Здание №1

Пример 1

В первом примере представлен код загрузки файла с помощью фреймворка Flask. Если в URL эндпоинтом будет “/share” выполниться функция sharedFiles, если ендпоинт-“/” выполниться функция index. Функция share берёт файл и возвращает прочитанный файл при этом обрабатывает путь к файлу на сервере (Get запрос). В функции индекс идёт логирование, клиент отправляет post запрос с файлом. После в качестве ответа возвращается html строка и вызывается функция Upload, в которая отображает форму загрузки файла.

<form action = "http://localhost:5000/" method="POST" enctype="multipart/form-data">
В данной строчке не проверяется тип файла. Есть возможность загрузить php, js скрипты, даже без изменения метаданных о файле. Уязвимость-File Upload(SSTI).

getFile = request.args.get("filename").replace('/','').replace('\\', '')

Также есть возможность эксплуатации Directory Traversal. При вводе var\/\www\/\file.txt есть возможность получить данные из файла по пути var/www/file.txt. Уязвимость-LFI.

Критичность в данном примере высокая, так как нарушается целостность и конфиденциальность. Есть возможность повлиять на поведение системы или получить данные как системы, так и других пользователей.

Чтобы предотвратить File Upload стоит реализовать проверку файлов и размещать файлы в изолированной среде.
Чтобы предотвратить LFI стоит фильтровать пользовательский ввод, ограничить права доступа пользователям, использовать белые списки.

Пример 2

Второй пример представляет собой форму подписки на рассылку.

email = '<?php echo htmlspecialchars($_GET['email']);?>'
Уязвимость-XSS. Htmlspecialchars недостаточно, чтобы избежать xss.

<script>function a() {alert('xss');}</script>

email = '<?php echo htmlspecialchars($_GET['a()']);?>'
Данный скрипт несмотря на проверку исполнится.

Сложно определить критичность из-за недостатка информации о системе.

Чтобы предотвратить XSS стоит реализовать защиту от внедрения скриптов, ограничить права доступа к данным, блокировать запрос, который содержит символы <,>.

Пример 3

В третьем примере происходит создание html страницы с формой и js скриптом, который ожидает получения сообщения с данными и выводит их на этой странице . Когда происходит событие

message, происходит подключение к серверу. В качестве аргумента передаётся event, который содержит адрес, который и выводится на странице.
Уязвимость-XSS.

<script>

alert(XSS');

</script>
Нет никакой проверки вводимых данных.

Критичность может быть высокой в зависимости от реализации системы.

Чтобы предотвратить XSS стоит реализовать защиту от внедрения скриптов, ограничить права доступа к данным, блокировать запрос, который содержит символы <,>.

Пример 4

В данном примере представлено получение панели администратора по куке. Идёт проверка является ли пользователь администратором. Роль определяется кукой. Также админ проверяется по ip адресу.
Данный код имеет уязвимость Cookie Spoofing и BrokenAuth.

Ip адресс должен быть 127.0.0.1 или localhost. То есть можно использовать заголовок X-Forwarded-For:127.0.0.1. На счёт куки точно не могу сказать h.Role.Value нужно подменить на admin.
Критичность высокая, так как в зависимости от реализации сервиса может пострадать и конфиденциальность, и целостность, и доступность.

Для более безопасного доступа к админке можно реализовать аутентификацию и авторизацию.

Пример 5

5-й пример представляет из себя проверку некого кода. Сначала функцией GetOPT принимается код пользователя. Далее принимается пользовательский ввод. Пользователю даётся 3 попытки ввода. Если значение root не нулевое или значение ввода и кода совпали, то загружается некая панель.
В целом если этот код можно зареверсить. If в данном случае при дизассемблировании будет командой JZ. Можно поменять либо флаг jz либо заменить его на jnz. Также можно поменять значение root. Нету защиты от реверса.
Но в рамках задания уязвимостью является BufferOverflow. Функция gets уязвима. Если передать в неё код, состоящий более чем из 4 символом этом, может привести к непредсказуемым последствиям. Точно сказать не могу, что произойдёт из-за недостатка опыта, но могу предположить. Язык си низкоуровневый и взаимодействует с памятью через указатели. Таким образом, скорее всего, либо все данные, хранящиеся после места в памяти, куда gets сохранит пароль либо сдвинутся, либо символы после 4 буду продолжать записываться в память. Точно последствия предсказать не могу. Также функция strcmp уязвима для переполнения буфера, но в данном случае непонятно, как это эксплуатировать.
Чтобы предотвратить данную уязвимость стоит использовать безопасные функции. В данном случае можно использовать fgets. А также стоит проверять количество вводимых символов.

Критичность высокая, так как может пострадать целостность системы.

Пример 6

В данном коде коде запускается сервер идёт настройка параметров id, username, session все параметры статические и берутся из бд(насколько я понял). Также идёт настройка CORS:

```
const headers = {
    "Access-Control-Allow-Origin": origin,
    "Access-Control-Allow-Credentials": "true",
    "Content-Type": "application/json",
  };
```

Отсюда и уязвимость CORS misconfig. Если значение origin = “” или “*”, то "Access-Control-Allow-Origin": origin позволяет любому домену делать запросы к серверу и получить доступ к его ресурсам.

Критичность высокая, так как могут быть нарушены все 3 критерия.

Пример 7

В данном коде идёт открытие файла с сервера. Идёт фильтрация пути файла. Отбрасываются большинство спец символов.

Уязвимость LFI и, возможно, Comandinjections.

Можно с помощью URL encoding или двойного енкодинга открыть файлы из других директорий. Можно при указания файла, использовать ввод такого типа: “file.txt’&whois--”

Таким образом критичность высокая наущаются все 3 критерия.

Для предотвращения этой уязвимости стоит ограничить доступ, поменять black list на white list запросов путей. Отбрасывать все символы “ ’ ”.

Пример 8

В данном случае идёт перенаправление на некий URL. Уязвимость-Open redirect.

http://example.com/?r=http%3a%2f%2fopen%2Ecom

Также здесь есть возможность эксплуатации XSS.

```
<script>
  alert(‘XSS');
</script>
```

Чтобы избежать этого стоит, как и в прошлом примере организовать white list, также есть вариант с добавление в blacklist символа %, а также стоит реализовать защиту от xss атак.

Критичность высокая, так как могут быть нарушены все 3 критерия.

Также я, возможно, ошибся с open redirect. Если пользователь перенаправляет себя же на какой-то другой сайт смысла в эксплуатации этой уязвимости нет.

Пример 9

Представленный код представляет Flask-приложение, которое обрабатывает HTTP-запросы и возвращает соответствующий HTML-контент в зависимости от запроса пользователя. На сколько я понимаю, код представляет собой некий сервис. В бд хранятся продукты и ссылки на их сайт.

Уязвимость-Open Redirect и SQl inj.
Например, можно создать продукт со ссылкой на вредоносный сайт.
Можно построить такой вектор атаки. Создать вредоносный сайт: http://tiazko.com. В поле inpt = html.escape(request.args.get('search')) вводим ‘UPDATE “products” SET column= http://tiazko.com where column=”a”(где a этозначение полученное из SHOW COLUMNS FROM products;). Однако если есть фильтрация sql запросов этот вектор может не сработать.

В целом экчплойт данной уязвимости будет работать по принципц подмены URL для перехода.

Критичность высокая, так как страдает целостность системы.