Hi,

My name is Jonathan Fitz-Gerald and I'm a member of the IBM Security Systems Ethical Hacking team. Today I'll be talking to you about insecure direct object references, which is currently ranked as 4th on the OWASP top 10 page.

In this video, we will cover what an insecure object reference is, how it can be exploited, how it can be mitigated and how we can detect it with AppScan.

So, what is a direct object reference?

In many cases, your application must perform some sort of action based on user-supplied parameters. This might happen when loading a particular resources such as a file. In these cases, parameters may be used internally within the application in ways that the developer didn't anticipate, giving attackers access to sensitive parts of the application.

In this example, we'll be demonstrating that the content parameter is vulnerable to insecure direct access reference.

The value has minimal validation and no access checks are performed on the value supplied by the user. In this example, we've combined the attack with a null byte, which allows us to smuggle values past the simple validation function that checks for .htm or .txt at the end of the file name. We use these two vulnerabilities to show the contents of the default.aspx.cs file which serves as the main page for this website.

If we look carefully at the sidebar, we can see that the target of the links is the default.aspx file with one parameter named content. There is a simple validation check on the filename to verify that content contains a filename that ends in .txt or .html. For the purpose of this example, we'll get around that with a null byte attack. Briefly put, ./net will see the null byte as part of the string. When the contents of content are handed to the OS's API it will think the string ends at the null byte and the .htm portion of the string will be trimmed off.

Let's navigate to the following URL which will show the contents of the default.aspx.cs file. As you can see in the code, we can directly set the fileToLoad parameter with the content parameter. This is interesting, but maybe we can get something a little more juicy.

Let's try to get the web.config file.

Now that we have web.config file, we can see the path to the Microsoft Access database that drives this application. Let's see if we can get a little bit more info.

Now that we've used the default.aspx to load the Microsoft Access database, we can see that there is some sensitive information in the results, such as user names and passwords.

Now that we've demonstrated the vulnerability, let's talk about how to protect against it. At the simplest level to prevent this vulnerability, we need to use an associative array to create a map of objects.

Here's an example of what the key value pairs might look like for such an array.

When default is passed as the value of the content variable, then the corresponding value for that key would be used. In this case, the corresponding value is index.html.

Now that we've discussed this vulnerability and how to mitigate it , lets see if we can detect this vulnerability using AppScan.

AppScan has a number of tests that detect direct object references of different types. One such test is “Path Traversal”. “ Path Traversal” could be considered a direct access reference attack because of the nature of this particular vulnerability in Altoro Mutual. Had the developers used an associative array, hash map or something similar, then this vulnerability would not be possible.

Now that my AppScan is running, let's configure a quick scan. I’m going to set the scan target to Altoro Mutual and I'm going to configure the scan to only scan for path traversal, just to keep things simple.

As you can see, AppScan detected the parameter that is vulnerable. On the Issues tab, you can get a breakdown of tests that were ran to detect the issue and on the Fix Recommendations tab you can get advice on how to fix it.

I'd like to thank you for watching this video and I hope you found this video informative.