My name is Warren Moynihan and I am a member of the Security Systems Ethical Hacking team. In the following video I will be talking about one of the more common weaknesses found in insecure code: Cross-Site Scripting.

This type of vulnerability is currently ranked #3 on the OWASP Top 10 chart and is a very commonly exploited vulnerability type.

So what is cross-site scripting? Well, basically, an attacker can inject untrusted snippets of JavaScript into your application without validation. This JavaScript is then executed by the victim who is visiting the target site. There are 3 Primary types of XSS, Reflected, Stored, and DOM based.

In Reflected XSS, an attacker sends the victim a link to the target application through email, social media, etc. This link has a script embedded within it which executes when visiting the target site.

In Stored XSS, the attacker is able to plant a persistent script in the target website which will execute when anyone visits it.

With DOM Based XSS, no HTTP request is required, the script is injected as a result of modifying the DOM of the target site in the client side code in the victim's browser and is then executed.

Approximately 17% of all applications tested are vulnerable to cross-site scripting.

So what are the potential risks of cross-site scripting? Well, an attacker could compromise or take over the victim’s user account in the application. They could retrieve data from the target web application, modify content on the target page, redirect the victim to another malicious or spoof site, or use it as a platform to install other malware on the victim's system.

The consequences of any of the above can seriously impact your ability to conduct business, your customers, and your organization’s reputation.

So how can you protect yourself from cross-site scripting? Well, there’s many different strategies. One common approach to testing for XSS involves attempting to inject a harmless JavaScript alert box into entities sent in a request. If this alert is reflected in the response, and executable in a browser, it offers some basic validation that injecting a script is possible. An attacker would typically write a more elaborate script to perform a variety of malicious functions.

Use a vetted library or framework that does not allow this weakness to occur, or provides constructs that make it easier to avoid. In any situation where data will be output to a page, especially data that was received from external inputs, use encoding for non-alphanumeric characters. Set all cookies with the HTTP only attribute. This helps the user’s cookie to be protected from being accessible to malicious client side scripts that use document.cookies. Assume all input from external sources is malicious. White list inputs that conform to strict standards, and reject or transform anything that doesn’t.
In the following example I’ll demonstrate a couple of ways an attacker can use cross site scripting.

In this first example I’m going to demonstrate a basic cross-site scripting vulnerability on demo.testfire.net. for this example, we’ll need access to a local proxy. In this case, I’m using AppScan Traffic Viewer. In a browser navigate to demo.testfire.net. The password for jsmith is demo1234. You cn see here that my proxy has trapped incoming requests. And in this case it has attacked amCreditOffer. I’m going to insert a basic script tag in an alert window to ensure the script will execute and save the request. And tell the proxy to release it. There you have it. The alert box has been executed in my browser displaying 123.

In this next example I’ll execute a cross-site scripting attack directly in a browser. In the search window of demo.testfire.net, I’ll enter a cross-site script with an alert window displaying 456. Great. You can see that this area is vulnerable to cross-site scripting.

Now let’s try something a little more fun. Let’s see if we can redirect our victim to a site of our choice. Apparently we’ve been redirected to a hit video form the 1980’s, Rick Astley’s “Never Gonna Give You Up.” We can now embed a link in a screen, in an email or an online form. Now, most attackers won’t have this sense of humor and won’t redirect you to a funny music video. They’re after something more valuable. Let’s see if we can use this attack and steal some document cookies. Now instead of using Traffic Viewer as a proxy, I’ll set it up as a server. In this injection I’m going to redirect the user to my evil server, but I’m going to pass along ‘document.cookie’ as a parameter. Someone logged into this site could unwittingly put this link in another tab and these cookies are then passed to the attacker. As you can see, my evil server did receive the request from the victim and here are their cookies. Now I can use these cookies to impersonate the victim and log into their bank account and pose as them.

So how can we test for these vulnerabilities? AppScan's test policy includes a wide variety of Cross-site scripting tests, as well as an Cross-site scripting analyzer which dynamically decides from millions of potential variants the best ones to test. You can use the search dialogue in the test policy window to search for specific tests or variants you may want to test and then enable/disable them as desired. Also under 'Test Options', be sure to check the box for 'Enable JavaScript Security Analysis' if you want AppScan to do a code scan of the client side JavaScript for DOM-Based XSS issues.

Here's how to configure AppScan to automatically detect an XSS vulnerability. {demo} In AppScan, we'll type our starting url and record a login: jsmith, and demo1234. It's always important to record a login with in-session detection. In the test policy window there are a number of options in the default policy. Most policies include cross-site scripting. To save time, I'll disable these tests and only enable cross-site scripting in the filter. At this point I can run a full scan. To save time I'll perform a manual explore of the areas I'm concerned with. I'll login and the first cross-site scripting we manually demonstrated on the apply.aspx page. In the search window, we'll search an arbitrary string here. The scan is analyzing our manual explore for data. Ok you can see that AppScan detected and sure enough, the amCreditOffer is vulnerable.

This concludes our coverage of the 3rd OWASP Top 10 category. I hope you found this information useful. I wish you best of luck in writing and maintaining secure software!