**Задание 1.** Дайте ответы на следующие вопросы.

Вопрос 1. Назовите принципы объекто-ориентированного программирования (ООП), прямым образом относящиеся к языку программирования С:

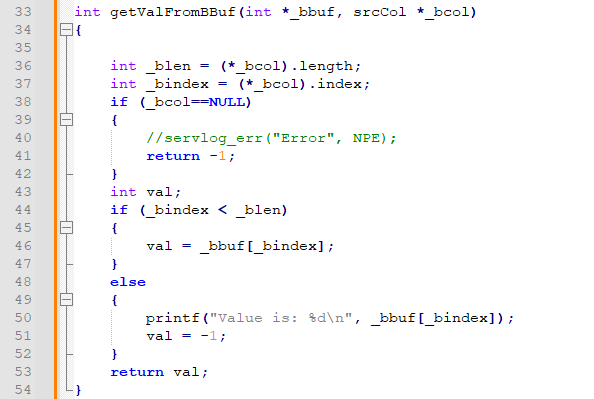
Среди выделеных принципов общепринятыми в ООП являются:

*А. Полиморфизм;  
B. Наследование;   
D. Инкапсуляция;*

Всех их вполне возможно в той или иной степени реализовать в СИ*.*

*См пример в https://github.com/Great-Ping/KeysCases/Question1.GoshaCar*

Вопрос 2. Представлен участок кода на языке программирования C:



В данном коде присутсвуют дефекты

*А. CWE-476 (36 – 37 строчки);   
D. CWE-125 (50 строка).*

В будущем возможна ошибка связанная с неверным применеием метода, в связи с тем что нет ни описани, ни в названии метода не указано что в случае исключительно ситуации будет возвращен “-1”.

В связи использованием для буфера простой структуры данных, с неизвестной длинной не исключено повторение ошибки CWE-125, либо получение CWE-476, в связи с тем что буффер будет являтся NULL

Примерное решение (Я не знаю причин использования структуры srcCol, поэтому выпилил ее за ненадобностью).

#include<stdio.h>

#include "GoshaCar.h"

typedef struct {

int\* ptr;

int length;

} Buf;

//Вернет значение из буфера либо -1

int getValFromBBufOrError(Buf \_bbuf, int index)

{

if (\_bbuf.ptr == NULL)

{

return -1;

}

if (index < \_bbuf.length && index >= 0)

{

return \_bbuf.ptr[index];

}

else

{

//не уверен что это нужно, ведь это побочный эффект

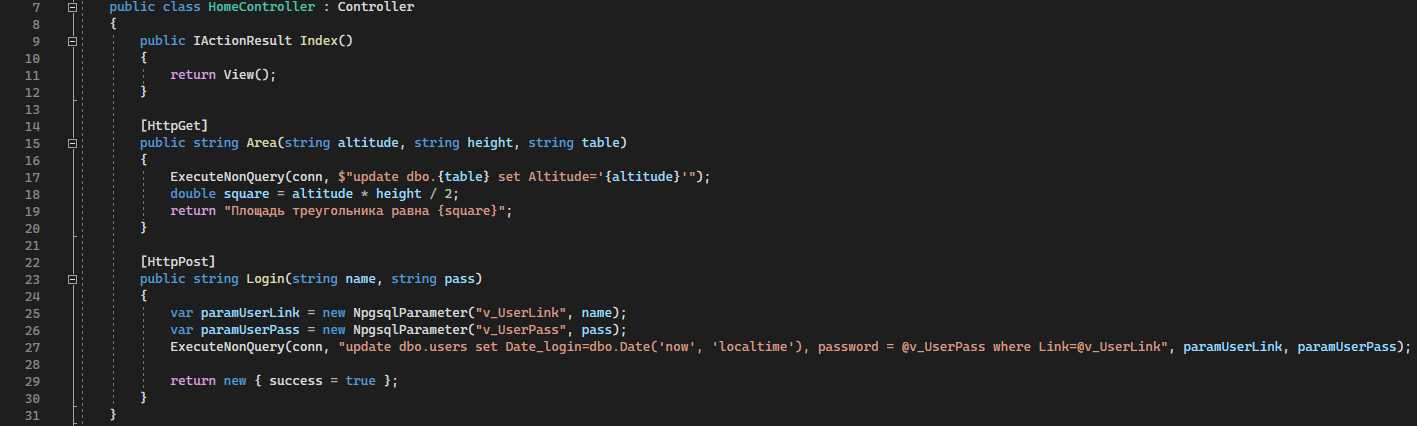
printf("Index out of range(0, %i)\n", \_bbuf.length);

return -1;

}

}

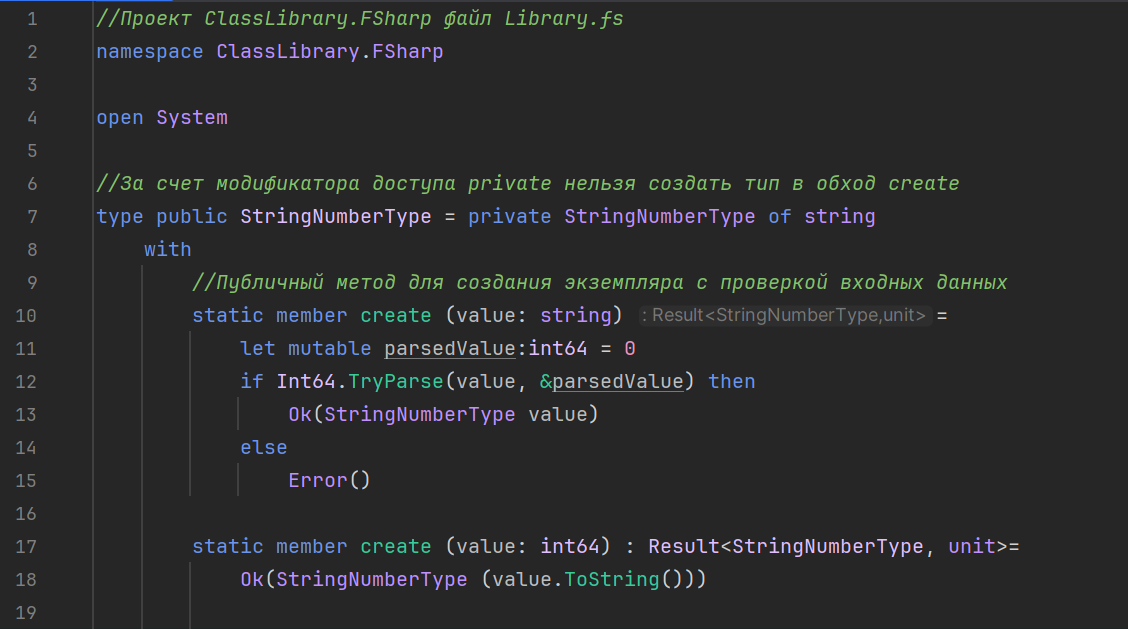
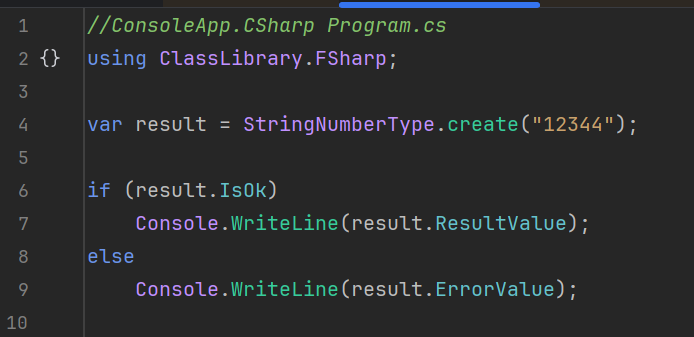
Вопрос № 3. Представлен фрагмент исходного кода, написанный на языке программирования С#. Проанализируйте фрагмент исходного кода и определите, содержится ли в нем какая-либо уязвимость/дефект/баг.



Крайне соминительные структуры данных возвращаемые контролером (полагаю на это делается скидка, т.к. это всего лишь простейший пример).

В методе Area возможна SQL инъекция. Я вижу три способа решения подобных проблем, все в конечном итоге сводятся к использованию провалидированых параметров:

1. Учитывая 18 строчку кода, наиболее подходящим решением для данного примера было бы изменить типы параметров метода на double, для table можно было бы завести отдельное перечисление (enum) со всеми допустимыми значениями
2. Использование записи подобной той что в методе логин, в наждежде на наличие экранирования строковых параметров внутри библиотеки.
3. Писать доменную область на F# :). Можно ограничить возможные значения для полей обернув их в специализированый тип для каждого поля свой (в данной ситуации uint/ulong, об этом позже), данный подход часто встречается в DDD и позволяет убедиться в валидности полей, ограничить возможные операции и значения полей. В родственном для С# языке F# как раз таки имеется очень удобный механизм оборачивания примитивных типов. Пример (https://github.com/Great-Ping/KeysCases/Question3.FSharpInterop):



Данный пример гарантирует исключение SQL инъекции, однако такое возможно не во всех ситуациях, так что лучше еще подстраховаться используя второй пункт.

Почему в качестве параметра v\_UserLink передается параметр name, я не знаю подробностей реализации, но не исключено наличие поля name в модели пользователя и что оно будет перепутано с полем Link.

Не стоит ли шифровать пароль пользователя, на случай утечки данных, в Microsoft.AspNetCore.Identity уже имеется готовый инструментарий в виде [PasswordHasher<TUser>](https://learn.microsoft.com/ru-ru/dotnet/api/microsoft.aspnetcore.identity.passwordhasher-1?view=aspnetcore-9.0) (по сути реализация PBKDF2)

Так же бы обратил внимание на выделение Методов и Классов средой разработки, в в данной ситуации белый цвет может указывать на то что не был произведен нормальный семантический анализ (возможно вручную отключено отображение ошибок), однако исходя из того что мы видим я бы присмотрелся к следующим моментам: возможно не подключены пространства имен Microsoft.AspNetCore.Mvc и Npgsql, а так же не реализован метод ExcecuteNonQuery.

Вопрос № 4. Выберите флаги безопасности cookie:

*А. Secure*

*C. HttpOnly;*

Вопрос № 5. Укажите существующие разновидности XSS:

*А. DOM-based;*

*B. Reflected;*

*D. Stored;*

Вопрос № 6. Какие существуют способы защиты от SQLi

*А. Параметризованные запросы;*

**Задание 2.** Реализуйте предложенный сценарий (рекомендуется реализовать в ОС Linux Ubuntu версии не ниже 20.04).

1. Установить docker (Docker Engine, <https://docs.docker.com/engine/install/ubuntu/>).
2. Сформировать докерфайл, запустить контейнер, войти в контейнер.
3. В контейнере:
   1. Выполнить загрузку файлов исходных текстов веб-сервера nginx;
   2. Выполнить компиляцию загруженных файлов исходных текстов nginx;
   3. Запустить скомпилированный веб-сервер, убедиться в его работе.

По результатам выполнения сценария, требуется:

* Задокументировать перечень применяемых команд в ходе реализации сценария;
* Прикрепите промежуточные результаты выполненных действий в виде скриншотов (на каждое выполненяемое действие).

**Задание 3.** Выполните [тестовый пример](https://github.com/antonio-morales/Fuzzing101/tree/main/Exercise%201) по выявлению уязвимости CVE-2019-13288 в консольной утилите XPDF версии 3.02 с применением фаззера AFL++ по следующему сценарию:

1. Компиляции исходных файлов XPDF с инструментацией;
2. Статуса фаззера после нахождения падения и остановки;
3. Воспроизведения выявленного аварийного завершения XPDF;
4. Выявить проблемное место в исходном коде XPDF, где возникла ошибка.

По результатам выполнения сценария требуется:

* Задокументировать перечень применяемых команд в ходе реализации сценария с дополнительных описанием выполняемых действий;
* Прикрепите промежуточные результаты выполненных действий в виде скриншотов (на каждое выполненяемое действие).