

# **Guidance on crawler identification and authentication**

**A component of the ACAP Technical Framework**

*Version 1.0, 24 February 2008*

## Document history

<b>Version</b>	<b>Release date</b>
Version 0.9 (draft 1)	2008-01-23
Version 0.9 (draft 2)	2008-02-04
Version 0.9 (draft 3)	2008-02-11
Version 1.0	2008-02-24

# 1 Introduction

ACAP (Automated Content Access Protocol) is being developed as an open industry standard to enable the providers of all types of content to communicate permissions information (relating to access to and use of that content) in a form that can be readily recognized and interpreted by crawlers operated by a search engine or other kind of aggregation service. ACAP provides a technical framework that allows content owners to express access and use policies in a language that machines can read and understand.

ACAP is predominantly concerned with communication, not enforcement. ACAP is about enabling more content to be available for indexing by search engines and for other types of aggregation, not about locking content away where it cannot realise its potential value in the market. However, for content to be accessible by aggregators, it has to be made accessible to their automated agents of content acquisition, the web crawlers. And for crawlers to be given access to high-value content, they have to be known and they have to be trusted, as agents of a legitimate set of aggregation services that they represent.

Delivery of content from provider to aggregator begins when the crawler requests that the web server deliver a specified content resource, typically a web page. How does the web server determine the identity of this crawler and how to respond?

The main part of this guidance document is concerned with how to determine and authenticate the identity of crawlers. Appendix A describes how this approach can be applied to the management of crawler access to protected content.

## 2 Crawlers

A web crawler is a software system that automatically retrieves resources from the web. Crawlers are among the most powerful automated computer processes in current use outside military and intelligence applications, capable of accessing billions of web resources and feeding content through to the indexing and other processes.

The name "crawler" is one of many for this kind of computer system. Other names include "spider" and "robot", or simply "bot". In technical parlance they are generally referred to as "automated user agents", where a "user agent" is a software application that acts on behalf of a user (human or machine) to access resources on the Internet. User agents take many forms, but the most familiar to human users of the Internet are web browsers.

Crawlers retrieve resources from web servers in exactly the same way that web browser do: by sending a message to the web server containing a request for a specified resource. The message uses a standard Internet protocol, either the

Hypertext Transfer Protocol (HTTP) or its secure alternative (HTTPS), and the specified resource is identified by a Uniform Resource Identifier (URI)<sup>1</sup>.

## 2.1 Crawler names and addresses

The web works because of the universal adoption of web standards such as HTTP, which ensure that all user agents and web servers know how to communicate at a technical level<sup>2</sup>. One of the features of HTTP is that requests always take a predictable form, typically containing:

- the requested resource is identified by its URI;
- the requesting host device (to which the web server is to respond) is identified by its unique Internet (IP) address;
- the type of user agent (software application running on the host device) making the request is usually indicated.

Here is a typical HTTP request:

```
GET /index.htm HTTP/1.0
Host: 123.45.67.89
User-Agent: Mozilla/5.0 (comments about the user agent)
```

The user agent type ("Mozilla/5.0" in the above example) gives the web server information that can be used to determine how to respond in a way that the user agent software will understand. Note that the user agent type may be followed by free text comments – as will be seen below, such comments sometimes contain information that can be used in identifying a crawler making a request.

Most web servers create a log of the content of these requests. Here is a typical entry from such a log:

```
123.45.67.89 - - [21/01/2008:12:34:56 +0000] "GET /index.htm
HTTP/1.0" 200 5123 "-" "Mozilla/5.0 (comments about the user
agent) "
```

The URI of the requested resource, the user agent host IP address and the user agent type are highlighted in both the above examples.

---

<sup>1</sup> The term "URL" – Uniform Resource Locator – has now officially been superseded by "URI" in web standard specifications, but informally is still quite widely used by web users.

<sup>2</sup> In fact, HTTP is just the top layer of several standardised communication "layers", that ensure that messages can find their way through any arrangement of wireless, wired and fibre optic cable networks, between machines that use different hardware, different electrical standards, different operating systems and different application software.

Note that, while the user agent type can be anything that the requesting user agent chooses to insert into the request (and could therefore be deliberately misleading<sup>3</sup>), the IP address must be valid for the user agent's host device, since that is the address to which the web server responds.

Crawlers, as automated user agents, are mostly identifiable in HTTP requests, although HTTP doesn't require this. The name of the crawler is generally embedded in the comments that follow the user agent type, but some crawlers have specific user agent types that identify them. Most crawler names remain fixed, but some may change over time.

For example, the following web server log entries identify a selection of search engine crawlers:

```
65.214.45.34 - - [15/Jun/2007:00:57:53 +0000]
"GET /robots.txt HTTP/1.0" 404 8234 "-" "Mozilla/2.0
(compatible; Ask Jeeves/Teoma;
+http://about.ask.com/en/docs/about/webmasters.shtml)"

66.249.66.240 - - [15/Jun/2007:01:07:15 +0000]
"GET /robots.txt HTTP/1.1" 404 8240 "-" "Mozilla/5.0
(compatible; Googlebot/2.1; +http://www.google.com/bot.html)"

207.46.98.111 - - [15/Jun/2007:05:50:05 +0000]
"GET /robots.txt HTTP/1.0" 404 8234 "-" "msnbot/1.0
(+http://search.msn.com/msnbot.htm)"

64.34.145.197 - - [15/Jun/2007:09:35:54 +0000]
"GET /robots.txt HTTP/1.0" 404 8234 "-"
"SBIder/SBIder-0.8.2-dev (http://www.sitesell.com/sbider.html)"

64.1.215.164 - - [15/Jun/2007:03:48:51 +0000]
"GET /robots.txt HTTP/1.0" 404 8234 "-" "Mozilla/5.0
(Twiceler-0.9 http://www.cuill.com/twiceler/robot.html)"

38.113.234.180 - - [15/Jun/2007:06:00:20 +0000]
"GET /robots.txt HTTP/1.0" 404 8240 "-" "voyager/1.0"

81.52.143.16 - - [15/Jun/2007:08:41:23 +0000]
"GET /robots.txt HTTP/1.1" 404 8234 "-" "Mozilla/5.0
(Windows; U; Windows NT 5.1; fr; rv:1.8.1) VoilaBot BETA 1.2
(http://www.voila.com/)"
```

---

<sup>3</sup> It is a relatively simple task to forge the User-agent field using the standard web browser Mozilla Firefox, but most non-technical users wouldn't know how to do so.

```
74.6.75.48 - - [15/Jun/2007:04:55:35 +0000] "GET /robots.txt
HTTP/1.0" 404 8234 "-" "Mozilla/5.0 (compatible; Yahoo! Slurp;
http://help.yahoo.com/help/us/ysearch/slurp)"
```

While the crawler name cannot always be determined reliably, the IP address to which the web server is requested to respond is always easy to determine.

However, the IP address in an HTTP request may not identify the crawler. The crawler may be operating behind an intermediary, a "proxy" server that relays the request on the crawler's behalf. Such a technique may be used entirely legitimately by crawlers for load-balancing and performance optimization. On the other hand, the same technique can be used by malicious crawlers to disguise their identity, using proxy servers (unwittingly or otherwise) to relay HTTP requests.

In order to identify whether an IP address genuinely identifies a crawler and, if so, which crawler, it is necessary to compare the information within the request – the IP address and any identification information provided by the User-agent field – with information that can be obtained about the identity of the Internet host machine with which the IP address is associated.

## 2.2 DNS lookup

The standard technique for determining the network domain in which an IP address is located is to consult the distributed online directory of Internet domain and host names known as the Domain Name Service (DNS) using a technique known as "double reverse DNS lookup". Using this technique, the name of the host that corresponds to a specific IP address can be obtained and verified.

For example, the host IP address 64.1.215.164 in one of the above web server log entries can be looked up, and the host name that is reported<sup>4</sup> is `crawl-8.cuill.com`. This is clearly consistent with the content of the User-agent field as indicated in the same web server log entry, where the domain `cuill.com` is mentioned.

Having found the host name, this should also be looked up, to ensure that the two directory entries – one for the IP address and one for the host name – are consistent. If they are not consistent, this would indicate that at least one of the DNS entries is unreliable.

If the lookup in either direction fails, this indicates that the DNS entries for the host in question are not properly set up. While this doesn't necessarily indicate that there is a deliberate attempt to hide the host's identity, it certainly suggests that the request is from an indeterminate or unreliable source.

---

<sup>4</sup> Correct at the time of writing. DNS directory entries may change over time, although in the case of legitimate web crawlers this happens relatively infrequently.

Although this technique provides a reasonably reliable way of determining the real identity of the machine that sent the HTTP request to the server, it is too slow a technique to be carried out in real time between receiving a request and responding to it. It will generally take several seconds to perform the entire lookup process, in which time the crawler will probably have given up.

However this technique can be employed as an "off-line" process, to build and maintain a simple, local list of IP addresses associated with known crawlers, which can be stored on the web server and consulted quickly to enable real-time response.

## 2.3 Limits of reliability in crawler identification

Any request that purports to come from a named crawler is most likely to be genuine, since the technique described above can be used to check that the IP address and crawler name given in the request are consistent with one another. The limits of reliability in crawler authentication occur when a crawler cannot be distinguished from other web users because it does not identify itself as a crawler.

There is no completely reliable process of determining that an HTTP request came from a crawler, if the crawler is actively disguising its identity. A crawler may have legitimate reasons for disguising its identity – for example, it may be aggregating publicly available business information from corporate websites and wishes to avoid being recognised as a crawler and thereby being fed different information from human web users. Major search engines use similar techniques to check web servers that are suspected of "cloaking" the real nature of the content that they serve to the general web user.

Malicious crawlers are most likely to disguise their requests to look as if they came from a standard web browser operated by a human user. Malicious crawlers may be viruses that are running on infected host machines, so nothing can be gained from knowing the IP address, other than the host machine name, which is most likely a proxy server or NAT router<sup>5</sup>. Detection of crawlers in such cases may be very difficult, and would involve techniques such as web server traffic analysis to identify specific patterns in requests. Positive identification of a specific crawler would probably be impossible.

---

<sup>5</sup> Network Address Translation (NAT) is a widely used technique that enables host machines on a private network (e.g. on a domestic or small business wireless network) to use the public Internet by using a router to act as a gateway between the private and public networks. All HTTP requests from web browsers within the private network appear to a web server to have been sent by the router.

## 3 Web server actions and responses

When a web server receives an HTTP request, its response will depend upon a number of factors, but principally the answers to the following questions:

- does the requested resource exist on this site?
- is the requesting user entitled to request this resource?
- must the user authenticate themselves before receiving (a copy of) the resource?

Crawlers aren't normally set up to respond to authentication requests, so the first two factors are the ones that concern us here.

A web server must first check that the crawler is entitled to request the resource. To check this, the server must, if possible, verify the identity of the crawler against a local check-list of known crawlers, then respond accordingly.

If the identity of the crawler is not known, the web server will on this occasion have to respond in some default way defined for unknown crawlers (which may be to refuse access to the resource, but this doesn't have to be the response). When at some later time the web server logs are analysed to determine which crawlers are visiting the site, the IP addresses of unknown crawlers can be looked up using DNS and, if they prove to be genuine, can be added to the local check-list.

Most web servers can be configured to respond to requests in a variety of ways, depending upon the source of the request, the resource requested and other factors. To vary the response in this way will have some impact upon server response times, but the impact can be minimised by implementing in an appropriate way. It should always be remembered that crawlers don't have time to wait for a response while web servers perform complex checks on their identity, so web servers need to be configured to check a crawler's identity without significantly delaying response.

### 3.1 How and when to check crawler identity

As already stated, a web server could make a thorough check on the identity of a crawler whenever it visits the site, but this would significantly delay response and risk causing the crawler to abandon crawling that site.

The easiest crawlers to identify are those that make use of the 'robots.txt' file to determine what resources they may crawl. Regular checks, by DNS lookup, should be made on the identity of all hosts that have requested this file. This should not be done at the time of the request but as a separate process, based upon an analysis of web service access log entries.

Any request for access to protected content will involve authentication. If certain crawlers are to be allowed privileged access protected content, the IP address of the

requesting host should be checked against the local list of IP addresses of privileged crawlers before responding to the request. If the IP address is in the local list, the crawler can be allowed access to the requested resource without further delay. If the IP address is identified to be the address of a non-privileged crawler, access would normally be denied, since the crawler will not be able to respond to a request for username and password authentication. If the IP address does not appear to be that of a crawler, the normal response to regular web users can be made (e.g. requesting username and password authentication).

If different 'robots.txt' files are to be served to different crawlers, according to their identity, a similar procedure could be followed to identify the crawler requesting 'robots.txt' and to determine which file to serve in response. The host IP address would be looked up in the local list to determine which crawler is making the request, and the response determined according to the result of the lookup.

The aim should be to make it as simple as possible for a web server to determine how to respond to each request that it receives, so that responses are not delayed any more than is necessary. The lookup table should therefore be arranged as far as possible according to the categories of response that are possible. In the case of requests for 'robots.txt' the categories of response could be:

- serve the requested resource as is (likely to be the default response)
- serve a different resource according to category (response to privileged crawlers)
- deny access according to category (response to blacklisted crawlers).

The best architecture for storing a local lookup table will depend upon the overall architecture of the web server. See below for an example of how this might work in an installation of the Apache web server.

## 4 Example approach to implementation of a procedure for crawler authentication

The following implementation of a procedure for crawler authentication assumes the use of the Apache web server, an architecture that is similar to what has been tested in a pilot implementation by the Belgian newspaper group *De Persgroep*. We are grateful to them for their assistance in providing an example for this guidance document.

**WARNING! This section goes into significant technical detail of the example implementation and is not suitable for the non-technical reader.**

This implementation specifically involves use of the Apache module 'mod\_rewrite'. As those who have configured an Apache server will realise, Apache as a whole and this module in particular are highly configurable, so a number of other implementation approaches could probably be adopted with equal success.

However, some approaches are definitely better than others, and this implementation includes a number of approaches that were tried and found not to be successful.

#### 4.1 *Bad approach: authentication of remote IP address using regular expressions*

The remote IP address is available in the rewrite engine as a server variable: `$_{REMOTE_ADDR}`. This makes it possible to write checks against it. There are three options using regular expressions:

**First option:** the use of a regular expression matching individual IP addresses:

```
RewriteCond $_{REMOTE_ADDR} (12\.34\.56\.78|12\.34\.56\.88|12\.35\.67\.89)
```

Clearly this is impractical for more than a very small number of IP addresses.

**Second option:** the use of the `[OR]` directive, creating a single line per IP address:

```
RewriteCond $_{REMOTE_ADDR} 12\.34\.56\.78 [OR]
RewriteCond $_{REMOTE_ADDR} 12\.34\.56\.88 [OR]
RewriteCond $_{REMOTE_ADDR} 12\.35\.67\.89
```

This is more readable than the first option, but is still only practical for a very small number of IP addresses without creating performance and maintenance issues.

**Third option:** the use of subnet masks to match multiple IP addresses with a single pattern:

```
RewriteCond $_{REMOTE_ADDR} 12\.34\.56\..* [OR]
RewriteCond $_{REMOTE_ADDR} 12\.35\..*
```

This will match any IP address in subnets matching a range of IP addresses on each line, but crawlers may use isolated IP addresses, in which case this option could match IP addresses that aren't associated with crawlers.

**Conclusion:** All of the above have to be avoided because of the following reasons:

1. All these options are costly in performance terms because of the large number of regular expressions that would need to be involved. Executing a few hundred checks for each request on the Apache web server would not be practical.
2. Whenever you want to add or remove an IP address the web server would have to be re-started in order to re-read the Apache configuration file 'httpd.conf'.
3. The configuration file 'httpd.conf' would probably have to be edited by hand.

The last two objections could be eliminated by placing the regular expressions in an external file, but in that case a program would need to be executed using a `RewriteMod` rule, and the performance issues would still remain.

## 4.2 *Good approach*: authentication of remote IP addresses by testing whether a file exists

A `RewriteCond` rule can be used to test whether a file exists or not. If a set of files are created whose names correspond to known crawler IP addresses, it becomes a simple matter to check whether an IP address is that of a known crawler:

```
RewriteCond /crawlers/ip/{REMOTE_ADDR} -f
```

Creating these files (which can be empty and so take up very little storage space) is very easy and can be automated. They can be created and removed by a separate process without stopping and starting the web server.

Checking for the existence of a file is relatively slow compared to checking a single regular expression, because it involves a disk-read the first time the file is read by the server. However, it is faster than having to check possibly hundreds of regular expressions, and many operating systems would optimise performance by putting regularly-accessed files into memory cache, thereby avoiding the disk-read on second and subsequent checks on the same file.

By placing files in different directories a number of rules could be created that determine how to respond to the same crawler in different circumstances. For example, if different 'robots.txt' files are to be served to different crawlers according to crawler name, this could be done by creating files for all IP addresses associated with a given crawler in a single directory corresponding to the crawler name, e.g.:

```
RewriteCond /crawlers/searchbot/{REMOTE_ADDR} -f  
RewriteRule ^robots\.txt$ /crawlers/searchbot/robots.txt
```

The same approach would work for other categorisations.

## 4.3 Analysing an Apache web server access log file

The following short program, written in "pseudo-code", reads an access log ("log-file"), finds all entries in which the file /robots.txt is requested, checks that each IP address is genuine by performing a double reverse DNS lookup, and if it is, creates a file whose name is the IP address in a directory 'all-ok-bots' of crawler IP addresses. If the domain of the remote host is 'search.com', the program additionally creates an identical file in a directory 'searchbot' containing all the IP addresses associated with the crawler of that name, or in a directory 'other-bots' in all other cases. IP addresses that do not appear to be genuine results in the creation of files in a directory 'suspect-bots'.

```
for-each (ip-address in log-file) {
  if (requested-resource == '/robots.txt') {
    let reverse-result = reverseDNSlookup(ip-address);
    let regular-result = regularDNSlookup(reverse-result);
    if (regular-result == ip-address) {
      // The reverse result is reliable
      create-file('/all-ok-bots/', ip-address)
      if (reverse-result contains-string '.search.com')
        create-file('/searchbot/', ip-address)
      else
        create-file('/other-bots/', ip-address)
    }
  } else
    // The reverse result is unreliable
    create-file('/suspect-bots/', ip-address)
}
```

This program has been deliberately kept short for clarity. A well-engineered program would be more complicated, because various error conditions would need to be caught and handled gracefully. There would clearly need to be an iterative process for handling more categories of crawlers, including different crawlers within the same remote host domain, and the information about what to do with crawlers in each category would probably need to be maintained in a separate configuration file. It is reasonable to assume that functions or methods equivalent to 'reverseDNSlookup' and 'regularDNSlookup' will be available in the function or object library of whatever programming language is actually used to develop a working program.

## 4.4 Example applications

### 4.4.1 Allowing a specific crawler access to protected resources

Web servers such as Apache can be configured to rewrite the path to the requested resource before either fetching and serving the resource or passing the rewritten request to an application server running behind the web server.

In the simplest case the request is simply rewritten by changing the path and/or filename of the requested resource so that it can be served without need for further authentication, e.g.

```
RewriteCond /crawlers/searchbot/%{REMOTE_ADDR} -f
RewriteRule ^/protected/.*/authenticated_access/$1
```

The requested resource is served without redirecting the client. If this method were used to serve a different resource to the one served to regular web users, this would be considered to be "cloaking". However, if the method is used to serve the same

resource, but without the need for username and password authentication, it is a legitimate technique.

There are several ways of passing information to an application using rewrite rules. Which works best will depend upon the server architecture. If the requested resource is executable (e.g. PHP) and environment variables can be passed to the application, a rewrite rule can be set up to pass an environment variable with the request, e.g.

```
RewriteCond /crawlers/searchbot/%{REMOTE_ADDR} -f  
RewriteRule ^/.* - [E=crawler:searchbot]
```

In this case the environment variable 'crawler' is given the value 'searchbot'.

If there is an Application Server running behind the web server, an alternative would be to rewrite the path to identify the crawler, e.g.

```
RewriteCond /crawlers/searchbot/%{REMOTE_ADDR} -f  
RewriteRule ^/.* /searchbot/$1
```

In this case the path to the resource has '/searchbot' prepended to it, which can be picked up by the application server in order to determine which resource to serve in response to the rewritten request.

## 5 References

1. Apache HTTP Server Version 2.2, Module mod\_rewrite – see [http://httpd.apache.org/docs/2.2/mod/mod\\_rewrite.html](http://httpd.apache.org/docs/2.2/mod/mod_rewrite.html).