Why your extension will not be enabled on Wikimedia wikis in its current state!

(and what you can do about it)

Technical advice for extension developers

Roan Kattouw - Wikimania 2010 - Gdańsk, Poland

Focus of this talk

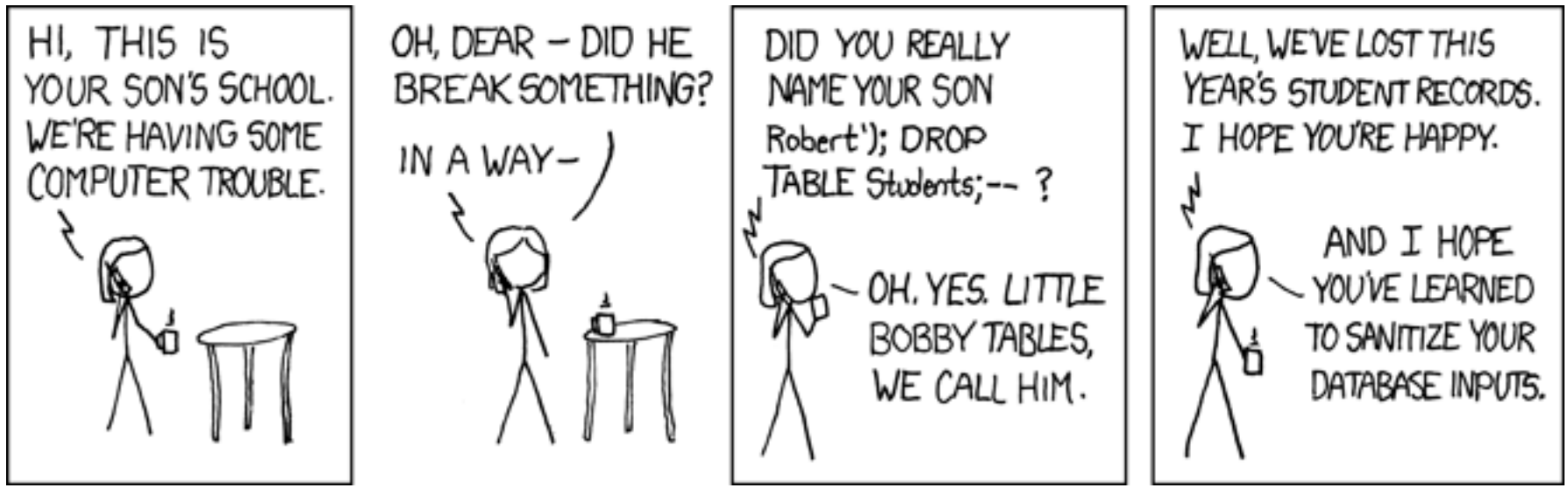
Most common issues with writing code for large wiki farms:

- Security
- Scalability / performance
- Security
- Concurrency
- Security

Security

- **Security is important. Really.**
- People rely on developers to write secure software, so:
  - Insecure extension in SVN = security risk for unwitting third-party wiki admins and their users
  - Insecure extension on Wikipedia = potential security risk for 300 million people

SQL Injection



SQL Injection

What happens here:

```
$sql = "INSERT INTO Students VALUES ( $name, ... );";
```

If **$name** is not escaped, you'll get this query:

```
INSERT INTO Students VALUES ( 'Robert' );
DROP TABLE Students; --', ... );
```

You need to escape the input:

```
INSERT INTO Students VALUES
    ( 'Robert\'); DROP TABLE Students; --' , ... );
```

Use MediaWiki's DB Functions

Evil:

```
$dbr->query( "SELECT * FROM foo WHERE foo_id=' $id'" );
```

Acceptable:

```
$escID = $dbr->addQuotes( $id );
$dbr->query( "SELECT * FROM foo WHERE foo_id= $escID" );
```

Correct:

```
$dbr->select( 'foo', '*', array( 'foo_id' => $id ) );
```

- The database functions handle query building and parameter escaping for you.
- There are docs at Doxygen and usage examples are all over the place in core.

Cross site scripting (XSS)

```
$val = $wgRequest->getVal( 'input' );
$wgOut->addHTML( "<input type=\"text\" value=\" $val \" />" );
```

But what if the user submits `"/><script>evilness</script>`"?

```
<input type="text" value=""/><script>evilness</script>"/>
```

The evil script gets executed and has access to the victim's login cookies

Like with SQL injection, you need to escape your inputs:

```
value="&lt;script&gt;evilStuff();&lt;/script&gt;"
```

Use MediaWiki's HTML functions

Reverted:

```
$html = "<input type=\"text\" name=\"foo\" value=\" $val\" />";
```

Passes code review:

```
$val = htmlspecialchars( $val );
$html = "<input type=\"text\" name=\"foo\" value=\" $val\" />";
```

Tim likes you, kind of:

```
$html = Html::input( 'foo', $val );
```

Cross site request forgery (CSRF)

```
<form id="myForm" method="POST" action=".....">
    <input type="hidden" name="title" value="Foo" />
    <input type="hidden" name="action" value="delete" />
    <input type="hidden" name="wpReason" value="MWAHAHA" />
</form>
<script>
    $( '#myForm' ).submit();
</script>
```

- If an administrator visits this page, they will unwittingly be deleting [[Foo]].
- MediaWiki core is secured against this, but if your extension implements state-changing actions over HTTP, it may be vulnerable.

Using tokens to protect against CSRF

Add a session-dependent token to the form and refuse to carry out the action if it doesn't come back.
Cross-domain tricks can't use the result of the first request to build the second.

```
$html .= Html::hidden( 'token', $wgUser->editToken() );
...
if ( !$wgUser->matchEditToken( $token ) ) {
        // refuse edit
```

Remember that bad token errors can be caused by session timeouts, so use a **nice** error message:

Sorry! We could not process your edit due to a loss of session data. Please try again.
If it still does not work, try logging out and logging back in.

General notes on security

- Don't trust anyone, not even your users
- Escape all inputs
- When in doubt, err on the side of caution
  - But watch out for double escaping!
- Write code that is *demonstrably secure*
- Read [[mw:Security for developers]]
- And best of all: **try to break/hack your code**

Scalability & Performance

- Wikipedia's kinda.... huge
  - 50k-100k requests per second
  - 2k-4k of those fire up MediaWiki
  - enwiki has 20M pages and almost 400M revisions
- For your code to hold up in these circumstances, you need to pay attention to scalability and performance.
- **Performance:** your code runs (relatively) fast
- **Scalability:** your code doesn't get much slower on larger wikis

Easy basic things

- Run code / load stuff only when necessary
  - i.e. **not** on every request if at all avoidable
- Assume nothing
  - "there's not gonna be that many pages with X"
  - "very few people will actually use X"
  - "my users can't be **that** stupid"
- These things sound (and are) obvious, but are not always followed in practice

Optimize database queries

- Certain DB queries operate on way too many rows:
  - Full table scans: all rows in a table
  - Unindexed WHEREs: all rows in the result set
  - Filesort: sorts all rows in the result set (n log n)
- This is bad because:
  - enwiki.revision has ~350M rows
  - [[en:Category:Living_people]] has ~450K members
- EXPLAIN reveals these things:

```
mysql> EXPLAIN SELECT * FROM page WHERE page_len > 100000 ORDER BY page_namespace, page_title;
+----+-------------+-------+-------+---------------+----------+---------+------+-------+----------------------------+
| id | select_type | table | type  | possible_keys | key      | key_len | ref  | rows  | Extra                      |
+----+-------------+-------+-------+---------------+----------+---------+------+-------+----------------------------+
|  1 | SIMPLE      | page  | range | page_len      | page_len | 4       | NULL | 61968 | Using where; Using filesort |
+----+-------------+-------+-------+---------------+----------+---------+------+-------+----------------------------+
1 row in set (0.00 sec)
```

Optimize database queries

- Don't:
  - ORDER BY an expression, unindexed fields or mix ASC with DESC
  - Use unindexed WHEREs (unless the condition drops very few rows)
  - Use LIKE with wildcards (%) that are not at the end
  - Use OFFSET 50 or LIMIT 50, 10 for paging
  - Write queries that scan or return a potentially unlimited number of rows
- Do:
  - LIMIT your queries (usually 50 or 500)
  - Use a unique index for paging
  - **ASK FOR ADVICE**. This is a complex subject

Cache stuff

- If it's expensive to generate, **cache it!**
- In-code caching (within the same request)
  - Query results
  - Results of processing
  - Look for wasted/duplicated effort
- Caching between requests
  - memcached for things that persist between requests
  - MW uses memcached for: parser cache, diff cache, user objects, etc.
  - MW has transparent caching layer supporting alternatives to memcached as well (APC, database, etc.)

Using memcached

Memcached keys:

```
$key = wfMemcKey( 'mything', 'someID' );
```

Getting a value:

```
$val = $wgMemc->get( $key );
if ( $val === null || $val === false ) {
    // Value not in cache
}
```

Setting a value:

```
$wgMemc->set( $key, $val ); // No expiry
$wgMemc->set( $key, $val, 3600 ); // Cache for 1 hour
```

Concurrency

- WMF has ~180 Apache servers running multiple Apache processes
- Potentially lots of instances of your code running at the same time
- Results in weird bugs that don't happen locally, so difficult to test

Common concurrency issues

- Slave lag
  - WMF has master/slave DB setup
  - Use DB_MASTER if data **must** be up-to-date, DB_SLAVE otherwise
- Updating things like counters
  - Needs to account for concurrent updates
  - Use e.g. timestamp-based smartness
- Cache stampeding
  - MW doesn't even handle this properly itself (Michael Jackson incident)
  - There is a framework to prevent this now, but it's not used anywhere yet
  - This will hopefully become standard practice in the future

Closing notes

- This talk is **incomplete**
  - There are more issues, avoiding the ones I mentioned is no guarantee
- This subject is **hard**
  - Understanding e.g. concurrency or database performance is not easy
- Ask the **experts**...
  - There's quite a few of them on IRC and wikitech-l
- ...but do what **you can do** first
  - Faster than waiting for people to get back to you
  - Saves the experts' time...
  - ...and they'll like you for trying first

Useful links

- References
  - http://svn.wikimedia.org/doc
    - Particularly, the DatabaseBase and Html classes
  - http://www.mediawiki.org/wiki/Security_for_developers
  - Other documentation on http://www.mediawiki.org
- Contact
  - #mediawiki on irc.freenode.net
  - wikitech-l@lists.wikimedia.org
- Credits
  - http://xkcd.com/327/
    - Their license says I have to credit them :)