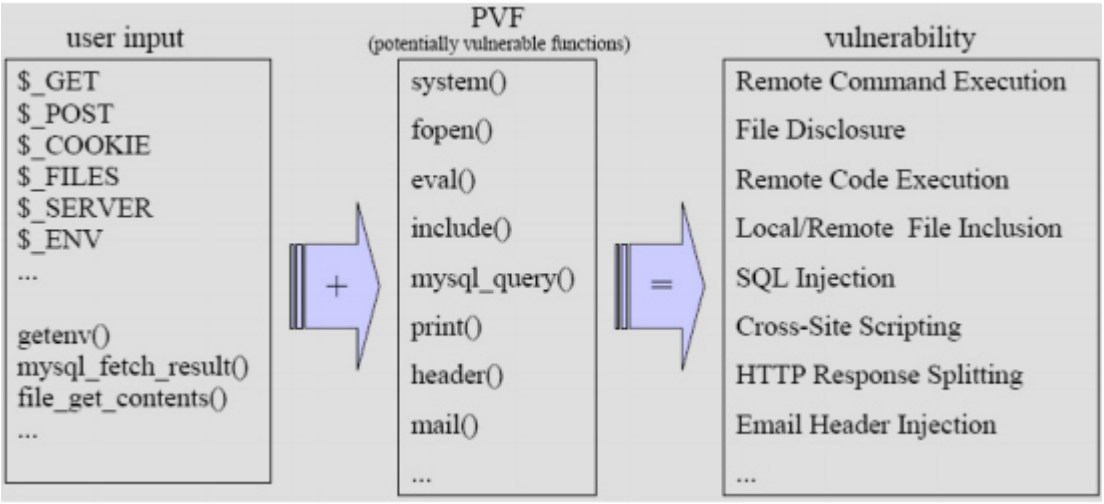


Рисунок 1.2 – Потенциально уязвимые фунции

Чтобы привести пример уязвимости, которые затрагивают клиент, и уязвимости, которые затрагивают сервер существуют две очень распространенные уязвимости: межсайтовый скриптинг и SQL-инъекции.

Межсайтовый скриптинг (XSS) атакует клиент веб-приложения и происходит, когда пользовательский ввод внедряется в результат HTML небезопасно. Злоумышленник может злоупотребить этим поведением, внедрив вредоносный код HTML или Javascript для локального искажения результатов веб-приложений или для других атак. Встраивая вредоносный Javascript, можно получить доступ к куки-файлам клиентского браузера и отправить их злоумышленнику, чтобы украсть сессионную информацию[7].

```
<?php
    print('<h1>Welcome ' . $_GET['name'] . '</h1>');
?>
```

Рисунок 1.3 – Небезопасный вывод

В листинге 3.1 показана типичная уязвимость XSS. Пользователь может указать имя с помощью параметра GET и имя встроено в страницу результатов HTML. Вместо выбора буквенно-цифрового имени он также может выбрать имя, состоящее из HTML и кода Javascript:

http://www.example.com/?name=foo</h1><script>alert(1)</script>

Код будет встроен в вывод и отправлен обратно в браузер клиента, который будет анализировать HTML и выполнит код Javascript. Злоумышленник может злоупотребить этим, отправив эту измененную ссылку его внедренном файле в параметре GET к жертве, которая выполнит код Javascript. Созданный результат HTML: Welcome foo < s c r i p t > a l e r t ( 1 ) < / s c r i p t >

Чтобы исправить эту уязвимость, выходные данные должны быть проверены. Символы, такие как <and >, а также кавычки могут быть заменены их HTML-сущностями. Таким образом, символы будут отображаться браузером, но не отображается как теги HTML. В PHP встроенная функция

htmlentities () и htmlspecialchars () могут быть использованы для проверки вывода.

Тонкости языка PHP.

PHP - самый быстрорастущий и самый популярный язык сценариев для веб-приложений. Это очень динамичный язык со множеством сложной семантики, которая часто используется современными веб-приложениями[1]. В этом разделе мы представим наиболее важные языковые функции, которые наш инструмент должен точно моделировать, чтобы правильно идентифицировать поток испорченных данных в чувствительные приемники. В частности, поток испорченных строк представляет интерес для уязвимостей в стиле taint.

1) Динамическая и слабая типизация: PHP представляет собой язык с динамической типизацией и не требует явного объявления переменных. Тип переменной выводится при первом присваивании во время выполнения. Кроме того, PHP является слабо типизированным языком, и его переменные не привязаны к конкретному типу данных. Таким образом, типы данных могут быть смешаны с другими типами данных во время выполнения. Строковый тест оценивается как 0, чтобы соответствовать математической операции, и добавляется к 1. Целочисленный результат сохраняется в переменной $ var2, предыдущий тип данных которой был string:

*1 $var1 = 1; $var2 = 'test';*

*2 $var2 = $var1 + $var2;*

2) Переменные: переменные обычно вводятся с символом доллара, за которым следует буквенно-цифровое, чувствительное к регистру имя. Однако в PHP имя также может быть выражением, например, извлеченным из другой переменной или возвращаемым значением вызова функции, который известен только во время выполнения. Это крайне затрудняет статический анализ языка PHP:

*1 $name = "x"; $x = "test";*

*2 echo $$name; // test 3 $y = ${getVar()};*

3) Динамические массивы. Массивы - это хеш-таблицы, которые отображают числа или строки (называемые ключами) в значения. Имя ключа может быть опущено при инициализации массива и сгенерировано во время выполнения. Кроме того, ключи и значения могут быть динамическими, как и само имя массива. При выполнении статического анализа сложно точно моделировать такую структуру динамического массива и динамический доступ к ней:

*1 $var = 6;*

*2 $arr = array('a', "4" => $var, 'foo' => 'c', 'd');*

*3 $arr[] = 'e';*

*4 // Array ([0] => a [4] => 6 [foo] => c [5] => d [6] => e)*

*5 print $arr[$var];*

4) Динамические константы: в PHP можно определить постоянные скалярные значения, как и в других языках программирования, таких как C.

Однако имя константы может быть динамически определено встроенной функцией define () и динамически доступно встроенной постоянная функции (). Хотя константа может не измениться после ее определения, можно определить константы условно в потоке программы или динамически сгенерировать с помощью пользовательского ввода.

5) Динамические функции: несколько функций с одинаковым именем могут быть определены разработчиком условно. Таким образом, может быть вызвана совершенно другая функция в зависимости от потока программы. Также возможно определить функцию B () в другой функции A (), которая присутствует только во время выполнения A (). Кроме того, встроенные функции func_get_arg () и func_get_args () позволяют динамически извлекать аргументы вызова функции по индексу:

*1 $name = 'step' . (int)$_GET['id'];*

*2 $name();*

*3 array_walk($arr = array(1), $name);*

6) Динамический код: оператор eval и встроенная функция assert () позволяют напрямую оценивать код PHP, который передается в виде строки его первому аргументу. Другие функции, такие как preg_replace (), позволяют выполнять динамический код PHP при использовании с некоторыми модификаторами. Динамически сгенерированный код очень сложно проанализировать, если исполняемый код PHP известен только во время выполнения и не может быть реконструирован статически. Кроме того, он вводит критические уязвимости безопасности.

7) Динамический include: код больших проектов PHP часто делится на несколько файлов и каталогов. Во время выполнения код можно объединить и выполнить условно. Оператор PHP include открывает указанный файл, оценивает его код PHP и возвращает код после оператора включения. Его можно использовать как выражение в любом другом выражении. Кроме того, имя файла включения может быть построено динамически, что означает, что его сложно статически реконструировать в сложных приложениях. Во время статического анализа очень важно разрешить все включения файлов, чтобы правильно проанализировать код PHP. Кроме того, испорченные данные в имени файла приводят к уязвимости включения файлов.

8) Встроенные функции: в зависимости от конфигурации и версии, PHP поставляется с несколькими расширениями, которые предоставляют разработчикам встроенные функции. Всего задокументировано 228 расширений с 5 701 встроенными функциями [9]. Статический анализ кода теряет точность всякий раз, когда вызывается встроенная функция, которая не настроена (или неисправна) в инструменте. Существует множество встроенных функций, которые должны распознаваться как чувствительные приемники или источники испорченных данных:

1 list($day, $month, $year) = $_GET['time'];

2 printf("Today is %d %s %d", $day, $month, $year);

9) Суперглобальные переменные: суперглобальные переменные - это встроенные массивы, которые инициализируются из интерпретатора PHP и доступны во всех областях. Они обеспечивают быстрый доступ к заголовку HTTP-запроса, среде и глобальной области видимости, которые могут содержать испорченные пользовательские данные. Часто разработчики рассматривают только значения $ _GET, $ _POST, $ _COOKIE и $ _REQUEST в качестве источника, но забывают о ключах $ _FILES и $ _SERVER. В частности, $ _SERVER ключи PHP_SELF и HTTP_HOST предполагаются как статические значения, но могут быть изменены злоумышленником[2].

Веб-приложения часто подключаются к внешним ресурсам, таким как базы данных, для обработки больших наборов данных. Для запроса базы данных используется структурированный язык запросов (SQL). Если веб-приложение создает запрос SQL во время выполнения с небезопасным пользовательским вводом, который злоумышленник может внедрить в собственный SQL синтаксис для изменения запроса SQL. Эта уязвимость называется SQL-инъекцией.

```
<?php
$id =  $_POST['id'];
$query = "SELECT id,title,content FROM news WHERE id = $id";
mysql_query($query);
?>
```

Рисунок 1.4 – Пример SQL-инъекции

Пример показывает уязвимость SQL-инъекций. Пользователь может указать идентификатор с помощью параметра POST, который встроен безанизированным в запрос SQL, и отправить его в базу данных. База данных ответит новостной статьей из таблицы новостей, соответствующей указанному идентификатору[8]. Злоумышленник может ввести собственный синтаксис SQL, чтобы изменить результат запроса и получить конфиденциальную информацию:

id=1+UNION+SELECT+1,username,password+FROM+users

В зависимости от системы управления базой данных, конфигурации и привилегий даже атаки на веб-сервер могут сопровождаться злоупотреблением функциями SQL. Чтобы исправить эту уязвимость, пользовательский ввод необходимо очистить перед его внедрением в запрос. Ожидаемые целочисленные значения должны быть приведены к типу int, чтобы гарантировать невозможность внедрения синтаксиса SQL. Ожидаемые строки обычно заключаются в кавычки как в рисунке 1.5.

```
<?php
$name =  mysql_real_escape_string($_POST['name']);
$query = "SELECT id,message FROM users WHERE name = '$name' ";
mysql_query($query);
?>
```

Рисунок 1.5 – SQL запрос

В этом случае достаточно избежать разрыва этих кавычек путем экранирования символа кавычки, чтобы он обрабатывался как обычный символ, а не как синтаксис SQL. В PHP встроенная функция mysql_real_escape_string () может использоваться для экранирования строк перед помещением их в запрос MySQL, как показано в листинге 5.

Существует много других уязвимостей в стиле “taint”, таких как File Disclosure и File Manipulation, которые позволяют злоумышленнику читать и записывать произвольные файлы. Уязвимости включения файлов позволяют злоумышленнику включать произвольные файлы PHP и выполнять код PHP. Необычные уязвимости, такие как удаленное выполнение кода или удаленное выполнение команд, даже позволяют злоумышленнику выполнять произвольные системные команды на веб-сервере. Однако все уязвимости в «грязном стиле» основаны на том же принципе, что и инъекции XSS и SQL: PVF вызывается с ненадежными и неанизированными данными, что позволяет злонамеренным пользователям изменять поведение и действия этого PVF в своих интересах. В зависимости от уязвимости, они могут быть встроенными функциями PHP, которые могут предотвратить этот тип уязвимости[3].

### 1.2 Статический анализ исходного кода

Во время статического анализа исходного кода исходный код компьютерной программы анализируется без его выполнения. Выполнение программы заставит ее работать по одному пути, в зависимости от ввода во время этого выполнения, и, следовательно, инструменты, которые анализируют программу во время ее выполнения (динамический анализ), могут проверять только те части программы, которые были задействованы во время этого конкретного запуска , Статический анализ позволяет сразу исследовать все различные пути выполнения в программе и делать оценки, применимые к каждой возможной перестановке входных данных. Статический анализ широко используется для различных целей, таких как выделение синтаксиса, проверка типов, оптимизация, а также поиск ошибок и безопасности. В общем, любой инструмент, который проверяет исходный код программы, не выполняя его, может быть классифицирован как инструмент статического анализа. [4] Существует четыре основных этапа статического анализа исходного кода:

1)конфигурация;

2)построение модели;

3)анализ ;

4)обработка результатов.

На этапе анализа настроенные знания безопасности могут использоваться для обнаружения уязвимостей.

Сначала инструмент должен иметь точный набор правил, чтобы знать, что искать. Для анализа уязвимости необходимо указать, какие типы уязвимостей следует обнаруживать и как можно определить чувствительные приемники. Также должны быть объявлены источники (пользовательский

ввод), которые могут испортить данные. Наконец, необходимо настроить и обнаружить действия по обеспечению безопасности, которые могут быть предприняты разработчиком, чтобы предотвратить получение защищенных частей программы в результате анализа.

Инструмент статического анализа должен преобразовывать исходный код в программную модель. Это абстрактное внутреннее представление исходного кода. Качество анализа инструмента во многом зависит от качества программной модели инструмента, поэтому очень важно, чтобы инструмент анализа хорошо понимал семантику языка. Построение программной модели требует следующих шагов:

Лексический анализ - инструмент должен разбивать исходный код на токены, чтобы правильно идентифицировать языковые конструкции. Незначительные токены, такие как пробелы и комментарии, обычно удаляются, чтобы правильно идентифицировать подключенные токены

Семантический анализ - инструмент анализа проверяет представление каждого токена. В примере важно выбрать между вызовом функции print и тем же словом, которое используется в строке. Также могут быть определены типы переменных.

Анализ потока управления - все возможные пути, которые можно пройти через программу, определены. Это включает случайные прыжки и вызовы функций. Затем пути объединяются в несколько графов потоков управления (CFG), которые представляют все возможные пути потоков данных.

Анализ потока данных - инструмент использует анализ потока данных для каждого CFG, чтобы определить, как данные перемещаются по всей программе. В анализаторе уязвимостей безопасности анализ заражений используется для определения места возникновения уязвимостей. Анализ потока данных основан на результатах семантического анализа и анализа потока управления, и поэтому важно, чтобы предыдущие результаты были максимально точными[10].

### 1.2.1 Анализ потенциально уязвимых функций

Когда модель построена, инструмент может выполнить анализ заражения на каждом найденном PVF. Он также должен выполнять внутрипроцедурный анализ для анализа вызова PVF в отдельной функции и межпроцедурный анализ для выполнения анализа посредством взаимодействия между несколькими функциями.

Во время анализа заражения определяется, где испорченные данные достигают чувствительных приемников, и, следовательно, может существовать недостаток безопасности. Инструмент должен различать нечистые чувствительные приемники и незапятнанные чувствительные приемники на этапе анализа потока данных. На рисунке 1.6 и рисунке 1.7 показаны два сценария PHP, которые используют систему PVF (), которая выполняет системные команды .

```
1   <?php
2       $a = $_GET['a'];
3       $b = $a;
4       system($b, $ret);
5   ?>
```

Рисунок 1.6 - уязвимость удаленного выполнения команды

```
1   <?php
2       $a = $_GET['a'];
3       $b = 'date';
4       system($b, $ret);
5   ?>
```

Рисунок 1.7- Безопасное выполнение статических команд

Хотя в рисунке 1.6 показана уязвимость удаленного выполнения команд, когда пользователь может указать любую команду, которая будет выполняться с помощью параметра GET a, рисунок 1.7 не является уязвимостью, поскольку выполняемая команда является статической и не может подвергаться влиянию злоумышленника. Анализируя поток данных параметра PVF, инструмент может определить, попадают ли ненадежные данные в приемник или вызывается PVF с безвредными статическими данными[5].

1.2.2 Внутрипроцедурный и межпроцедурный анализ

Во время анализа заражения инструмент также должен выполнять внутрипроцедурный анализ. Этот анализ включает в себя отслеживание данных в определенной пользователем вызванной функции. Если в объявлении функции обнаружен чувствительный приемник, инструмент должен решить, при каких обстоятельствах эта функция должна быть вызвана, чтобы вызвать уязвимость в чувствительном приемнике. Это в основном тот случай, когда чувствительный приемник зависит от параметров пользовательских функций, которые должны вызываться с испорченными данными. Инструмент также должен иметь возможность определить, не испортили ли испорченные данные функцию или ее очистили, выполнив эту функцию. Также функция может возвращать испорченные данные, хотя они не были вызваны с испорченными данными (например, когда новый пользовательский ввод принят во время выполнения функции). Межпроцедурный анализ - это понимание контекста вне функции во время вызова. В зависимости от состояния программы вызов может вести себя по-разному или внешние переменные в глобальной области видимости могут быть изменены. Поскольку при статическом анализе исходного кода предполагается, что все возможные данные CFG могут проходить, может быть очень трудно выполнить оба анализа без чрезмерного увеличения времени анализа.

### 1.3 Обработка результатов анализа

Последняя важная часть статического анализа исходного кода состоит в том, чтобы представить результаты пользователю таким образом, чтобы он мог быстро обнаружить критические недостатки. Отображение только соответствующих частей уязвимого кода, а также выделение синтаксиса помогает проанализировать и понять проблему. Также полезно предоставить автоматизированную информацию, как этот недостаток может быть исправлен. Если беспроблемный участок кода ненадлежащим образом помечен как уязвимый, мы говорим о ложном срабатывании или ложной тревоге. Если инструмент не может определить уязвимость, когда на самом деле она есть, мы говорим о ложном отрицании. И наоборот, правильно идентифицированная уязвимость известна как истинно положительная, в то время как соответствующее отсутствие предупреждений в защищенном разделе кода называется истинно отрицательной. Ложные срабатывания обычно рассматриваются как навязчивые и нежелательные и могут привести к отказу от инструмента. Ложные негативы, возможно, еще хуже, потому что они могут дать пользователю ложное чувство безопасности. [6] Чем лучше этот инструмент, тем больше истинных положительных результатов можно определить и тем меньше будет отображено ложных положительных результатов. Статические инструменты анализа исходного кода часто генерируют много ложных срабатываний и предупреждений. Это может произойти из-за неправильного семантического или потокового анализа или обстоятельств, которые необходимы для правильной идентификации уязвимости, но которые не могли быть оценены инструментом правильно. Поэтому важно представлять результаты таким образом, чтобы пользователь мог легко выбирать между правильным обнаружением уязвимости или самим ложным срабатыванием.

## 2 Реализация концепции статического анализа на основе лексера

В этой главе описывается, как теоретические концепции статического анализа исходного кода реализованы в данном дипломном проекте. Анализатор написан на PHP, а результатом является HTML-файл с управлением, написанным на Javascript, предлагающий несколько вариантов просмотра деталей результата. Разработанная программа не требует каких-либо условий, кроме локального веб-сервера с установленным интерпретатором PHP и веб-браузером.

В результате проектирования программы была выполнена программная реализация статического анализа исходного кода на языке PHP.

Программная реализация состоит из двух основных модулей выполненных на языке php:

Первый модуль – токенизатор(лексер),который разделяет программный код на токены(метки) которые компонуются в массив с идентификатором токена, который в свою очередь можно превратить в имя токена, вызвав значение токена и номер строки для полседующего анализа.

Второй модуль – анализатор , функция которого анализировать список токенов каждого файла, проходя по ним и определяя важные токены по имени. Для каждого отсканированного файла он создает стек зависимостей, стек файлов, список объявленных переменных и несколько регистров, которые указывают, сканирует ли он в функции, классе или любой другой языковой структуре.

### 2.1 Обзор программы

Изначально при сканировании каталогов расширения файлов настроены на анализ только файлов PHP. Чтобы идентифицировать потенциально уязвимые функции(PVF), в файлах исходного кода, создается огромный список, которые классифицируются по типам уязвимостей, чтобы пользователь мог сканировать только определенную уязвимость. Помимо имени функции также настраиваются значимые параметры. Это предотвращает ложные срабатывания, когда испорченные данные в первом параметре приводят к уязвимости, но испорченные данные во втором параметре не являются критическими. Кроме того, это повышает производительность, поскольку не связанные параметры не отслеживаются. Универсальный параметр 0 может быть выбран, когда все параметры потенциально уязвимы и, следовательно, значимы. Это важно для функций, которые могут иметь переменное количество параметров, таких как print (). В программе имеется 186 PVF, разделенных на следующие типы уязвимостей:

- межсайтовый скриптинг (5);
- SQL-инъекция (54);
- раскрытие файла (37);
- манипулирование файлами (20);
- включение файлов (7);

- удаленное выполнение кода (17);
- удаленное выполнение команд (8);
- подключение обработки (28);
- XPath-инъекция (3);
- другое (7).

Также для каждого PVF настраивается список функций защиты встроенных PHP. Это важно для предотвращения ложных срабатываний, когда разработчик уже предпринимает меры безопасности. Настроенная запись PVF реализована в виде простого списка массивов. В рисунке 2.8 показан пример записи PVF для функции system () со значимым параметром (the first) и функциями защиты

(escapeshellarg () и escapeshellcmd ()).

```
"system" => array(
    array(1), array("escapeshellarg", "escapeshellcmd")
);
```

Рисунок 2.8 – Запись потенциально уязвимых функций

Кроме того, определяется глобальный список защитных функций, содержащий функции, которые будут предотвращать эксплуатацию каждого PVF. Обычно это функции, которые возвращают только целые числа, такие как strlen () или очищенные строки, такие как md5 (), которые будут препятствовать внедрению кода. Для каждого типа уязвимости также имеется краткое описание, пример кода, пример исправления и пример использования, настроенные для того, чтобы помочь пользователю понять обнаруженную уязвимость. Все источники заражения (пользовательский ввод) должны быть настроены для выявления уязвимости во время обратной трассировки параметров PVF. Пользовательский ввод обычно передается через параметры GET, POST и Cookie, а также через загруженные имена файлов и переменные среды сервера, такие как пользовательский агент или строка запроса. В PHP глобальные переменные $ _GET, $ _POST, $ _COOKIE и $ _FILES, а также переменные $ _SERVER и $ ENV обрабатывают эти данные [2]. Также испорченный пользовательский ввод может поступать из файлов или баз данных, когда пользователь записывает или вставляет их ранее в этот внешний ресурс. Для этого случая также настраиваются функции, которые читают из файлов или баз данных. Для сбора информации объявлен список интересных функций (41), которые будут создавать примечание в результате сканирования каждый раз, когда обнаруживается вызов функции из этого списка. Примерами являются session_start (), который указывает на управление сеансом, или mysql_connect (), который указывает на использование системы управления базами данных (СУБД) MySQL.

## 2.2 Построение модели исходного кода для последующего анализа

2.2.1 Лексико-семантический анализ

Чтобы правильно проанализировать PHP-скрипт, код разбит на токены модулем токенизатора. Для этого используется функция PHP token_get_all () список меток на рисунке 2.11 [5]. Каждый токен представляет собой массив с идентификатором токена, который можно превратить в имя токена, вызвав функцию buildin token_name () [6], значение токена и номер строки. Отдельные символы, представляющие семантические коды, отображаются в виде строки в списке токенов.На рисунке 2.10 показан пример кода PHP, а на рисунке 2.11 представлен его репрезентативный список токенов, сгенерированный функцией token_get_all ().

```
1   <?php
2       $a = $_GET['a'];
3       system($a, $ret);
4   ?>
```

Рисунок 2.10 – Пример вывода

```
name: T_OPEN_TAG                    value: <?php     line: 1
name: T_VARIABLE                    value: $a        line: 2
name: T_WHITESPACE                  value:           line: 2
                                          =
name: T_WHITESPACE                  value:           line: 2
name: T_VARIABLE                    value: $_GET,    line: 2
                                           [
name: T_CONSTANT_ENCAPSED_STRING    value: 'a',      line: 2
                                           ]
                                           ;
name: T_WHITESPACE                  value:           line: 2
name: T_STRING                      value: system    line: 3
                                           (
name: T_VARIABLE                    value: $a        line: 3
                                           ,
name: T_WHITESPACE                  value:           line: 3
name: T_VARIABLE                    value: $ret      line: 3
                                           )
                                           ;
name: T_WHITESPACE                  value:           line: 3
name: T_CLOSE_TAG                   value: ?>        line: 4
```

Рисунок 2.11.Список токенов сгенерированный token_get_all ()

Как только список токенов PHP-скрипта получен, было сделано несколько улучшений для правильного анализа токенов. Это включает в себя замену некоторых специальных символов именами функций (например, `$ a` на backticks ($ a), которые представляют выполнение команды [7]) или добавление фигурных скобок в конструкции потока программы, где никакие скобки не использовались (например, if или switch условия только с одной

условной линией, следующей [8]). Также все пробелы, встроенный HTML и комментарии удаляются из списка токенов, чтобы уменьшить накладные расходы и правильно идентифицировать подключенные токены. Затем исходный код можно проанализировать токеном по токену [9].

2.2.2 Разбор исходного кода

Цель разработанной программы - анализировать список токенов каждого файла только один раз, чтобы повысить скорость. Он проходит по списку токенов и определяет важные токены по имени. Для каждого отсканированного файла он создает стек зависимостей, стек файлов, список объявленных переменных и несколько регистров, которые указывают, сканирует ли он в функции, классе или любой другой языковой структуре. Несколько действий выполняются, когда идентифицирован один из следующих токенов:

T_INCLUDE Если будет найдено включение файла, токены включенного файла будут добавлены в текущий список токенов, а также дополнительный токен, который идентифицирует конец включенных токенов. Также есть примечание об успешном включении, добавляемом в вывод, если сбор информации включен. Если имя файла состоит из переменных и строк, имя файла может быть восстановлено динамически. Внутренний указатель файла отслеживает текущую позицию во включенных файлах. Также каждое включение файла проверяется на уязвимость включения файла.

T_FUNCTION Если объявлена новая функция, имя и параметры анализируются и сохраняются для дальнейшего анализа.

T_CLASS Если объявлен новый класс, имя сохраняется для дальнейшего анализа и создается новый список потенциально уязвимых функций класса. Этот список используется для сохранения уязвимых пользовательских функций в зависимости от их класса, поскольку несколько классов могут называть свои функции одинаковыми, что может привести к ложным срабатываниям.

T_RETURN Если пользовательская функция возвращает переменную, эта переменная будет отслеживаться в обратном направлении и проверяться на предмет безопасности действий. Если возвращаемая переменная очищается с помощью защитной или нейтрализующей функции, такой как md5 (), или с помощью защитного действия, такого как приведение типов, эта функция добавляется в глобальный список функций защиты, чтобы можно было определить определяемые пользователем функции очистки. Если возвращаемое значение испорчено пользовательским вводом, функция добавляется в список функций, которые могут испортить другие переменные при назначении им.

T_VARIABLE Если объявление переменной идентифицировано, текущая область проверяется, и объявление переменной добавляется либо в список локальных (если токен найден в объявлении функции), либо в список

глобальных переменных вместе с соответствующей строкой исходного кода. , В рисунке 2.12 показаны некоторые примеры объявлений переменных в PHP.

```
$a          =>  $a = $_GET['a'];
$b          =>  $b = '';
$b          =>  $b.=$a;
$c['name']  =>  $c['name'] = $b;
$d          =>  while($d = fopen($c['name'], 'r'))
```

Рисунок 2.12. Примеры для объявления переменных в PHP

Примеры показывают, что недостаточно только разобрать = and; в списке токенов, чтобы правильно идентифицировать каждое объявление переменной. RIPS использует этот список для отслеживания переменных, найденных в вызовах PVF, в обратном направлении к их источнику во время анализа заражения. Также все зависимости добавляются в каждое объявление переменной, чтобы сделать возможным отслеживание различных потоков программы. RIPS избегает использования потоковых графиков управления из-за соображений производительности, однако это может привести к ложным негативам в очень специфических сценариях.

Кроме того, токены, которые идентифицируют особенности PHP, такие как extract (), list () или define (), оцениваются для улучшения правильности результатов. Также определен список функций, которые идентифицируют использование СУБД или сеанса, и обнаруженные вызовы добавляются к выводу с комментарием, если для этого задан уровень детализации.

2.2.3 Анализ потока управления в исходном коде

Для анализа потока управления используются следующие токены:

Фигурные скобки {} Весь поток программы обнаруживается фигурными скобками, и поэтому токены должны быть подготовлены в ситуациях, когда программист не использовал фигурные скобки. Управляющие структуры, такие как if и switch, добавляются в текущий стек зависимостей. Если PVF обнаружен в том же блоке фигурных скобок, к этой находке будут добавлены текущие зависимости. Закрывающая скобка обозначает конец структуры управления, и последняя зависимость удаляется из стека зависимостей.

Метки T_EXIT, T_THROW могут привести к выходу из потока программы, и последняя найденная управляющая структура, от которой зависит выход (например, оператор if или switch), добавляется в текущий стек зависимостей. Если в объявлении функции найден выход, эта функция добавляется в список функций с пометкой о возможном выходе. Благодаря этому пользователь может получить представление о том, какие условия должны быть выполнены, чтобы перейти к нужному вызову PVF в потоке программы, не прерывая выполнение программы.

### 2.3 Анализ массива токенов

Фаза модуля анализа начинается каждый раз, когда обнаруживается вызов PVF во время анализа списка токенов. Это имеет то преимущество, что

список токенов нужно анализировать только один раз. Однако ранее построенная модель на этапе анализа должна быть завершена таким образом, чтобы анализ текущего PVF был правильным.

Вызов функции обнаруживается токеном T_STRING. Если обнаружен вызов функции, инструмент проверяет, находится ли имя функции в определенном списке PVF и, следовательно, вызов функции для дальнейшего сканирования. Создается новый родительский элемент, и все параметры, настроенные как ценные, будут отслеживаться в обратном порядке, просматривая имена переменных в списке глобальных или локальных переменных. Результаты добавляются в дерево PVF как дочерний элемент. Все переменные в ранее найденных объявлениях также будут найдены в списке переменных и добавлены к соответствующему родительскому элементу. Если во время анализа строки объявления переменной обнаруживаются защитные действия, дочерний элемент помечается красным. Если пользовательский ввод найден, дочерний элемент помечается белым, и к результату добавляется дерево PVF. При желании параметры могут быть помечены как испорченные, если они испорчены функциями, которые читают результаты SQL-запроса или содержимое файла. Следовательно, можно идентифицировать уязвимости с помощью постоянного хранилища полезных данных.

```
1   <?php
2       $a = $_GET['a'];
3       $b = $a;
4       system($b, $ret);
5   ?>
```

Рисунок 2.13 - Пример кода с PVF system()

В Рисунке 2.13 показан пример кода, который использует PVF _system (), которая выполняет системные команды [9]. Как только PVF обнаружен, следующим шагом является определение его сконфигурированных значимых параметров. Функция system () выполняет только первый параметр ($ b), в то время как второй обрабатывает только результат ($ ret). Поэтому только первый параметр $ b отслеживается в обратном направлении. Соответственно строка 3 находится в списке глобальных переменных, который присваивает переменную $ a переменной $ b. Снова $ a будет сравниваться с ранее объявленными переменными и так далее. Если параметр произошел из пользовательского ввода, вызов PVF рассматривается как потенциальная уязвимость. Затем дерево отслеживаемых параметров показывается пользователю, который может выбрать между реальной уязвимостью или ложным срабатыванием. Выходные данные 5.5 показаны на рисунке 2.14

```
4:    system($b, $ret);
    3:    $b = $a;
        2:    $a = $_GET['a'];
```

Рисунок 2.14 - Ложное срабатывание

2.3.2 Внутрипроцедурный и межпроцедурный анализ

Если отслеживаемая переменная PVF в объявлении пользовательской функции зависит от параметра этой функции, объявление добавляется как дочерний и помечается желтым. Затем эта пользовательская функция добавляется в список PVF с соответствующим списком параметров. Список функций защиты адаптирован из функций защиты, определенных для PVF, найденного в этой пользовательской функции. В конце все необходимые на данный момент зависимости в потоке программы добавляются.

```
1  <?php
2      function myexec($a, $b, $c)
3      {
4          exec($b);
5      }
6
7      $aa = "test";
8      $bb = $_GET['cmd'];
9      myexec($aa, $bb, $cc);
10 ?>
```

Рисунок 2.15 - PVF exec () вызывается в пользовательской функции

При обнаружении вызова PVF в строке 4 в рисунке 2.15 параметр $ b отслеживается в обратном направлении. Обнаружено, что $ b зависит от параметра функции объявления функции myexec (). Теперь функция myexec () добавлена в список PVF, второй параметр определен как ценный, а функции защиты определены для exec (). Пользовательская функция myexec () теперь обрабатывается как любая другая функция PVF рисунок 2.16. Если обнаружен звонок с пользовательским вводом, звонок и уязвимость добавляются к выводу. Этот вызов может происходить в другой пользовательской функции, так что межпроцедурный анализ возможен в вызовах связанных функций.

```
4:   exec($b);
     2:   function myexec($a, $b, $c)
```

Рисунок 2.16 - Оригинальный PVF

```
9:   myexec($aa, $bb, $cc);
     8:   $bb = $_GET['cmd'];
```

Рисунок 2.17 - Показан вызов функции, которая вызывает уязвимость

Кроме того, переменные, которые отслеживаются и объявляются в структуре кода, отличной от той, в которой был найден вызов PVF, будут прокомментированы зависимостью для объявления переменной. Зависимости, которые влияют на оба, остаются глобальной зависимостью для этого родителя рисунок 2.17.

Для правильного внутрипроцедурного анализа необходимо учитывать также глобальные переменные. Они обнаруживаются путем идентификации переменных T_VARIABLE и предыдущего токена T_GLOBAL. Ключевое слово global объявляет переменную, которая будет использоваться в глобальной, а не в области видимости локальной переменной [8]. Все переменные, объявленные в функции как глобальные, отслеживаются в списке локальных переменных функции и в списке глобальных переменных. Также все изменения в переменной, объявленной как глобальная в функции, остаются после ее возврата. Поэтому изменения сохраняются в дополнительном списке переменных, который извлекается в глобальный список переменных после вызова функции.

### 2.4 Веб-интерфейс

Сканер может полностью контролироваться веб-интерфейсом. Чтобы начать сканирование, пользователь просто должен указать имя файла или каталога, выбрать тип уязвимости и нажать «Сканировать» рисунок 2.18. Программа будет сканировать только файлы с расширениями, которые были настроены. Кроме того, уровень детализации может быть выбран для улучшения результатов:

- уровень детализации по умолчанию 1 сканирует только те вызовы PVF, которые испорчены пользовательским вводом без каких-либо обнаруженных защитных действий в трассировке;
- второй уровень детализации также включает файлы и содержимое базы данных как потенциально злонамеренный ввод данных пользователем. Этот уровень важен для поиска уязвимостей с постоянным хранилищем полезной нагрузки, но он может увеличить уровень ложных срабатываний;
- третий уровень детализации также будет выводить защищенные вызовы PVF. Эта опция важна для обнаружения недостаточных средств защиты, которые трудно обнаружить при статическом анализе исходного кода;
- четвертый уровень детализации также показывает дополнительную информацию, собранную во время сканирования. Это включает в себя найденные выходы, заметки об успешном анализе включенных файлов и вызовы функций, которые были определены в массиве интересных функций. В больших PHP-приложениях такой сбор информации может привести к очень большому и нечеткому результату сканирования;
- последний уровень многословия 5 показывает все вызовы PVF и их следы, независимо от того, испорчены пользовательские данные или нет. Это может быть полезно в тех случаях, когда представляет интерес список статического ввода для вызовов PVF. Тем не менее, этот уровень многословия приведет к множеству ложных срабатываний.

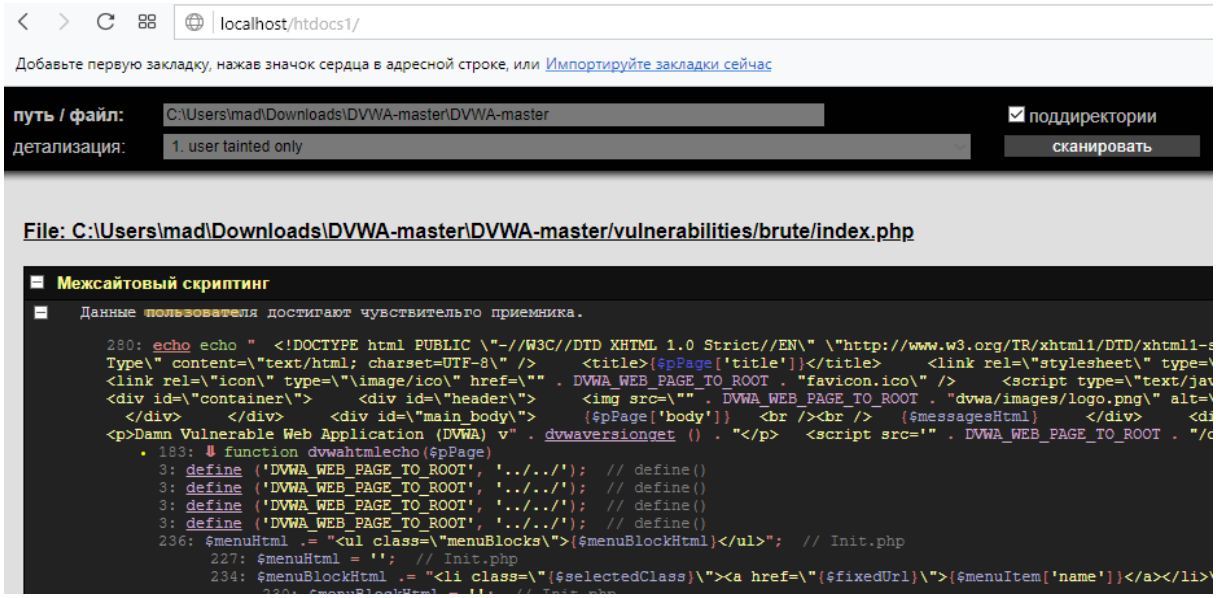


Рисунок 2.18 – Веб интерфейс

Все найденные вызовы PVF и их следы отображаются выделенным синтаксисом и разбиваются на блоки для пользователя. Пользователь может решить показать уязвимости в обратном направлении (как это было отсканировано из PVF в пользовательский ввод) или в прямом направлении (как разработчик мог бы прочитать). Подсветка синтаксиса кода PHP может быть изменена на лету, выбрав из 7 различных таблиц стилей. Цветовые схемы были адаптированы вручную из Pastie (pastie.org) и интегрированы в собственную подсветку синтаксиса RIPS. Для каждого имени токена атрибут CSS определяет цвет, чтобы можно было легко создавать собственные стили. Загрязненные данные особо выделяются, а также обнаруживаются защитные действия, предпринятые разработчиком.

Также можно открыть окно перетаскивания, чтобы увидеть исходный код, нажав на значок файла. Все строки, используемые в вызове PVF и его трассе, выделены красным цветом в исходном коде, и средство просмотра кода автоматически переходит к вызову PVF, что позволяет быстро и легко просмотреть трассировку. Он также предлагает возможность выделять определенные переменные одним щелчком, чтобы облегчить понимание потока кода.

Для дальнейшего изучения сложных уязвимостей пользователь может открыть список пользовательских функций, список точек входа в программу (пользовательский ввод) и список всех отсканированных файлов, что позволяет ему непосредственно переходить в код, нажимая на кнопку. элементы списка. Также все пользовательские функции, вызываемые в результате сканирования, могут быть проанализированы путем наведения курсора мыши на имя функции. Затем код объявления функции отображается в слое при наведении курсора рисунок 2.19. Также возможен переход между выводами в пользовательской функции и соответствующим вызовом этой функции.

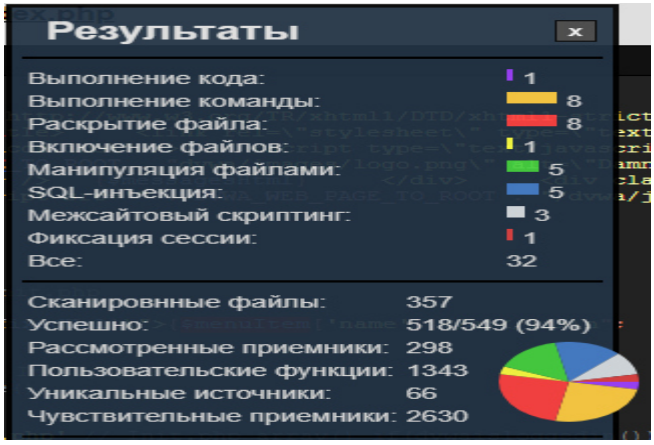


Рисунок 2.19 - Результаты

Типы уязвмости:

1) XSS.

Злоумышленник может выполнить произвольный код HTML / JavaScript в контексте браузера клиента с помощью этой уязвимости безопасности. Данные, испорченные пользователем, внедряются приложением в вывод HTML и обрабатываются браузером пользователя, что позволяет злоумышленнику встраивать и обрабатывать вредоносный код. Подготовка вредоносной ссылки приведет к выполнению этого вредоносного кода в контексте браузера другого пользователя при нажатии на ссылку. Это может привести к порче локального веб-сайта, фишингу или краже файлов cookie и краже сеанса рисуно 2.20. Рекомендации: Кодируйте все испорченные данные пользователя с помощью встроенных PHP-функций, прежде чем встраивать данные в вывод. Обязательно установите параметр ENT_QUOTES, чтобы избежать инъекций обработчика событий в существующие атрибуты HTML и укажите правильный набор символов .

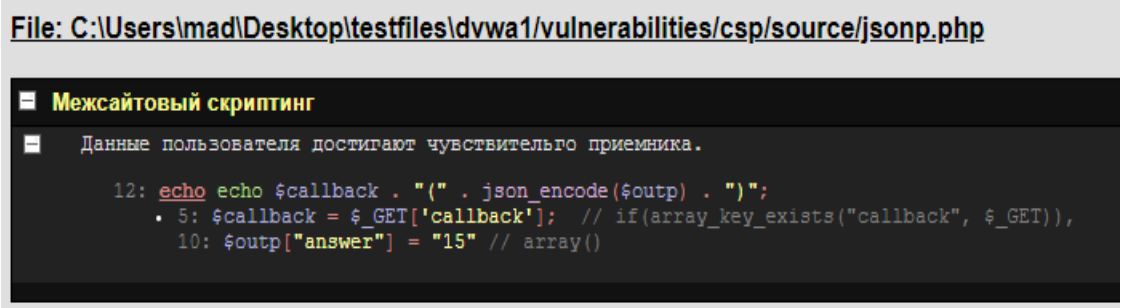


Рисунок 2.20 - Пример уязвимости

2) HTTP_HEADER

Злоумышленник может добавить произвольные заголовки в заголовок ответа HTTP. Это может быть использовано для перенаправления при добавлении заголовка «Location:» или помощи в атаке с фиксацией сеанса, когда добавлен заголовок «Set-Cookie:». Кроме того, ответ HTTP может быть перезаписан, а JavaScript может быть введен, что приводит к атакам на межсайтовый скриптинг. В версии PHP ниже 4.4.2 или 5.1.2 символы \ n \ r

(LF CR) могут использоваться для завершения строки заголовка (кросс-браузер). В PHP ниже 5.4 символ \ r (CR) все еще может использоваться для завершения строки заголовка (Chrome, IE). Рекомендации: Обновите PHP, чтобы предотвратить внедрение заголовка или реализовать белый список.

3)SESSION_FIXATION

Злоумышленник может заставить пользователя использовать определенный идентификатор сеанса. Когда пользователь входит в систему, злоумышленник может использовать ранее зафиксированный идентификатор сеанса для доступа к учетной записи. Рекомендации: Не используйте токен сеанса, предоставленный пользователем.

4)Code_Injection

Злоумышленник может выполнить произвольный код PHP с этой уязвимостью. Данные, испорченные пользователем, встроены в функцию, которая компилирует код PHP на ходу и выполняет его, позволяя злоумышленнику внедрить собственный код PHP, который будет выполняться. Эта уязвимость может привести к полной компрометации сервера. Рекомендации: Создайте белый список для положительного кода с помощью регулярных выражений (например, только буквенно-цифровых символов) или массивов. Не пытайтесь занести в черный список злой код PHP.

5) Reflection_injection

Злоумышленник может выполнить произвольные функции с этой уязвимостью. Данные, испорченные пользователем, используются в качестве имени функции. Это может привести к неожиданному поведению приложения. Рекомендации: Создайте белый список для разрешенных функций.

6) exploiting-php-file-inclusion

Злоумышленник может включить локальные или удаленные файлы PHP или прочитать файлы, не относящиеся к PHP, с этой уязвимостью. Данные, испорченные пользователем, используются при создании имени файла, которое будет включено в текущий файл. PHP-код в этом файле будет оценен, не-PHP-код будет встроен в вывод. Эта уязвимость может привести к выходу из строя сервера. Рекомендации: Создайте белый список для положительных имен файлов. Не ограничивайте только имя файла конкретными путями или расширениями.

7) FILE_READ

Злоумышленник может читать локальные файлы с этой уязвимостью. Данные, испорченные пользователем, используются при создании имени файла, который будет открыт и прочитан, что позволяет злоумышленнику читать исходный код и другие произвольные файлы на веб-сервере, что может привести к новым векторам атаки. Например, злоумышленник может обнаружить новые уязвимости в файлах исходного кода или прочитать учетные данные пользователя. Рекомендации: Создайте белый список для

положительных имен файлов. Не ограничивайте только имя файла конкретными путями или расширениями.

8) FILE_AFFECT

Злоумышленник может записать в произвольные файлы или внедрить произвольный код в файл с этой уязвимостью. Данные, испорченные пользователем, используются при создании имени файла, который будет открыт, или при создании строки, которая будет записана в файл. Злоумышленник может попытаться записать произвольный код PHP в файл PHP, что позволяет полностью скомпрометировать сервер. Рекомендации: Создайте белый список для положительных имен файлов. Не ограничивайте только имя файла конкретными путями или расширениями. Если вы пишете в файлы PHP, убедитесь, что злоумышленник не может написать собственный код PHP. Используйте белый список с массивами или регулярными выражениями (например, только буквенно-цифровой).

9) EXEC

Злоумышленник может выполнить произвольные системные команды с этой уязвимостью. Данные, испорченные пользователем, используются при создании команды, которая будет выполняться в базовой операционной системе рисунок 2.21. Эта уязвимость может привести к полной компрометации сервера. Рекомендации: Ограничьте код до очень строгого набора символов или создайте белый список разрешенных команд. Не пытайтесь фильтровать «злые команды». Старайтесь по возможности избегать использования функций, выполняющих системные команды.

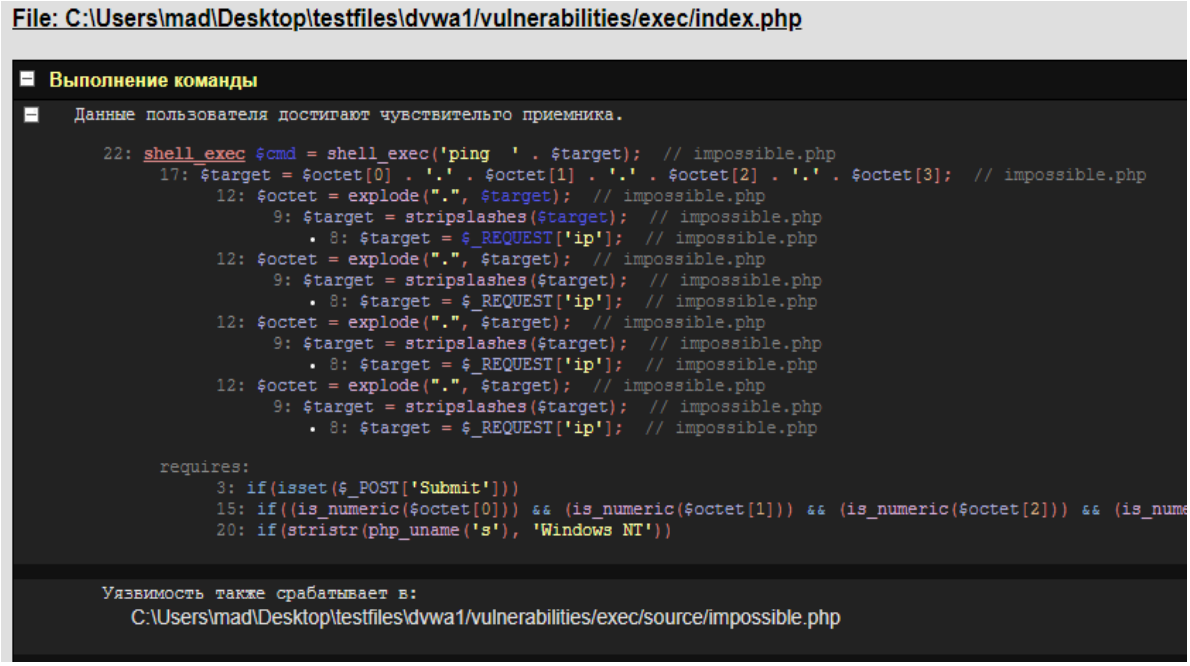


Рисунок 2.21 – Уязвимость доступа к командной строке

10) DATABASE

Злоумышленник может выполнить произвольные команды SQL на сервере базы данных с этой уязвимостью. Данные, испорченные

пользователем, используются при создании запроса к базе данных, который будет выполняться в системе управления базами данных (СУБД). Злоумышленник может внедрить собственный синтаксис SQL, таким образом, инициировать чтение, вставку или удаление записей базы данных или атаковать базовую операционную систему в зависимости от запроса, СУБД и конфигурации рисунок 2.22. Рекомендации: Всегда вставляйте ожидаемые строки в кавычки и экранируйте строку с помощью встроенной функции PHP, прежде чем встраивать ее в запрос. Всегда вставляйте ожидаемые целые числа без кавычек и вводите данные в целое число, прежде чем встраивать их в запрос. Экранирование данных, но встраивание их без кавычек небезопасно.

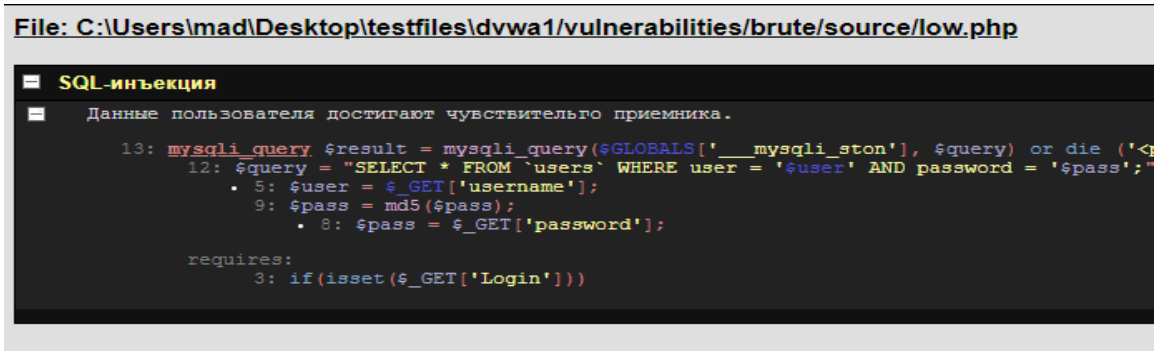


Рисунок 2.22 - Пример обнаруженной SQL-иньекции

11) XPATH

Злоумышленник может выполнить произвольные выражения XPath с помощью этой уязвимости. Данные, испорченные пользователем, используются при создании выражения XPath, которое будет выполняться на ресурсе XML. Злоумышленник может ввести собственный синтаксис XPath для чтения произвольных записей XML. Рекомендации: Всегда вставляйте ожидаемые строки в кавычки и экранируйте строку с помощью встроенной функции PHP, прежде чем встраивать ее в выражение. Всегда вставляйте ожидаемые целые числа без кавычек и вводите данные в целое число, прежде чем встраивать их в выражение. Экранирование данных, но встраивание их без кавычек небезопасно.

12) LDAP

Злоумышленник может выполнить произвольные выражения LDAP с этой уязвимостью. Данные, испорченные пользователем, используются при создании фильтра LDAP, который будет выполняться на сервере LDAP. Злоумышленник может внедрить собственный синтаксис LDAP для чтения произвольных записей LDAP. Рекомендации: Ожидаемые строки не вставляются в кавычки в LDAP. Ограничьте набор символов ввода буквенно-цифровым (если возможно), чтобы предотвратить внедрение синтаксиса фильтра.

13) CONNECT

Злоумышленник может изменить параметры обработки соединения или данные, передаваемые с помощью этой уязвимости. Данные, испорченные

пользователем, используются при выборе параметров или создании данных, которые будут переданы, что позволяет злоумышленнику изменить их. В зависимости от типа соединения это может привести к дальнейшим атакам. Рекомендации:Нет обобщенных рекомндаций.

14) POP

Когда userinput анализируется функцией unserialize (), злоумышленник может злоупотребить этим, предоставляя сериализованные объекты, которые будут использоваться в текущей области приложения. Эти объекты могут быть только экземплярами классов этого приложения. Некоторые гаджеты, такие как функции __wakeup () или __destruct () этих классов, будут автоматически вызываться при воскрешении объекта во время десериализации, и переменные объекта, указанные злоумышленником, могут привести к уязвимости в этих гаджетах.Не используйте unserialize, потому что он содержит гораздо больше недостатков.

Обнаружение уязвимостей в в критических операциях приложения.

Уязвимость в системе безопасности возникает, когда данные, предоставленные пользователем, используются в критических операциях приложения и недостаточно очищены. Злоумышленник может воспользоваться этим недостатком, внедрив вредоносный ввод, который изменяет поведение или результат этой операции. Эти виды уязвимостей называются уязвимостями в стиле «Taint», поскольку ненадежные источники, такие как предоставленные пользователем данные, считаются испорченными и буквально попадают в уязвимые части программы (называемые чувствительными приемниками).

Недавняя работа в этой области была сосредоточена на обнаружении только ограниченного числа типов уязвимостей, таких как уязвимости межсайтового скриптинга (XSS) и SQL-инъекция (SQLi) или анализа процедур очистки. Кроме того, существующие подходы обычно неточны в том смысле, что некоторые языковые функции, такие как встроенные функции очистки или манипулирования строками и контексты разметки, не смоделированы точно. В результате, некоторые типы уязвимостей и санитарии не могут быть обнаружены такими подходами. Например, Saner опирается на сгенерированные вручную тестовые случаи, что подразумевает, что он может обнаруживать только уязвимости, закодированные в инструменте. Кроме того, другие подходы, такие как предложенный Се и Айкеном, не моделируют встроенные функции и, таким образом, пропускают важные векторы атаки и защиты. Коммерческие инструменты, поддерживающие язык PHP, направлены на обнаружение уязвимостей в трех и более языках программирования. Следовательно, эти инструменты создают более общую модель и в них отсутствуют многие специфичные для PHP уязвимости и характеристики.

Более конкретно, мы выполняем внутри- и межпроцедурный анализ потока данных, чтобы создать сводные данные о потоке данных в приложении, чтобы очень эффективно обнаруживать уязвимости в стиле

«Taint». Мы выполняем контекстно-зависимый анализ строк, чтобы уточнить результаты анализа заражений на основе текущего контекста разметки, типа источника и конфигурации PHP. Обобщение нашего подхода к различным языкам возможно путем моделирования его (менее разнообразных) встроенных функций, в то время как алгоритмы анализа остаются теми же.

Итоговый обзор программы:

Для каждого PHP-файла в проекте создается абстрактное синтаксическое дерево (AST), основанное на внутренних компонентах PHP с открытым исходным кодом. Кроме того, все пользовательские функции извлекаются, и соответствующая информация, такая как имя и параметры, сохраняется в среде. Тело функции сохраняется как отдельный AST и удаляется из основного AST анализируемого файла.

Начинаем преобразовывать каждый основной AST в график потока управления (CFG). Всякий раз, когда узел AST выполняет условный переход, создается новый базовый блок, который соединяется с предыдущим базовым блоком с помощью края блока. Условие перехода добавляется к краю блока, а следующие узлы AST добавляются в новый базовый блок.

Моделируем поток данных каждого базового блока, как только создается новый базовый блок. Основным преимуществом является то, что анализ базового блока зависит только от предыдущих базовых блоков при выполнении обратного анализа потока данных. Кроме того, результаты анализа интегрированы в так называемую сводку блоков, которая создается во время моделирования. Он суммирует поток данных в базовом блоке.

Если во время симуляции встречается вызов ранее неизвестной пользовательской функции, CFG строится из функции AST и сводка функции создается один раз с внутрипроцедурным анализом. Затем предварительные и последующие условия для этой функции могут быть извлечены из сводки и выполнен межпроцедурный анализ. Наконец, строительство основного CFG продолжается.

Проводим анализ заражения, начиная с моделируемого в настоящее время базового блока для каждого уязвимого параметра пользовательской функции или настроенного чувствительного приемника.

Этот подход использует блочные, функциональные и файловые сводки для хранения результатов анализа потока данных в каждом модуле и для построения абстрактной модели потока данных для эффективного анализа. Точнее, предпринимаются следующие шаги:

1)Загрузка PHP файлов рисунок 2.23.


Рисунок 2.23 – путь к файлам

2)Токенизация кода (рисунок 2.24) и построение абстракного синтаксичесокого древа (AST).

```php
<?php

class Tokenizer
{
    public $filename;
    public $tokens;

    function __construct($filename)
    {
        $this->filename = $filename;
    }

    // main
    public function tokenize($code)
    {
        $this->tokens = token_get_all($code);
        $this->prepare_tokens();
        $this->array_reconstruct_tokens();
        $this->fix_tokens();
        $this->fix_ternary();
        #die(print_r($this->tokens));
        return $this->tokens;
    }

```

Рисунок 2.24 - Токенизация кода

3)Разделение АСТ на базовые блоки рисунок 2.25 релизация.

```php
function getBraceEnd($tokens, $i)
{
    $c=1;
    $newbraceopen = 1;
    while( !($newbraceopen === 0 || $tokens[$i + $c] === ';') )
    {
        if( $tokens[$i + $c] === '(' )
        {
            $newbraceopen++;
        }
        else if( $tokens[$i + $c] === ')' )
        {
            $newbraceopen--;
        }
        if($c>50)break;
        $c++;
    }
    return $c;
}

function get_ini_paths($path)
{
    if(!preg_match('/([;\\\\]|\W*[C-Z]{1}:)/', $path))
        $path = str_replace(':', ';', $path);
    return explode(';', $path);
}
```

Рисунок 2.25 - Разделение на базовые блоки

4)Подключение базовых блоков к графику потока управления (CFG).
5)Анализ птока данных через CFG рисунок 2.26.

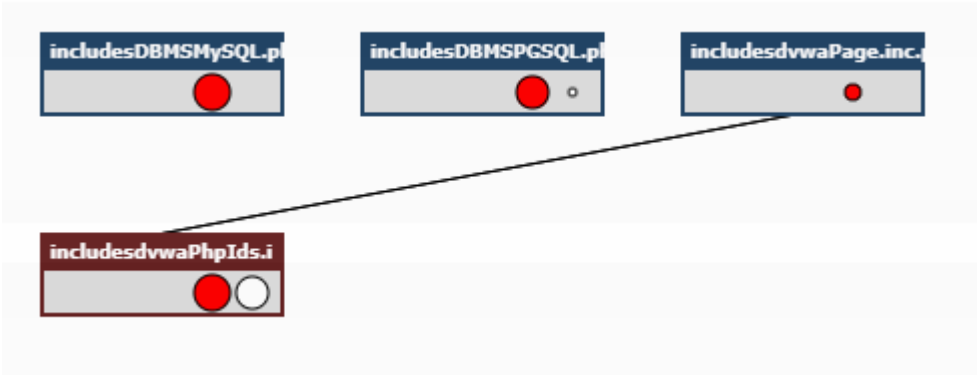


Рисунок 2.26 - Пример потока данных

6)Отчет, рисунок 2.27 - пример обнаруженных уязвимостей

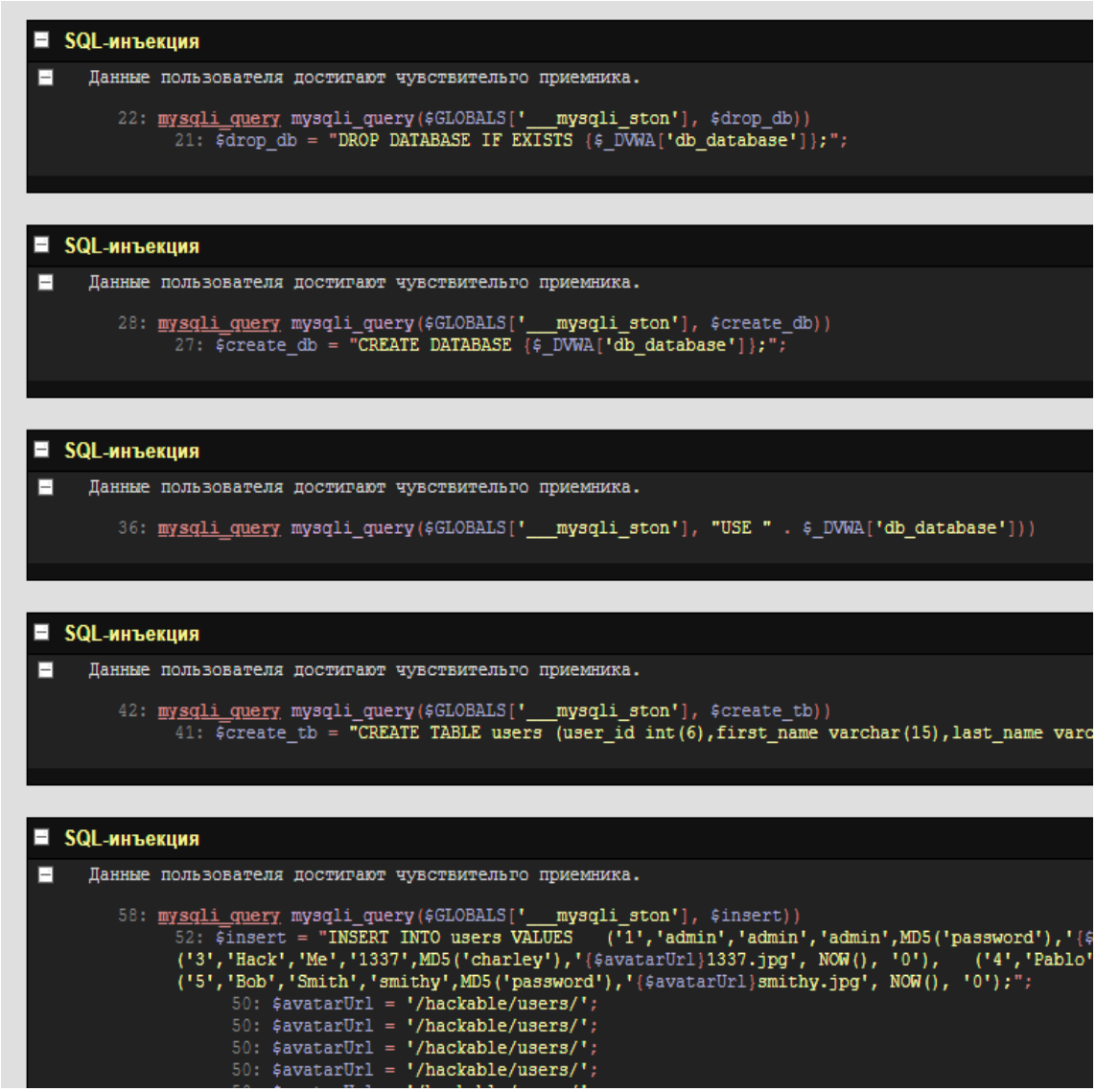

Рисунок 2.27 - Пример обнаруженных уязвимостей

**Вывод**

Путем токенизации и синтаксического анализа всех файлов исходного кода разработанная программа может преобразовывать исходный код PHP в программную модель и обнаруживать чувствительные приемники (потенциально уязвимые функции), которые могут быть испорчены пользовательским вводом (под влиянием злонамеренного пользователя) во время выполнения программы. Помимо структурированного вывода обнаруженных уязвимостей RIPS также предлагает интегрированную среду аудита кода для дальнейшего ручного анализа.

Чтобы сдерживать риски уязвимых программных обеспечений, исходный код должен быть проанализирован разработчиком или пентестером. Учитывая тот факт, что большие приложения могут иметь тысячи строк кода и ограниченными времязатратами, ручная проверка исходного кода может быть недостаточной. Этот интсрумент может помочь пентестерам минимизировать затраты времени за счет автоматизации процессов, требующих значительных затрат времени при просмотре исходного кода.

## 3 Технико-экономическое обоснование

Данный дипломный проект подразумевает разработку программного продукта для анализа исходного кода на языке php на уязвимости.

В экономической части дипломной работы будут рассчитаны все расходы на разработку программного продукта: затрат на материалы и устройства: затраты на программные обеспечения, затраты на электроэнергию и оплату труда, а также амортизацию основного оборудования[10].

### 3.1 Расчет трудоемкости разработки программного продукта

Приведен перечень основных этапов и работ, которые нужно выполнить для определения трудоемкости разработки программного обеспечения. Трудоемкость работы определялась согласно нормам времени на проведение расчетов, анализа и исследований. Форма разделения работ по этапам с указанием трудоемкости их выполнения приведена в таблице 3.1.

Таблица 3.1 - Распределение работ по этапам и оценка их трудоемкости

| Этапы разработки ПО | Вид работы | Трудоемкость, чел. час. |
|---|---|---|
| Этап 1 | Постановка задач | 12 |
| Этап 2 | Разработка и утверждение технического задания на разработку ПП | 27 |
| Этап 3 | Поиск и изучение подобных программ и устройств | 10 |
| Этап 4 | Оформление теоретической части темы дипломного проекта | 20 |
| Этап 5 | Разработка практической части дипломного проекта | 40 |
| Этап 6 | Выбор среды разработки программного обеспечения | 12 |
| Этап 7 | Реализация проекта | 38 |
| Этап 8 | Отладка программного обеспечения | 20 |
| Этап 9 | Оформление отчета и выводов | 17 |
| Этап 10 | Тестирование проекта | 22 |
| Итого: трудоемкость выполнения дипломного проекта | | 216 |

Продолжительность рабочего дня равна 8 часам (216/8=27).

### 3.2 Расчет материальных затрат на разработку программного продукта

Определение затрат на разработку программного продукта производится на основе существующей сметы, которая включает следующие статьи:

– материальные затраты;
– затраты на оплату труда;
– социальный налог;
– амортизация основных фондов;

Статья «Материальные затраты» состоит из основных и вспомогательных материалов, энергии, которые необходимы для разработки программного продукта. Расчет затрат на материальные ресурсы производится по форме, приведенной в таблице 3.2.

Таблица 3.2 – Затраты на материальные ресурсы

| Наименование материала | Марка | Ед. измерения | Количество | Цена за ед. в тенге | Сумма в тенге |
|---|---|---|---|---|---|
| Офисная бумага, А4 | Ballet Brilliant | Пачка | 2 | 2245 | 4490 |
| Тетрадь общая,А4, 80 листов | ACTION | Штук | 3 | 485 | 1455 |
| Блокнот (96 листов) | Notes | Штук | 1 | 485 | 485 |
| Ручка | MaxWriter | Штук | 3 | 150 | 450 |
| Карандаш | Staedtler | Штук | 3 | 200 | 600 |
| Компьютерная мышь (беспроводная) | Logitech MX MASTER 2S | Штук | 1 | 27990 | 27990 |
| Итого | | | | | 37930 |

При покупке нового ноутбука Toshiba Portege Z30-C-138 в нем предусмотрены встроенная операционная система и дополнительное программное обеспечение, поэтому затраты на покупку новой операционной системы Windows 10 enterprise и лицензионную MS Office производиться не будут.

Так же, в расчеты на затраты не входят дополнительные программные обеспечения для работы, так как они являются бесплатными и размещены в общем доступе

Таблица 3.3 – Затраты на ОС и ПО, необходимые для проекта

| Наименование материала | Марка | Ед. измерения | Количество | Цена за ед. в тенге | Сумма в тенге |
|---|---|---|---|---|---|
| Ноутбук | Toshiba Portege Z30-C-138 | Шт. | 1 | 690000 | 690000 |
| Принтер | Samsung M2070 | Шт. | 1 | 48500 | 48500 |
| Модем | TP-Link TL-WR740N | Шт. | 1 | 8900 | 8900 |
| Итого | | | | | 747400 |

Общая сумма затрат на материальные ресурсы ($З_м$) определяется по формуле:

$$З_м = \sum P_i \times Ц_i, \quad (3.1)$$

где $P_i$– расход i-го вида материального ресурса, натуральные единицы;
$Ц_i$– цена за единицу i-го вида материального ресурса, тг;
i – вид материального ресурса;
n – количество видов материальных ресурсов.
$З_м$ = 37930 + 747400+ = 785330 (тг)
Материальные затраты на разработку программного продукта составят 785330 тенге.

### 3.3 Расчет затрат на электроэнергию

Важно рассчитать затраты на электроэнергию, потому что в процессе работы используется электрооборудование. Время работы оборудования для разработки программного продукта берется равным 224 часов для ноутбуков и модема, данное количество часов было рассчитано в таблице 6.1. Для принтера время работы для разработки программного продукта берется равным 12 часов, так нет необходимости постоянного его использования[10].

$$Э = З_{эл.эн.обор} + З_{доп.нуж}, \quad (3.2)$$

где $З_{эл.эн.обор}$ – затраты на электроэнергию оборудования;
$З_{доп.нуж.}$ – затраты электроэнергии на дополнительные нужды.

Расходы электроэнергии на оборудование рассчитывается по формуле:

$$З_{эл.эн.обор} = \sum W \times К_{исп} \times S \times T, \quad (3.3)$$

гдеW – потребляемая мощность, Вт;
$К_{исп}$ – коэффициент использования ($К_{исп}$ = 0,7..0,9);

T – время работы;

S – тариф (1кВт/ч = 18,32 тг).

Сводные результаты расчета затрат на электроэнергию представлены в таблице 3.4.

Таблица 3.4 - Затраты на электроэнергию

| Наименование приборов | Паспортная мощность, кВт | Коэффициент мощности | Время работы оборудования, ч | Цена ЭЭ тг/кВтч | Сумма, тг |
|---|---|---|---|---|---|
| Ноутбук | 0,6 | 0,7 | 216 | 18,32 | 1661,90 |
| Модем | 0,08 | 0,9 | 100 | 18,32 | 131,90 |
| Принтер | 0,5 | 0,9 | 12 | 18,32 | 98,90 |
| Кондиционер | 0,8 | 0,9 | 200 | 18,32 | 2638,08 |
| Освещение | 0,3 | 0,7 | 216 | 18,32 | 830,90 |
| Итого | | | | | 5361,70 |

$З_{эл.эн.обор}$ = 1661,9+131,9+98,9+2638,08+830,9=5361,70 (тенге)

Затраты на дополнительные потребности берутся по укрупненному показателю в размере 5% от затрат на оборудование:

$$З_{доп.нуж} = 5\% \times З_{эл.эн.обор} \quad (3.4)$$

Затраты на дополнительные потребности рассчитаны по формуле (6.4):

$З_{доп.нуж}$ = 0,05 × 5361,7 = 268,08 (тенге)

Таким образом суммарные затраты на электроэнергию составляют:

Э = 5361,7 + 268,08= 5629,78(тенге)

**3.4 Расчет затрат на оплату труда**

Над разработкой проекта работают два сотрудника:

- руководитель проекта – он изучает предметную область, проводит анализ требований к системе, занимается внедрением и поддержкой;
- разработчик – создание и реализует модель, занимается тестировкой и отладкой продукта;

Общая сумма затрат на оплату труда ($З_{тр}$) определяется по формуле:

$$З_{тр}=\sum ЧС_i \times T_i \quad (3.5)$$

где $ЧС_i$– часовая ставка i-го работника, тг;

Ti – трудоемкость разработки модели, чел.×ч;

i – категория работника;

n – количество работников, занятых разработкой ПП.

На этапах разработки, участники разработки задействованы неравноценно, для этого необходимо рассчитать часовую ставку работника, а затем общий размер заработной платы.

Часовая ставка работника может быть рассчитана по формуле:

$$ЧС_i = \frac{ЗПi}{ФРВi} \quad (3.6)$$

где Зпi – месячная заработная плата i-го работника, тг;

ФРВi- месячный фонд рабочего времени i-го работника, час

Месячная заработная плата сотрудников:

Руководитель проекта – 180 000 тг;

Разработчик – 120 000 тг.

$ЧС_i$= 180 000 / 22 × 8 = 1 022,72 тг/ч

$ЧС_i$= 120 000 / 22 × 8 = 681,81 тг/ч

Часовая ставка научного руководителя составляет 1 022,72 (тг/ч), трудоемкость разработки – 90 ч. Часовая ставка разработчика составляет 681,81 (тг/ч), трудоемкость разработки – 216 ч.

Рассчитаем общую сумму затрат на оплату труда по формуле (3.5):

$З_{тр}$ = 1 022,72 × 90 + 681,81 × 216 = 239 315,76 (тенге)

Сводные результаты расчета затрат на оплату труда показаны в таблице 6.5.

Таблица 3.5 – Расчёт основной заработной платы разработчиков.

| Категория работника | Квалификация | Трудоемкость разработки ПП, час. | Часовая ставка, тг/ч | Сумма, тг. |
|---|---|---|---|---|
| Руководитель проекта | Инженер-программист | 90 | 1 022,72 | 92 044,80 |
| Разработчик | Программист | 216 | 681,81 | 147270,96 |
| Итого | | | | 239 315,76 |

**3.5 Расчет затрат по социальному налогу**

Социальный налог – согласно Налоговому кодексу Республики Казахстан составляет 9,5 % от ФОТ (фонда оплаты труда). Следует отметить, что пенсионные отчисления не облагаются социальным налогом.

$$С_н = (ФОТ- ПО)\times 0{,}095 \quad (3.7)$$

где ПО - отчисления в пенсионный фонд, 10% от ФОТ.

Социальный налог рассчитываем по формуле (3.7):

ПО = 239 315,76× 0,1 = 23 931,576 тенге;

$С_н$ = (239 315,76 – 23 931,576) × 0,095 = 20 461,49 тенге

Сводные результаты расчета затрат представлены в таблице 6.7.

Таблица 3.7 - Начисление социального налога

| Категория работника | Количество человек | Заработная плата, тг | Пенсионные отчисления, тг | Социальный налог, тг |
|---|---|---|---|---|
| Руководитель | 1 | 92 044,8 | 9 204,48 | 7 869,75 |

| | | | | |
|---|---|---|---|---|
| проекта | | | | |
| Разработчик | 1 | 147270,96 | 14 727,09 | 12591,66 |
| Итого | | | | 20 461,49 |

### 3.6 Амортизация основных фондов и прочие затраты

Годовые нормы амортизации ОФ принимаются по налоговому кодексу РК или определяются, исходя из возможного срока полезного использования ОФ. Амортизация основных фондов определяется:

$$А_г = \frac{С_{об} \times Н_а}{100} \quad (3.8)$$

где, $С_{об}$ – стоимость оборудования;

$Н_а$ – норма амортизации (норма амортизация = 20);

По формуле 3.8 рассчитаем сумму амортизационных отчислений за год для ноутбука:

$$А_г = \frac{190000 \times 20}{100} = 138\ 000 \text{ тг}$$

Рассчитаем сумму амортизации за время разработки:

$$А_р = \frac{138\ 000 \times 27}{365} = 2\ 811 \text{ тг}$$

Аналогичным способом рассчитаем сумму амортизации для остального оборудования[10].

Результаты расчетов приведены в таблице 6.6

Таблица 3.8 - Амортизация основных фондов

| Наименование оборудования и ПО | Стоимость оборудования и ПО, тг | Годовая норма амортизации, % | Сумма амортизации за год, тг | Сумма амортизации за время разработки, тг |
|---|---|---|---|---|
| Ноутбук | 690 000 | 20 | 138 000 | 10208,21 |
| Принтер | 48500 | 20 | 9700 | 744,1 |
| Модем | 8900 | 15 | 1355 | 102,4 |
| ИТОГО амортизация основных средств | | | 49055 | 11054,71 |

Смета затрат на разработку программного продукта

На основании полученных данных по отдельным статьям составляется смета затрат на разработку программного продукта по форме, приведенной в таблице[1].

Таблица 5.9 – Смета затрат на разработку программного продукта

| Статьи затрат | Сумма, тг | % |
|---|---|---|
| Затраты на оборудование и материальные ресурсы | 785330,00 | 73.96 |
| Затраты на оплату труда | 239 315,76 | 22.54 |
| Социальные налоги | 20 461,49 | 1.94 |
| Затраты на электроэнергию | 5629,70 | 0.53 |
| Амортизация основных фондов | 11054,71 | 1.04 |
| Итого по смете | 1061791,66 | 100 |

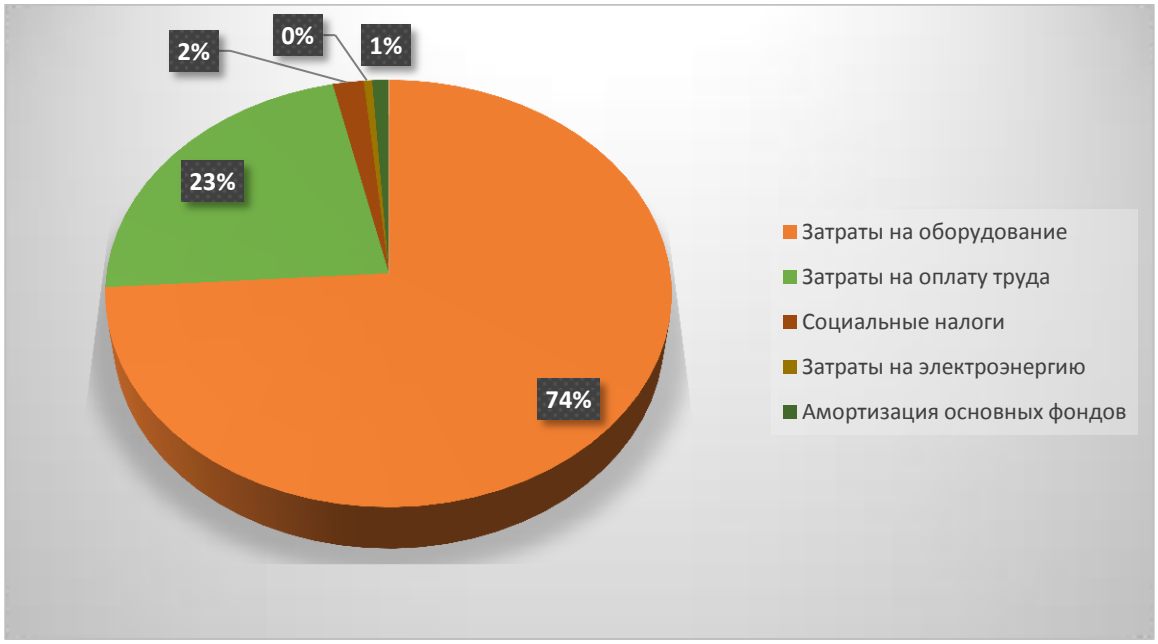


Диаграмма 3.1 - Диаграмма структуры затрат

### 3.7 Определение возможной (договорной) цены программного продукта

Величина возможной (договорной) цены программного продукта устанавливается на основе эффективности, качества и сроков её выполнения на уровне, отвечающем экономическим интересам заказчика (потребителя) и исполнителя.

Договорная цена ЦД для прикладных программных продуктов рассчитывается по формуле:

$$Ц_Д = З_{НИР}\left(1 + \frac{Р}{100}\right) \quad (3.9)$$

где $З_{НИР}$ - затраты на разработку ПП, тг; Р - средний уровень рентабельности ПП. % (принимается в размере 20%).

$Ц_Д$ = 1061791,66× (1 + 20/100) = 1061791,66+212358,33 = 1274149,99 тенге

Далее определяется цена реализации с учетом налога на добавленную стоимость (НДС), ставка (НДС) устанавливается законодательно. Налоговым Кодексом РК. На 2017 год ставка НДС установлена в размере 12%.

Цена реализации с учетом НДС рассчитывается по формуле:

$$Ц_р = Ц_Д + Ц_Д \times НДС \quad (3.10)$$

1274149,99 + 1274149,99 × 0,12 = 1274149,99 + 152898 = 1427047,99 тенге

Рассчитанную возможную цену ПП можно округлить до 1428000 тенге.

**Вывод**

Данная глава дипломного проекта содержит экономические расчеты, которые позволяют определить затраты необходимые для разработки программного продукта. Расчеты включают в себя:

- расчет трудоемкости разработки программного продукта;
- расчет затрат на разработку программного продукта;
- расчет затрат на электроэнергию;
- расчет затрат на оплату труда;
- расчет затрат по социальному налогу;
- амортизация основных фондов и прочие затраты.

Договорная цена программного продукта будет равна 1428000,00 тенге. Смета затрат на разработку программного продукта будет равна 1061791,66 тенге. Прибыль (рентабельность) будет равна 212358,33 тенге.

## 4 Безопасность жизнедеятельности

Основной целью данного дипломного проекта является исследование программного кода на уязвимость и последующее воздействие на систему. Актуальность данной темы заключается в том, что на текущий момент информация является ценным ресурсом, с которым необходимо правильно взаимодействовать, и если допустить вероятность потери данных, то это оценивается в огромном материальном ущербе.

Данное исследование может использоваться различными компаниями или просто пользователями, которым приходится сталкиваться с фишинговыми рассылками. Благодаря этому исследованию подверженные заражению системы могут быть спасены, поскольку будет известно какими методами можно вылечить систему, где хранится главный зараженный документ, какие профилактические меры стоит проводить, помимо общеизвестных и достаточно много других факторов, которые помогут сэкономить компании большие средства. Если вы не соблюдаете меры безопасности пользования интернет ресурсом, в частности почтой, то велик риск заражения именно тем вредоносным программным обеспечением, о котором и говорится в моем дипломном проекте[11].

### 4.1 Анализ условий труда

При исследовании кода программного обеспечения, работник вынужден долгое время взаимодействовать с компьютерной техникой. Рабочая зона – это та зона временного либо постоянного присутствия работника. Из за того что работник должен проводить длительное время на стуле в сидячем положении должны быть предусмотрены меры максимального удобства, которые позволят работать уютно и без вредного воздействия. Эти меры должны включать в себя: компьютерное оборудование и мебель необходимо размещать оптимальным образом, достаточное рабочее пространство, которые позволит работнику проделывать все необходимые действия и перемещения, работник должен получать необходимое количество световых лучей, чтобы максимально снизить нагрузку на зрение, на рабочем месте должна соблюдаться комфортная комнатная температура при которой работник сможет чувствовать себя комфортно.

Существует несколько типов освещения, естественное и искусственное.

Естественное освещение – освещение, которое проникает через световые проемы, и является дневным светом. Этот тип освещения меняется в связи с природными условиями, временем суток, временем года.

Искусственное освещение – освещение, в котором не участвует естественное освещение, и может использоваться в тёмное время суток и тогда, когда естественного освещения не хватает.

Использование искусственного и естественного освещения одновременно, называется комбинированным освещением[11].

Правильность выбора подсветки и освещения в виде светильников и ламп, а также корректное расположение обеспечит не привыкающую значения 40 кд/м$^2$ отраженность бликов на рабочей станции и рабочей поверхности.

Для искусственного освящения должны применяться люминесцентные лампы белого света. В производственной среде и административно-общественных помещениях можно использовать металл галогенные лампы мощностью до 250 Вт.

Характеристики рабочего помещения:

Рабочее помещение, в котором работник проводит свое исследование рассчитан на одно рабочее место. Располагается на 4 этаже жилого здания. Визуализированная модель помещения представлена на Рисунке 5.17.

Характеристики помещения: длина L = 10 метра, ширина B = 5 метров, высота H = 3 метра. Помещение было построено и оборудовано согласно санитарным требованиям от 01.12.2011 года, т.е. площадь одного рабочего места 4,5 метра в квадрате, а монитор должен находиться на расстоянии 60см от глаз. Рабочее пространство работника удовлетворяет требованиям и составляет 50м$^2$.

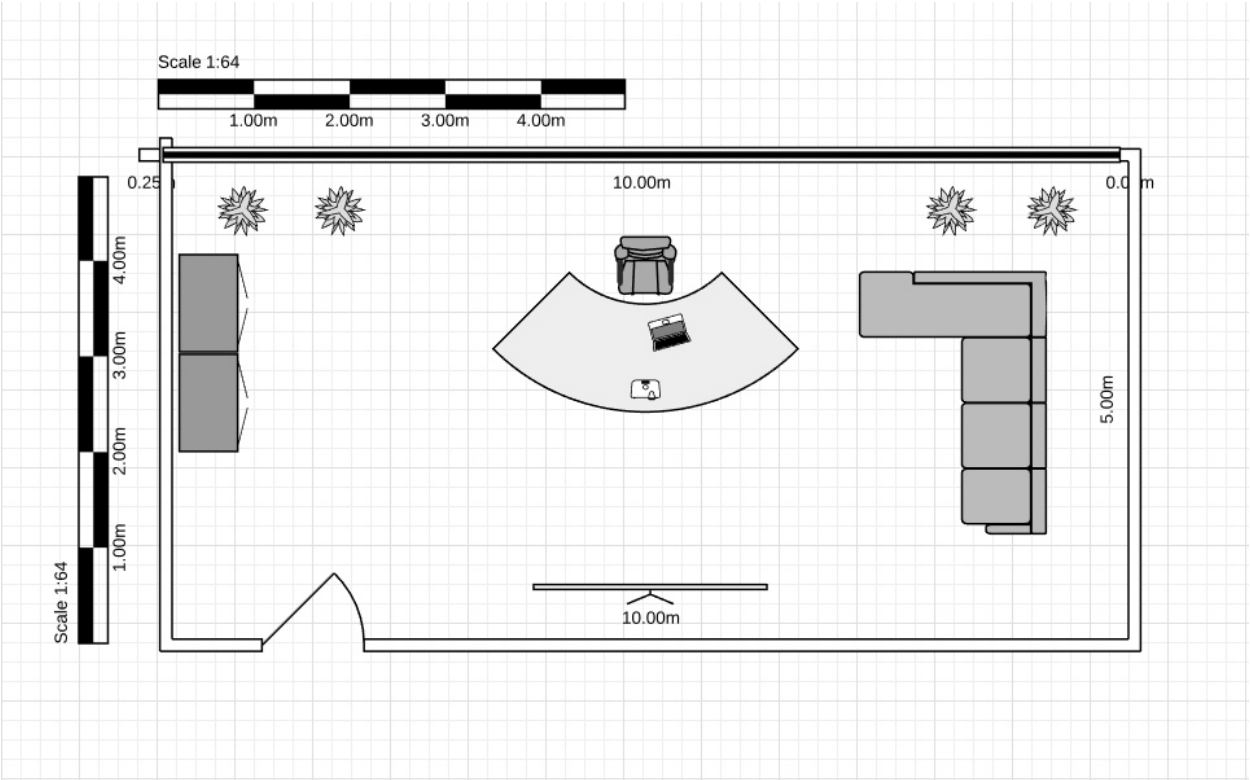


Рисунок 4.28 – Рабочее помещение

Используемое оборудование и его характеристики:

Ноутбук Toshiba Portege Z30-C-138. Технические характеристики устройства:

- Intel(R) Intel Core i7-6500U (CPU 2.5 GHz);
- Intel HD 4600;
- ОЗУ 8 ГБ;
- HDD 1 ТБ;

- электропитание: 220-250В, 50Гц, 400 Вт;
- габариты(мм): 270 - 414 – 32,5.

Модем: 4-х портовый с коммутатором 10/100 Мбит/с.

Стул: высота 0,6 м.

Стол: высота – 0,8 м, длина – 2.25 м, ширина – 1.25 м.

4.1.1 Расчет естественного освещения

Один из главных показателей, который должен находить на заданном уровне, это освещение. Качество освещения важно для создания удобства при работе. Хороший уровень освещения необходим для комфортной работы и исключения зрительного напряжения, которое в следствии может привести к ухудшению здоровья и сказаться на качестве работы. Согласно нормам освещенности (СНиП 11-4-79) и отраслевым нормам, работа инженера относится к четвертому разряду зрительной работы. Важно проводить расчеты по освещению, они необходимы для определения площади световых проёмов естественного освещения и характеристики искусственного освещения[11]. Формула (4,1) для расчета площади световых проемов при естественном освещении:

$$S = \frac{(S_n * e_n * K_n * h_0 * K_{зо})}{(100 * t_0 * r_1)} \quad (4.1)$$

где,

$S_n$ – площадь помещения, м$^2$;

$e_n$ – нормированное значение КЕО, %;

$K_n$ – коэффициент запаса;

$h_0$ – световая характеристика окон (6,5 - 29);

$K_{зо}$ – коэффициент затемнения окон зданиями стоящими напротив (1,0-1,7);

$r_1$ – коэффициент повышение КЕО за счет отраженного света от поверхности помещения (1,05 - 1,7);

$t_0$ - общий коэффициент светопропускания равен от 0,1-0,8.

Полагаясь на данные характеристики, длина помещения равна 10 метров, а ширина равно 5 метра, можно найти площадь пола по следующей формуле:

$$S_n = L * B \quad (4.2)$$

$$S_n = 5 * 10 = 50 \text{ м}^2$$

Для данного рабочего пространства площадь световых проемов естественного бокового освещения определяется формулой (4,1), необходимы следующие значения:

Где,

$e_n = 1{,}5$ %;

$K_n = 1{,}5$;
$h_0 = 29$;
$K_{зо} = 1{,}1$;
$r_1 = 1{,}3$;
$t_0 = 0{,}7$.

Теперь необходимо подставить значение данных коэффициентов в формулу (4,1) и вычислить площадь световых проемов:

$$S = \frac{50 * 1{,}5 * 1{,}5 * 29 * 1{,}1}{100 * 0{,}7 * 1{,}3} = 39{,}43 \text{ м}^2$$

Рассчитав площадь оконного пространства, значение вышло $39{,}43 \text{ м}^2$, из чего следует вывод что необходимо искусственное освещение так как окна площадью 30м$^2$ недостаточно для создания комфортных условий освещения.

4.1.2 Расчет искусственного освещения

Основываясь на норму СНиП 11-4-79 для четвертого класса зрительных работ освещенность помещения должно быть не менее 200 Лк. Номинальная освещенность рабочего места определяется формулой:

$$E = \frac{Ф_{св} * n * N * 1}{s * K_з * Z} \quad (4.3)$$

Где,

$Ф_{св}$ – световой поток от ламп, Лк;

N – количество светильников;

$K_з$ – коэффициент, учитывающий запыленность светильников;

n – коэффициент использования светильников;

s – площадь помещения, м$^2$;

z – коэффициент неравномерности освещения.

Основываясь на норму СНиП 11-4-79 для типа ламп, который будут использоваться в ЛАЗ КДП (светильник типа УСП-35): $K_з = 1{,}4{:}\,1{,}5$ при нормальной эксплуатации светильников; z = 1,1:1,2 при оптимальном их размещении. В данном случае коэффициент n полностью зависит от светильников и их типа, коэффициенты отражения светового потока от потолка - $p_2$, от пола $p_3$, от стен $p_1$, зависят от размера помещения, учитывающих величиной I, где I это индекс помещения.

$$I = \frac{(A * B)}{h_c * (A + B)} \quad (4.4)$$

где A, B - параметры помещения, м;

$h_c$ - высота светильников над рабочей поверхностью.

Расчёт высоты светильников над рабочей поверхностью выводится по формуле:

$$h_c = H_{помещения} - H_{свеса} - H_{р.п.} \quad (4.5)$$

где $H_{свеса}$ = 0,4 - высота свеса ламп, м;

$H_{р.п.}$ = 0,8 - расстояние рабочей поверхности над полом, м;

$H_{помещения} = 3$ - высота помещения, м.

Основываясь на формулу (4.5) определяется высота светильников над рабочей поверхностью:

$$h_{расч} = 3 - 0{,}4 - 0{,}8 = 1{,}8 \text{ м}$$

Зная что параметры помещения равны 5м × 10м и высота светильников над рабочей поверхностью $h_c = 1{,}8$м, по формуле (4.4):

$$I = \frac{(10 * 5)}{2 * (10 + 5)} = 1{,}67$$

Основываясь на таблицу (4.1) определяется коэффициент использования светового потока n, учитывая, что коэффициенты $p_1$ = 30%, $p_2 = 50\%$, $p_3 = 10\%$.

Таблица 5.3 - Значения коэффициента использования светового потока

| Коэффициент I | 0,5 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| Коэффициент использования светового X потока, h | 0,22 | 0,36 | 0,48 | 0,54 | 0,59 |

Коэффициент n = 0,3 для рабочего места. От лампы iPower IPIL200W-GKE световой поток равняется 16700 Лк, в совокупности от 4 ламп световой поток будет равняться 66800 Лк. Основываясь на все вычисления и данные можно определить номинальную освещенность рабочего места по формуле

$$E = \frac{16700 \cdot 0{,}3 \cdot 4}{50 \cdot 1{,}4 \cdot 1{,}2} = 262{,}43 (Лк) \quad (4.3)$$

Значение, полученное в ходе расчётов, соответствует нормальным условиям освещенности и создается комфортную для работы обстановку[4].

### 4.2 Расчет воздухообмена в рабочем помещении

При работе важной характеристикой является температура и вентиляция. Условия труда в которых используется оборудование для

регулирования температуры, такие как вентиляторы, кондиционеры или естественная вентиляция, представляемая от окон или дверей, необходимо рассчитывать по формуле:

$$Q = \frac{g}{X - X_n} \tag{4.6}$$

где Q - потребный воздухообмен, $м^3/ч$;

g - количество вредных веществ, л/ч;

X – предельно допустимая концентрация вредных веществ в помещения, $л/м^3$;

$X_n$ – предельно допустимая концентрация вредных веществ в наружном воздухе, $л/м^3$.

Кратность воздухообмена n, показывает количества раз в течении часа воздух обязан смениться.

$$n = \frac{Q}{Q_{пот}} \tag{4.7}$$

где $Q_{пот}$ это объем помещения;

Объем помещения $Q_{пот}$ определяется формулой:

$$Q_{пот} = L * H * B \tag{4.8}$$

$$Q_{пот} = 5 * 10 * 3 = 150 м^3$$

Количество углекислого газа, который выделяется человеком при работе в расслабленном состоянии равен 23 л/ч, в помещении предельная концентрация не должна превышать 1 $л/м^3$, предельно допустимая концентрация в наружном воздухе 0,5 $л/м^3$. Потребный воздухообмен определяется по формуле (4.6):

$$Q = \frac{23}{1 - 0{,}5} = 46\ (\frac{м^3}{ч})$$

Формулой (4.7) определяет кратность воздухообмена n:

$$n = \frac{46}{150} = 0{,}3$$

Воздухообмен для удаления избыточного тепла рассчитывается по формуле:

$$Q_{изб} = \frac{L_{изб}}{G_B * C_B * d_t} \quad (4.9)$$

где $L_{изб}$ - избыточное тепло, ккал/ч;
$G_B$ = 1,207 кг/м$^3$ - удельная масса приточного воздуха;
$C_B$ = 0,25 ккал/кг·°С - теплоемкость воздуха;
$d_t$ - разность температур удаленного и приточного воздуха;
$d_t$ зависит от тепло напряжённости воздуха - $L_H$. Если $L_H$ больше 20 ккал/ч, то $d_t$ = 8°С. Если $L_H$ меньше 20 ккал/ч, то $d_t$ = 6°С.
Тепло напряжённости воздуха определяется формулой:

$$L_H = \frac{L_{изб}}{Q_{пот}} \quad (4.10)$$

Количество избыточного тепла определяется формулой:

$$L_{изб} = L_{об} + L_{ос} + L_л + L_р + L_{отд} \quad (4.11)$$

где $L_{об}$ - тепло от оборудования, ккал/ч;
$L_{ос}$ - тепло от системы освещения, ккал/ч;
$L_л$ - тепло, выделяемое людьми, ккал/ч;
$L_р$ - тепло от солнечной радиации, ккал/ч;
$L_{отд}$ – теплоотдача естественным путем, ккал/ч.
Тепло от оборудования определяется формулой:

$$L_{об} = 860 * P_{об} * f \quad (4.12)$$

где $P_{об}$ - номинальная мощность оборудования, Вт;
f - коэффициент передачи, f = 0,25.
Тепло от системы освещения определяется формулой:

$$L_{ос} = 860 * P_{ос} * a * b * cof(f) \quad (4.13)$$

где $P_{ос}$ - номинальная мощность освещения, кВт;
a – коэффициент перевода электрической энергии в световую, 0,46;
b – коэффициент одновременной работы ламп, b = 1;
cos(f) – коэффициент мощности, cos(f) = 0,3.
Тепло выделяемое людьми определяется формулой:

$$L_л = n * g \quad (4.14)$$

где n - количество человек;

g - тепловыделение одного человека, g=50 (ккал/ч).

Тепло от солнца для одного окна определяется формулой:

$$L_р = F * D_{ос} \quad (4.15)$$

где F - площадь окна, м$^2$;

$D_{ос}$ - солнечная радиация, $D_{ос}$ = 65 (ккал/ч).

Тепло излучающее системой, оборудованием, людьми и солнцем для помещения определяется формулами (4.12 – 4.15):

$L_{об}$ = 860 · 1 · 0,25 = 215 (ккал/ч)

$L_{ос}$ = 860 · (0,8 · 4) · 0.46 · 1 · 0.3 = 379,77 (ккал/ч),

$L_л$ = 0,7 · 50 = 35 (ккал/ч),

$L_р$ = 30 · 65 = 1950 (ккал/ч).

Естественную теплоотдачу приравнивается к $L_р$ в холодное время года и в погоду равной нулю в теплое время года, определяю по формуле (4.11) можно узнать количество избыточного тепла:

$L_{изб}$ = 215+379,77 +35+1950+0 = 2579,77 (ккал/ч).

Воздушную тепло напряжённость определяется формулой (0.10):

$$L_Н = \frac{2579,77}{150} = 17,19 \text{ (ккал)}$$

Поскольку тепло напряжённость воздуха меньше 20, $d_t = 6°C$. Используется формула (0.9) для того, чтобы произвести расчет значения необходимого для воздухообмена и удаления избыточного тепла:

$$Q_{изб} = \frac{2579,77}{(1,206 \cdot 0,24 \cdot 6)} = 1485,49 \text{ (м3/ч)}$$

Исходя из полученных результатов, для удаления лишнего тепла и очистки воздуха нужно использовать вентиляционную систему, которая способна обеспечить требуемую подачу воздуха $Q_{изб} = 208,3$ (м3/ч). В данном случае подойдет кондиционер Hisense. Данный кондиционер способен обеспечить подачу воздуха до 600 м$^3$ /ч.


Рисунок 4.29 – Кондиционер DAIKIN FTXZ50N

Технические характеристики:
- мощность (охлаждение): 3.75 кВт;
- мощность (обогрев): 3.8 кВт;
- потребляемая мощность при охлаждении: 5000 Вт;
- потребляемая мощность при обогреве: 6300 Вт;
- обслуживаемая площадь: 50 м2;
- уровень шума внутреннего блока: 21-23 дБ;
- уровень шума внешнего блока: 55 дБ;
- цвет: белый.

Характеристики подключения:
- вентиляция: 600 м3/час;
- класс энергоэффективности при охлаждение/обогреве: А++/А+;
- напряжение/частота: 220 В / 50 Гц;
- энергопотребление в режиме ожидания не более 1 Вт.

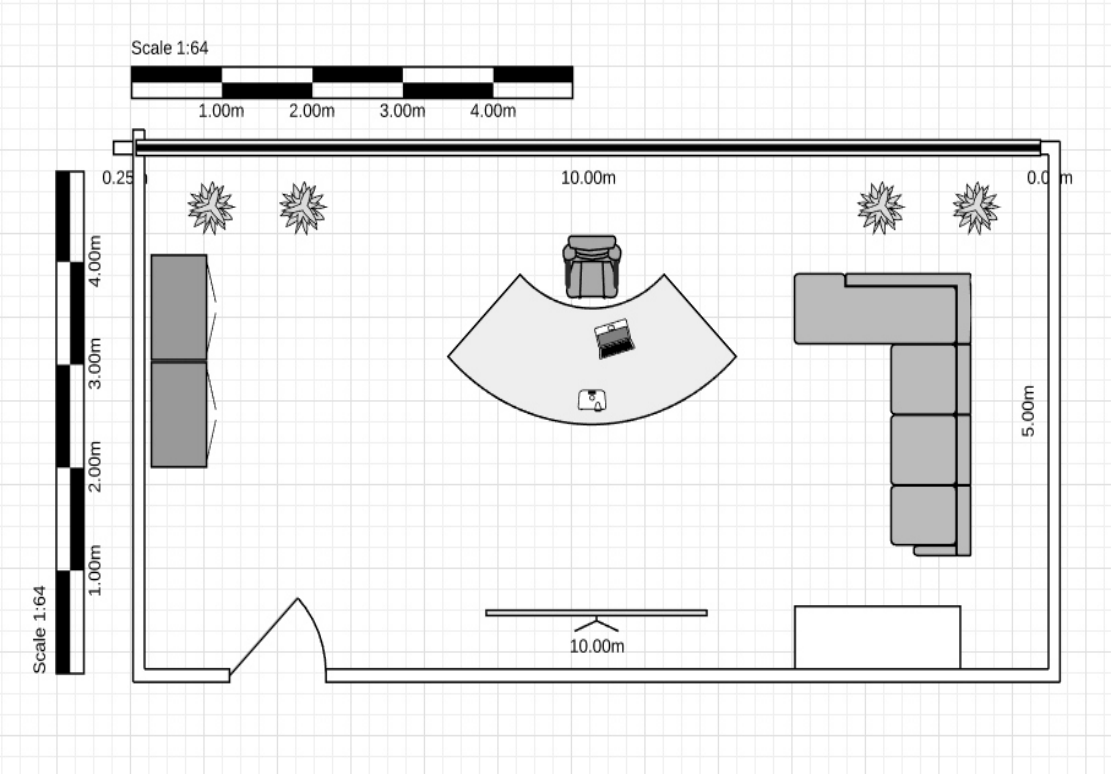


Рисунок 4.30 – Итоговый план рабочего помещения

**Вывод**

В этом разделе моего дипломного проекта я рассмотрел и рассчитал световые и воздушные показатели для условий труда. Эти показатели одни из важнейших при организации работы, должны всегда соответствовать рамкам стандартов и нормы, поскольку это будет способствовать созданию благоприятных условий для работника и не будет мешать работе затормаживая ее. Основываясь на расчёты, могу сказать, что, для того чтобы осветить комнату площадью 50 $м^2$ абсолютно не хватает естественного освещения, предоставляемого с окна размером 10 метра в длину и 3 в ширину. Для удобной работы необходимо комбинированное освещение, в которое включено как естественное, так и искусственное освещение. Полагаясь на расчёты полученные в ходе выполнения этого раздела должно использоваться 3 лампы в моем случае это ДРЛ-80 3800 Лк. Если необходимые условия буду соблюдены, то работник может проводить все необходимые работы и исследование в ночное время.

Следующим важным моментом является расчёт воздухообмена и вентиляции. Полагаясь на результаты расчётов, могу сказать, что для создания хороших условий труда необходим один кондиционер с подачей воздуха не менее 208,3 $м^3$ /ч, в моем случае используется кондиционер Hisense с подачей воздуха до 600 $м^3$ /ч.

**Заключение**

На данный момент стоит острая необходимость в проверке кода ПО на уязвимость, так как из-за уязвимостей компании теряют огромную прибыль по всему миру. Различные типы уязвимостей могут привести к утечке данных, изменению или даже взлому сервера. Учитывая тот факт, что большие приложения могут иметь тысячи строк кода, ручная проверка может быть очень неэффективной.

Поскольку статический анализ исходного кода может дать сбой при очень сложных уязвимостях, цель - сделать все возможное, чтобы автоматически находить недостатки, а также предоставлять как можно больше информации и вариантов, чтобы сделать дальнейший анализ максимально простым и быстрым.

В ходе выполнения дипломного проекта была изучена концепция статического анализа исходного кода программного обеспечения на скриптовом языке PHP а также была выполнена ее программная реализация и тестирование:

- Изучены особенности скриптового языка PHP, для эффективного процесса разработки программы.
- Произведена организация базовых функций для работы с файлами.
- Реализован модуль для токенизации исходного кода на языке php.
- Реализован модуль для анализа массива токенов.
- Спроектирован и реализован модуль вывода результатов анализа.
- Произведены экономические расчеты эффективности.
- Произведены расчеты безопасности жизнедеятельности.

Разработанный инструмент проверки программного обеспечения может помочь пентестерам минимизировать затраты времени за счет автоматизации процессов проверки исходного кода на языке PHP на уязвимость.

## Список литературы


1 Костарев А. Ф. PHP 5. - СПб.: «БХВ-Петербург», 2008. - С. 1104.

2 Устинов, Г.Н. Уязвимость и информационная безопасность телекоммуникационных технологий/ Г.Н. Устинов М.: Радио и связь, 2016. - 342 с.

3 Боршевников А. Е. Сетевые атаки. Виды. Способы борьбы // Современные тенденции технических наук: материалы Междунар. науч. конф. (г. Уфа, октябрь 2014 г.). – Уфа: Лето, 2014. – С. 8-13. – URL https://moluch.ru/conf/tech/archive/5/1115/ (дата обращения: 29.05.2019).

4 Информационная безопасность и защиты информации. Учебное пособие. Ростов-на-Дону, 2015.

5 Официальный сайт PHP, URL: https://www.php.net (дата обращения: 15.03.2019).

6 Официальный сайт *owasp,* URL: https://www.owasp.org (дата обращения 15.03.2019).

7 Моделирование встроенных функций PHP для точного статического анализа кода, URL: http://citeseerx.ist.psu.edu (дата обращения 15.03.2019)

8 Уязвимости в PHP-скриптах, URL: http://www.php-security.org/downloads/rip (дата обращения 15.03.2019)

9 Бекишева А.И. Методические указания к выполнению экономической части дипломной работы для бакалавров специальности 5В0703 – Информационные системы – Алматы: АУЭС, 2013.

10 Аманжолова К. Б., Алибаева С. А. Экономика предприятии телекоммуникации: Учебное пособие. - Алматы: АИЭС, 2003

11 И. Ф. Мазалов, К. Г. Мустафин, Е. М. Тыщенко, М. А. Сералиева Методические указания по выполнению РГР для студентов специальности 5В0731100-БЖ. – Алматы: АУЭС, 2015. – 38 с.

ПРИЛОЖЕНИЕ А

Листинг

```
<?php

class Tokenizer
{
	public $filename;
	public $tokens;

	function __construct($filename)
	{
		$this->filename = $filename;
	}
	public function tokenize($code)
	{
		$this->tokens = token_get_all($code);
		$this->prepare_tokens();
		$this->array_reconstruct_tokens();
		$this->fix_tokens();
		$this->fix_ternary();
		#die(print_r($this->tokens));
		return $this->tokens;
	}
	function wrapbraces($start, $between, $end)
	{
		$this->tokens = array_merge(
			array_slice($this->tokens, 0, $start), array('{'),
			array_slice($this->tokens, $start, $between), array('}'),
			array_slice($this->tokens, $end)
		);
	}
	function prepare_tokens()
	{
		for($i=0, $max=count($this->tokens); $i<$max; $i++)
		{
			if( is_array($this->tokens[$i]) )
			{
				if( in_array($this->tokens[$i][0],
Tokens::$T_IGNORE) )
					unset($this->tokens[$i]);
				else if( $this->tokens[$i][0] === T_CLOSE_TAG )
```

```
                                        $this->tokens[$i] = ';';
                                    else if( $this->tokens[$i][0] ===
T_OPEN_TAG_WITH_ECHO )
                                        $this->tokens[$i][1] = 'echo';
                                }
                                else if($this->tokens[$i] === '@')
                                {
                                    unset($this->tokens[$i]);
                                }
                            else if( $this->tokens[$i] === '{'
                                && isset($this->tokens[$i-1]) && ((is_array($this-
>tokens[$i-1]) && $this->tokens[$i-1][0] === T_VARIABLE)
                                || $this->tokens[$i-1] === ']') )
                                {
                                    $this->tokens[$i] = '[';
                                    $f=1;
                                    while($this->tokens[$i+$f] !== '}')
                                    {
                                        $f++;
                                        if(!isset($this->tokens[$i+$f]))
                                        {
                                            addError('Could not find closing brace
of '.$this->tokens[$i-1][1].'{}.', array_slice($this->tokens, $i-1, 2), $this->tokens[$i-
1][2], $this->filename);
                                            break;
                                        }
                                    }
                                    $this->tokens[$i+$f] = ']';
                                }
                            }

                            $this->tokens = array_values($this->tokens);
                        }
```

Листинг -4 .

Анализ базовых блоков

```
<?php


class Analyzer
{
```

```
		function get_tokens_value($file_name, $tokens, $var_declares,
$var_declares_global, $tokenid, $start=0, $stop=0, $source_functions=array())
		{
			$value = '';
			if(!$stop) $stop = count($tokens);
			for($i=$start; $i<$stop; $i++)
			{
				if( is_array($tokens[$i]) )
				{
					// trace variables for its values
					if( $tokens[$i][0] === T_VARIABLE
					|| ($tokens[$i][0] === T_STRING
					&& $tokens[$i+1] !== '(' ) )
					{
						if(!in_array($tokens[$i][1],
Sources::$V_USERINPUT))
						{
							// constant CONSTANTS
							if ($tokens[$i][1] ===
'DIRECTORY_SEPARATOR')
								$value .= '/';
							else if ($tokens[$i][1] ===
'PATH_SEPARATOR')
								$value .= ';';
							// global $varname -> global scope,
CONSTANTS
							else if( (isset($tokens[$i-1]) &&
is_array($tokens[$i-1]) && $tokens[$i-1][0] === T_GLOBAL) || $tokens[$i][1][0]
!== '$' )
								$value .=
self::get_var_value($file_name, $tokens[$i], $var_declares_global,
$var_declares_global, $tokenid);
							// local scope
							else
								$value .=
self::get_var_value($file_name, $tokens[$i], $var_declares, $var_declares_global,
$tokenid);
						} else
						{
							if(isset($tokens[$i][3]))
								$parameter_name =
str_replace(array("'",'"'), '', $tokens[$i][3][0]);
							else
								$parameter_name = '';
```

```
                                        // mark userinput for quote analysis
                                        if( ($tokens[$i][1] !== '$_SERVER' ||
(empty($parameter_name) || in_array($parameter_name,
Sources::$V_SERVER_PARAMS) || substr($parameter_name,0,5) === 'HTTP_'))
                                        && !((is_array($tokens[$i-1])
                                        && in_array($tokens[$i-1][0],
Tokens::$T_CASTS))
                                        || (is_array($tokens[$i+1])
                                        && in_array($tokens[$i+1][0],
Tokens::$T_ARITHMETIC))) )
                                                $value.='$_USERINPUT';
                                        else
                                                $value.='1';
                                }
                        }
                        else if( $tokens[$i][0] ===
T_CONSTANT_ENCAPSED_STRING
                        && !($tokens[$i-2][0] === T_STRING &&
$tokens[$i-2][1] === 'define'))
                        {
                                $value .= substr($tokens[$i][1], 1, -1);
                        }
                        else if( $tokens[$i][0] === T_FILE
                        && ($tokens[$i-2][0] === T_STRING &&
$tokens[$i-2][1] === 'dirname'))
                        {
                                $value = dirname($file_name).'/';
                        }
                        else if( $tokens[$i][0] === T_LNUMBER ||
$tokens[$i][0] === T_DNUMBER || $tokens[$i][0] === T_NUM_STRING )
                        {
                                $value .= round($tokens[$i][1]);
                        }
                        else if( $tokens[$i][0] ===
T_ENCAPSED_AND_WHITESPACE )
                        {
                                $value .= $tokens[$i][1];
                        }
                        else if( $tokens[$i][0] === T_AS )
                        {
                                break;
                        }
```

```
                                else if($tokens[$i][0] === T_STRING &&
$tokens[$i+1] === '(')
                                {
                                        if (in_array($tokens[$i][1],
Sources::$F_DATABASE_INPUT) || in_array($tokens[$i][1],
Sources::$F_FILE_INPUT) || isset(Info::$F_INTEREST[$tokens[$i][1]]))
                                        {
                                                break;
                                        }
                                        else if(in_array($tokens[$i][1],
$source_functions))
                                        {
                                                $value .= '$_USERINPUT';
                                        }
                                }
                        }
                }

                return $value;

        }
        function get_var_value($file_name, $var_token, $var_declares,
$var_declares_global, $last_token_id, $source_functions=array())
        {
                $var_value = '';
                if($var_token[1][0] !== '$')
                        $var_token[1] = strtoupper($var_token[1]);
                if( isset($var_declares[$var_token[1]]) )
                {
                        foreach($var_declares[$var_token[1]] as $var_declare)
                        {
                                $array_key_diff = false;
                                if( isset($var_token[3]) && !empty($var_declare-
>array_keys) )
                                        $array_key_diff =
array_diff_assoc($var_token[3], $var_declare->array_keys);

                                if( $var_declare->id < $last_token_id &&
empty($array_key_diff))

                                        $var_value .=
self::get_tokens_value($file_name, $var_declare->tokens, $var_declares,
$var_declares_global, $var_declare->id, $var_declare->tokenscanstart,
$var_declare->tokenscanstop, $source_functions);
```

```
                        if($var_value)
                                break;
                }
        }
        return $var_value;
}
function getBraceEnd($tokens, $i)
{
        $c=1;
        $newbraceopen = 1;
        while( !($newbraceopen === 0 || $tokens[$i + $c] === ';') )
        {
                if( $tokens[$i + $c] === '(' )
                {
                        $newbraceopen++;
                }
                else if( $tokens[$i + $c] === ')' )
                {
                        $newbraceopen--;
                }
                if($c>50)break;
                $c++;
        }
        return $c;
}

function get_ini_paths($path)
{
        if(!preg_match('/([;\\\\]|\W*[C-Z]{1}:)/', $path))
                $path = str_replace(':', ';', $path);
        return explode(';', $path);
}
}

?>
```