request you are making and then Grep for “Access-Control-Allow-Origin”. Even if you discover a certain endpoint which does contain this header but it doesn't contain any sensitive information, spend time trying to bypass it. Remember **developers reuse code** and this “*harmless*” bypass may be useful somewhere later down the line in your research.

SQL Injection

One thing to note is typically **legacy code** is more vulnerable to SQL injection so keep an eye out for old features. SQL injection can simply be tested across the site as most code will make some sort of database query (*for example when searching, it will have to query the database with your input*). When testing for SQL injection yes you could simply use ‘ and look for errors but a lot has changed since the past & these days a lot of developers have disabled error messages so I will always go in with a sleep payload as usually these payloads will slip through any filtering. As well as this it is easier to indicate if there is a delay on the response which would mean your payload was executed blindly. I typically use sleep payloads such as: (*I will use between 15-30 seconds to determine if the page is actually vulnerable*)

```
' or sleep(15) and 1=1#
' or sleep(15)#
' union select sleep(15),null#
```

When testing for SQL injection I will take the same approach as XSS and test throughout the web application. **Being honest I do not have as much success finding SQL as I do with other vulnerability types**.

Business/Application Logic

Why create yourself work when all the ingredients are right in front of you? By simply understanding how a website should work and then trying various techniques to create weird behaviour can lead to some interesting finds. For example, imagine you're testing a target that gives out loans and they've got a max limit of £1,000. If