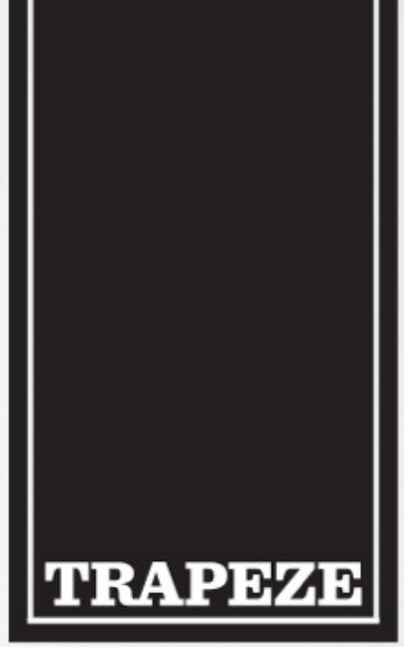


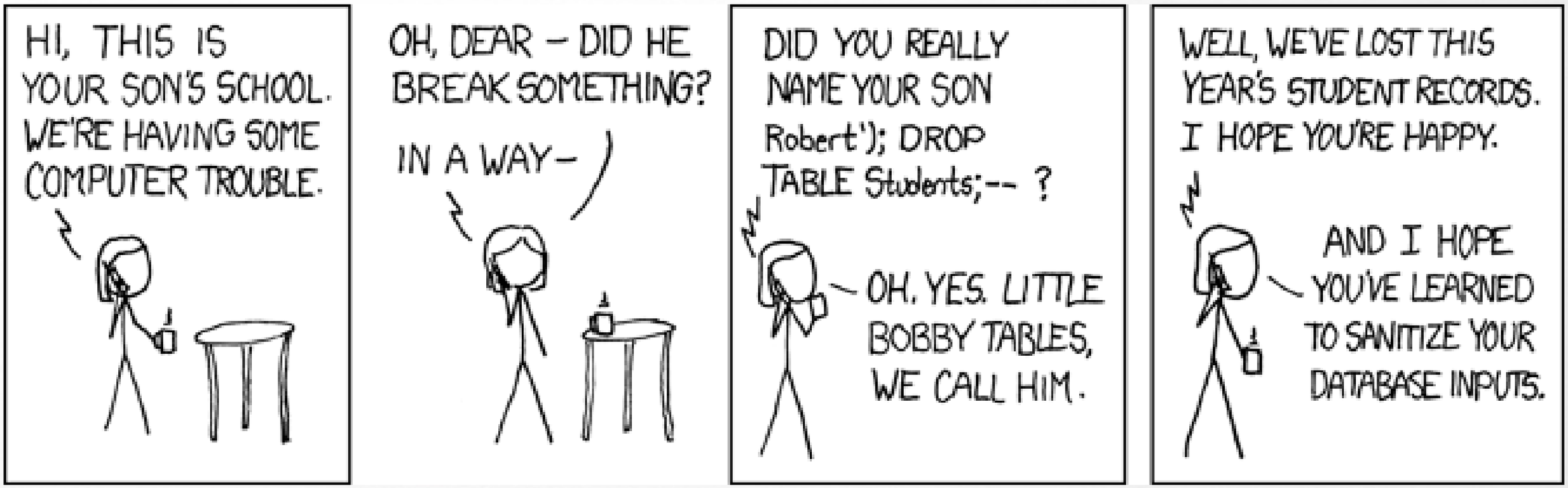
Django Security Tips (and Tricks)

Django is designed to automatically protect you from the most common vulnerabilities:

- SQL code injection
- Cross Site Scripting (XSS)
- Cross Site Request Forgery (CSRF)
- HTTP Header injection

So you don't have to worry about security right?

SQL injection



http://xkcd.com/327/

Django QuerySets automatically escape all special SQL characters.

Except in:

- raw queries (Managers.raw, cursor.execute(), ...)
- where and select arguments to extra()

```
User.objects.raw('SELECT * FROM users WHERE name = ' + name)

User.objects.raw('SELECT * FROM users WHERE name =  %s' % name)

User.objects.raw('SELECT * FROM users WHERE name =  %s', [name,])
```

Lesson learned (SQL Injection):

*Never, **never**, **NEVER** use Python string concatenation (+) or string parameters interpolation (%) to pass variables to a SQL query string. Not even at gunpoint.*

http://initd.org/psycopg/docs/usage.html#passing-parameters-to-sql-queries

XSS

1. Bob has a website where users can create their own profile pages
2. Mallory submit some javascript code in the description field of his profile page
3. Alice visits Mallory's profile page, Mallory's javascript get's executed and send Alice's cookies (sessionid) to Mallory's server.
4. Mallory can impersonate Alice on Bob's website

Django templating engine automatically escapes all special HTML characters.

django/utils/html.py

```
def escape(html):
    """
    Returns the given HTML with ampersands, quotes and angle brackets
encoded.
    """
    return mark_safe(force_unicode(html).replace('&', '&amp;').replace('<',
'&lt;').replace('>', '&gt;').replace('"', '&quot;').replace("'", '&#39;'))
```

Demo...

Lessons Learned (XSS)

- Always escape User Generated Content (UGC)

- Do NOT pass UGC through markup filters

- Django auto escape **only protects you in HTML context!** (inside HTML element content)

  – Use **urlize** and **escapejs** template filters

  – Do not output UGC anywhere else (CSS, HTML comments, ...)

  – SESSION_COOKIE_HTTPONLY = True (Django 1.3)

http://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet

http://ha.ckers.org/xss.html

CSRF

1. Alice has admin rights on her Django site
2. Mallory has a website with a form that POST to Alice's website admin and create a new admin account. *The form is automatically submitted via javascript.*
3. Alice visits Mallory's website.
4. Mallory has admin rights on Alice's website.

In order to prevent forged POST requests, Django CSRF protection:

1. Generate a large random number (csrf token)
2. Set a cookie to the value of the csrf token
3. Make the csrf token available in the context so that it can be added to every form submission
4. For every POST, verify: COOKIES[csrf] == POST[csrf]

browser same origin policy → csrf cookie not available to Mallory.

Demo...

Lessons Learned (CSRF)

- Ensure non POST requests are side-effect free
- Enable the CsrfViewMiddleware in your settings
- Use @csrf_protect decorator in reusable apps
- Keep control of subdomains
- Upgrade to Django 1.3 or 1.2.5 (CSRF AJAX issue)
- Prevent XSS: if there is a XSS, there is a CSRF (MySpace Samy worm).

HTTP Header Injection

1. Bob has a website that redirect users back to the original page after login
   http://bob.com/login?next=/blog/1/
2. When the user is already logged in, login step is skipped and redirection happen immediately
3. Mallory trick Alice into visiting:
   http://bob.com/login?next=%0D%0A%0D%0A<script>...</script>

4. Alice receives the following HTTP Response

HTTP/1.0 302 FOUND
Location:
<script>...</script>

5. Mallory can impersonate Alice on Bob's website.

Django filters all HTTP Response Headers.

See HttpResponse.__setitem__ in django/http/__init__.py

Python Cookie module automatically quotes any non-text character

Looks like we really don't have to worry about HTTP Header injection when using Django! :-)