Script Injection Attack: Smoking Gun?

Posted At : September 18, 2009 1:07 PM | Posted By : Mark Kruger
Related Categories: Coldfusion Security

Many of you may know there is a web server attack going on in the wild that involves appending a JS script to all the htm, php, cfm, js, jsp files found on a server. If you are unfamiliar with this attack see some of my previous posts like **this one** for more of an explanation. While I have found the script that actually does this dirty deed and I have combated this issue on numerous servers by now, I have never really been confident that I have discovered where the attack actually begins (i.e. how this file gets on the server to begin with). Yesterday I was made aware of a technique that *might* be the smoking gun. It has been tested by some folks I trust and I want to give a full explanation here to assist all those Muse readers who battle the bad guys at the server level.

If you are a technician or network operations professional who is trying to scan your way out of this attack, I'm afraid you are probably out of luck (but keep reading anyway). This attack specifically targets application code - not just CF but ASP, JSP, PHP and any others. All of them can be subject to this problem because it has to do with insecure coding, not specific platform vulnerabilities. I would add that if you find your code vulnerable don't feel too bad. This exploit is clever enough to get by code that *seems secure* as we shall see. If you are a web developer of any stripe you should **definitely read this post**. The examples are in ColdFusion, but you will be able to extrapolate for your own language or technology pretty easily.

Uploading User Files

Allowing users to upload files is a pretty typical feature on the web these days - and not just with ColdFusion. The ability to add content or resources in the form of a file is a common feature of forums, email applications, galleries, video sites, document sharing sites et al. Naturally it is very important that you are able to secure your server from malicious files uploaded using the tools you provides. In ColdFusion you can use the "accept" attribute to specify MIME types to accept. The *browser* determines the MIME type on upload using the file extension. Not only is this useless from a security standpoint, developers have found that this is not one hundred percent reliable (depending as it does on the client) so they have taken to writing custom code to filter out pernicious attempts to infiltrate the server. Consider this code for example:

```
<!--- allow these extensions --->
<cfset extlist = 'jpg,gif,doc,xls,pdf'/>

<!--- Place the file --->
<cffile action="UPLOAD"
        filefield="myFile"
        destination="#expandpath('../files')#">

<!--- Delete files that don't match extensions --->
<cfif NOT listfind(extlist,listlast(file.serverfile,'.'))>
    <cffile action="DELETE" file="#expandpath('../files')#/#file.serverfile#">
</cfif>
```

This code checks the extension of the file and excludes any files that don't match a list of allowed extensions. So, for example, if a user wanted to upload a file with a .cfm

extension and run it on your server they would be unable to do so - right? That makes this code safe... or does it?

There are a couple of reasons why this code may *not be safe* - not the least of which is that code could be uploaded with a spurious extension and used in some other way. For example, in my post on the **Iframe injection hack** a user was able to upload a file using ASP that *seemed* to be a GIF file, but in actuality was a CDX file which an IIS server in its "default configuration" (and no server should *ever be left in it's default configuration!*) is set up to handle using the asp.dll. In that case a malicious user was able to fool the upload script into thinking it's handling a gif when in actuality it is handling and executing malicious code. But that is not the exploit we are discussing here. In fact the exploit we are discussing here is even more devious and clever.

Step 1: Prepare the File

The way this works is pretty simple. A malicious user prepares a CFM script that does something naughty like emails your directory structure to himself or downloads additional files to the server via cfftp. He calls his script something random that is not likely to be on your server. Let us suppose he names his .cfm file "bob.cfm". Having his ignoble file in hand he must now figure out how to get it on your server where it can be executed.

Step 2: Innocuous Probe

Our hacker registers for an account on that site you built for mole lovers (Ilovemoles.com) and pokes around a bit (or should I say "digs" around a bit). Part of your site allows you to upload pictures of moles that you think are cute or unique. He uploads a photograph, and then views it his browser. He takes note that your code is serving the picture from www.ilovemoles.com/userphotos/. That path becomes his target directory. So his first attempt is to simply upload something other than a photo. He tries uploading Bob.cfm. Naturally, because you are not *just* a mole lover but also a good programmer you have precluded the upload of .cfm files. Our too clever hacker moves on to step 2 - the load test scenario.

Step 3: Load Test preparation

Our hacker sets up his load test tool. Load test tools can be made to do different things, but one thing they do well is to simulate a barrage of HTTP requests at a certain level. Our hacker begins by setting up his load tester to test against one URL - www.ilovemoles.com/userphotos/bob.cfm. He sets it to run for 3 or 4 minutes and configures it to simulate a fairly high level of traffic (even a low level might work with repetition).

Step 5: Penetration

Now our hacker (I'm starting to feel an avuncular fondness for him) begins his load test. Naturally the load test is returning a boat load of 404 errors which it is dutifully logging. With the load test steaming along he logs into his ilovemoles.com account and uploads bob.cfm as if it were a photo.

Hold the Phone!

Let's stop right here and consider what is happening under the hood with our code.

- *CFFILE action = "upload"* - ColdFusion Creates a file handle, copies in the binary

to the location where the disk subsystem handles the linkage of all the little block locations.
- The file handle is released and ColdFusion receives back basic meta data about the file and the upload operation in the form of the "file" object (file.clientfile, file.serverfile, file.size etc).
- *CFIF Statement* - The code Checks the extension on the file and says to itself "Hmmmm... this isn't a jpg or gif or office document. I better delete it."
- *CFFILE Action = "delete"* - CF Gets a new file handle using the path and file name and "deletes" the file - an operation which simply removes the file location marker from the disk system index.

The thing to take careful note of here is that ***in between file handle 1 and file handle 2 this file exists on the disk and is accessible as a URL***. Not only that, but file operations being what they are the process has a natural latency to it from marshalling data in and out of the underlying disk subsystem. So between the release of file handle 1 and the acquisition of file handle two there might be 10, 20, 50 milliseconds or more - a number that would vary wildly based on capacity, usage and the type of server.

It is this tiny little window of opportunity that the hacker uses to his advantage. His load test code happily churning out 404 errors is firing off http requests at a rapid clip and one of them can hit the file at the right moment and grab a handle. The file is compiled into a CF class and executed - meanwhile your delete code will simply sit there and wait for the operating system to return a new file handle for it's delete operation. The OS obliges and does exactly that. As soon as CF is done compiling the code to Java Bytecode your excellent validation code happily deletes the evidence that the file was ever there to begin with. In fact, if you eschew web logging because you use something like Google analytics you will *never* know this file was being probed for or was ever on your disk.

Lessons Learned

Before we talk about prevention let me reiterate again that this approach is not a "ColdFusion" flaw - even though my example uses ColdFusion. Any web application that allows for upload directly into a web accessible directory that is configured with script permissions will potentially be vulnerable (PHP, ASP, JSP et al). Here's my list of preventative measures:

- **Protect this House** - Do not upload directly into a web accessible directory. In my view the proper approach is to upload to a directory that is *outside* the web root and then *move* the file to its final destination once it is checked. ColdFusion help docs often suggest the temp directory (use the function getTempDirectory()) and that is an acceptable solution to be sure - but any directory is fine as long as it is not accessible via a URL.
- **Consider Cfcontent** - In fact, unless you are serving *just* images you should consider using CFCONTENT for things like office docs and other file types. That way you can store your user files completely outside of the web root. CFCONTENT will insure that the file is delivered as binary content to the web browser and not executed on the server.
- **Script and Execute Permissions** - For directories where you are storing user content you should disable execute and script permissions. And FYI - you typically don't need execute permissions under any circumstances.
- **Remove Superfluous Handlers** - IIS comes installed with a boat load of handlers

for various things. They should all be removed unless you are explicitly going to use them for something. Stuff like .cer, .cdx, .htr etc - are all virtually *never* used, let alone on a web server open to the public. These mappings are there for various internal and domain networking usage. I know it comes as a shock, but Windows Server is primarily used in enterprise networking for managing the needs of large groups or *internal* users for things like access, printing and file sharing. A web server is only one of it's uses (and not always it's best one :).

Some Final Thoughts

I'm always looking for a smoking gun. With the recent spate of attacks I have been frustrated by the fact that I am unable to determine the attack vector. I have suspected FTP, FCK Editor and a variety of insecure coding techniques. This explanation comes as close to explaining the "stealth" attacks as any I have run across. I have to say I'm still not completely convinced in spite of how neatly this all fits together. Still, it is one more thing to examine. Finally I would like to express my thanks to a group of ColdFusion developers who shall remain nameless but helped me uncover and vet this attack (thanks Gurus).