Using Resource Obfuscation to Reduce Risk of Mass SQL Injection

Lori MacVittie, 2009-05-03

One of the ways miscreants locate targets for mass SQL injection attacks that can leave your applications and data tainted with malware and malicious scripts is to simply seek out sites based on file extensions. Attackers know that .ASP and .PHP files are more often than not vulnerable to SQL injection attacks, and thus use Google and other search engines to seek out these target-rich environments by extension.

Using a non-standard extension will not eliminate the risk of being targeted by a mass SQL injection attack, but it can significantly reduce the possibility because your site will automatically turn up in cursory searches seeking vulnerable sites. As Jeremiah Grossman often points out, while cross-site scripting may be the most common vulnerability discovered in most sites, SQL injection is generally the most exploited vulnerability, probably due to the ease with which it can be discovered, so anything you can do to reduce that possibility is a step in the right direction.

You could, of course, embark on a tedious and time-consuming mission to rename all files such that they do not show up in a generic search. However, this requires much more than simply replacing file extensions as every reference to the files must also necessarily be adjusted lest you completely break your application. You may also be able to automatically handle the substitution and required mapping in the application server itself by modifying its configuration.

Alternatively there is another option: resource obfuscation. Using a network-side scripting technology like iRules or mod_rewrite, you have a great option at your disposal to thwart the automated discovery of potentially vulnerable applications.

HIDE FILE EXTENSIONS

You can implement network-side script functionality that simply presents to the outside world a different extension for all PHP and ASP files. While internally you are still serving up a*pplication.php* the user – whether search engine, spider, or legitimate user – sees *application.zzz*. The network-side script must be capable of replacing all instances of ".php" with ".zzz" in responses while interpreting all requests for ".zzz" as ".php" in order to ensure that the application continues to act properly.

The following iRule shows an example of both the substitution in the response and the replacement in the request to enable this functionality:

```
when HTTP_REQUEST {

   # This replaces ".zzz" with ".php" in the URI
   HTTP::uri [string map {".zzz" ".php"} [HTTP::uri]]
}

when HTTP_RESPONSE {

  STREAM::disable

  If {[HTTP::header value "Content-Type"] contains "text" } {

    STREAM::expression "@.php@.zzz@"
    STREAM::enable
  }
}
```

One of the benefits of using a network-side script like this one to implement resource obfuscation is that in the event that the bad guys figure out what you're doing, you can always change the mapping in a centralized location and it will immediately propagate across all your applications – without needing to change a thing on your servers or in your application.

HIDE YOUR SERVER INFORMATION

A second use of resource obfuscation is to hide the server information. Rather than let the world know you're running on IIS or Apache version whatever with X and Y module extensions, consider changing the configuration to provide minimal – if any – information about the actual application infrastructure environment.

For Apache you can change this in *httpd.conf:*

```
ServerSignature Off
ServerTokens Prod
```

These settings prevent Apache from adding the "signature" at the bottom of pages that contains the server name and version information and changes the HTTP Server header to simply read "Apache".

In IIS you can disable the Server header completely by setting the following registry key to "1".

```
HKLM\SYSTEM\CurrentControlSet\Services\HTTP\Parameters\DisableServerHeader
```

If you'd rather change the IIS Server header instead of removing it, this KnowledgeBase Note describes how to use URLScan to achieve your goals.

If you'd like to change the HTTP Server header in a centralized location you can use mod_security or network-side scripting to manipulate the Server header. As with masking file extensions, a centralized location for managing the HTTP Server header can be beneficial in many ways, especially if there are a large number of servers on which you need to make configuration changes.

Using iRules, just replace the header with something else:

```
when HTTP_RESPONSE {
  HTTP::header replace Server new_value
}
```

Using mod_security you can set the SecServerSignature directive:

```
SecServerSignature "My Custom Server Name"
```

These techniques will not prevent your applications from being exploited nor do they provide any real security against an attack, but they can reduce the risk of being discovered and subsequently targeted by making it more difficult for miscreants to recognize your environment as one that may be vulnerable to attack.

F5 Networks, Inc. | 401 Elliot Avenue West, Seattle, WA 98119 | 888-882-4447 | f5.com

F5 Networks, Inc. Corporate Headquarters info@f5.com	F5 Networks Asia-Pacific apacinfo@f5.com	F5 Networks Ltd. Europe/Middle-East/Africa emeainfo@f5.com	F5 Networks Japan K.K. f5j-info@f5.com