Уметалиев Алишер

Анализ кода на Python для Cloud.ru

При анализе кода, выявлено, что одиннадцатая (11) строка **«output = Template('Hello ' + name + '! Your age is ' + age + '.').render()»**  захватывает ввод юзера без каких-либо фильтров и изменений (Рис. 1).

from flask import Flask, request

from jinja2 import Template

import os as os

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)

Рис.1 Фрагмент уязвимого кода.

**Найденные уязвимости:**

Такой подход ведет к возможности эксплуатации сразу и клиентской стороны и серверной, таким образом охватывая как минимум 3 критические уязвимости:

1. *Server-Side Template Injection (SSTI).* Уязвимость сервер-сайда, в данном случае эксплуатирующая Flask на предмет модификации действующего темплейта для использования удаленных команд.
2. *Reflected Cross-Site Scripting (XSS)*. Уязвимость клиенто-серверного типа, которая позволяет использовать произвольный JavaScript код в целях компрометации сервиса.
3. *SQL Injection (SQLi).* Уязвимость в конфигурации валидирования данных пользователя позволят злоумышленнику манипулировать данными, находящимися в базе данных, что приводит к компрометации данных пользователей, сервисов, а также возможности получения полного контроля над уязвимым сервисом.

Сейчас я продемонстрирую, как именно потенциальный злоумышленник может заэксплуатировать эти уязвимости. Начнем с SSTI. Для примера я возьму базовый пэйлоад **{{7\*7}},** который выполнит умножение, а значит воспроизведет код внутри скобок – это и будет знаком компрометации. (Рис .2)  
Изображение выглядит как текст, снимок экрана, Мультимедийное программное обеспечение, Шрифт

Автоматически созданное описание

Рис 2. PoC эксплуатации SSTI.

Как и следовало ожидать, мы ввели наш пэйлоад и умножение выполнилось, а значит есть возможность ввести вредоносный код, который обязательно будет исполнен сервером: output = Template('Hello ' + name + '! Your age is ' + age + '.').render()

Теперь подробнее разберем, что же такого опасного может ввести злоумышленник и к каким последствиям это приводит:

1. Возможность Command Execution (использование команд) предоставит хакеру возможность к получению чувствительной информации, через команду ls (и более масштабные подходы),
2. Доступ во внутреннюю часть инфраструктуры, внутри которой можно выполнить LPE (Локальное Поднятие Привилегий), что приведет к получению звания «root/System/Authority» для дальнейшей компрометации всей инфраструктуры.
3. Отправка вредоносных файлов (ВПО) в систему, которая может миновать корреляционные правила защиты Firewall.

Теперь разберем возможность использования Reflected XSS в данном сервисе. Для этого воспользуемся показательным payload-ом, который вызывает всплывающее уведомление с произвольным текстом (cloudructf{syst3m\_1s\_c0mpr0m1s3d}):

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

Рис 3. PoC эксплуатации Reflected XSS.

А теперь последствия подобной атаки. Так как последствия атаки сразу отражаются на странице (исходя из названия), то и атаки тоже будут появляться мгновенно:

1. Злоумышленник может украсть cookies пользователя, которые часто содержат сессионные токены или другие учетные данные, позволяя атакующему осуществить несанкционированный доступ к учетной записи жертвы.
2. Мошенник может создать копию входной страницы сервиса и перенаправить пользователя на нее, тем самым получив доступ к логину и паролю жертвы.
3. Вредоносный скрипт может считывать и передавать личную информацию, введенную пользователем на страницах веб-сайта, такую как данные кредитных карт, адреса и телефонные номера.
4. Злоумышленник может выполнить действия на сайте от имени пользователя, включая изменение пароля, отправку сообщений или выполнение транзакций.
5. Reflected XSS может быть использован для распространения вредоносного кода среди пользователей веб-сервиса, увеличивая масштаб компрометации.

На данном этапе, когда основные вектора для атаки были освещены, следует обратить внимание на более мелкие проблемы, например Error-based attacks, которые основываются на получении информации о веб-сервисе, исходя из сообщений об ошибке. Один из примеров неправильной конфигурации – debug=’True’:  
  
На 15-й строке (**app.run(debug=True)**) замечено, что включен режим дебаггинга, что означает подробное описание ошибок прямо на веб-странице. Несмотря на то, что это удобно в целях оптимизации кода, это может послужить отличной отправной точкой для хакера, позарившегося на ваш веб-сайт (Рис. 4).  
  
Изображение выглядит как текст, снимок экрана, Шрифт, веб-страница

Автоматически созданное описание

Рис 4. Error-Based Attack

**Исправление данных уязвимостей:**

* **SSTI.** При использовании Flask и Jinja2 обратите внимание на безопасные функции или фильтры, которые автоматически экранируют потенциально опасные символы:

from jinja2 import escape

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

* **Reflected XSS.** Проверяйте и очищайте все данные, которые выводятся на экран пользователя, чтобы предотвратить выполнение вредоносного кода. Решение может включать в себя создание белых списков, или использование динамический библиотек проверки (ORM, GORM).
* from flask import escape
* @app.route("/page")
* def page():
* name = escape(request.values.get('name'))
* age = escape(request.values.get('age', 'unknown'))
* return f"Hello {name}! Your age is {age}."
* **Выключите режим отладки при развертывании.** В вашем коде есть строка **app.run(debug=True)**, которая полезна при разработке, но очень опасна в продакшене, так как может показывать злоумышленникам внутреннюю структуру вашего приложения:

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

    app.run(debug=False)