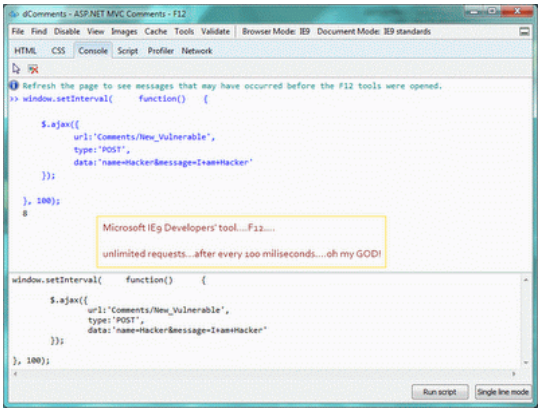
ASP.NET MVC security and hacking: Defense-in-depth

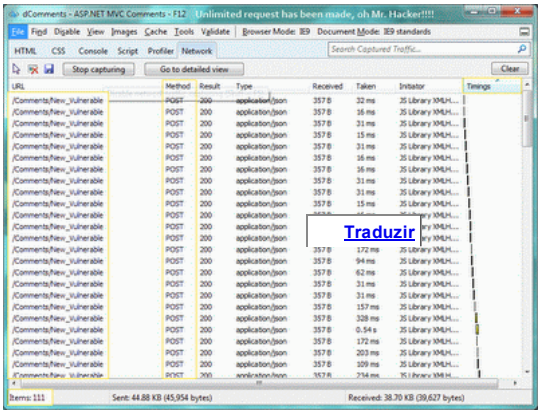
In this post, I'll try to explorer some common and well-known attacks and hacking techniques regarding ASP.NET MVC web app.

Note: *Asp.NET MVC 4 shipped with the support of HTML Sanitization and Anti-XSS libraries. So now you can easily protect your website against a lot of well-known XSS/CSRF attacks.*

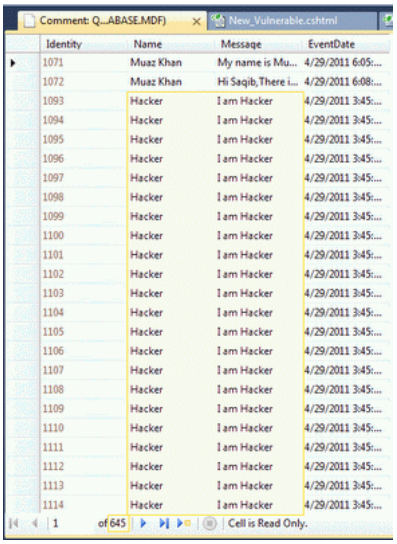
Cross-Site Scripting (XSS) attack

In simple words, injection of malicious scripts via input. Here is how to attack a JSON request.



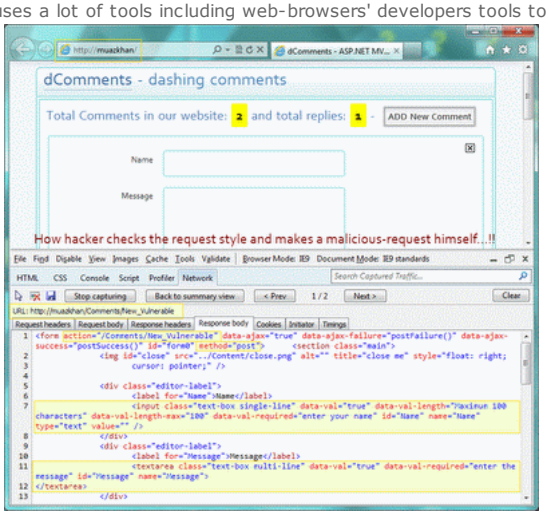


Don't forget to check the database table after the attack!



How hacker start attacks?

Actually hacker uses a lot of tools including web-browsers' developers tools to hack websites.



Hacker checks the:

- **Posting-Url** ---- i.e. form-action (Http POST, GET, DELET etc.)
- **Number of parameters and their types** ---- required or optional

As above figure shows that:

- **Posting-Url**: http://muazkhan/comments/new_vulnerable
- **Parameters**: 1) Name 2) Message
- There is no Anti-Forgery token

The absolute URL of the request is:

```
http://muazkhan/comments/new_vulnerable?name=something&message=something
```

Also hacker can make unlimited requests that cause the Daniel of Service (DoS) attack or Buffer Overrun attack.

Another XSS Attack: Redirect User to hacker's webpage

I suggest you never allow HTML in the input and use Wiki Mark-up instead!

Here is how you are vulnerable if you allow HTML in the input:

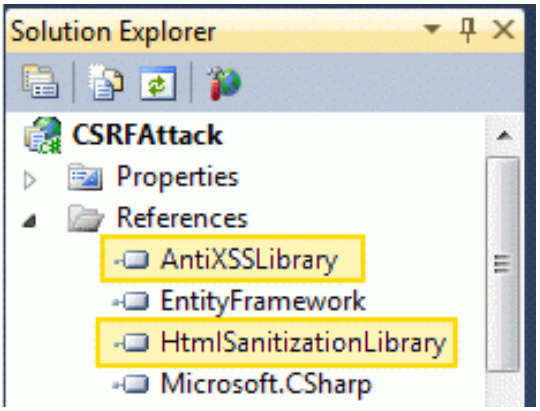
```
<script>
    (function ()
    {
        location = 'http://hacker.com';
    } )();
</script>
```

What has happened? The hacker has injected a malicious script via input and if this is a comments section or discussion forum then on each page load user is redirected to hackers given webpage!

Security against XSS attacks?

ASP.NET MVC 4 natively supports below mentioned libraries however for old version, you've to reference them yourself and their usage is highly recommended because these libraries are well-designed to help protect given input from well-known XSS/CSRF attacks.

- Anti XSS Library (for .NET)
- HTML Sanitization Library (for .NET)



Some good rules for avoiding XSS attacks

These rules are taken from http://www.educatedguesswork.org/.

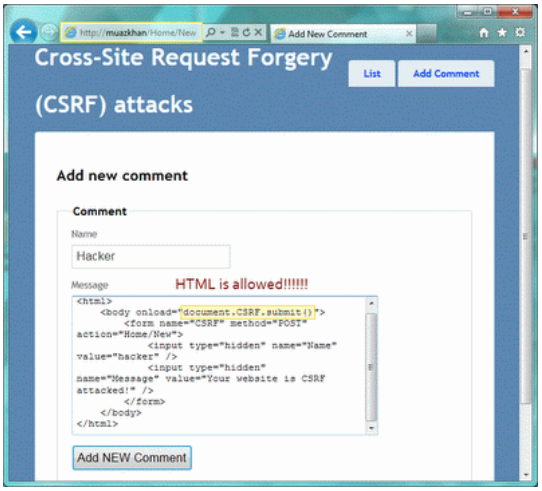
- Untrusted data MUST NOT be transmitted in the same HTTP responses as HTML or JavaScript. In particular, the main HTML document SHOULD be static (and therefore cacheable for a long time).
- When transmitted from the server to the client, untrusted data MUST be properly encoded in JSON format and the HTTP response MUST have a Content-Type of application/json.
- When introduced into the DOM, untrusted data MUST be introduced using one of the following APIs:
- 1) Node.textContent 2) document.createTextNode 3) Element.setAttribute (second parameter only)

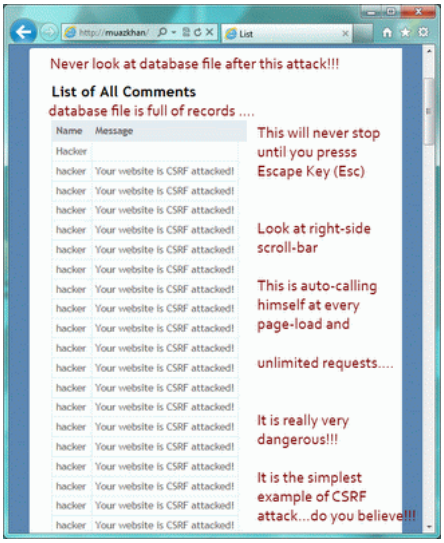
Cross-Site Request Forgery (CSRF) Attack

First of all, please Download the demo project and try CSRF attacking yourself.

```
<html>
    <body onload="document.CSRF.submit()">
        <form name="CSRF" method="POST" action="Home/New">
            <input type="hidden" name="Name" value="hacker" />
            <input type="hidden" name="Message" value="Your website is CSRF attacked!" />
</form>
    </body>
<html>
```

Copy above HTML code and paste into the **Message** textarea:





Security against CSRF attacks

You need to use AntiXSSLibrary or HtmlSanitizationLibrary for security against XSS (Cross-Site Scripting) attacks as well as CSRF or XSRF (Cross-Site Request Forgery) attacks.

```
comment.Message = Sanitizer.GetSafeHtml(comment.Message);
```

After using above mentioned libraries, hacker's malicious-code is no-more malicious:

```
<html>
    <body onload="document.CSRF.submit()">
        <form name="CSRF" method="POST" action="Home/New">
            <input type="hidden" name="Name" value="hacker" />
            <input type="hidden" name="Message" value="Your website is CSRF attacked!" />
        </form>
    </body>
<html>
```

The method (`Sanitizer.GetSafeHtml(..)`) transforms and filters HTML of executable scripts. A safe list of tags and attributes are used to strip dangerous scripts from the HTML. HTML is also normalized where tags are properly closed and attributes are properly formatted.

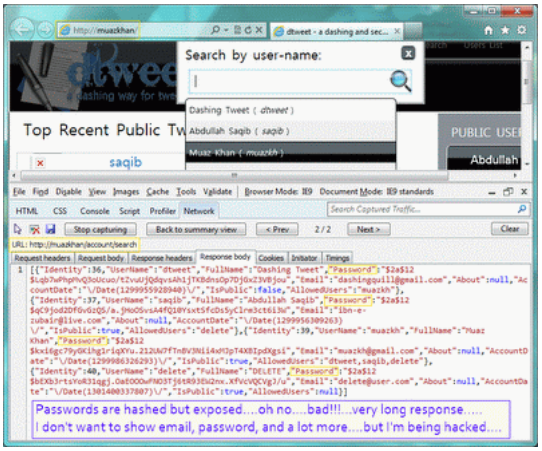
ValidateAntiForgeryToken attribute and CSRF attacks

`ValidateAntiForgeryToken` attribute helps protect you against CSRF attacks but not protect you against XSS attacks, so please use it with conjunction to AntiXSSLibrary and HtmlSanitizationLibrary.

```
using Microsoft.Security.Application;

[HttpPost, ValidateInput(enableValidation:false), ValidateAntiForgeryToken]
public ActionResult New(Comment comment)
{
    comment.Message = Sanitizer.GetSafeHtml(comment.Message);
    _comments.Add(comment); _comments.Save();
    return RedirectToAction("List");
}

---------------------------

@using (Html.BeginForm()) {
    @Html.AntiForgeryToken()
    ...
}

---------------------------

<input name="__RequestVerificationToken" type="hidden" value="BvzGsNxa0Jmrw5XZfMIIsPFyt0qNac0LYxJqBJiOahrnGvG7+Uh9qXckeF6iVa7(
```

Common Mistakes or Common Security Holes

In this section, we'll try to know how our common coding mistakes help the hacker to see our secure data or data that we never want to show. We'll use an Open-Source project: dtweet, to know how our secure data is exposed to hacker:



BCrypt library helped me hash the password; however, please do your best to protect your code against known and possible attacks. Right! Following code shows what was my mistake and how I can mitigate the risk:

```
public JsonResult Search() /* Exposed to Hack */
{
    /* I am passing whole 'Users' object */
    return Json(new dtweetDataContext().Users);
}
public JsonResult Search() /* Secured */
{
    /* I am passing required fields only, not whole object!! */
    return Json(new dtweetDataContext().Users.Select(u => new{u.UserName, u.FullName}));
}
```

And now hacker is being shamed:

Some good Rules

- Never, ever trust any data your users give you. Ever.
- It is recommended to know yourself and your unexpected enemy.
- Hackers can easily hack your websites because of your bad assumptions, misinformation, and lack of education.

Today, security is possible with CSP (Content Security Policy)!
Read my full blog post:

Exploring CSP (Content Security Policy) using ASP.NET MVC

Comentários

Você não tem permissão para adicionar comentários.

Muaz Khan 09:25 18/10/2012 • Comentários desativados
ASP.NET MVC Security tutorials: http://www.asp.net/mvc/tutorials/security

Overview of ASP.NET MVC security:--------- http://www.asp.net/mvc/overview/security