12. DATATABLES Security.

12.1. Security.

Security is a fundamental topic in web-development and is a topic that should not be overlooked by any developer, from interns to CTOs. High profile hacking cases are frequently in headlines around the world, but with some careful thinking and planning you can quickly create secure applications.

12.2. Overview

This page will discuss web security attacks that are directly relevant to DataTables, along with methods for how you can combat them. Web security is a very wide ranging topic and it would be impossible to cover all topics here. For more general information about web and software security, refer to the excellent **OWASP** site. https://www.owasp.org/

12.3. Software versions

The first thing to do when considering software security is always to **run the latest version of the software that is available**. The latest versions will contain fixes for known issues, while those issues may be present in older versions. For DataTables the latest version is always available on the download page. https://datatables.net/download

12.4. Attack types

There are typically two attack types that are important to consider when working with DataTables:

- **Cross-Site Scripting (XSS)**

  https://en.wikipedia.org/wiki/Cross-site_scripting

- **Cross-Site Request Forgery (CSRF)**

  https://en.wikipedia.org/wiki/Cross-site_request_forgery

There are other forms of data leaks which can also occur, such as allowing non-logged in users to access sensitive data, allowing privilege escalation (*viewing data someone shouldn't be allowed to*) and SQL injection attacks. These are primarily issues for the application being created which uses DataTables.

12.5. Cross-Site Scripting

An **XSS attack** can be performed by allowing arbitrary Javascript or HTML to be executed on your own site. The injected Javascript could then perform actions with the current user's account or steal information. In the case of DataTables, an XSS attack could potentially be performed if you allow editing of your table's content or other input of data into your table by a user.

Consider for example if you allow editing for a cell in a table and the user were to enter: **<script>alert('Hi');</script>**. When the cell is displayed in the table, the ***alert***() would be triggered if the data is not encoded. Although an exceptionally simple example, if this were successful, a much more potent attack would also be possible.

12.6. Prevention

There are two options to stop this type of attack from being successful in your application:

- Disallow any harmful data from being submitted
- Encode all untrusted output using a rendering function.

For the first option your server-side script would actively block all data writes (*i.e. input*) that contain harmful data. You could elect to simply disallow all data that contains any HTML, or use an HTML parser to allow "*safe*" tags. It is strongly recommended that you use a known and proven security library if you take this approach - do not write your own!

The second option to **use a rendering function** will protect against attacks when displaying the data (*i.e. output*). DataTables has two built in rendering functions that can be used to prevent against XSS attacks; **$.fn.dataTable.render.text** and **$.fn.dataTable.render.number**.

The rendering functions can be used simply by assigning them to **columns.render** when creating your table - e.g.:

Example:

```
{
    data: 'product',
    render: $.fn.dataTable.render.text()
}
```

12.7. Cross-Site Request Forgery

A **CSRF attack** will force an end user (*typically without their knowledge - it all happens in the background!*) into executing unwanted actions on a site or application on which they are currently authenticated. For example, consider if you are logged into your online banking and then visit another, apparently harmless, page. If that page were to execute a transaction on your bank account, in **a hidden iframe**, the bank would accept the transaction thinking it came from you!

Prevention

To protect against this type of attack most systems will use a token that will be submitted on each data request in order to ensure that the end user is who they claim to be.

The DataTables ajax configuration object can be used as an object which accepts all of the same options as the **$.ajax method**, including the ability to submit headers and data. Depending on how your application expects the CSRF token, you can use one of a number of methods:

Setting a global header (*this ensures that all Ajax requests from the page have the CSRF token*):

Example:

```
$.ajaxSetup( {
    headers: {
        'CSRFToken': TOKEN
    }
} );
```

Submitting the token as a header value from the ajax configuration:

Example:

```
$('#myTable').DataTable( {
    ajax: {
        url: '...',
        headers: {
            'CSRFToken': TOKEN
        }
    }
} );
```

Submitting the token as part of the request data:

Example:

```
$('#myTable').DataTable( {
```

```
        ajax: {
            url: '...',
            data: function ( d ) {
                d.CSRFToken = TOKEN;
            }
        }
    } );
```

where in all cases **TOKEN** is the CSRF token (*again, how you get this token is a matter for the application or framework you are using*).