**Ответы на тестовое задание Стажировки Cloud.ru**

**Данные кандидата:**

1. Сафаргалиев Данил
2. +7-(910)-440-44-12
3. [danilsafargaliev@yandex.ru](mailto:danilsafargaliev@yandex.ru)
4. Ник в Телеграме: @Patrick89077

**1 ВОПРОСЫ ДЛЯ РАЗОГРЕВА**

1. **Расскажите, с какими задачами в направлении безопасной разработки вы сталкивались?**

Во время учёбы на практике занимался такими вещами как, Оценка уязвимостей веб-приложений и их исправление.

Анализировал код на предмет уязвимостей, таких как XSS и SQL-инъекции и т. д.

И внедрял меры безопасности, таких как мониторинг атак и анализ журналов, также изучал такие вещи как STRIDE, CI CD и OWASP, Juice Shop (уязвимый сайт).

1. **Если вам приходилось проводить security code review или моделирование угроз, расскажите, как это было?**

Доводилось проходить моделирование угроз, то есть потенциальные угрозы для системы, идентифицируются слабые места и разработка мер для их устранения.

И частично проведение Code Review ,то есть он позволяет обнаружить потенциальные уязвимости в исходном коде приложения и также предотвратить возможное атаки.

1. **Если у вас был опыт поиска уязвимостей, расскажите, как это было?**

Прежде всего поиск уязвимости требует внимательности, тщательности и знания специфики различных типов атак.

И поиск уязвимостей часто означает анализ системы на предмет существующих уязвимостей. И по итогу их эксплуатацию и рекомендации по их устранению.

1. **Почему вы хотите участвовать в стажировке?**

Стажировка может даст мне возможность изучить передовые методы обеспечения безопасности информации, научиться применять их на практике и наладить контакты с профессионалами в это области

**2 SECURITY CODE REVIEW**

**ЧАСТЬ 1. SECURITY CODE REVIEW: GO**

**В данном фрагменте присутствуют следующие уязвимости:**

1. **query := fmt.Sprintf("SELECT \* FROM products WHERE name LIKE '%%%s%%'", searchQuery)**

Она означает необработанные пользовательские данные в SQL-запросе.

То есть уязвимость заключается в том, что переменная searchQuery, полученная из пользовательского запроса, подставляется напрямую в SQL-запрос без должной обработки. Это может привести к SQL-инъекции, когда злоумышленник может выполнить вредоносный SQL-код.

Последствия эксплуатации уязвимости: злоумышленник может осуществить SQL-инъекцию, получить несанкционированный доступ к базе данных, изменить данные или даже удалить их.

**Способы исправления уязвимости:**

- Использовать параметризованные запросы, чтобы передавать параметры отдельно от запроса и избежать конкатенации строк. Например, использование подготовленных запросов.

- Использовать ORM (Object-Relational Mapping) для работы с базой данных, что позволит избежать проблем с обработкой пользовательского ввода вручную.

Наилучшим способом исправления данной уязвимости будет использование параметризованных запросов или ORM.

Параметризованные запросы предпочтительны, поскольку они позволяют явным образом передавать параметры без возможности модификации структуры самого запроса, что обеспечивает большую безопасность.

ORM также может быть хорошим вариантом, если в проекте используется активное взаимодействие с базой данных через объекты и методы, а не напрямую с SQL-запросами.

1. **db, err = sql.Open("mysql", "user:password@/dbname")**

Проблема заключается в том, что хранение происходит прямо в исходном коде. Если злоумышленник получит доступ к нашему исходному коду (например, через утечку или же несанкционированный доступ к файлам), то он сможет увидеть наш пароль к БД.

**Способы исправления уязвимости:**

- Использовать переменные среды для хранения конфиденциальной информации, такой как пароли. Различные платформы предоставляют средства для безопасного хранения конфиденциальных данных.

- Использовать инструменты управления секретами, такие как Vault

Исправление уязвимости путем удаления пароля из кода и использования безопасного механизма хранения конфиденциальных данных будет наилучшим решением в данном случае.

**ЧАСТЬ 2 SECURITY CODE REVIEW: PYTHON**

**ПРИМЕР №2.1**

from flask import Flask, request

from jinja2 import Template

**app = Flask(name)**

@app.route("/page")

def page():

name = request.values.get('name')

age = request.values.get('age', 'unknown')

**output = Template('Hello ' + name + '! Your age is ' + age + '.').render()**

**return output**

**if name == "main":**

app.run(debug=True)

Прежде всего в это коде не только обнаружены уязвимости и опечатки в имении переменной или же переменной которые нету в deg page(), то есть return output **(ошибки будут выделены синим, а красные уязвимости).**

Эти ошибки, проще говоря не запустят код.

Уязвимости же к инъекциям в шаблоне: Использование шаблонизатора Jinja2 без должного предварительного экранирования или фильтрации данных (например escape или safe фильтры), что может привести к уязвимости XSS или же SSTI (далее будет подробнее).

**Далее хочется сказать про SSTI**

output = Template('Hello ' + name + '! Your age is ' + age + '.').render()

Уязвимость SSTI возникает, когда злоумышленник может внедрить и выполнить произвольный код в контексте шаблонизатора.

И по итогу эта строчка потенциально рискованная, так как она использует значения name и age, полученные из запроса, напрямую в шаблоне без должного экранирования или фильтрации.

Также для выявления SSTI существуют инструменты, например Tplmap помогает в использовании уязвимостей внедрения кода и шаблонов на стороне сервера с помощью нескольких методов выхода из изолированной среды для получения доступа к базовой операционной системе.

**Исправленный вариант кода:**

Конечно, можно сделать куда лучше (т. к. я только учусь), но благодаря этому можно избежать этих уязвимостей XSS и SSTI

from flask import Flask, request

from jinja2 import Template, escape

app = Flask(\_\_name\_\_)

@app.route("/page")

def page():

name = request.values.get('name') or 'Unknown'

age = request.values.get('age') or 'Unknown'

template = Template('Hello {{ name }}! Your age is {{ age }}.')

output = template.render(name=escape(name), age=escape(age))

return output

if \_\_name\_\_ == "\_\_main\_\_":

app.run(debug=True)

**ПРИМЕР №2.2**

from flask import Flask, request

**import subprocess**

**app = Flask(name)**

@app.route("/dns")

def dns\_lookup():

hostname = request.values.get('hostname')

**cmd = 'nslookup ' + hostname**

**output = subprocess.check\_output(cmd, shell=True, text=True)**

**return output**

**if name == "main":**

app.run(debug=True)

В этом коде также если переменные, которые записаны неправильно, что приведёт к не запуску кода **(эти строки будут выделены синим)**.

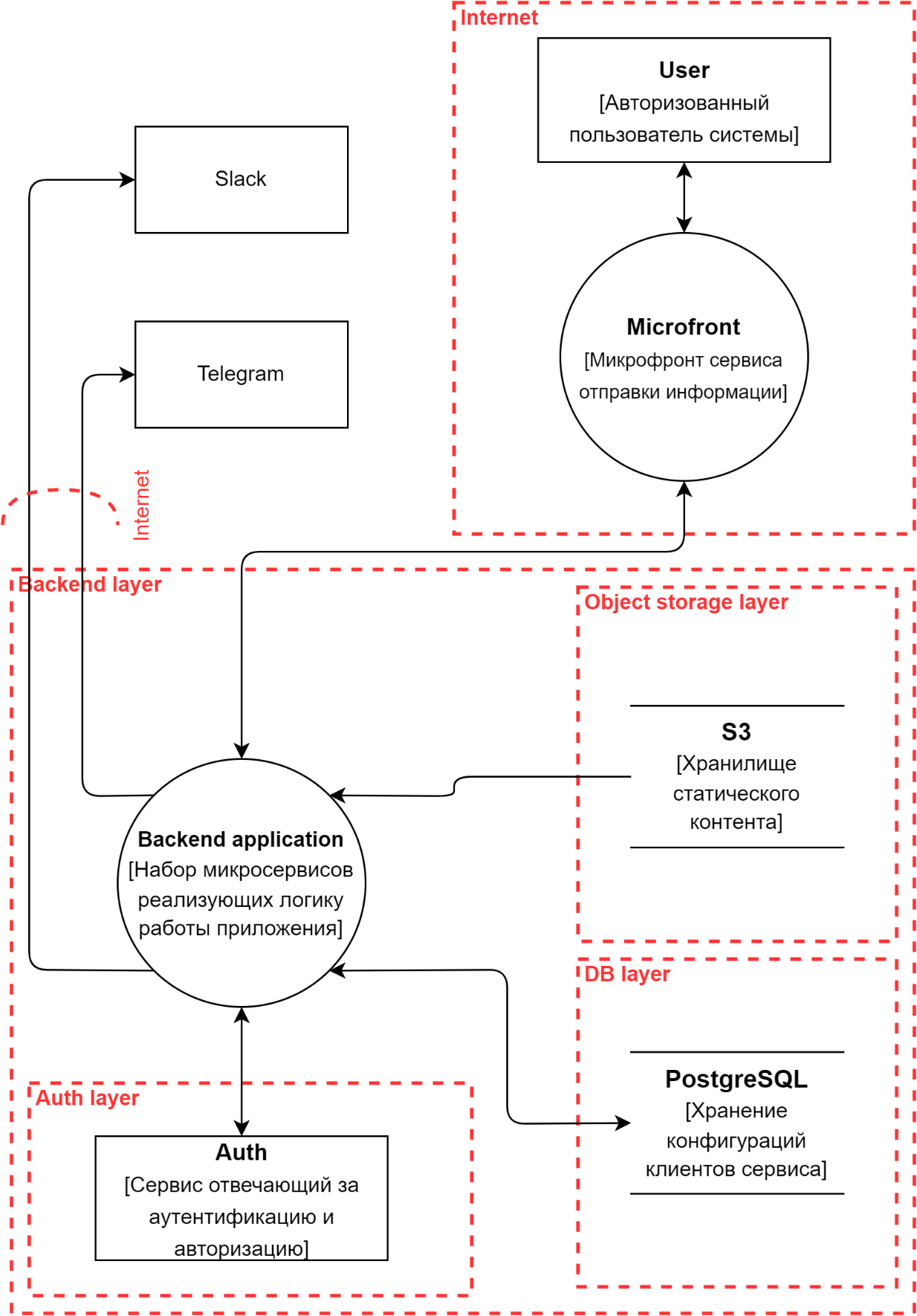
Также в этом коде присутствуют уязвимости **они будут выделены красным в коде.**

Прежде всего главная уязвимость это Comman Injection, то есть значение переменной hostname вставляется непосредственно в команду nslookup. То есть злоумышленник может использовать специально сконструированное значение hostname для выполнения произвольных команд на сервере (в командной строке).

Для решения уязвимости лучше использоваться библиотеки, которые обеспечивают безопасное выполнение внешних команд.

**3 МОДЕЛИРОВАНИЕ УГРОЗ**

Далее будет показана диаграмма потоков данных (Data Flow Diagram, DFD) сервиса, обеспечивающего отправку информации в Telegram и Slack:



Во-первых, у каждого сервиса существуют какие проблемы, нету такого сервиса, где бы ни нарушилась безопасность.

1. **User [Авторизованный пользователь системы]:**

* 1. У пользователя могут перехватить личные данные, то есть злоумышленники могут попытаться перехватить личные данные пользователя (логины, пароли, финансовые данные и т. д.), с помощью фишинга, вредоносного ПО, физический контакт, подслушать или атака на сеть и т. д.;
  2. Атака на уровне приложений: пользователь может столкнуться с угрозами, связанные с уязвимостями в самом приложении или серверной части, что может привести к краже данных
  3. Утечка данных: то есть возможность утечки данных из-за атаки на серверы приложения или неправильной настройки безопасности.

**Для защиты прежде всего можно использовать двухфакторную аутентификацию, обновлять ПО или же следить за активностью в аккаунте.**

1. **Microfront [Микрофронт сервиса отправки информации]:**
   1. Уязвимости в микросервисе: уязвимости в самом сервисе, включая инъекции, переполнения буфера и другие виды атак.
   2. Недостаточная безопасность сети
   3. DDoS атаки

**Для защиты и минимизации угроз необходимо проводить аудит безопасности, также обновлять ПО, использовать шифровку данных и контролировать доступ к сервису.**

1. **S3 [Хранилище статического контента]**
   1. Ошибки конфигурации: они считаются настраиваемыми, но они также создают риск неправильных настроек облачной безопасности
   2. Недостаточная видимость: то есть это модель совместной ответственности за облако, в соответствии которой облачный клиент несёт частичную ответственность за безопасность своей инфраструктуры.
   3. Вредоносные загрузки: проще говоря они не настроены на запрет доступа, то есть они общедоступные, что означает, что любой может получить к ним доступ напрямую из Интернета.

**Для обеспечения безопасности прежде всего стоить уделить вниманию управления доступом, принудительное использование наименьших привилегий, шифрование данных и внедрение MFA (Многофакторной аутентификации).**

1. **PostgreSQL [Хранение конфигурации клиентов сервиса]**
   1. Кража данных
   2. Несанкционированный доступ
   3. Неправильное использование

**Для защиты этого хранилища нужна правильная настройка параметров безопасности и авторизации доступа к базе данных. Мониторинг PostgreSQL. Резервное копирование данных для обеспечения их восстановлениях в случае утери или повреждения.**

1. **Auth [Сервис отвечающий за аутентификацию и авторизацию]**

Прежде всего этот сервис играет важную роль в обеспечении безопасности веб-приложений и сервисов. По сути, тут главная угроза — это нарушение конфиденциальности, и чтобы решить это я приведу следующие пункты:

**- Механизмы аутентификации, такие как пароль, многофакторная аутентификация, OAuth и OpenID Connect, должны быть правильно настроены.**

**- Защита от перебора паролей и других атак на аутентификацию.**

**- Четкое определение прав доступа для различных ролей пользователей.**

**- Контроль доступа к конфиденциальным данным и функциональности приложения.**

**- Мониторинг активности пользователей и обнаружение подозрительных действий.**

**- Активное обнаружение и устранение уязвимостей, связанных с аутентификацией и авторизацией.**

И все эти вещи в backend application которые реализуют логику работы приложений.

**Напишите список уточняющих вопросов, которые вы бы задали разработчикам данного сервиса?**

Прежде всего я бы задал такие важные вопросы:

1. Настроен ли White List в боте телеграмма?
2. Не получит ли данные другой пользователь?
3. И как злоумышленник может получить эти данные?
4. Какие уязвимости прежде всего могут существовать в этого бота?