

# Flickr's API Signature Forgery Vulnerability

**Thai Duong and Juliano Rizzo**

Date Published: Sep. 28, 2009

Advisory ID: MOCB-01

Advisory URL: [http://netifera.com/research/flickr\\_api\\_signature\\_forgery.pdf](http://netifera.com/research/flickr_api_signature_forgery.pdf)

Title: Flickr's API Signature Forgery Vulnerability

Remotely Exploitable: Yes

## **1. Vulnerability Description**

Flickr is almost certainly the best online photo management and sharing application in the world. As of June 2009, it claims to host more than 3.6 billion images. In order to allow independent programmers to expand its services, Flickr offers a fairly comprehensive web-service API that allows programmers to create applications that can perform almost any function a user on the Flickr site can do.

The Flickr's API consists of a set of callable methods, and some API endpoints. To perform an action using the Flickr's API, you need to select a calling convention, send a request to its endpoint specifying a method and some arguments, and will receive a formatted response.

Many methods require the user to be logged in. At present there is only one way to accomplish this. Users should be authenticated using the Flickr Authentication API. Any applications wishing to use the Flickr Authentication API must have already obtained a Flickr's API Key. An 8-byte long 'shared secret' for the API Key is then issued by Flickr and cannot be changed by the users. This secret is used in the signing process, which is required for all API calls using an authentication token. In addition, calls to the flickr.auth.\* methods and login URLs pointing to the auth page on Flickr must also be signed. For more details, please read the Flickr Authentication API Spec [1].

This advisory describes a vulnerability in the signing process that allows an attacker to generate valid signatures without knowing the shared secret. By exploiting this vulnerability, an attacker can send valid arbitrary requests on behalf of any application using Flickr's API. When combined with other vulnerabilities and attacks, an attacker can gain access to accounts of users who have authorized any third party application. Additionally, if an application uses PHPFlickr  $\geq$  1.3.1, an attacker can trick users of that application to visit arbitrary web sites. This may apply for other Flickr's API libraries and applications as well.

## **2. Vulnerable Web Services**

A lot of other web sites provide API service whose architecture is the same as Flickr's API. They are potentially vulnerable. We don't have a complete list, but here are some notable web sites:

- \* DivShare <http://www.divshare.com/>
- \* iContact <http://www.icontact.com/>
- \* Mindmeister <http://www.mindmeister.com/>
- \* Myxer <http://www.myxer.com/>
- \* Remember The Milk <http://www.rememberthemilk.com/>
- \* Scribd <http://www.scribd.com/>
- \* Vimeo <http://www.vimeo.com/>
- \* Voxel <http://www.voxel.net/>
- \* Wizehive <http://www.wizehive.com/>
- \* Zoomr <http://www.zoomr.com/>

Please note that we haven't tested these web sites. They are included here because they describe the same signing process in their API documentation.

## **3. Vendor Information**

An initial notification was sent to Yahoo! Flickr on Sep. 5, 2009. A copy of this advisory was sent to Yahoo! Flickr on Sep. 13, 2009. Yahoo! Flickr replied on Sep. 14, 2009 to acknowledge the vulnerability. Yahoo! Flickr sent us an email on Sep. 23, 2009 to say that they were going to deploy a fix in the same week.

An initial notification was sent to the vendors listed in Section 2 on Sep. 17, 2009. Another copy of this advisory was sent to them on Sep. 24, 2009. Here are the responses from some of them:

\* Remember The Milk said that they have investigated and confirmed that the Remember The Milk API is not vulnerable to this particular known issue.

\* Vimeo tried to fix the issue by making sure that the first parameter after sorting is always `api_key` and failing if it isn't. This fix doesn't work because we can make the first parameter be `api_key` and still append new data to the request.

No other vendor provided details about their plans to deploy fixes.

## 4. Credits

This vulnerability was found and researched by Thai Duong from VNSecurity/HVAOnline and Juliano Rizzo from Netifera. Greeting to all members of VNSecurity, HVAOnline and Netifera.

The authors would like to thank Huong L. Nguyen, rd, Gunther, Bruce Leidl, and Alex Sotirov for reading and editing the draft of this advisory.

## 5. Technical Description

In Section 5.1 we introduce Flickr's API request signing process and its vulnerabilities. In Section 5.2 we describe the length-extension attack against Merkle-Damgård hash. In Section 5.3 we describe our attack against Flickr's API. In Section 5.4 we discuss some exploitations, and finally in Section 5.5 we suggest some solutions that may help to fix the vulnerability.

### 5.1 Flickr's API Request Signing Process

Flickr requires that all API calls using an authentication token must be signed. In addition, calls to the flickr.auth.\* methods and the URLs that bring users to the application authorization page must also be signed. The process of signing is as follows.

- \* Sort your argument list into alphabetical order based on the parameter name.
- \* e.g. foo=1, bar=2, baz=3 sorts to bar=2, baz=3, foo=1
- \* concatenate the shared secret and argument name-value pairs
- \* e.g. SECRETbar2baz3foo1
- \* calculate the md5() hash of this string
- \* append this value to the argument list with the name api\_sig, in hexadecimal string form, e.g. api\_sig=1f3870be274f6c49b3e31a0c6728957f

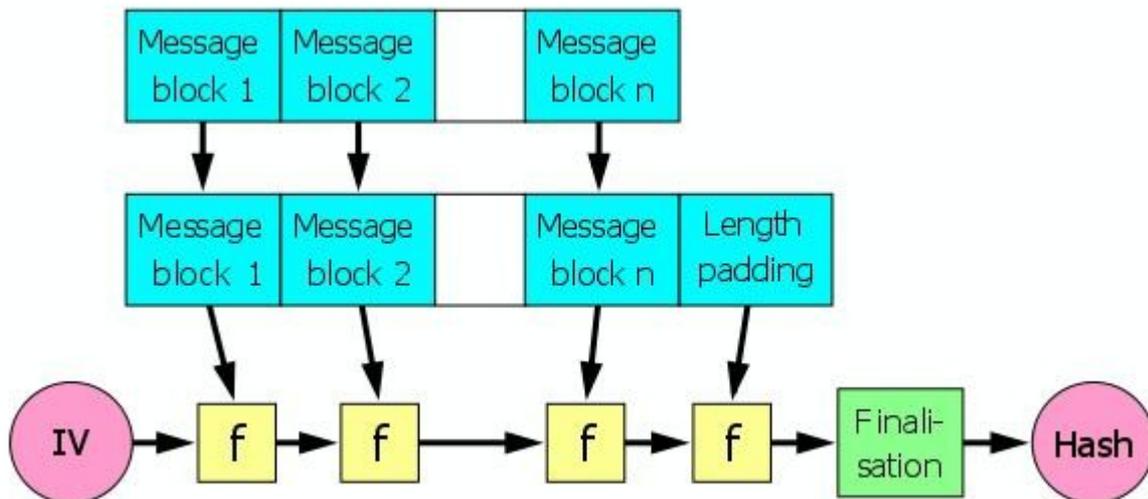
There are two vulnerabilities in this signing process:

- As there are **no delimiters** between the keys and values, the signature for "foo=bar" is identical to the signature for "foob=ar"; moreover, the signature for "foo=bar&fooble=baz" is the same as the signature for "foo=barfooblebaz". See [2] for a similar vulnerability of Amazon Web Service.
- As MD5 is vulnerable to **length-extension attack** (see Section 5.2), one can append arbitrary data to the request yet still can generate valid signature without knowing the secret key. When combining with the first vulnerability, one can easily forge any request on behalf of any Flickr application.

## 5.2 Length-Extension Attack On MD5

In short, the length-extension attack on one-way hash construction is that you can, given  $h(m)$  and  $\text{len}(m)$ , you are able to compute  $h(m\|\text{pad}(m)\|m')$  for any  $m'$  ( $\|$  stands for concatenation), even if you don't know the entire message  $m$ . This attack works on all Merkle-Damgård hash (see [4, 5]) such as MD0-MD5 and SHA0-SHA2. This is also called "message extension" or "padding" attack (see [6]). The rest of this section describes how this attack works on Flickr's API's MD5 signature. What follows is technical. You may want to skip reading it until you need it.

MD5 ([3]) follows the Merkle/Damgård iterative structure, where the hash is computed by the repeated application of a compression function to successive blocks of the message. (See Figure 1.) For MD5, the compression function takes two inputs — a 128-bit chaining value and a 512-bit message block — and produces as output a new 128-bit chaining value, which is input into the next iteration of the compression function. The message to be hashed is first padded to a multiple of 512 bits, and then divided into a sequence of 512-bit message blocks. Then the compression function is repeatedly applied, starting with an initial chaining value and the first message block, and continuing with each new chaining value and successive message blocks. After the last message block has been processed, the final chaining value is output as the hash of the message.



**Figure 1. Merkle-Damgård hash construction (copied from Wikipedia)**

According to [7], because of the iterative design, it is possible, from only the hash of a message and its length, to compute the hash of longer messages that start with the initial message and include the padding required for the initial message to reach a multiple of 512 bits. (See Figure 2.) Applying this to Flickr's API request signature, it follows that from  $\text{MD5}(\text{SECRET}\|m)$ , one can compute  $\text{MD5}(\text{SECRET}\|m')$  for any  $m'$  that starts with  $m\|p$ , where  $p$  is the Merkle-Damgård padding on  $\text{SECRET}\|m$ . To compute  $p$ , one just needs to know the length of  $\text{SECRET}\|m$ , which is easy to calculate in Flickr's API's case. In other words, from the signature of  $m$ , one can forge the API signature of  $m\|p\|x$  for any  $x$ , without even knowing the shared secret key, and without breaking MD5 in any sense.

# One-Way Hash Function MAC Broken With Merkle-Damgaard Strengthening

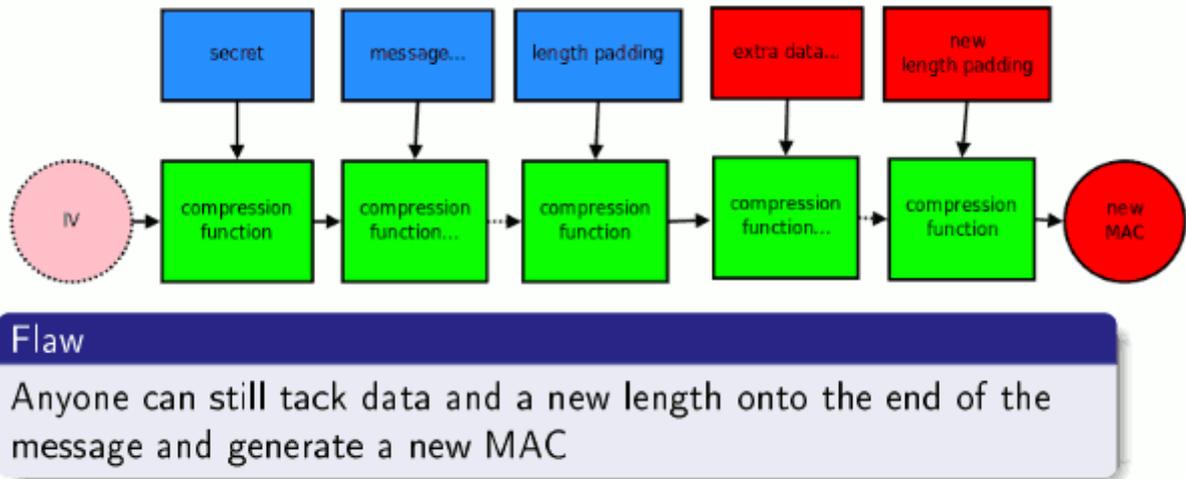


Figure 2. Length-extension attack on  $MAC = MD(KEY||msg)$  (copied from [9])

## 5.3 Our Attack

As described in Section 5.2 we require a message  $m$  and its signature to produce a longer message with a valid signature. It is easy to obtain a request signed by the target third party application provider.

Flickr and many other Web 2.0 sites allows users to share data with third party applications without divulging the user's credentials. Users are transported to the Flickr web site where they are asked whether he/she wants to allow the application to access their data. To achieve this the application providers ask the user to follow a link like this:

```
http://flickr.com/services/auth/?api_key=[api_key]&perms=[perms]&api_sig=[sig]
```

The *api\_key* 16 bytes value identifies the application asking for permissions and *api\_sig* is the signature of the request calculated using the secret shared between the application's developer and Flickr. This link is called login URL and is also a signed message which is all we need to perform the attack.

And we start working on our message forgery code, let's see what we have:

```
Message: SECRETapi_key[api_key]permsread (sorted and concatenated without '&=')  
Signature: [api_sig]  
Length: Length of Message + Length of SECRET
```

SECRET is the shared secret that Flickr and application owner don't want to share with us and that we don't need anyway.

Although the length of the padding is always between 9 and 64 bytes we need to know the exact size of the hashed data to reconstruct the sequence used as padding because this value is included in the ending 64 bits. Based on the API documentation we can assume that the secret is 16 bytes.

We can append anything to the request and calculate its signature but we must keep the same prefix including the padding, fortunately we have a simple way to avoid that prefix being a limitation. We can use the first char of the first variable as a variable name and all the rest of the original request including the hash padding as its value:

```
a=apikey[api_key]permsreadapi_sig[api_sig]
[padding]&api_key=[api_key]&perms=delete&new_key=new_value...
```

The annoying padding that includes non alphanumeric values and always contains null bytes becomes part of a variable that will be ignored. The only limitation is that we cannot use new variable names that after sorting fall before the 'a' variable but this is not a problem in practice because 'a' is the first letter of the alphabet and there isn't any numeric API parameter name being used.

## 5.4 Exploitations

There are many ways one can exploit this vulnerability for fun and profit. Below are what we have come up with. Others may have better ways to exploit this vulnerability. Please note that what we write here apply only for Flickr's API. This vulnerability may become much more-or-less dangerous in the context of other Flickr copycat API services. However, it would be up to those who have more time and/or interest than us to test these services.

First off, an attacker can send arbitrary yet valid requests on behalf of any third party application. This can be exploited to send requests using commercial API key which is AFAWK the same as non-commercial keys at the moment but this may change in the future. This may also make the application be blocked by Flickr if the attacker sends abusive requests that violating TOS of Flickr's API.

It is important to stress that this vulnerability alone doesn't give an attacker immediately access to accounts of Flickr users, but being able to send arbitrary requests on behalf of any application brings him much closer to that goal. In order to compromise the account of an user, the attacker needs to capture a 'frob' or an 'auth\_token' from that user. He can do that by attacking either the third party application or the user using techniques such as network sniffing or ARP/DNS spoofing or as simple as phishing. Google may help too, as always.

Additionally, if an application uses PHPFlickr >= 1.3.1, an attacker can trick users of that application to visit arbitrary web sites. The login url accepts an undocumented 'extra' parameter which is passed back by Flickr to the calling applications after the users finish authorizing the application. PHPFlickr >= 1.3.1 will automatically treat the 'extra' parameter as an URL, and redirect the users to it.

So if users click on a link like this:

```
http://www.flickr.com/services/auth/?
a=pi_key[api_key]permsdelete[padding]&api_key=[api_key]&perms=read&api_sig=[api
_sig]&extra=http://evil.com
```

where *api\_key* belongs to some third party application using PHPFlickr, they'll be immediately redirected to <http://evil.com> if they have already authorized that application. This may lead to phishing or browser exploitation attacks.



## 6. Conclusion

Length-extension attack on MAC implementation based on MD hash construction has been known in crypto academic community as early as 1992 (see [6]). After 17 years, however, we still have a large number of systems vulnerable to this simple attack. What is even more surprising is the fact that we were the first to identify this vulnerability in such popular system like Flickr.

Since August 2009 we have been carrying out a research in which we test-run a number of identified practical crypto attacks on random widely-used software systems. To our surprise, most, if not all, can be attacked by one or more of well-known crypto bugs. This case is just one example.

We hope that publishing this vulnerability and other future results from our research would encourage the security community in taking a more serious look at crypto bugs in software system which is as pervasive as SQL Injection or XSS in early 2000.

We hope you enjoy reading this advisory as much as we enjoy writing it.

## 7. References

- [1] Cal Henderson et al. Flickr Authentication API Spec. Retrieved September 6, 2009, from <http://www.flickr.com/services/api/auth.spec.html>.
- [2] Colin Percival. AWS signature version 1 is insecure. Retrieved September 6, 2009, from <http://www.daemonology.net/blog/2008-12.html>.
- [3] R. Rivest. RFC 1321: The MD5 Message-Digest Algorithm. RSA Data Security, Inc., April 1992.
- [4] I.B. Damgård. A design principle for hash functions. In G. Brassard, editor, *Advances in Cryptology: Proceedings of CRYPTO '89*, volume 435 of *Lecture Notes in Computer Science*, pages 416–427. Springer-Verlag, New York, 1990.
- [5] R. Merkle. One way hash functions and DES. In G. Brassard, editor, *Advances in Cryptology: Proceedings of CRYPTO '89*, volume 435 of *Lecture Notes in Computer Science*, pages 428–446. Springer-Verlag, New York, 1990.
- [6] G. Tsudik. Message authentication with one-way hash functions. *ACM Computer Communications Review*, 22(5):29–38, 1992.
- [7] B. Kaliski and M. Robshaw. Message Authentication with MD5. *RSA Labs' CryptoBytes*, Vol. 1 No. 1, Spring 1995.
- [8] M. Bellare, R. Canetti, and H. Krawczyk. RFC 2104 HMAC: Keyed-Hashing for Message Authentication, February 1997.
- [9] H. Travis. *Web 2.0 Cryptology, A Study in Failure*. OWASP, 2009. Retrieved September 13, 2009, from [http://www.subspacefield.org/security/web\\_20\\_crypto.pdf](http://www.subspacefield.org/security/web_20_crypto.pdf).
- [10] B. Preneel, P.C. van Oorschot. MDx-MAC and building fast MACs from hash functions. *Advances in Cryptology, Lecture Notes in Computer Science 963*, D. Coppersmith, Ed., Springer-Verlag, 1995, pp. 1-14.