Race condition в веб-приложениях

Спасибо Константину Попову за помощь в подготовке!

Agenda

0x1 WTF Race condition

0x2 Примеры Server-side гонок

0x3 Поиск и эксплуатация гонок

0x4 Demo

0x1 WTF Race Condition

0x1 WTF Race Condition

Свойство системы, при котором её **поведение зависит от неконтролируемых внешних событий**

0x1 WTF Race Condition

Несинхронизированный доступ нескольких параллельных процессов к переменной, при котором как минимум один из процессов осуществляет запись

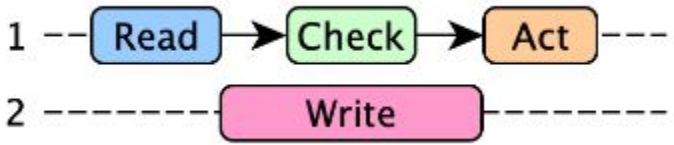
0x1 WTF Race Condition - причины

- Непредсказуемый порядок наступления (атомарных) событий
- Параллельное выполнение без использования необходимых примитивов синхронизации
- Один из возможных порядков выполнения приводит к **нежелательному** (wtf?) поведению
- Одновременное взаимодействие с **общими ресурсами** (какими?)
- Нарушение **инвариантов** (wtf?)

0x1 WTF Race Condition - виды (в веб-приложениях)*

Read - Check - Act (RCA/TOCTOU)

Читаем, проверяем условие, делаем действие. На момент действия истинность условия может измениться.



Read - Update - Write (RUW)

Читаем, меняем, записываем обратно. Параллельные чтения видят старое состояние, параллельные записи теряются.

**https://essay.utwente.nl/78020/1/Emous_van_MA_EEMCS.pdf*

0x1 WTF Race Condition - общие ресурсы

- ~~Общая память (общая переменная)~~
- **Объект в БД**
- Файл в ФС
- Сессия
- ...

0x2 Примеры*

* Публичные репорты с h1 (не мои)

0x2. Пример: reverb.com

https://hackerone.com/reports/759247

- Login
- Buy a gift card
- Redeem it multiple times in parallel
- ….
- PROFIT: *In my case I bought one gift card which was worth of **25$** and as we can see from the picture I was able to redeem it 7 times which makes 25*7 = **175$***.

Аналогичные примеры:

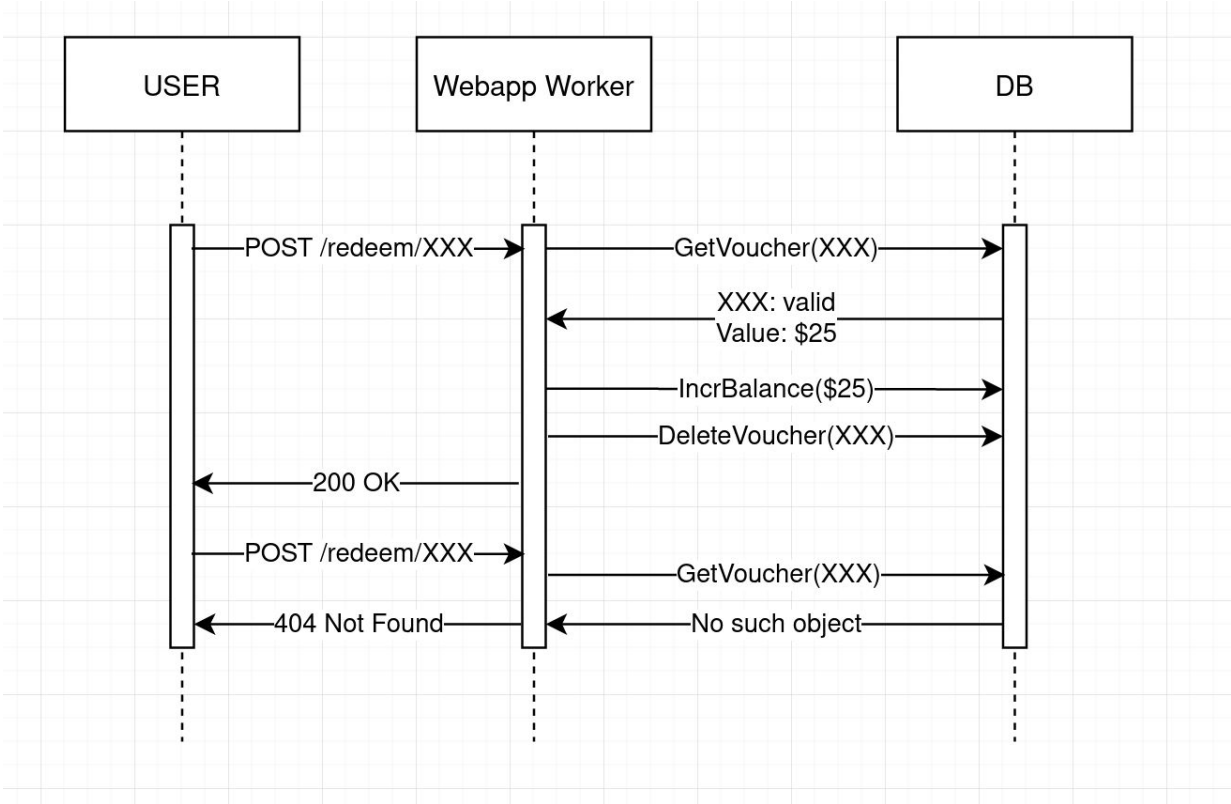
https://hackerone.com/reports/927384 (Follow одного и того же пользователя несколько раз, every.org)

https://hackerone.com/reports/165570 (получение welcome-credit несколько раз, slack)

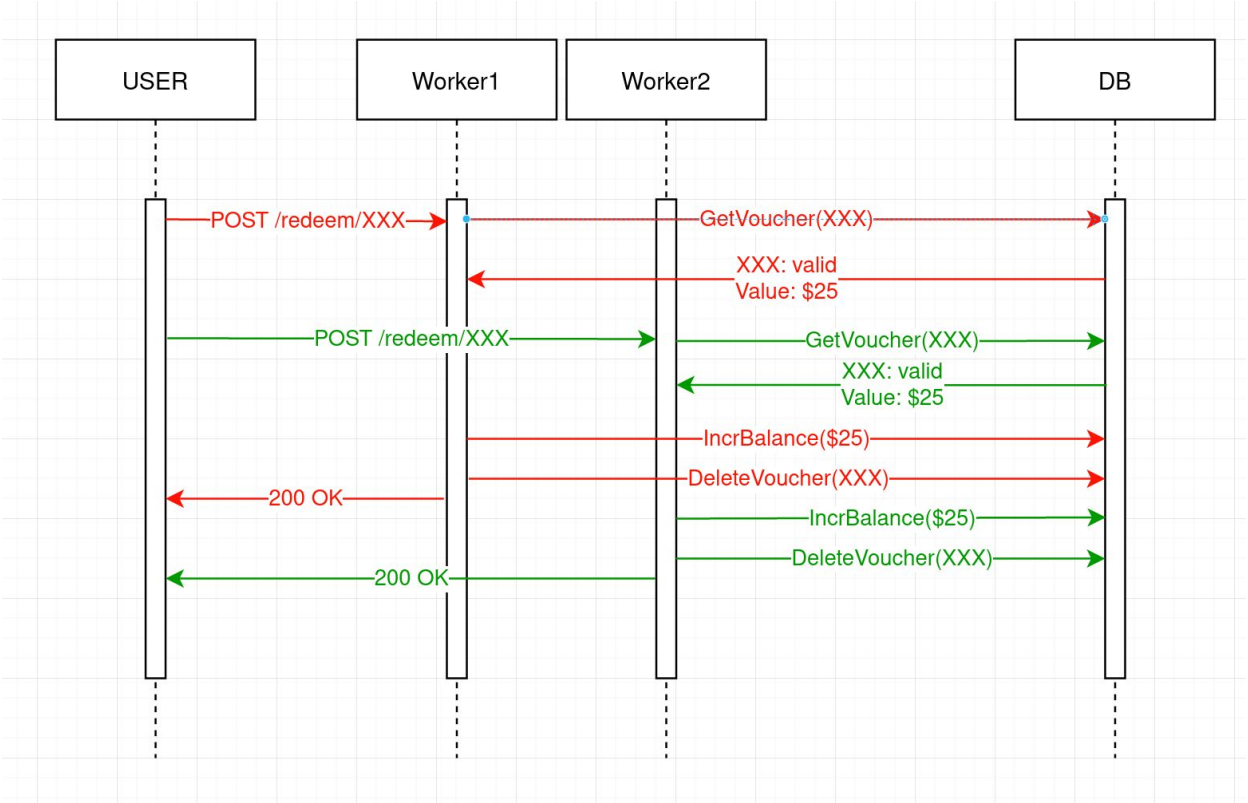
https://blog.avuln.com/article/4 (гонка при обмене code на access-токен, различные OAuth 2.0-провайдеры)

https://hackerone.com/reports/157996 (гонка на применение купонов)

0x2. Пример: reverb.com. Ожидаемый сценарий



0x2. Пример: reverb.com. **Нежелательное** поведение



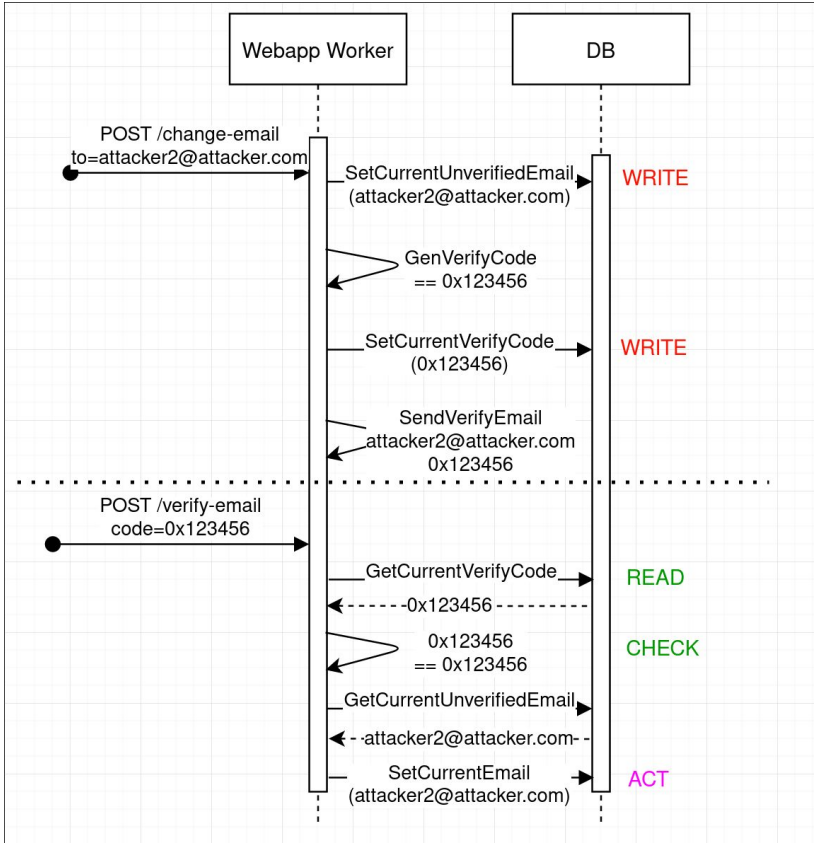
0x2. Пример 2: shopify

Ability to bypass partner email confirmation to take over any store given an employee email

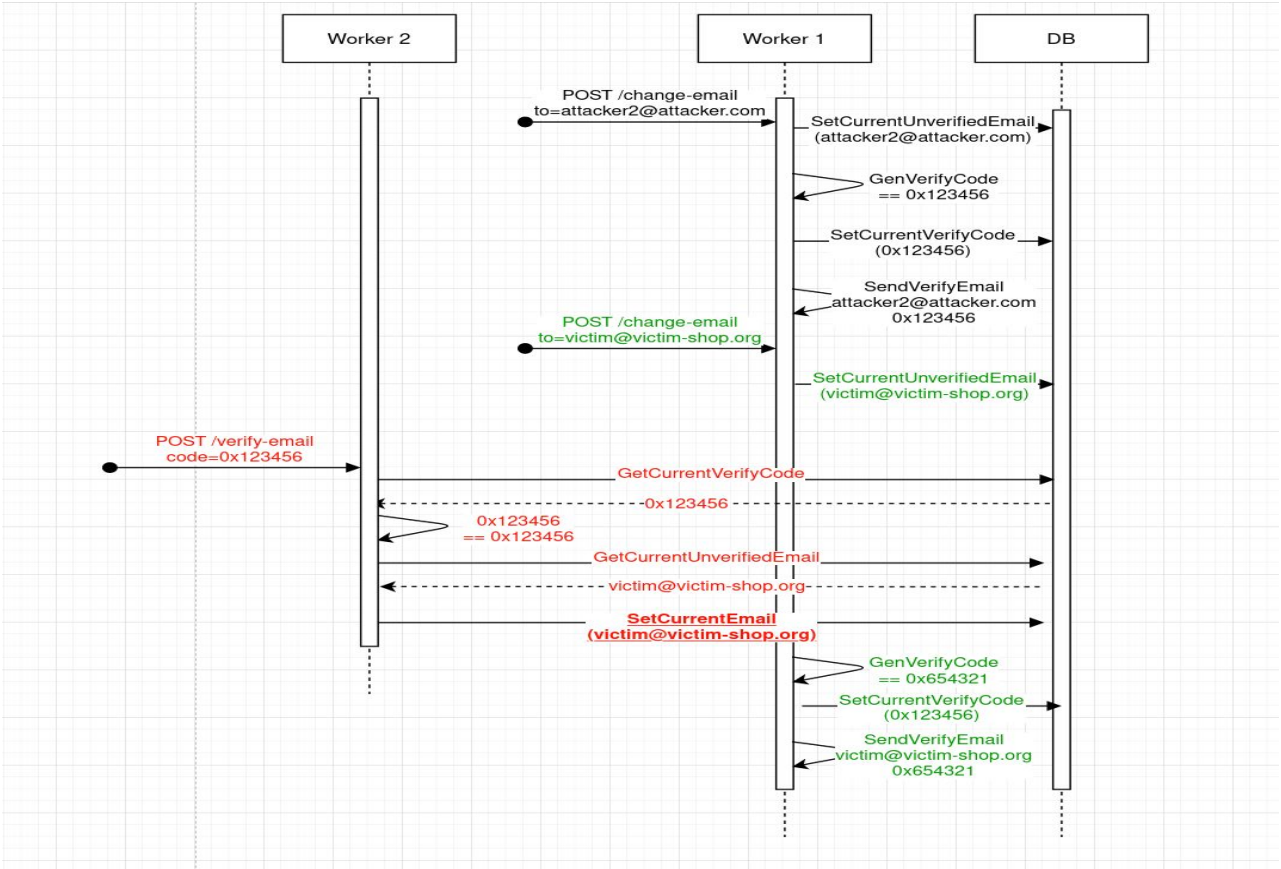
- victim@victim-shop.org - сотрудник в магазине victim shop.
- attacker@attacker.org - **партнер** shopify (помогает бизнесам создавать магазины или управлять ими).
- Партнеры могут **запрашивать доступ** к магазинам через специальное API. Владелец магазина должен подтвердить.
- Если email партнера совпадает с почтой одного из существующих сотрудников магазина, подтверждение не требуется.
- Есть функция смены почты, для завершения смены нужно перейти по ссылке из письма.

https://hackerone.com/reports/300305

0x2. Пример 2: shopify - ожидаемое поведение

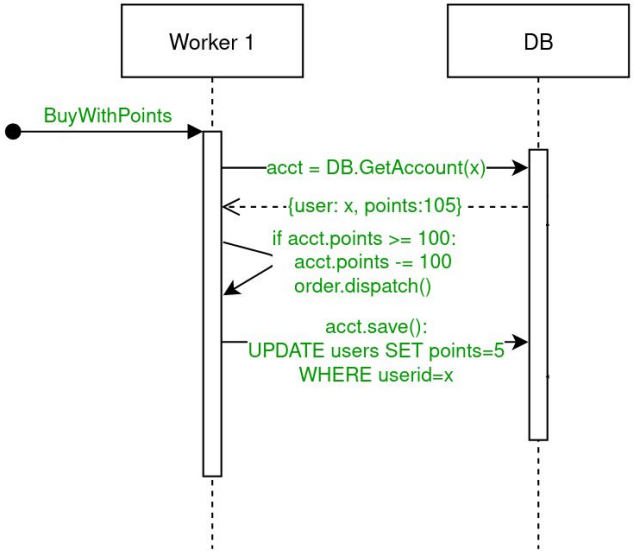


0x2. Пример 2: shopify - **нежелательное** поведение



0x2. Пример 3: Delivery club

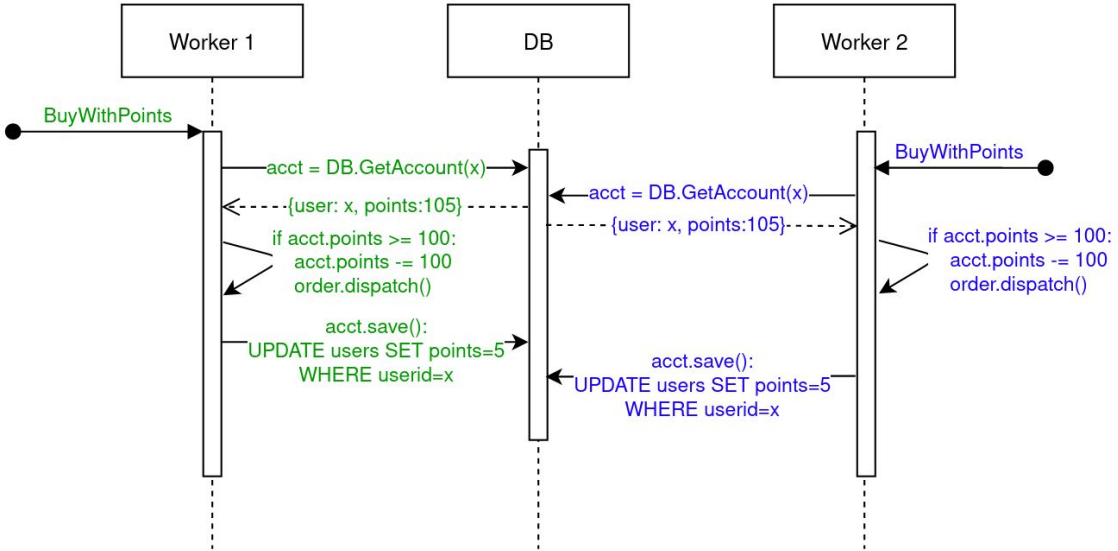
У меня на счету было 105 баллов. Запустив 50 запросов на покупку товара (за 100 баллов) в 50 потоков, я получил 7 ответов сервера о покупке. Выигрыш от эксплуатации уязвимости составил ***600 баллов****.*



https://hackerone.com/reports/700833

0x2. Пример 3: Delivery club

У меня на счету было 105 баллов. Запустив 50 запросов на покупку товара (за 100 баллов) в 50 потоков, я получил 7 ответов сервера о покупке. Выигрыш от эксплуатации уязвимости составил ***600 баллов****.*



https://hackerone.com/reports/700833

0x3 Поиск и эксплуатация

0x3 Поиск гонок - типичные сценарии

- Объекты “одноразового применения” - купоны, разовые ссылки, одноразовые коды;
- Счетчики (в том числе антибрутфорс);
- Операции с балансом (деньги или любой расходуемый/пополняемый ресурс - баллы, бонусы);
- Попытка нарушить инварианты уникальности, существования, единственности;
- Действия со сложной логикой (когда за один HTTP-запрос происходит много атомарных действий) - импорт/загрузка из файла, каскадные удаления, батч-действия и т.п.
- Действия, которые затрагивают несколько различных типов объектов.

0x3 Поиск гонок - методика?

- Нормальной методики поиска гонок не существует (ни blackbox, ни whitebox);
- Нет способа доказать, что некая функциональность не содержит гонок;
- Большинство гонок трудно или невозможно эксплуатировать, даже если они есть;
- Решающее значение имеет интуиция и хорошее общее понимание того, как работает приложение.

0x3 Поиск гонок - black-box методика (kind of)

1. **Изучаем приложение** - платформа, функциональность, виды объектов;
2. Выделяем функции и объекты, которые могут быть использованы для гонок, **формулируем** предположительный **сценарий**;
3. Выделяем **запросы или их последовательности**, которые участвуют в сценарии;
4. Делаем много запросов параллельно;
5. Оцениваем результат.

0x3 Про БД, ORM и транзакции

- На практике в 9 из 10 случаев объект гонки - это объект в БД;
- Возможность гонок на объекты в БД зависит от того,
  - … как модель данных отображается на структуру БД;
  - … какие инструменты БД и как использует ваш код (транзакции, select for update, lock row, атомарные операции);
  - … какие гарантии предоставляет сама БД (уровень изоляции*) - с точки зрения защиты от гонок минимально необходимым является уровень **repeatable reads**;
  - … использует ли код какую-то дополнительную синхронизацию, помимо средств БД.
- Использование **нормальных** ORM **правильным** образом в сочетании с **нормальной** настройкой СУБД как правило обеспечивает невозможность гонок на уровне **одной** сущности БД;
- Если речь о нескольких связанных сущностях разного типа, всё становится существенно сложнее и все обычно забивают.

* https://en.wikipedia.org/wiki/Isolation_(database_systems)#Isolation_levels

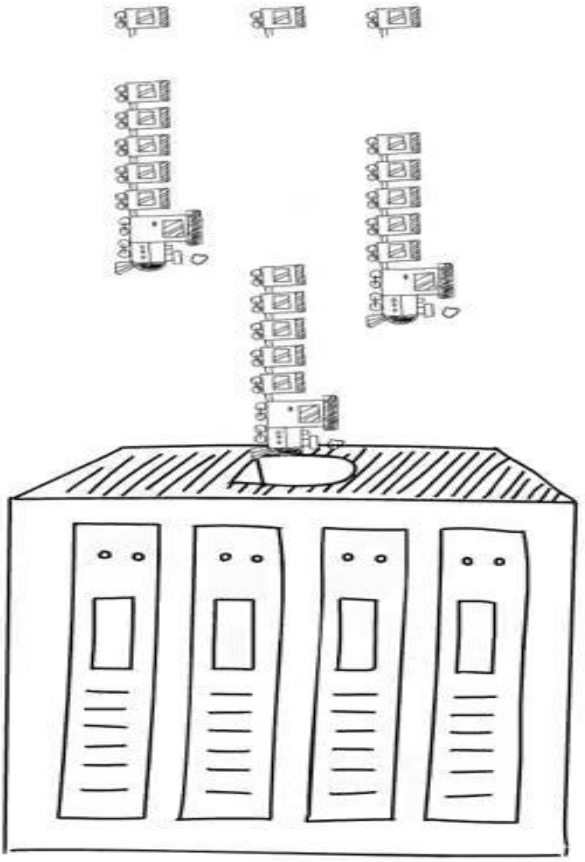
0x3 Эксплуатация - выполнение запросов

Нам могут потребоваться два примитива:

1. Сделать так, чтобы обработка запросов x, y, z произошла как можно более одновременно (вариант - с некой как можно более константной относительной задержкой);
2. Сделать так, чтобы несколько запросов выполнились один за другим, и между ними не попали бы другие;

0x03 Эксплуатация - выполнение запросов

- Одновременное выполнение: пользуемся тем, что запрос не будет обработан, пока не будет весь прочитан:
  - в N потоков посылаем все байты нужных запросов, кроме последнего;
  - досылаем одновременно последний байт во всех запросах.
  - Тулзы: turbo intruder (плагин для Burp Suite), racepwn, ~~ваш скрипт на питоне~~
- Последовательное выполнение: HTTP keep-alive (несколько запросов в одном TCP-соединении)



0x3 Эксплуатация - увеличиваем шансы

- Замедляем выполнение конкретного запроса (пример: deflate-бомба в PNG);
- Замедляем выполнение всего приложения в целом (загружаем базу апдейтами);
- Избегаем ненужных блокировок, используя отдельные сессии;
- Ищем благоприятное сетевое расположение;
- Тестируем в правильное время (в час пик или наоборот ночью).

0x4 Demo

https://github.com/computestdev/CompuRacer/tree/master/TestWebAppVouchers/app

0x5 Bonus: client-side race

0x5 Client-Side Race

- Есть client-side уязвимость - CSRF, XSS, небезопасный postMessage*;
- Окно возможностей по эксплуатации ограничено тем, насколько быстро выполнится действие в браузере пользователя;
- Злоумышленнику в контексте js-скрипта, выполняющегося в браузере пользователя в соседней вкладке, нужно выиграть гонку.

* postMessage race - см https://hackerone.com/reports/381356

0x5 Client-Side Race: пример