Security in Django

Chapter 10

Objectives

- Discussion on the following attacks:
  - XSS
  - CSRF
  - SQL injection
  - Clickjacking
  - SESSION HIJACKING

THE THEME OF WEB SECURITY

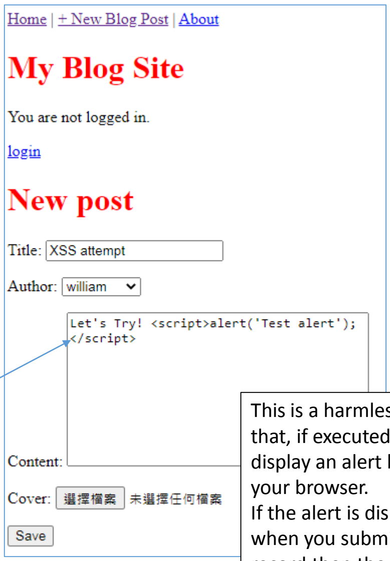
- Never — under any circumstances — trust data from the browser.
- You *never* know who's on the other side of that HTTP connection. It might be one of your users, but it just as easily could be a nefarious cracker looking for an opening.
- Any data of any nature that comes from the browser needs to be treated with a healthy dose of paranoia.
- This includes data that's both "in band" (i.e., submitted from Web forms) and "out of band" (i.e., HTTP headers, cookies, and other request information).

Django's built in security features

- Ensuring that the sites you build are secure is of the utmost importance to a professional web applications developer.
- The Django framework is very mature and the majority of common security issues are addressed in some way by the framework itself.
- However no security measure is 100% guaranteed and there are new threats emerging all the time.
- This chapter includes an overview of Django's security features and advice on securing a Django powered site that will protect your sites 99% of the time.
- But it's up to you to keep abreast of changes in web security.

Cross Site Scripting (XSS)

- **Cross Site Scripting** (**XSS**) is found in Web applications that fail to escape user-submitted content properly before rendering it into HTML.
- This allows an attacker to insert arbitrary HTML into your Web page, usually in the form of <script> tags --- that is, inject client side scripts into the browsers of other users.
- Attackers often use XSS attacks to steal cookie and session information, or to trick users into giving private information to the wrong person (aka *phishing*).
- Let's take a look at a typical example. We can easily insert <script> tags that will run some client-side script!



Home | + New Blog Post | About

My Blog Site

You are not logged in.

login

New post

Title: XSS attempt

Author: william

Content:

```
Let's Try! <script>alert('Test alert');
</script>
```

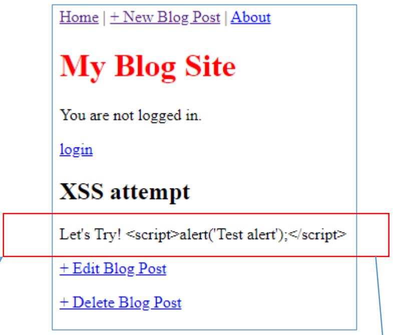
Cover: 選擇檔案 未選擇任何檔案

Save

This is a harmless script that, if executed, will display an alert box in your browser.
If the alert is displayed when you submit the record then the site is vulnerable to XSS threats.

Django's protection against XSS

- The problem gets worse if you store this data (malicious scripts) in the database and later retrieved and displayed to other users, or by getting users to click a link which will cause the attacker's JavaScript to be executed by the user's browser.
- Using Django templates protects you against the majority of XSS attacks.
- Django templates escape specific characters which are particularly dangerous to HTML.



Home | + New Blog Post | About

My Blog Site

You are not logged in.

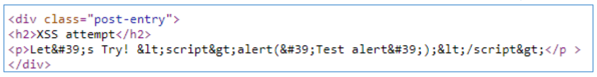
login

XSS attempt

Let's Try! <script>alert('Test alert');</script>

+ Edit Blog Post

+ Delete Blog Post



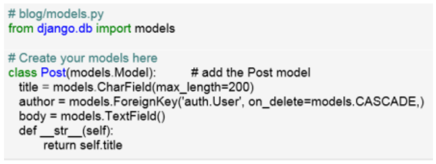
```
<div class="post-entry">
<h2>XSS attempt</h2>
<p>Let&#39;s Try! &lt;script&gt;alert(&#39;Test alert&#39;);&lt;/script&gt;</p >
</div>
```

Cross Site Request Forgery (CSRF)

- **Cross Site Request Forgery** (**CSRF**) attacks allow a malicious user to execute actions using the credentials of another user without that user's knowledge or consent.
- For example consider the case where we have a hacker who wants to create additional blog posts for our application.
- A more ambitious hacker could use the same approach on other sites to perform much more harmful tasks (e.g. transfer money to their own accounts, etc.)
- Django has built-in protection against most types of CSRF attacks, providing you have enabled and used it where appropriate.
- CSRF protection works by checking for a nonce in each POST request. This ensures that a malicious user cannot simply replay a form POST to your website and have another logged in user unwittingly submit that form. The malicious user would have to know the nonce, which is user specific (using a cookie).

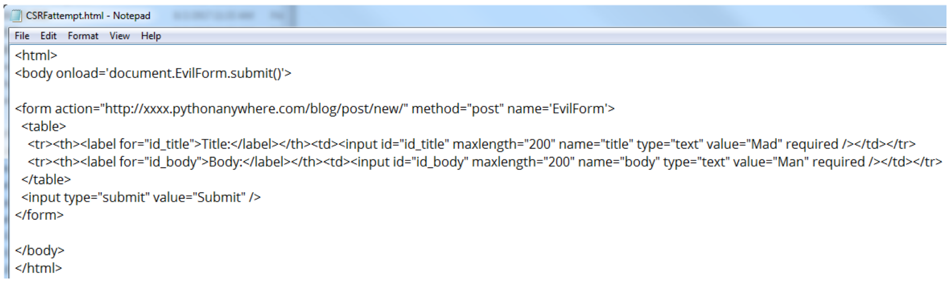
7

CSRF example



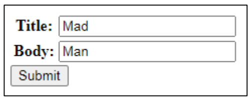
```
# blog/models.py
from django.db import models

# Create your models here
class Post(models.Model):         # add the Post model
   title = models.CharField(max_length=200)
   author = models.ForeignKey('auth.User', on_delete=models.CASCADE,)
   body = models.TextField()
   def __str__(self):
        return self.title
```

- In order to do this, they create an HTML file, which contains an post-creation form that is submitted as soon as the file is loaded. They would then send out the file.
- If the file is opened by any logged in user, then the form would be submitted with their credentials and a new blog post would be created.



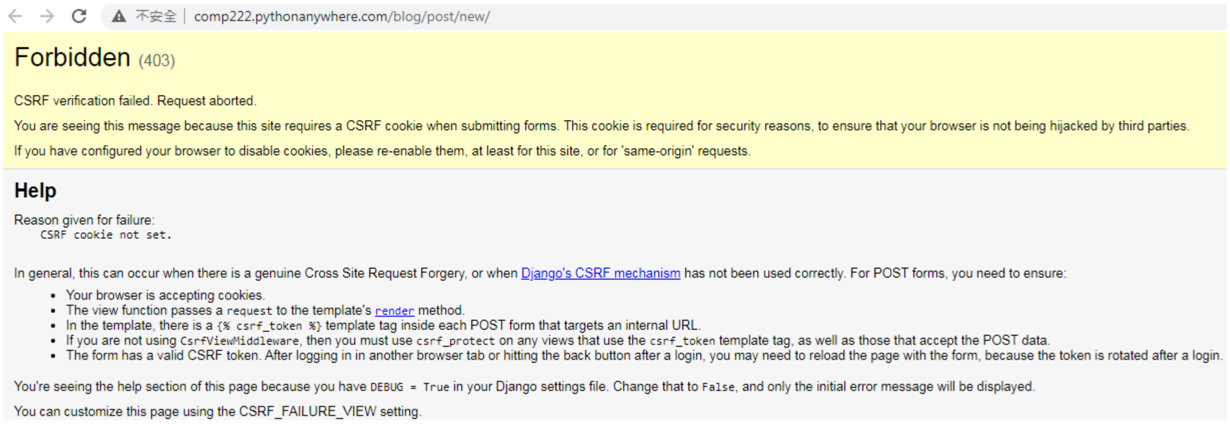
```
<html>
<body onload='document.EvilForm.submit()'>

<form action="http://xxxx.pythonanywhere.com/blog/post/new/" method="post" name='EvilForm'>
 <table>
  <tr><th><label for="id_title">Title:</label></th><td><input id="id_title" maxlength="200" name="title" type="text" value="Mad" required /></td></tr>
  <tr><th><label for="id_body">Body:</label></th><td><input id="id_body" maxlength="200" name="body" type="text" value="Man" required /></td></tr>
 </table>
 <input type="submit" value="Submit" />
</form>

</body>
</html>
```

Django’s protection against CSRF



Title:	Mad
Body:	Man

- When you open the HTML on the previous slide,
- You should get a CSRF error, because Django has protection against it.



Forbidden (403)

CSRF verification failed. Request aborted.

You are seeing this message because this site requires a CSRF cookie when submitting forms. This cookie is required for security reasons, to ensure that your browser is not being hijacked by third parties.

If you have configured your browser to disable cookies, please re-enable them, at least for this site, or for 'same-origin' requests.

Help

Reason given for failure:

```
    CSRF cookie not set.
```

In general, this can occur when there is a genuine Cross Site Request Forgery, or when Django's CSRF mechanism has not been used correctly. For POST forms, you need to ensure:

- Your browser is accepting cookies.
- The view function passes a `request` to the template's `render` method.
- In the template, there is a `{% csrf_token %}` template tag inside each POST form that targets an internal URL.
- If you are not using `CsrfViewMiddleware`, then you must use `csrf_protect` on any views that use the `csrf_token` template tag, as well as those that accept the POST data.
- The form has a valid CSRF token. After logging in in another browser tab or hitting the back button after a login, you may need to reload the page with the form, because the token is rotated after a login.

You're seeing the help section of this page because you have `DEBUG = True` in your Django settings file. Change that to `False`, and only the initial error message will be displayed.

You can customize this page using the CSRF_FAILURE_VIEW setting.

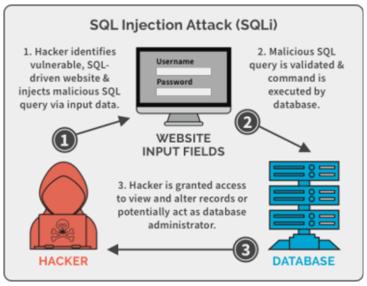
Django’s protection against CSRF

- The way the protection is enabled is that you include the {% csrf_token %} template tag in your form definition.
- This token is then rendered in your HTML as shown below, with a value that is specific to the user on the current browser.
  <input type='hidden' name='csrfmiddlewaretoken' value='0QRWHnYVg776y2l66mcvZqp8alrv4lb8S8lZ4ZJUWGZFA5VHrVfL2mpH29YZ39PW' />
- Django generates a user/browser specific key and will reject forms that do not contain the field, or that contain an incorrect field value for the user/browser.
- To use this type of attack, the hacker now has to discover and include the CSRF key for the specific target user.

Difference between CSRF and XSS

- Unlike cross-site scripting (XSS), which exploits the trust a user has for a particular site, CSRF exploits the trust that a site has in a user's browser.
- In a CSRF attack, an innocent end user is tricked by an attacker into submitting a web request that they did not intend. This may cause actions to be performed on the website that can include inadvertent client or server data leakage, change of session state, or manipulation of an end user's account.
- https://owasp.org/www-community/attacks/csrf

SQL injection


SQL Injection Attack (SQLi)
1. Hacker identifies vulnerable, SQL-driven website & injects malicious SQL query via input data.
Username
Password
2. Malicious SQL query is validated & command is executed by database.
1
2
WEBSITE INPUT FIELDS
3. Hacker is granted access to view and alter records or potentially act as database administrator.
3
HACKER
DATABASE

- SQL injection is a type of attack where a malicious user is able to execute arbitrary SQL code on a database. This can result in records being deleted or data leakage.
- A demo: https://www.youtube.com/watch?v=TSqXkkOt6oM

SQL injection

- This vulnerability commonly crops up when constructing SQL “by hand” from user input.
- For example, imagine writing a function to gather a list of contact information from a contact search page.

```
def user_contacts(request):
    user = request.GET['username']
    sql = "SELECT * FROM user_contacts WHERE username = '%s';" % username
    # execute the SQL here...
```

- What happens if an attacker types "' OR 'a'='a" into the query box. In that case, the query that the string interpolation will construct will be:

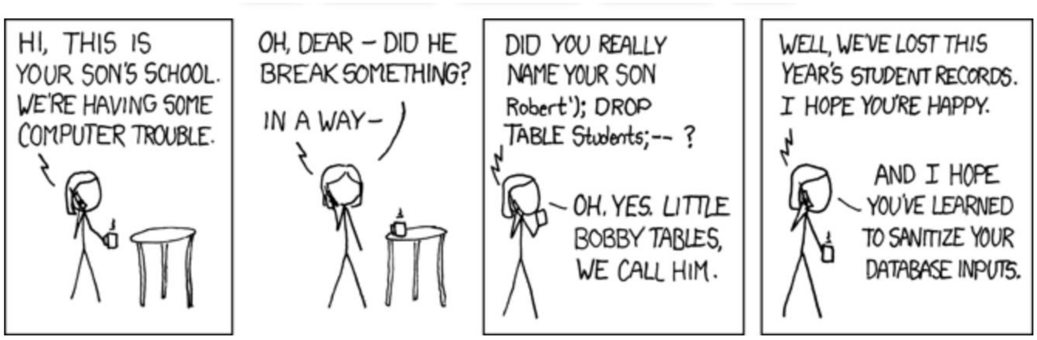
```
SELECT * FROM user_contacts WHERE username = '' OR 'a' = 'a';
```

- Because we allowed unsecured SQL into the string, the attacker’s added OR clause ensures that every single row is returned.

SQL injection (cont’d)

- Imagine what will happen if the attacker submits "'; DELETE FROM user_contacts WHERE 'a' = 'a'".

```
SELECT * FROM user_contacts WHERE username = ''; DELETE FROM
user_contacts WHERE 'a' = 'a';
```


HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY–
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH. YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

Django’s protection against SQL injection

- Although this problem is sometimes hard to spot, the solution is simple: *never* trust user-submitted data, and *always* escape it when passing it into SQL.
- By using Django's querysets, the resulting SQL will be properly escaped by the underlying database driver.
- However, Django also gives developers power to write raw queries or execute custom SQL. These capabilities should be used sparingly and you should always be careful to properly escape any parameters that the user can control.

Clickjacking protection

- Hijacking is to vehicles as Clickjacking is to clicks.
- They are also known as “UI redress attacks," where the attacker renders a concealed layer on your website, in the hope of deceiving the client into clicking on to it, which they have loaded in a hidden frame or iframe, and redirects it to another page that is owned by another application, domain, or both.
- Suppose this new endpoint’s functionality is to install a script introducing a worm on your machine, which, in this case, is connected to your production server network. This worm will be the cause of replication of itself on every other host with which it can communicate with, resulting in big trouble.
- Similarly, keyboard strokes can also be hijacked. With a carefully crafted combination of *stylesheets, iframes, and text boxes,* a user can be deceived into typing their password to their social account, or banking websites when they are actually typing into the attacker’s form input, thus giving them access to the secret data from the user.
- Django contains clickjacking protection in the form of the X-Frame-Options middleware which in a supporting browser can prevent a site from being rendered inside a frame. It is possible to disable the protection on a per view basis or to configure the exact header value sent.

    ‘django.middleware.clickjacking.XFrameOptionsMiddleware’ to MIDDLEWARE in settings.py

SSL/HTTPS

- It is always better for security, though not always practical in all cases, to deploy your site behind HTTPS.
- Without this, it is possible for malicious network users to sniff authentication credentials or any other information transferred between client and server, and in some cases-active network attackers–to alter data that is sent in either direction.
- If you want the protection that HTTPS provides, and have enabled it on your server.

17

SESSION HIJACKING

This is a general class of attacks on a user's session data. It can take a number of different forms:

- *Session forging*, where an attacker uses a session ID (perhaps obtained through a man-in-the-middle attack) to pretend to be another user.
    - An example would be an attacker in a coffee shop using the shop's wireless network to capture a session cookie. That stolen cookie can then be used to impersonate the original user.

18

SESSION HIJACKING (cont'd)

- *Session fixation*, where an attacker tricks a user into setting or reseting the user's session ID. For example, PHP allows session identifiers to be passed in the URL (e.g., http://example.com/?PHPSESSID=fa90197ca25f6ab40bb1374c510d7)
  - An attacker who tricks a user into clicking a link with a hard-coded session ID will cause the user to pick up that session.
  - Session fixation has been used in phishing attacks to trick users into entering personal information into an account the attacker owns. He can later log into that account and retrieve the data.

19

SESSION HIJACKING -- The Solution

There are a number of general principles that can protect you from these attacks:

- Never allow session information to be contained in the URL. Django's session framework simply doesn't allow sessions to be contained in the URL.
- Don't store data in cookies directly; instead, store a session ID that maps to session data stored on the back-end.
- Notice that none of those principles and tools prevents man-in-the-middle attacks.
- These types of attacks are nearly impossible to detect. If your site allows logged-in users to see any sort of sensitive data, you should *always* serve that site over HTTPS.
- Additionally, if you have an SSL-enabled site, you should set the SESSION_COOKIE_SECURE setting to True; this will make Django only send session cookies over HTTPS.

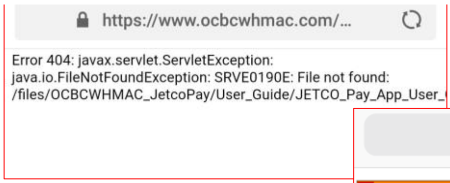
20

EXPOSED ERROR MESSAGES

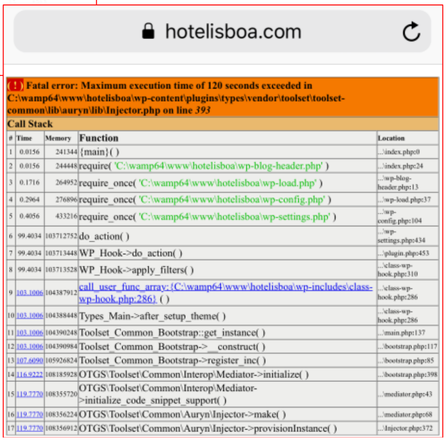
- During development, being able to see tracebacks and errors live in your browser is extremely useful. Django has “pretty” and informative debug messages specifically to make debugging easier.
- However, if these errors get displayed once the site goes live, they can reveal aspects of your code or configuration that could aid an attacker. Django’s philosophy is that site visitors should never see application-related error messages.
- Naturally, of course, developers need to see tracebacks to debug problems in their code. So the framework should hide all error messages from the public, but it should display them to the trusted site developers.
- **The Solution:** uses the simple flag that controls the display of these error messages, i.e. the DEBUG setting.

21

Never deploy a site into production with DEBUG turned on



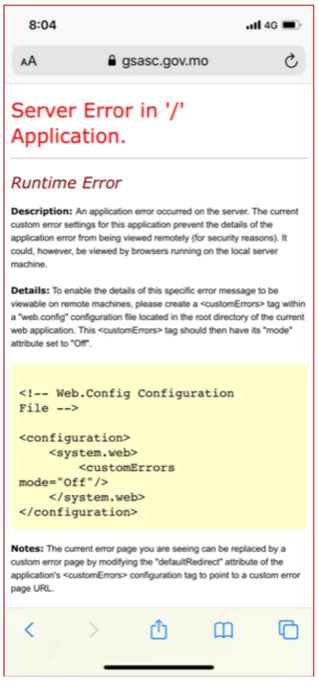
```
Error 404: javax.servlet.ServletException:
java.io.FileNotFoundException: SRVE0190E: File not found:
/files/OCBCWHMAC_JetcoPay/User_Guide/JETCO_Pay_App_User_
```



(!) Fatal error: Maximum execution time of 120 seconds exceeded in C:\wamp64\www\hotelisboa\wp-content\plugins\types\vendor\toolset\toolset-common\lib\auryn\lib\Injector.php on line 393

Call Stack

#	Time	Memory	Function	Location
1	0.0156	241344	{main}( )	...\index.php:0
2	0.0156	244448	require('C:\wamp64\www\hotelisboa\wp-blog-header.php')	...\index.php:24
3	0.1716	264952	require_once('C:\wamp64\www\hotelisboa\wp-load.php')	...\wp-blog-header.php:13
4	0.2964	276896	require_once('C:\wamp64\www\hotelisboa\wp-config.php')	...\wp-load.php:37
5	0.4056	433216	require_once('C:\wamp64\www\hotelisboa\wp-settings.php')	...\wp-config.php:104
6	99.4034	103712752	do_action( )	...\wp-settings.php:434
7	99.4034	103713448	WP_Hook->do_action( )	...\plugin.php:453
8	99.4034	103713528	WP_Hook->apply_filters( )	...\class-wp-hook.php:310
9	103.1006	104387912	call_user_func_array:{C:\wamp64\www\hotelisboa\wp-includes\class-wp-hook.php:286} ( )	...\class-wp-hook.php:286
10	103.1006	104388448	Types_Main->after_setup_theme( )	...\class-wp-hook.php:286
11	103.1006	104390248	Toolset_Common_Bootstrap::get_instance( )	...\main.php:137
12	103.1006	104390984	Toolset_Common_Bootstrap->__construct( )	...\bootstrap.php:117
13	107.6090	105926824	Toolset_Common_Bootstrap->register_inc( )	...\bootstrap.php:85
14	116.9222	108185928	OTGS\Toolset\Common\Interop\Mediator->initialize( )	...\bootstrap.php:398
15	119.7770	108355720	OTGS\Toolset\Common\Interop\Mediator->initialize_code_snippet_support( )	...\mediator.php:43
16	119.7770	108356224	OTGS\Toolset\Common\Auryn\Injector->make( )	...\mediator.php:68
17	119.7770	108356912	OTGS\Toolset\Common\Auryn\Injector->provisionInstance( )	...\Injector.php:372



Server Error in '/' Application.

Runtime Error

Description: An application error occurred on the server. The current custom error settings for this application prevent the details of the application error from being viewed remotely (for security reasons). It could, however, be viewed by browsers running on the local server machine.

Details: To enable the details of this specific error message to be viewable on remote machines, please create a <customErrors> tag within a "web.config" configuration file located in the root directory of the current web application. This <customErrors> tag should then have its "mode" attribute set to "Off".

```
<!-- Web.Config Configuration
File -->

<configuration>
    <system.web>
        <customErrors
mode="Off"/>
    </system.web>
</configuration>
```

Notes: The current error page you are seeing can be replaced by a custom error page by modifying the "defaultRedirect" attribute of the application's <customErrors> configuration tag to point to a custom error page URL.

Archive of security issues

- Django's development team is strongly committed to responsible reporting and disclosure of security-related issues, as outlined in Django's security policies.
- As part of that commitment, they maintain an historical list of issues which have been fixed and disclosed.
- For the up to date list, see the archive of security issues https://docs.djangoproject.com/en/3.0/releases/security/

23

Further Reading

- https://developer.mozilla.org/en-US/docs/Web/Security
- https://developer.mozilla.org/en-US/docs/Learn/Server-side/First_steps/Website_security
- https://docs.djangoproject.com/en/3.1/topics/security/
- OWASP (Open Web Application Security Project) **Top 10 Web Application Security Risks –** https://owasp.org/www-project-top-ten/

24