6/12/2021

CS437 Assignment 1

Group Number : 13
Group Members:
Tuğcan Barbin 25168
Emre Ekmekçioğlu 25223
Doğan Can Hasanoğlu 26809

Video Link:
https://drive.google.com/drive/folders/10-WCqJdRvCtnt4ttr0DMrJkON1jzeni0?usp=sharing
Responsibility : 8 / Injection XSS

Description: We created a vulnerable Django web application to demonstrate XSS vulnerability. First we created a Django project from scratch then added some features to create XSS vulnerability. VS Code was used for editing issues and SQLite for databases. Our web application was named LearnVul. Application consists of three main parts: Home, Search, Contact. On the home page, you can comment on topics and see previously made comments. Search tab is for searching as the name suggests. Contact is for suggesting new topics. For front-end purposes we used HTML and CSS.

XSS vulnerability arises when the application takes input from the user and uses it without sanitizing it. If this is the case an attacker can inject malicious payload to run javascript in the browser. From the attackers point of view, first the determination of the vulnerable input points should be performed to do this. The attacker will inject various payloads according to the context that user input is used in. The application has 3 examples of the XSS, one of them stored XSS in HTML context, the others are reflected XSS examples in different contexts (JavaScript context and tag attribute context.) After determining the exploitation should be done to steal users' cookies, to do that an attacker can open a server and inject a payload for stored XSS and prepare links for reflected ones. The difference between those types of XSS, at stored one attacker does not need to perform social engineering action so we can say it is a powerful vulnerability since it will affect multiple users at the same time. For the reflected XSS attacker should force the victim to click the link with social engineering skills. All the actions are performed during the demonstration of this assignment.

Code(s) associated with the responsibilities :

→ Emre Ekmekçioğlu, Doğan Can Hasanoğlu

- Back-end creation
- admin.py
- models.py
- forms.py

→ Doğan Can Hasanoğlu , Tuğcan Barbin

- Testing with Static Code Analyzers
- CSS of the web application

→ Tuğcan Barbin, Emre Ekmekçioğlu, Doğan Can Hasanoğlu

- add_comment.html
- base.html
- book_details.html
- contact.html
- contactresult.html
- Feed.html
- Home.html
- Search_result.html
- Searchbar.html
- urls.py
- views.py

Explanation of the code:

Stored XSS:

Django uses XSS protection by using automatic HTML escaping. However, we can disable this default protection by using some flags in html such as ‘autoescape off’ and ‘safe’ which make the input point vulnerable to possible malicious payloads.

LearnVulapp > templates > book_details.html > p > h2

```
{% extends 'base.html' %}
{% load static %}
{%block content%}
<p><h1 style= 'color: white'>Topic:</h1><h2>{{book.title}}<small>Published on {{book.pub_date}}</small></h2></p>
<p></p>
<br>
<br>
<!--Burada autoescape off edildiginden nickname ve  comment.body input  noktalari olasi XSS zaafiyet noktalaridir.
<h2>Comments</h2>
{% autoescape off %}
{%if not book.comments.all %}
    No comments for this book...<a href='add_comment'> Add Comment</a>
{%else%}
<a href={%url 'add_comment' book.pk %}>Add Comment</a>

    {%for comment in book.comments.all%}
    <div class='comment-box'>
        <label><strong>Nickname: </strong>{{comment.nickname}}</label>
        <center>
        <p id= '{{comment.body}}' >{{comment.body}}</p>
        </center>
    </div>
    {%endfor%}

{%endif%}
{% endautoescape %}

<br><br>
<a href="{%url 'feed'%}"> Back </a>
{%endblock content%}

```

The figure above includes the code of the book details page which contains all comments posted to this specific book. In this code segment, we used ’autoescape off’ to prevent default encoding of messages. So, an attacker can enter a malicious comment or nickname and create a stored XSS attack.When the other visitors of the site visit the page of the book which has malicious payload he/she will be affected. The vulnerability arises when some special characters are not escaped such as little then(<),greater than (>) etc. which is used a lot in XSS payloads.

Reflected XSS in tag attribute context:

Another attack we covered is reflected XSS in attribute context. In the figure below, users can create a contact message and send this message to admins.

LearnVulapp > templates > contact.html > ...

```
1   {% extends 'base.html' %}{%load static%}{% block content %}
2
3   <h2>You can contact us to suggest a topic</h2>
4
5   <form action= contact_result method = GET>
6       <p>Nickname: <input name=uname ></p>
7       <p>Topic: <input name=topic></p>
8       <button type=submit>Submit</button>
9   </form>
10  {%endblock content%}
```

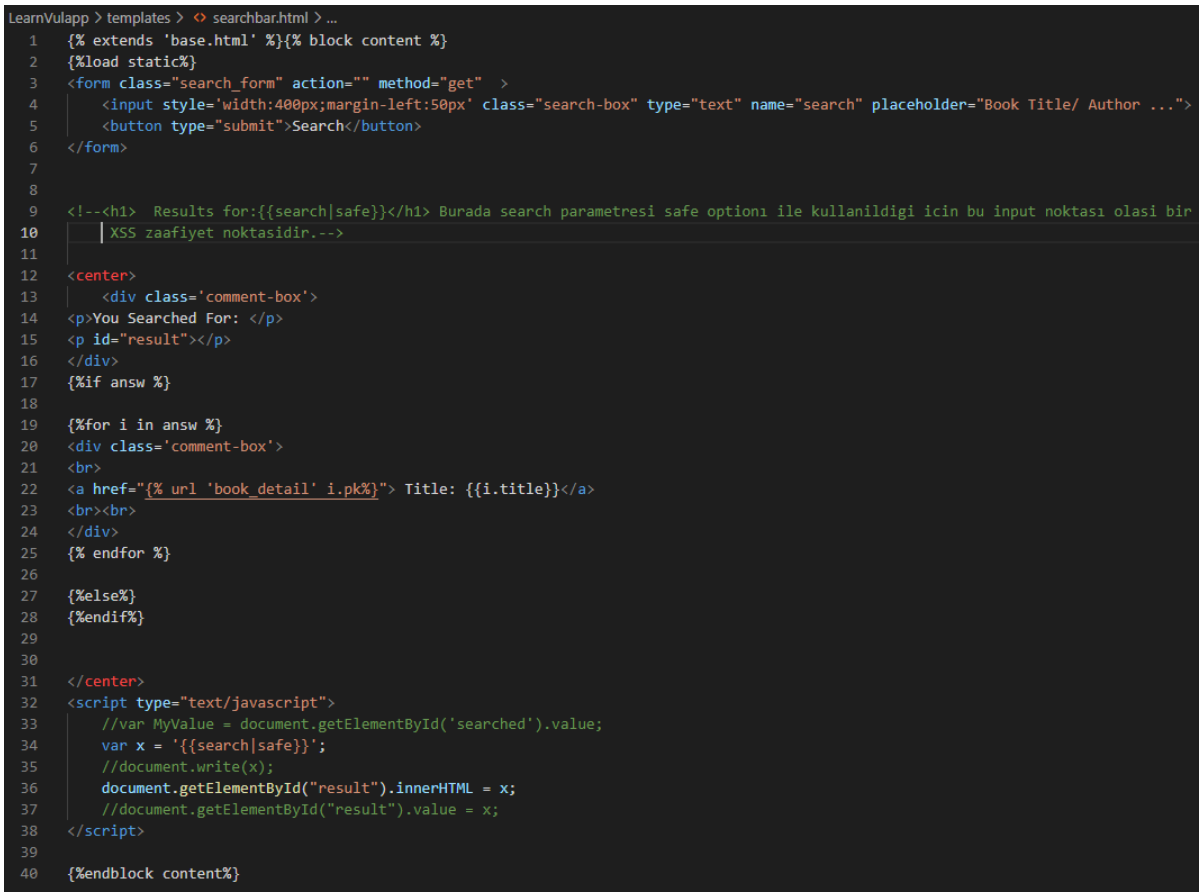
After fill the contact form, application redirect user to a page (contactresult.html) that include an input box which uses the name input (from the form filled in the contact.html) in value attribute of the input tag. Attacker can enter malicious input to name variable and can use payloads such as *"><script>alert(1)</script>* or *" onmouseover=alert(1)>//* to detect the XSS vulnerability.

LearnVulapp > templates > contactresult.html > ...

```
1   {% extends 'base.html' %}
2   {% load static %}
3   {%block content%}
4   <input ype="text" placeholder="Type " id="inputId" value="{{name|safe}}"
5   <h1>Thank you for your suggest</h1>
6
7   {%endblock content%}
```

Reflected XSS in JavaScript Context:

In addition, reflective XSS attacks can be made by using the search facility of the website. As shown in the figure below, the website takes an input from the user as a search parameter and handles this input in a javascript function. Then, the function passes the value to the html element as a searched input and displays the searched datas from the database. The XSS vulnerability arises from the usage of searched parameters with safe flag.So, a malicious link can be created by an attacker to make a reflective javascript XSS attack.



To prevent the vulnerabilities explained above, as we stated in the beginning of the report Django framework has automatic HTML escaping and if the developer of the site disable

auto escaping by using {% autoescape off %} option or by using safe flag (for example: {{query|safe}}) the application becomes vulnerable to possible XSS attacks. To avoid these situations, the developer should not use the flags above but if the user should use these flags it is recommended to use striptags flag to escape the special characters that are used in malicious payloads.

In the demonstration the determination, exploitation(stealing cookies with another server) and the preventions are shown.

Static Code Analyzers:
Analyzer tools were tested in Windows 10. All the screenshots will be included in another folder. Installation process for each of them is pretty straightforward except for RATS (pip install [package_name]). It will be explained later in that section. Usage can differ. Some of them need a recursive call to scan subfolders and files. Also some need framework specific usage, for instance Python Taint. Moreover, some scans yield no significant result whereas the others find a lot of issues. Regardless, it is best practice to review all of them.

- Bandit:
  We scanned the root directory then our app directory recursively. For the root directory, it scanned 119 lines and the only issue that it finds is that the secret key is hardcoded with low severity and medium confidence. For the app directory, it scans 175 lines and finds nothing significant.
- PYT(Python Taint):
  In total it finds 11 vulnerabilities. Mostly they are about input points and weaknesses about them. However, it does not directly point to the XSS and filtering & sanitizing issues. Detailed list can be found in the attachment folder.
- Rough-Auditing-Tool-for-Security(RATS):
  This is a legacy tool which was developed more than 10 years ago. In Windows, it requires an expat DLL file. Once it runs, it works too slow and yields no result other than a basic summary. Therefore, we did not try to run it on a Linux machine properly.
- Prospector:
  Prospector found several issues but they were all about code smells. Mostly unused variables, imports and arguments. Misconfiguration issues, bad indentation and inconsistent return issues. It does not provide any helpful information in terms of security rather than what a basic static analyzer tool does. Detailed list can be found in the attachment folder.

- Pylint:
  This one is the extra we found. First we tried tools such as SonarQube and Pythia for XSS. However configuration and usage were complex. That's why we chose Pylint. This one mostly reveals issues related to misuse, indentation, unused properties(variables, functions, arguments), missing things and then makes suggestions about them. Not a significant issue about XSS.

  Most of the tools do not suggest anything about XSS. That is mainly because XSS can be prevented by input sanitizing & filtering and encoding the data in HTML files. Static

tools scan the .py files not the HTML files. XSS can be detected by manually testing every entry point in the application and also by some combined static and dynamic analysis tools such as Burp Suite scanner.

Sources (Weblink, book or magazine) used to write the code:

- https://docs.djangoproject.com/en/3.2/topics/forms/
- https://www.geeksforgeeks.org/render-html-forms-get-post-in-django/
- https://www.youtube.com/watch?v=OuOB9ADT_bo&list=PLCC34OHNcOtr025c1kHSPrnP18YPB-NFi&index=36&ab_channel=Codemy.com
- https://github.com/villeheikkila/vulnerable-django
- https://www.stackhawk.com/blog/django-xss-examples-prevention/
- https://docs.djangoproject.com/en/3.2/topics/security/
- https://tonybaloney.github.io/posts/xss-exploitation-in-django.html