Developing with Catalyst

as securely as possible

What do I need to do?

- Catalyst generally tries to help you be secure and most of the plugins are pretty good
- Some of it is just stuff you setup at the start of the project
- Other bits you need to keep in mind during development

Password storage

Simple answer - use bcrypt

Longer answer - make sure you use a crypto hash with salt that's well stretched. Scrypt and PBKDF2 with SHA512 are other reasonable alternatives.

DBIx::Class and password auth

Add a column to your password class,

```
"password",
 {
     encode_column => 1,
     encode_class  => 'Crypt::Eksblowfish::Bcrypt',
     encode_args   => { key_nul => 0, cost => 8 },
     encode_check_method => 'check_password',
 }
```

Tweak the cost to be as high as you can afford (it slows the function down, and hence slows down brute force).

Prevent Click Jacking

Use a combination of browser headers and javascript for older browsers.

The answer used to be `X-Frame-Options`. See the OWASP click jacking prevention guide. Now CSP frame-ancestors property appears to be the answer. Links at end.

Ensure you’re not exposing files

Be careful how the Static plugin works.

Don’t expose an sqlite auth database by accident.

Keep HTTPS encrypted

If your app needs to be on HTTPS make sure it stays that way. Take a look at HSTS.

Look at the `Strict-Transport-Security` header

HTTPS content

Pay attention to caching. As we've moved to a greater use of https the browsers have started to change to cache https content.

Ensure sensitive data is not cached

Check every layer of your stack.

Prevent content type sniffing

Prevent the browser from looking at text files and executing them as javascript because they look like it

X-Content-Type-Options: nosniff

Session rotation

Rotate session ids when users login and log out. If you have a high value site rotate them more frequently. OWASP suggests,

- If the level of privilege changes, logging in, sudo equivalent etc.
- After significant actions
- After a period of time or a number of accesses

CSRF

Using Plack::Middleware::CSRFBlock module is the simplest option

Barbie mentioned this one at a previous talk

Only downside is a second session cookie unless you deal with linking the plack and catalyst sessions

Error pages

Don’t expose too much information to the users (or hackers)

Make sure you log everything, but keep things nice for the users.

Catalyst::Plugin::CustomErrorMessage is the simple answer, or it can be used as inspiration

XSS

We always need to keep on top of XSS. Getting developers to put ‘| `html`’ on the end of everything is not much fun.

Template::Toolkit

Template::AutoFilter

Stick ‘| none’ when you want to avoid escaping

```
[% form.render | none %]
```

TT Alternative

Template::Alloy with AUTO_FILTER

Allows TT syntax, and similar ‘| none’

One caveat is that whenever a filter is specified it overrides the html filter you specify. If you’re doing something like html_linebreak you need to re-apply the filter, e.g.

```
[% var | html | html_line_break %]
```

Generating HTML in Perl

I generally discourage it.

If you do, make sure you encode your entities.

Simple answer is to use HTML::Entities.

There is a potentially faster module if that is important.

General theme of XSS protection

Use templates with auto filters to make it just work

Make avoiding encoding the exception, and ensure you think about what you’re doing

Make sure everyone is constantly aware

SQL Injection

DBI and DBIx::Class have always made it easy to avoid SQL injection.

That doesn’t mean you can’t accidentally introduce it.

SQL Injection

Be careful when generating queries. Don’t trust user input for field names. i.e. trusting names from HTML forms as the field names in your DB.

You know the column names you want to allow, start from that point and see if the user input mentions that.

DBIx::Class example

```
$rs->search({ $untrusted => $val });
```

If you're not quoting names that's a really simple SQL injection hole. (Quoting names is not the answer to ensure security)

Another DBIC clanger

Don’t do this,

```
my $found_rate = $tariffs_rs->search
 (
     {
         "'$destination_number'" => { 'like' => \"number_match || '%'" }, # user input on the left
     },
 )->first;
```

It’s almost an argument to turn on `quote_names` in itself.

```
return $self->search( \[ "? like number_match || '%'", [ dummy => $user_agent ]] )->first;
```

Don’t allow POST to be GET

POST actions should not be possible by GET with the possible exception of logout.

It shouldn’t be possible for a hacker to get a user to click on a link that does something the user doesn’t expect

3rd Party Extensions

- Watch release notes of projects
- Sign up to security lists if available
- Perhaps use https://www.perlmodules.net/
- Consider what your possible exposure is

Monitor your logs

Watch out for attacks

- Lots of 500’s from the same host may suggest an attack

What haven’t I covered?

- Hosting 3rd party content
- Hosting user generated content
- User entry forms - they generally deal with XSS threats fine
- Cookie entropy - it looks fine from the outside; if a pentest suggests you have 0 bytes of entropy, they likely goofed.

… continued

- SSL setup
- Testing
- Firewalls
- **Content Security Policy** <- look this up

That’s not to say these aren’t important. They’re just not in this talk.

Links

- https://www.owasp.org/index.php/Main_Page
- http://www.html5rocks.com/en/tutorials/security/content-security-policy/
- https://www.owasp.org/index.php/Clickjacking_Defense_Cheat_Sheet
- https://developer.mozilla.org/en-US/docs/Web/HTTP/X-Frame-Options
- https://www.owasp.org/index.php/Session_Management#Rotate_Session_Identifiers
- http://blogs.msdn.com/b/ieinternals/archive/2010/04/21/internet-explorer-may-bypass-cache-for-cross-domain-https-content.aspx
- http://stackoverflow.com/questions/174348/will-web-browsers-cache-content-over-https
- http://blogs.perl.org/users/olaf_alders/2012/07/using-plackmiddlewarecsrfblock-and-jquery-to-deal-with-cross-site-request-forgery.html
- http://www.html5rocks.com/en/tutorials/security/content-security-policy/

Modules

Template::Alloy

Template::AutoFilter

HTML::Entities

Crypt::Eksblowfish::Bcrypt

Catalyst::Plugin::CustomErrorMessage

Plack::Middleware::CSRFBlock