Wednesday, October 15, 2014

How I made two PHP files with the same MD5 hash

I recently posted a link on twitter to two PHP scripts which have different behaviours but the same MD5 hash. To verify this download the files:

```
$ wget https://s3-eu-west-1.amazonaws.com/md5collisions/a.php
$ wget https://s3-eu-west-1.amazonaws.com/md5collisions/b.php
```

Then check their MD5 hash:

```
$ md5 a.php b.php
MD5 (a.php) = 62e1d0d1620581693435aa75f5b6c964
MD5 (b.php) = 62e1d0d1620581693435aa75f5b6c964
```

Now run each file:

```
$ php a.php
Behaviour A
$ php b.php
Behaviour B
```

The files behave differently and yet hash to the same hex string using MD5. A quick check with another hash algorithm (SHA1 in this case) confirms the two files are in fact different:

```
$ shasum a.php b.php
6ef81f904b84d6ea53048004a26157816a85cfa3  a.php
d191a5eb7557231f461c09cb4dc565a23336778a  b.php
```

So how were these files made? Well if you look at the source of file you will find there is very little difference between the two files.

```
cat a.php
```

```
<?php

$space = "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA";
if ('?G.???-)?????+?B?8????A???P?;"??P?ZU?5?$q4F?1c?-ŏC?S
?e?MIM???~?2?:?]b??\??/5???^?gn~3D7' == '?G.???-)?????+?B?8????A???P?;"??P?ZU?5?$q4F?1c?-ŏC?S
?e?MIM???~?2?:?]b??\??/5???^?gn~3D7' ) {
    echo 'Behaviour A',PHP_EOL;
} else {
    echo 'Behaviour B',PHP_EOL;
}
```

```
$ cat b.php
```

```
<?php


$space = "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA";
if ('?G.???-)??????+?B?8????A???P?;"??Py[U?5?$q4F?1??-ŏC?S
?e?MIM???~?2?:?]b?p?\??/5???^?g?~3D7' == '?G.???-)?????+?B?8????A???P?;"??P?ZU?5?$q4F?1c?-ŏC?S
?e?MIM???~?2?:?]b??\??/5???^?gn~3D7' ) {
    echo 'Behaviour A',PHP_EOL;
} else {
    echo 'Behaviour B',PHP_EOL;
}
```

Spot any difference? It may be hard to spot but there are actually subtle differences in the first of the chunks of binary data.

So whats with all that binary data? The binary data represents a collision in the MD5 algorithm. Collisions are surprisingly easy to create for MD5 following an attack on MD5 first published by Wang in 2004. What's more you can choose the starting value for the collision you want to create so it does not have to be at the start of the file. This is because MD5 in common with most hash algorithms operates on one block of data (64 bytes in MD5s case) at a time and then uses this value as the starting point for the next block. The next two blocks differ between the two files, however, they have been specially created to collide when inserted at this location. All subsequent blocks are the same in the two files. This is the method I used to generate the files.

1) First I created a file which is exactly 64 bytes long and contains all of the content up to the first single quote. The A’s are just there to make it exactly 64 bytes long

```
<?php

$space = "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA";
if (‘
```

2) Run this content though the MD5 algorithm but a slightly modified version which does not add the length or any padding to the message, I used my own implementation of MD5 in PHP and commented out the parts where padding is appended. You can get the code for this here https://gist.github.com/natmchugh/fbea8efeced195a2acf2.This produces the hash 32f6df28e4a110970fb7f9938bee1686.

3) Use this hash as the initial value to find an MD5 collision. The fastest collision script I have found is Marc Stevens fastcoll See https://marc-stevens.nl/research/ . If creating collisions in MD5 sounds like it will take a long time - it doesn’t. It varies how it takes but in this case it took about 15 seconds to produce 2 messages that collide with the initial value of 32f6df28e4a110970fb7f9938bee1686.

```
$ time ./fastcoll –i 32f6df28e4a110970fb7f9938bee1686 –o a b
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Using output filenames: 'a' and 'b'
Using initial value: 32f6df28e4a110970fb7f9938bee1686

Generating first block: ....................................
Generating second block: S10.............................................
Running time: 14.9727 s

real     0m14.981s
user     0m14.969s
sys     0m0.006s
```

4) I then appended the two messages produced to file created in 1 to give two files.

5). The two files should now be of equal length and have the same MD5 however they will not run as PHP. So I added the statement between the two lots of binary data.
‘ == ‘

6) Next I appended the collision a to both files. This produces a statement that will be true in file a (since the 2 binary blobs are indeed the same) but false in file b.

7) I appended the same PHP to both messages to complete the if statement and produce the output you see. So effectively we have two files where the first 64 bytes are identical. The next 128 bytes are a collision in MD5 and then all other text following that is identical. Not surprisingly this produces the same MD5 hash when passed though the whole MD5 algorithm including padding.

While this attack may appear useless since the files are filled with gobbledygook and rely on a fairly obvious if statement there are methods to create files with the same MD5 hash which do not rely on a switch and where the rest of the file contents can be completely different

Nat at 8:49 PM

Share

No comments:

New comments are not allowed.

❮

Home

View web version