Тестовое задание на стажировку AppSecCloudCamp

Отчёт о выполненной работе

**1. ВОПРОСЫ ДЛЯ РАЗОГРЕВА**

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

С данными задачами встречаюсь впервые.

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

­

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

Участвовал в некоторых CTF, последний значительный был от Тинькофф в прошлом году. Проходил различные задания и комнаты, на таких сервисах как TryHackMe и HackTheBox. Вот ссылка на мой профиль THM: <https://tryhackme.com/p/Nmazgaleev>, сразу скажу, что давно не выполнял там задания.

* Почему вы хотите участвовать в стажировке?

На данный момент времени, я обучаюсь на 3 курсе по специальности «Обеспечение информационной безопасности автоматизированных систем» и хотел бы выбрать что-то определенное для себя. Меня всегда интересовала разработка, ещё со школы занимался программированием и изучал всё связанное с этим самостоятельно. На протяжении пары лет писал различные проекты: подобие физических движков, песочниц, симуляций различных физических явлений. Меня интересует, как информационная безопасность, так и разработка различных сервисов.

Через рекламу на веб-ресурсах узнал о стажировке в AppsSec, подал заявку. Я бы хотел связать свою дальнейшую карьеру с интересующими меня направлениями, поэтому я рассматриваю данную стажировку как возможность усовершенствовать свои существующие навыки и начать карьеру в AppsSec.

**2. SECURITY CODE REVIEW**

**Часть 1. Security code review: GO**

Рассмотрев представленный в примере код, я нашёл только одну уязвимость. В функции «searchHandler» выполняется запрос к базе данных, на выходе мы получаем список товаров. Сам SQL запрос можно увидеть на 38 строчке, где из заголовка GET-запроса приложение получает наименование товара и совмещает это с заготовленным запросом. Рассмотрев всё это, можно сказать, что данный фрагмент кода уязвим к SQLi. Данная уязвимость позволяет злоумышленнику изменять содержимое SQL запроса и получать данные, к которым он не имеет доступ.

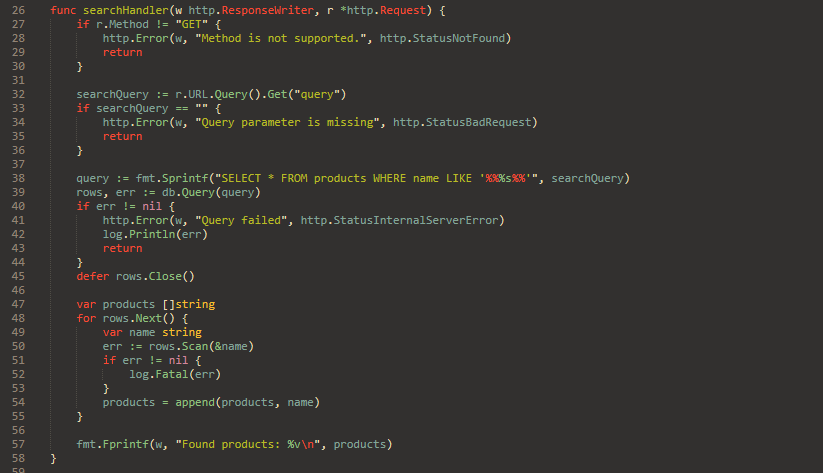


Рис.1 Функция, содержащая уязвимый фрагмент кода

Для устранения данной уязвимости можем предоставлять значения параметров SQL в качестве аргументов функции, так как пакет sql в golang осуществляет проверку на данную уязвимость. Решая данную задачу, я даже наткнулся на страницу документации golang, где представлен пример как не следует выполнять SQL запросы, ознакомиться можно по ссылке <https://go.dev/doc/database/sql-injection>. Исправленный фрагмент кода представлен на рисунке №.

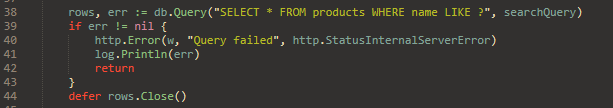


Рис.2 Исправленный фрагмент кода

**Часть 2. Security code review: Python**

Рассмотрим пример 2.1, код которого представлен на рисунке №. После анализа, я заметил возможность наличия уязвимости на строчках 9-11, где по значениям в заголовке GET-запроса с аргументами «name», «age» и «unknown» устанавливаются значения переменным «name» и «age». После чего данное веб-приложение отображает содержимое этих переменных без каких-либо проверок, что позволяет нам воспользоваться XSS-атакой.

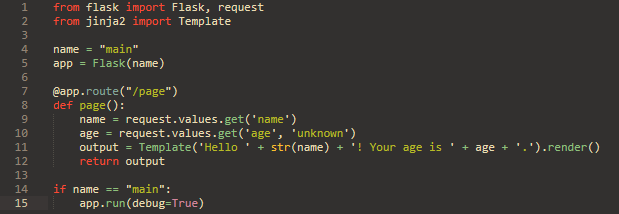


Рис.3 Код программы из примера 2.1

Запустив локально программу, я проверил наличие XSS-уязвимости, указав в качестве имени «<script>alert(document.cookie)</script>», что привело к результату, представленному на рисунке №. Данная уязвимость может позволить злоумышленнику провести кражу авторизационных cookie пользователей веб-приложения.

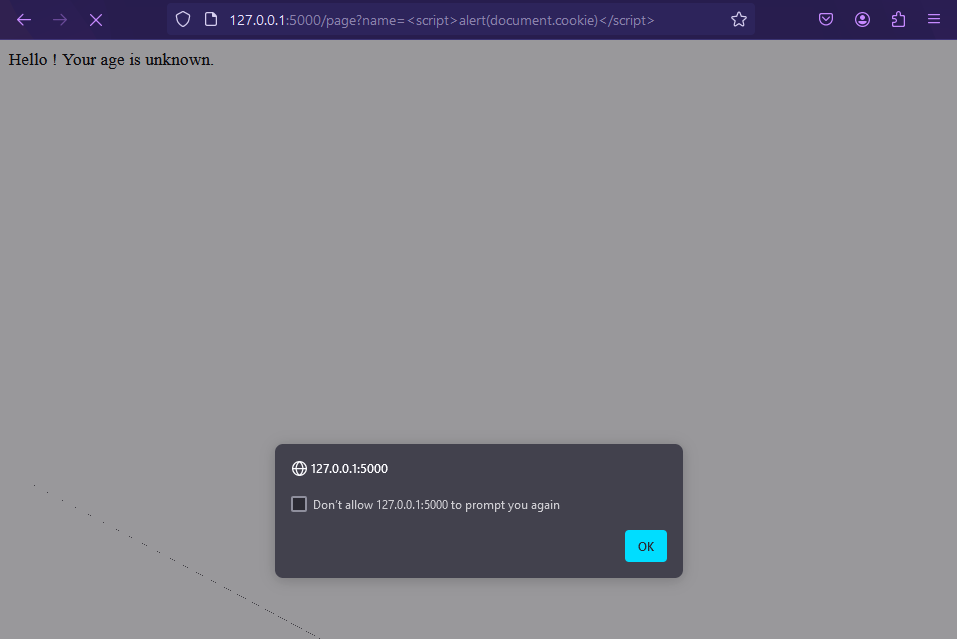


Рис.4 Результат выполнения XSS-атаки

Для устранения данной уязвимости я решил производить проверку вводимых параметров на наличие специальных символов. В нашем случае приложение получает имя и возраст, так что возможность удаления лишних символов из имени исключена. На рисунке № представлен код с проверкой вводимых значений. После чего я повторно запустил приложение и провёл попытку XSS-атаки, результат представлен на рисунке №.

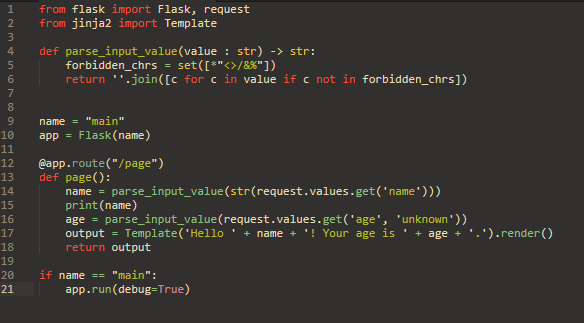


Рис.5 Исправленный код

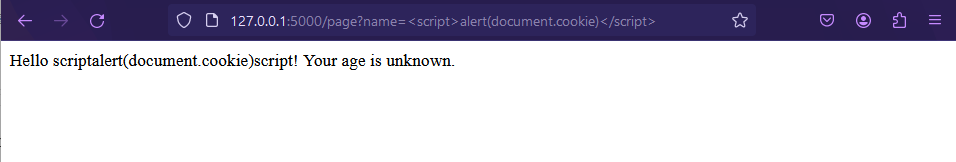


Рис.6 Результат выполнения повторной XSS-атаки

Также, одним из вариантов устранения уязвимости может использоваться библиотека escape. Она не позволяет выполнять код в данных, переданных пользователем. Реализация представлена на рисунке №, результат попытки атаки на рисунке №.

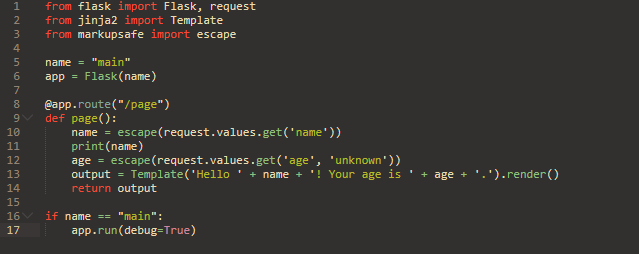


Рис.7 Исправленный код

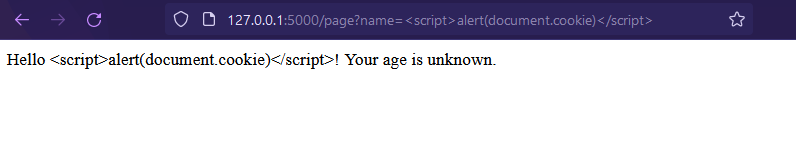


Рис.8 Результат выполнения повторной XSS-атаки

Если выбирать наилучший вариант реализации исправления данной уязвимости, то я бы выбрал второй вариант с использованием дополнительного функционала flask. По сути, обе реализации осуществляют схожие действия: они обе проверяют данные на наличие дополнительных символов, но реализация flask не удаляет эти символы, а заменяет их на безопасные для HTML-последовательности.

Рассмотрим пример 2.2, код которого представлен на рисунке №. На строках 8-11 я обнаружил возможную уязвимость. На 8 строчке приложение берёт значение «hostname» из заголовка GET-запроса пользователя, после чего совмещает с командой «nslookup» и запускает в командной строке. Результат выполнения команды отображается на сайте.

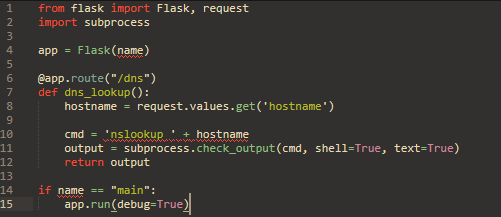


Рис.9 Код из примера 2.2

Запустив данное приложение локально, я отправил запрос, указав в качестве «hostname» значение «8.8.8.8 & ipconfig», тем самым получив информацию о сетевых устройствах системы, на которых работает данный сервис. Злоумышленник может воспользоваться данной уязвимостью для запуска любого кода на системе сервера, имея соответствующие права. Вкратце, получаем полный доступ к терминалу хоста, после чего можем даже настроить reverse shell, тем самым уже получаем полный доступ к системе с правами сервера.

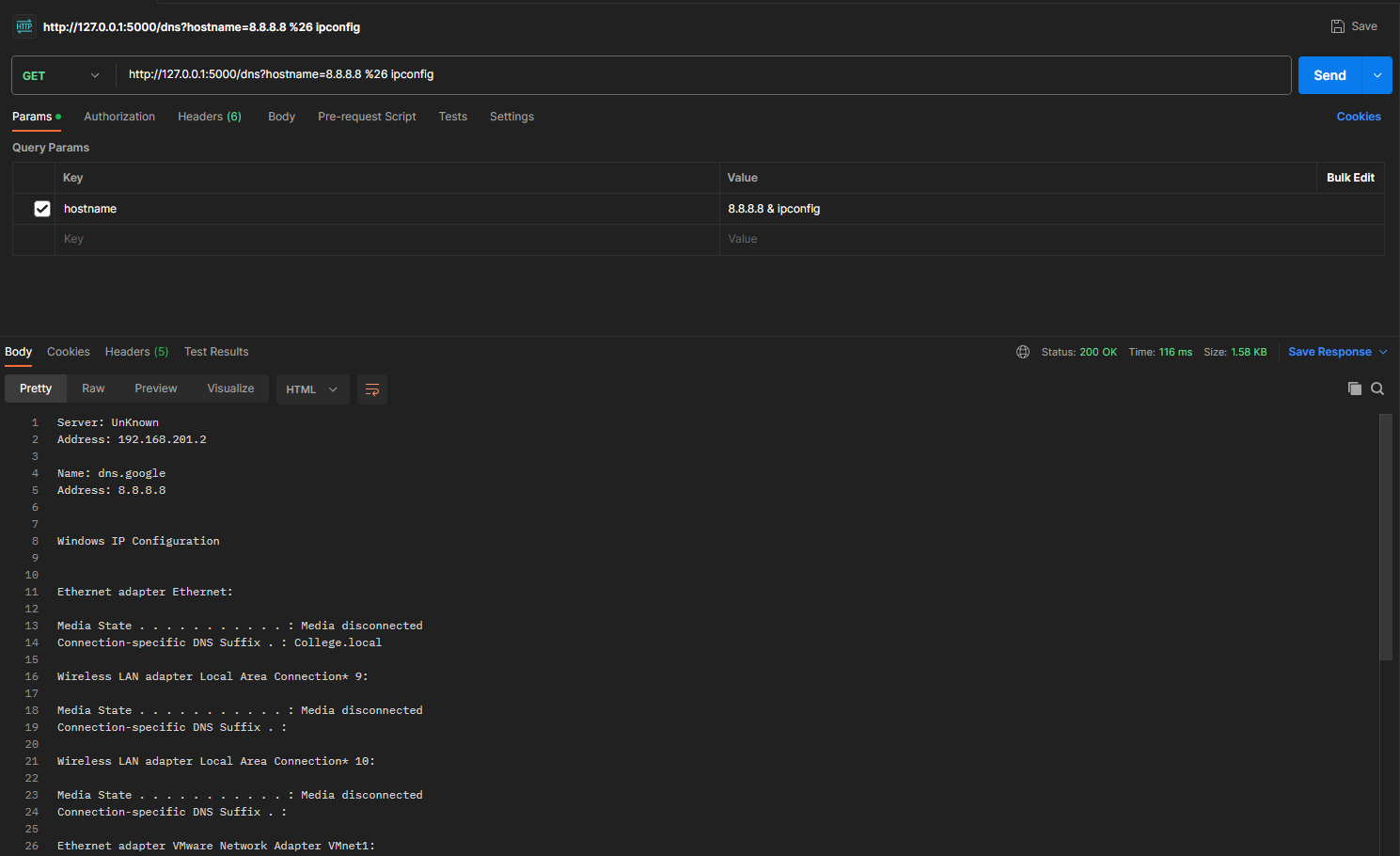


Рис.10 Результат исполнения атаки

Для устранения данной уязвимости я решил провести проверку на вводимых значениях «hostname», а именно удалять часть строки после специального символа «&» и «;». Так, при любом запросе от пользователя, будет выполняться только команда «nslookup». Код представлен на рисунке №, результат попытки эксплуатировать уязвимость после изменений представлен на рисунке №.

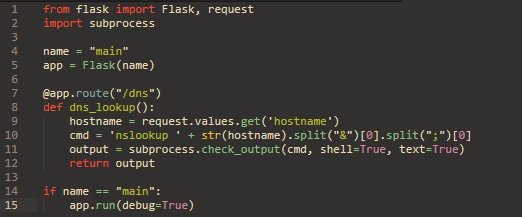


Рис.11 Исправленный код

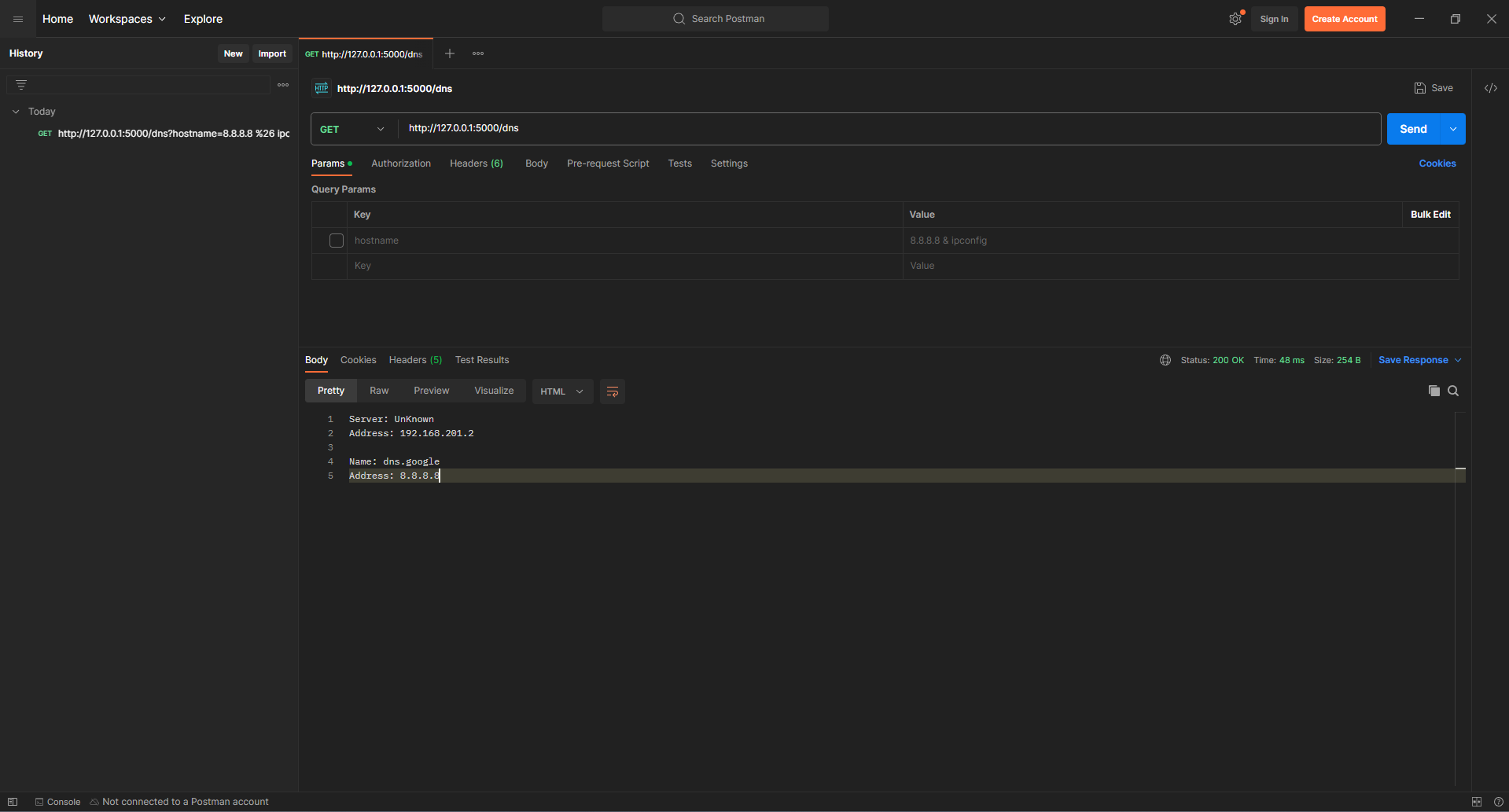


Рис.12 Проверка на уязвимость

Также, в качестве решения уязвимости можно, аналогично первому примеру на Python, удалять специальные символы из вводимых пользователем данных.

**3. МОДЕЛИРОВАНИЕ УГРОЗ**

Начну с того, что я раньше не занимался моделированием угроз для сервисов, так что основываться мне не на чем. Данный отчёт будет моей, так сказать, первой попыткой.