

МИНОБРНАУКИ РОССИИ  
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ  
ВЫСШЕГО ОБРАЗОВАНИЯ  
“ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ”

Факультет компьютерных наук

Кафедра Технологии обработки и защиты информации

*Статический анализ кода*

*Анализ уязвимостей и защита программного обеспечения*

*10.03.01 Информационная безопасность*

*Безопасность компьютерных систем*

Обучающийся \_\_\_\_\_ *Н.С.Колпаков, 4 курс, д/о*

Руководитель \_\_\_\_\_ *М.А.Дрюченко, к.т.н., доцент*

Воронеж 2023

## **Введение**

Стремительный рост количества встраиваемых систем актуализирует вопрос качества их программного кода. Если же ещё принимать в расчёт специфику разработки (затруднительная отладка, высокая цена ошибки и т.д.), то использовать специальные инструменты и отдельное ПО, которые позволяют повысить качество написанного кода, становится просто необходимо.

Зачастую, даже от глаз опытных разработчиков ускользают непримечательные недостатки кода, которые, впрочем, могут быть следствием уже куда серьёзных проблем. В таком случае следует подвергнуть код автоматизированной проверке.

Одним из таких решений является использование статического анализа кода. О его структуре и пользе для встраиваемых систем и пойдёт речь в данном докладе.

## **Определение статического анализа**

Статический анализ – анализ ПО, производимый без реального выполнения исследуемых программ.

Инструменты статического анализа проводят куда более глубокую проверку и аналитику исходного кода, чем это делает компилятор, занимающийся, по-большому, лишь нахождением синтаксических ошибок. Принцип работы статического анализа:

- Принимает на входе исходный код программы (в идеале, компилируемый);
- Преобразует этот код в специальную модель для дальнейшего анализа;
- Проводит поиск дефектных мест, применяя к модели набор диагностических правил, в основе которых лежат различные методологии;
- Сохраняет все полученные предупреждения и нарушения в нужный для разработчика формат;

Уже после всего разработчик изучает полученный от анализатора данные и исправляет проблемные места.

## **Методы статического анализа**

Существует несколько типов методов статического анализа-

- Контрольный анализ

Это программное обеспечение фокусируется на изучении элементов управления, используемых в структуре вызова, анализе потока управления и анализе перехода состояния. Структура вызова связана с моделью путем идентификации вызывающего и структуры вызова. Вызывающая структура может быть процессом, подпрограммой, функцией или методом. Анализ потока управления проверяет последовательность передач управления. Кроме

того, это неэффективные конструкции в модели. Создается граф модели, в котором условные ветви и соединения модели представлены узлами.

- Анализ данных

Обеспечивает правильную работу с объектами данных, такими как структуры данных и связанные списки. Кроме того, этот метод также гарантирует, что определенные данные используются должным образом. Анализ данных включает в себя два метода, а именно анализ зависимости данных и анализ потока данных. Зависимость от данных необходима для оценки точности синхронизации между несколькими процессорами. Анализ потока данных проверяет определение и контекст переменных.

- Анализ неисправностей/отказоустойчивости

Он анализирует ошибки (неправильный компонент) и сбой (неправильное поведение компонента модели) в модели. Этот метод использует описание преобразования ввода-вывода для определения условий, которые являются причиной сбоя. Для определения отказов в определенных условиях проверяется спецификация конструкции модели.

- Анализ интерфейса

Это программное обеспечение проверяет интерактивные симуляции и модели распространения для проверки кода. Существует два основных метода анализа интерфейса: анализ пользовательского интерфейса исследует интерфейсы подмоделей и определяет точность структуры интерфейса. Анализ пользовательского интерфейса исследует модель пользовательского интерфейса и меры предосторожности, предпринятые для предотвращения ошибок во время взаимодействия пользователя с моделью. Этот метод также фокусируется на том, насколько точно интерфейс интегрирован в общую модель и имитационное моделирование.

Исходя из вышесказанного, ряд задач, выполняемыми статическими анализаторами, можно обозначить как:

- Поиск ошибок в программном коде, что является самой распространённой и основной задачей анализатора;
- Улучшение качества кода. Сюда относится повышение читаемости, поддерживаемости, уровня связности и т.д.
- Поиск уязвимостей по версии OWASP.

Разберём каждую задачу отдельно.

## Поиск ошибок в программном коде

Существует множество примеров ошибок, на которые может среагировать статический анализатор кода. Вот некоторые из них:

- Неопределённое поведение. Использование неинициализированных переменных, обращение к NULL-указателям;
- Нарушение алгоритма пользования библиотекой. Например, при использовании функции `fopen`, после обязательно должно последовать использование `fclose`;
- Переполнение буфера. Запись данных за пределами выделенного в памяти буфера;

```
void doSomething(const char* x)
{
    char s[40];
    sprintf(s, "[%s]", x);    // sprintf в локальный буфер, возможно переполнение
    ....
}
```

- Ошибки, связанные с некросплатформенностью;

```
Object *p = getObject();
int pNum = reinterpret_cast<int>(p);    // на x86-32 верно, на x64 часть указателя будет потеряна; нужен intptr_t
```

- Ошибки форматных строк. В функциях наподобие `printf` могут быть ошибки с несоответствием форматной строки реальному типу параметров;

```
std::wstring s;  
printf ("s is %s", s);
```

## Улучшение качества кода

В данном разделе пойдет речь о стандарте написания кода Python PEP8. В качестве демонстрационного статического анализатора было решено представить консольную утилиту `pycodestyle`, проверяющую Python-код на соответствие PEP8.

На проверку был создан скрипт `example.py`, намеренно содержащий нарушения PEP8.

```
import os  
import notexistmodule  
  
def Function(num,num_two):  
    return num  
  
class MyClass:  
    """class MyClass """  
  
    def __init__(self,var):  
        self.var=var  
  
    def out(var):  
        print(var)  
  
if __name__ == "__main__":  
    my_class = MyClass("var")  
    my_class.out("var")  
    notexistmodule.func(5)
```

После в консоли вводим следующую команду:

```
$ python3 -m pycodestyle example.py
example.py:4:1: E302 expected 2 blank lines, found 1
example.py:4:17: E231 missing whitespace after ','
example.py:7:1: E302 expected 2 blank lines, found 1
example.py:10:22: E231 missing whitespace after ','
example.py:11:17: E225 missing whitespace around operator
```

Как видно из скриншота, программа выявила нарушения стандарта, однако список ошибок на этом не заканчивается. Например, в 1-ой и 2-ой строках были использованы ненужные импорты пакетов. В 1-м случае модуль `os` не использовался, а во втором такого модуля просто нет в системе. Что, впрочем, неудивительно, ведь данная программа проверяет код именно на соответствие стилю написания, а не поиск синтаксических и прочих ошибок.

Стоит также упомянуть, что во многих современных средах разработки присутствует встроенный статический анализатор, позволяющий также контролировать написание кода, согласно определённому стандарту. В данном случае рассмотрим среду разработки PyCharm Community Edition. Ниже приведён скриншот файла, помещённого в IDE:

```
import os
import notexistmodule

def Function(num,num_two):
    return num

class MyClass:
    """class MyClass """

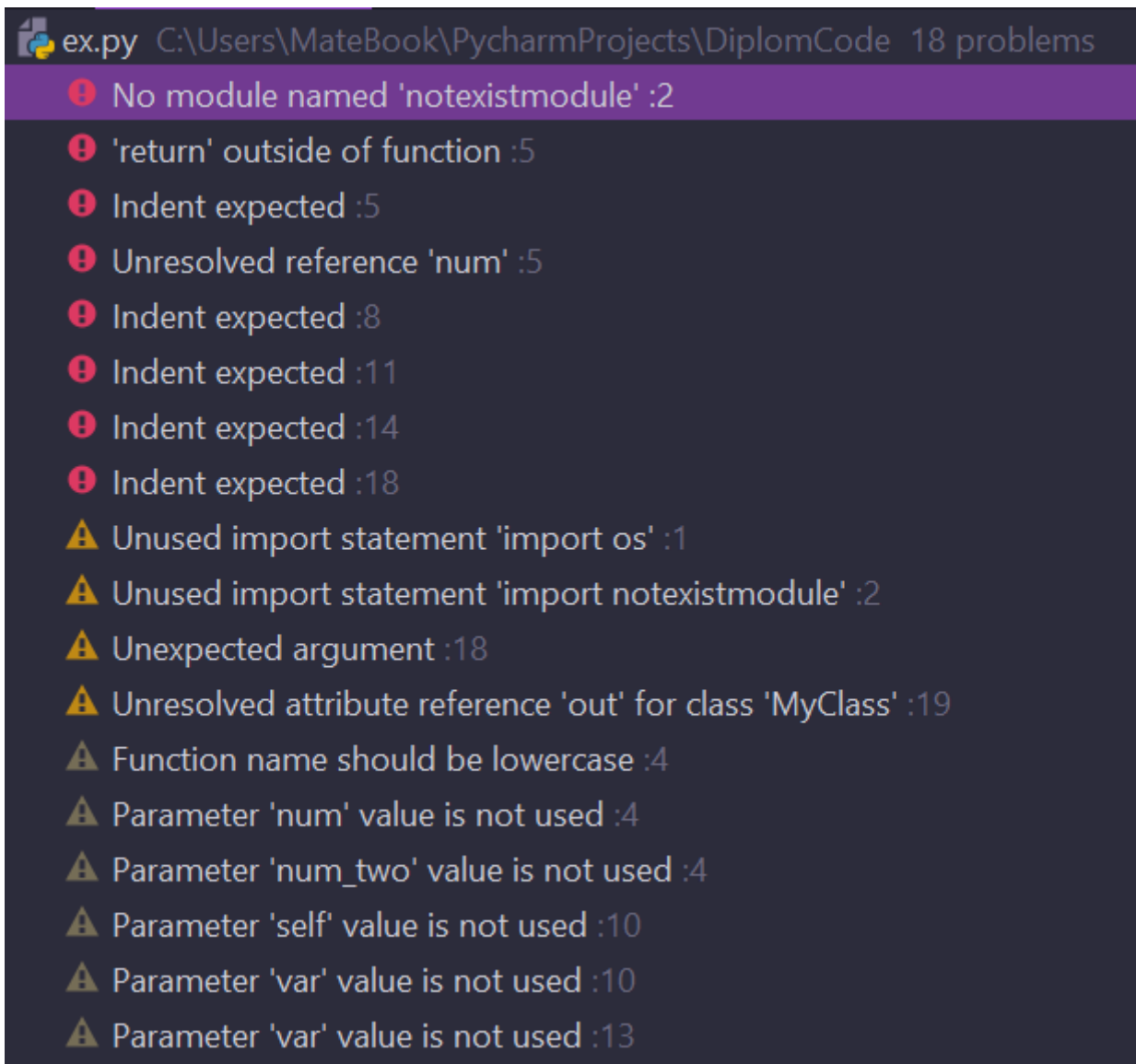
    def __init__(self,var):
        self.var=var

    def out(var):
        print(var)

if __name__ == "__main__":
    my_class = MyClass("var")
    my_class.out("var")
    notexistmodule.func(5)
```

Всего встроенный анализатор обнаружил 18 ошибок, различной степени критичности. Однако их хватает, чтобы программа не смогла даже запуситься.





The screenshot displays the 'Problems' window in PyCharm, listing 18 issues for the file `ex.py` located at `C:\Users\MateBook\PycharmProjects\DiplomCode`. The issues are categorized by severity: errors (red exclamation mark) and warnings (yellow triangle). The first error, 'No module named 'notexistmodule'', is highlighted in a purple bar. The list includes several indentation errors, unused imports, an unexpected argument, an unresolved attribute reference, and several parameters that are not used.

```
ex.py C:\Users\MateBook\PycharmProjects\DiplomCode 18 problems
❗ No module named 'notexistmodule' :2
❗ 'return' outside of function :5
❗ Indent expected :5
❗ Unresolved reference 'num' :5
❗ Indent expected :8
❗ Indent expected :11
❗ Indent expected :14
❗ Indent expected :18
⚠ Unused import statement 'import os' :1
⚠ Unused import statement 'import notexistmodule' :2
⚠ Unexpected argument :18
⚠ Unresolved attribute reference 'out' for class 'MyClass' :19
⚠ Function name should be lowercase :4
⚠ Parameter 'num' value is not used :4
⚠ Parameter 'num_two' value is not used :4
⚠ Parameter 'self' value is not used :10
⚠ Parameter 'var' value is not used :10
⚠ Parameter 'var' value is not used :13
```

## Поиск уязвимостей по версии OWASP

OWASP (Open Web Application Security Project) – открытая организация, сосредоточенная на анализе и улучшении безопасности программного обеспечения. Данная организация составила свой список из 10 самых опасных уязвимостей web-приложений.[4]

Статический анализ позволяет обнаружить следующие уязвимости OWASP:

- Внедрение SQL-инъекций (поиск предоставляемых пользователем строк, используемых в запросах);

- Межсайтовый скриптинг (XSS) (поиск манипуляций с DOM через innerHTML);
- Межсайтовая подделка запроса (CSRF) (поиск GET-запросов, меняющих состояния);
- Отражённый XSS (Поиск переменных, извлечённых из URL-адреса).

## Заключение

Польза от внедрения статического анализатора кода в разработку встроенного ПО очевидна, что и было доказано выше. Просуммировав всё сказанное ранее, можно выделить основные преимущества данного анализа:

- Использование статического анализа представляет собой тестирование вида, так называемого, «белого ящика», при котором после анализа даётся куда более полная картина о состоянии тестируемой системы;
- Возможность нахождения ошибок, паттерн которых до анализа был неизвестен;
- Возможность нахождения трудно заметных или даже неочевидных ошибок;
- Возможность внедрять статический анализ на самых ранних стадиях разработки, минимизируя стоимость найденных уязвимостей.

К сожалению, как и у любых вещей, у статического анализа имеются и недостатки, к которым можно причислить:

- Большое количество ложных срабатываний;
- Менее эффективная работа с динамическими языками (Python, PHP, JavaScript);

Тем не менее, несмотря на недостатки, статический анализ кода всё равно остаётся довольно эффективным средством контроля качества написания кода.

## **Список используемых источников**

1. Электронный ресурс - <https://www.geeksforgeeks.org/types-of-static-analysis-methods/>
2. Электронный ресурс - <https://habr.com/ru/company/solarsecurity/blog/439286/>
3. Электронный ресурс - <https://proglib.io/p/python-code-analysis>
4. Электронный ресурс - <https://owasp.org/Top10/>