INTERNET-DRAFT
Expires: December 1999
Category: Standards Track
draft-ietf-wrec-wpad-01.txt

Paul Gauthier
Inktomi Corporation
Josh Cohen
Microsoft Corporation
Martin Dunsmuir
RealNetworks, Inc.
Charles Perkins
Sun Microsystems, Inc.

Web Proxy Auto-Discovery Protocol

Status of This Memo

This document is a submission by the WREC Working Group of the Internet Engineering Task Force (IETF). Comments should be submitted to the wrec@cs.utk.edu mailing list.

Distribution of this memo is unlimited.

This document is an Internet-Draft and is in full conformance with all provisions of Section 10 of RFC2026. Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at:
http://www.ietf.org/ietf/1id-abstracts.txt
The list of Internet-Draft Shadow Directories can be accessed at:
http://www.ietf.org/shadow.html.

Abstract

A mechanism is needed to permit web clients to locate nearby web proxy caches. Current best practice is for end users to hand configure their web client (i.e., browser) with the URL of an "auto configuration file". In large environments this presents a formidable support problem. It would be much more manageable for the web client software to automatically learn the configuration information for its web proxy settings. This is typically referred to as a resource discovery problem.

Web client implementers are faced with a dizzying array of resource discovery protocols at varying levels of implementation and deployment. This complexity is hampering deployment of a "web proxy auto-discovery "facility. This document proposes a pragmatic

approach to web proxy auto-discovery. It draws on a number of proposed standards in the light of practical deployment concerns. It proposes an escalating strategy of resource discovery attempts in order to find a nearby web proxy server. It attempts to provide rich

mechanisms for supporting a complex environment, which may contain multiple web proxy servers.

Table of Contents

1. Conventions used in this document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in "Key words for use in RFCs to Indicate Requirement Levels" [KEYWORDS].

2. Introduction

The problem of locating nearby web proxy cache servers can not wait for the implementation and large scale deployment of various

upcoming resource discovery protocols. The widespread success of the HTTP protocol and the recent popularity of streaming media has placed unanticipated strains on the networks of corporations, ISPs and backbone providers. There currently is no effective method for these organizations to realize the obvious benefits of web caching without tedious and error prone configuration by each and every end user.

The de-facto mechanism for specifying a web proxy server configuration in web clients is the download of a script or configuration file named by a URL. Users are currently expected to hand configure this URL into their Browser or other web client. This mechanism suffers from a number of drawbacks:

- Difficulty in supporting a large body of end-users. Many users misconfigure their proxy settings and are unable to diagnose the cause of their problems.

- Lack of support for mobile clients who require a different proxy as their point of access changes.

- Lack of support for complex proxy environments where there may exist a number of proxy servers with different affinities for different clients (based on network proximity, for example). Currently, clients would have to "know" which proxy server was optimal for their use.

Currently available methods for resource discovery need to be exploited in the context of a well defined framework. Simple, functional and efficient mechanisms stand a good chance of solving this pressing and basic need. As new resource discovery mechanisms mature they can be folded into this framework with little difficulty.

This document is a specification for implementers of web client software. It defines a protocol for automatically configuring those clients to use a local proxy. It also defines how an administrator should configure various resource discovery services in their network to support WPAD compatible web clients.

While it does contain suggestions for web proxy server implementers, it does not make any specific demands of those parties.

3. Defining Web Proxy Auto-Discovery

As mentioned above, currently web client software needs to be configured with the URL of a proxy auto-configuration file or script. The contents of this script are vendor specific and not currently standardized. This document does not attempt to discuss the contents of these files (see[8] for an example file format).

Thus, the Web Proxy Auto-Discovery (WPAD) problem reduces to providing the web client a mechanism for discovering the URL of the

Configuration File. Once this Configuration URL (CURL) is known, the client software already contains mechanisms for retrieving and interpreting the Configuration File (CFILE) to enable access to the specified proxy cache servers.

It is worth carefully noting that the goal of the WPAD process is to discover the correct CURL at which to retrieve the CFILE. The client is *not* trying to directly discover the name of the proxy server. That would circumvent the additional capabilities provided by proxy Configuration Files (such as load balancing, request routing to an array of servers, automated fail-over to backup proxy server [6,8]).

It is worth noting that different clients requesting the CURL may receive completely different CFILEs in response. The web server may send back different CFILES based on a number of criteria such as the "User-Agent" header, "Accept" headers, client IP address/subnet, etc. The same client could conceivably receive a different CFILE on successive retrievals (as a method of round-robin load balancing, for example).

This document will discuss a range of mechanisms for discovering the Configuration URL. The client will attempt them in a predefined order, until one succeeds. Existing widely deployed facilities may not provide enough expressiveness to specify a complete URL. As such, we will define default values for portions of the CURL which may not be expressible by some discovery mechanisms:

http://<HOST>:<PORT><PATH>

<HOST> - There is no default for this potion. Any succeeding discovery mechanism will provide a value for the <HOST> portion of the CURL. The client MUST NOT provide a default.

<PORT> - The client MUST assume port 80 if the successful discovery mechanism does not provide a port component.

<PATH> - The client MUST assume a path of "/wpad.dat" if the successful discovery mechanism does not provide a path component.

4. The Discovery Process

4.1. WPAD Overview

This sub-section will present a descriptive overview of the WPAD protocol. It is intended to introduce the concepts and flow of the protocol. The remaining sub-sections (3.2-3.7) will provide the rigorous specification of the protocol details. WPAD uses a collection of pre-existing Internet resource discovery mechanisms to

perform web proxy auto-discovery. Readers may wish to refer to [1] for a similar approach to resource discovery, since it was a basis for this strategy. The WPAD protocol specifies the following:

- how to use each mechanism for the specific purpose of web proxy auto-discovery
- the order in which the mechanisms should be performed
- the minimal set of mechanisms which must be attempted by a WPAD compliant web client

The resource discovery mechanisms utilized by WPAD are as follows.
- Dynamic Host Configuration Protocol (DHCP, [3,7]).
- Service Location Protocol (SLP, [4]).
- "Well Known Aliases using DNS A records [5,9].
- DNS SRV records [2,9].
- "service: URLs" in DNS TXT records [10].

Of all these mechanisms only the DHCP and Well Known Aliases are required in WPAD clients. This decision is based on three reasons: these facilities are currently widely deployed in existing vendor hardware and software; they represent functionality that should cover most real world environments; they are relatively simple to implement.

DNS servers supporting A records are clearly the most widely deployed of the services outlined above. It is reasonable to expect API support inside most web client development environments (POSIX C, Java, etc). The hierarchical nature of DNS makes it possible to support hierarchies of proxy servers.

DNS is not suitable in every environment, unfortunately. Administrators often choose a DNS domain name hierarchy that does not correlate to network topologies, but rather with some organizational model (for example, foo.development.bar.com and foo.marketing.bar.com). DHCP servers, on the other hand, are frequently deployed with concern for network topologies. DHCP servers provide support for making configuration decisions based on subnets, which are directly related to network topology.

Full client support for DHCP is not as ubiquitous as for DNS. That is, not all clients are equipped to take advantage of DHCP for their essential network configuration (assignment of IP address, network mask, etc). APIs for DHCP are not as widely available. Luckily, using DHCP for WPAD does not require either of these facilities. It is relatively easy for web client developers to speak just the minimal DHCP protocol to perform resource discovery. It entails building a simple UDP packet, sending it to the subnet broadcast address, and parsing the reply UDP packet(s) which are received to extract the WPAD option field. A reference implementation of this code in C is available [11].

The WPAD client attempts a series of resource discovery requests,

using the discovery mechanisms mentioned above, in a specific order. Clients only attempt mechanisms that they support (obviously). Each time the discovery attempt succeeds; the client uses the information obtained to construct a CURL. If a CFILE is successfully retrieved

at that CURL, the process completes. If not, the client resumes where it left of in the predefined series of resource discovery requests. If no untried mechanisms remain and a CFILE has not been successfully retrieved, the WPAD protocol fails and the client is configured to use no proxy server.

First the client tries DHCP, followed by SLP. If no CFILE has been retrieved the client moves on to the DNS based mechanisms. The client will cycle through the DNS SRV, Well Known Aliases and DNS TXT record methods multiple times. Each time through the QNAME being used in the DNS query is made less and less specific. In this manner the client can locate the most specific configuration information possible, but can fall back on less specific information. Every DNS lookup has the QNAME prefixed with wpad to indicate the resource type being requested.

As an example, consider a client with hostname johns-desktop.development.foo.com. Assume the web client software supports all of the mechanisms listed above. This is the sequence of discovery attempts the client would perform until one succeeded in locating a valid CFILE:

- DHCP
- SLP
- DNS A lookup on QNAME=wpad.development.foo.com.
- DNS SRV lookup on QNAME=wpad.development.foo.com.
- DNS TXT lookup on QNAME=wpad.development.foo.com.
- DBS A lookup on QNAME=wpad.foo.com.
- DNS SRV lookup on QNAME=wpad.foo.com.
- DNS TXT lookup on QNAME=wpad.foo.com.

4.2. When to Execute WPAD

Web clients need to perform the WPAD protocol periodically to maintain correct proxy settings. This should occur on a regular basis corresponding to initialization of the client software or the networking stack below the client. As well, WPAD will need to occur in response to expiration of existing configuration data. The following sections describe the details of these scenarios. 3.2.1. Periodic Discovery

The web proxy auto-discovery process MUST occur at least as frequently as one of the following two options. A web client can use either option depending on which makes sense in their environment. Clients MUST use at least one of the following options. They MAY also choose to implement both options.
- Upon startup of the web client.
- Whenever there indication from the networking stack that the IP address of the client host either has, or could have, changed.

In addition, the client MUST attempt a discovery cycle upon expiration of a previously downloaded CFILE in accordance with HTTP/1.1.

4.2.1. Upon Startup of the Web Client

For many types of web client (like web browsers) there can be many instances of the client operating for a given user at one time. This is often to allow display of multiple web pages in different windows, for example. There is no need to re-perform WPAD every time a new instance of the web client is opened. WPAD MUST be performed when the number of web client instances transitions from 0 to 1. It SHOULD NOT be performed as additional instances are created.

4.2.2. Network Stack Events

Another option for clients is to tie the execution of WPAD to changes in the networking environment. If the client can learn about the change of the local host s IP address, or the possible change of the IP address, it MUST re-perform the WPAD protocol. Many operating systems provide indications of network up events, for example. Those types of events and system-boot events might be the triggers for WPAD in many environments.

4.2.3. Expiration of the CFILE

The HTTP retrieval of the CURL may return HTTP headers specifying a valid lifetime for the CFILE returned. The client MUST obey these timeouts and rerun the PAD process when it expires. A client MAY rerun the WPAD process if it detects a failure of the currently configured proxy (which is not otherwise recoverable via the inherent mechanisms provided by the currently active Configuration File).

Whenever the client decides to invalidate the current CURL or CFILE, it MUST rerun the entire WPAD protocol to ensure it discovers the currently correct CURL. Specifically, if the valid lifetime of the CFILE ends(as specified by the HTTP headers provided when it was retrieved),the complete WPAD protocol MUST be rerun. The client MUST NOT simply re-use the existing CURL to obtain a fresh copy of the CFILE.

A number of network round trips, broadcast and/or multicast communications may be required during the WPAD protocol. The WPAD protocol SHOULD NOT be invoked at a more frequent rate than specified above (such as per-URL retrieval).

4.3. WPAD Protocol Specification

The following pseudo-code defines the WPAD protocol. If a particular discovery mechanism is not supported, treat it as a failed discovery attempt in the pseudo-code.

In addition, this logic is expressed below in pseudo-code.
The following pseudo-code fragment defines WPAD. Unsupported
discovery mechanisms are treated as failure in the pseudo-code.

Two subroutines need explanation. The subroutine strip_leading_component(dns_string) strips off the leading characters, up to and including the first dot (`.') in the string which is passed as a parameter, and is expected to contain DNS name. The Boolean subroutine is_not_canonical(dns_string) returns FALSE if dns_string is one of the canonical domain suffixes defined in RFC 1591 [13] (for example, "com").

The slp_list and dns_list elements below are assumed to be linked lists containing a data field and a pointer to the next element. The data field contains the elements used to override the default values in creating a CURL, as detailed in section 3.5.

```
load_CFILE() {
  /* MUST use DHCP */
  curl = dhcp_query(/*WPAD option  (section 4.4.1) */);
  if (curl != null) {  /* DHCP succeeded */
     if isvalid (read_CFILE(curl))
        return SUCCESS;  /* valid CFILE */
  }

  /* Should use SLP */
  slp_list = slp_query(/*(WPAD attributes  (Section 4.4.2)*/);
  while (slp_list != null) {  /* test each curl  */
     if isvalid(read_CFILE(slp_list.curl_data))
        return SUCCESS;   /* valid CFILE */
     else
        slp_list = slp_list.next;
  }

  /* all the DNS mechanisms */
  TGTDOM = gethostbyname(me);
  TGTDOM = strip_leading_component(TGTDOM);

  while (is_not_canonical(TGTDOM)) {

     /* SHOULD try DNS SRV records */
     dns_list = dns_query(/*QNAME=wpad.TGTDOM.,
                                 QTYPE=SRV (section 4.4.4)*/);
     while (dns_list != null) { /* each TXT record */
        if isvalid(read_CFILE(dns_list, curl_data))
           return SUCCESS;   /* valid CFILE */
        else
           dns_list = dns_list.next;
     }

     /* SHOULD try DNS TXT records */
     dns_list = dns_query(/*QNAME=wpad.TGTDOM.,
```

```
                                  QTYPE=TXT (section 4.4.5)*/);
        while (dns_list != null) { /* each TXT record */
           if isvalid(read_CFILE(dns_list, curl_data))
              return SUCCESS;   /* valid CFILE */
```

```
            else
               dns_list = dns_list.next;
         }

         /* MUST try DNS A records */
         dns_list = dns_query(/*QNAME=wpad.TGTDOM.,
                                 QTYPE=A  (Section 4.4.3)*/);

         while (dns_list != null) { /* check each A record */
            if isvalid(read_CFILE(dns_list, curl_data))
               return SUCCESS;   /* valid CFILE */
            else
               dns_list = dns_list.next;
         }


         /* Still no match, remove leading component and iterate */
         TGTDOM = strip_leading_component(TGTDOM);

      } /* no A, TXT or SRV records for wpad.* */

      return FAILED;  /* could not locate valid CFILE */
   }
```

4.4. Discovery Mechanisms

Each of the resource discovery methods will be marked as to whether the client MUST, SHOULD, MAY, or MUST NOT implement them to be compliant. Client implementers are encouraged to implement as many mechanisms as possible, to promote maximum interoperability.

Discovery Mechanism	Status	Document Section
DHCP	MUST	4.4.1
SLP	SHOULD	4.4.2
"Well Known Alias"	MUST	4.4.3
DNS SRV Records	SHOULD	4.4.4
DNS TXT "service: URLs"	SHOULD	4.4.5

SUMMARY OF DISCOVERY MECHANISMS

4.4.1. DHCP

Client implementations MUST support DHCP. DHCP has widespread support innumerous vendor hardware and software implementations, and is widely deployed. It is also perfectly suited to this task, and is used to discover other network resources (such a time servers,

printers, etc). The DHCP protocol is detailed in RFC 2131 [3].
We propose a new DHCP option with code 252 for use in web proxy
auto-discovery. See RFC 2132 [7] for a list of existing DHCP

options. See "Conditional Compliance" for more information on DHCP requirements.

The client should obtain the value of the DHCP option code 252 as returned by the DHCP server. If the client has already conducted DHCP protocol during its initialization, the DHCP server may already have supplied that value. If the value is not available through a client OS API, the client SHOULD use a DHCPINFORM message to query the DHCP server to obtain the value.

The DHCP option code for WPAD is 252 by agreement of the DHC working group chair. This option is of type STRING. This string contains a URL which points to an appropriate config file. The STRING is of arbitrary size.
An example STRING value would be:
"http://server.domain/proxyconfig.pac"

4.4.2. Service Location Protocol /SLP

The Service Location Protocol [RFC2608] is a Proposed Standard for discovering services in the Internet. SLP has several reference implementations available; for details, check the following web page:

http://www.svrloc.org/

A service type for use with WPAD has been defined and is available as an Internet Draft.

Client implementations SHOULD implement SLP. SLP Service Replies will provide one or more complete CURLs. Each candidate CURL so created should be pursued as specified in section 4.5 and beyond.

4.4.3. DNS A/CNAME "Well Known Aliases

Client implementations MUST support this mechanism. This should be straightforward since only basic DNS lookup of A records is required. See RFC 2219 [5] for a description of using "well known" DNS aliases for resource discovery. We propose the "well known alias of "wpad" for web proxy auto-discovery.

The client performs the following DNS lookup:
QNAME=wpad.TGTDOM., QCLASS=IN, QTYPE=A

Each A RR, which is returned, contains an IP address which is used to replace the <HOST> default in the CURL.

Each candidate CURL so created should be pursued as specified in section 4.5 and beyond.

4.4.4. DNS SRV Records

Client implementations SHOULD support the DNS SRV mechanism. Details of the protocol can be found in RFC 2052 [2]. If the implementation language/environment provides the ability to perform DNS lookups on QTYPEs other than A, client implementers are strongly encouraged to provide this support. It is acknowledged that not all resolver APIs provide this functionality.

The client issues the following DNS lookup:
QNAME=wpad.tcp.TGTDOM., QCLASS=IN, QTYPE=SRV

If it receives SRV RRs in response, the client should use each valid RR in the order specified in RFC 2052 [2]. Each valid record will specify both a <HOST> and a <PORT> to override the CURL defaults.

Each candidate CURL so created should be pursued as specified in section 4.5 and beyond.

4.4.5. DNS TXT service: Entries

Client implementation SHOULD support this mechanism. If the implementation language/environment provides the ability to perform DNS lookups on QTYPEs other than A, the vendor is strongly encouraged to provide this support. It is acknowledged that not all resolver APIs provide this functionality.
The client should attempt to retrieve TXT RRs from the DNS to obtain service: URLs contained therein. The service: URL will be of the following format, specifying a complete candidate CURL for each record located:

service: wpad:http://<HOST>:<PORT><PATH>

The client should first issue the following DNS query:
QNAME=wpad.TGTDOM., QCLASS=IN, QTYPE=TXT

It should process each TXT RR it receives (if any) using each service:URL found (if any) to generate a candidate CURL. These CURLs should be pursued as described in section 3.5 and beyond.
Readers familiar with [1] should note that WPAD clients MUST NOT perform the QNAME=TGTDOM., QCLASS=IN, QTYPE=TXT lookup which would be suggested by that document.

4.4.6. Fallback

Clients MUST NOT implement the "Fallback" mechanism described in [1]. It is unlikely that a client will find a web server prepared to handle the CURL request at a random suffix of its FQDN. This will only increase the number of DNS probes and introduce an excess of spurious "GET" requests on those hapless web servers.

Instead, the "Well Known Aliases method of section 3.4.4 provides

equivalent functionality.

4.4.7. Timeouts

Implementers are encouraged to limit the time elapsed in each discovery phase. When possible, limiting each phase to 10 seconds is considered reasonable. Implementers may choose a different value which is more appropriate to their network properties. For example, a device implementation, which operated over a wireless network, may use a much larger timeout to account for low bandwidth or high latency.

4.5. Composing a Candidate CURL

Any successful discovery mechanism response will provide a <HOST>(perhaps in the form of an IP address). Some mechanisms will also provide a <PORT> and/or a <PATH>. The client should override the default CURL fields with all of those supplied by the discovery mechanism.

4.6. Retrieving the CFILE at the CURL

The client then requests the CURL via HTTP.
When making the request it MUST transmit HTTP "Accept" headers indicating what CFILE formats it is capable of accepting. For example, Netscape Navigator browsers with versions 2.0 and beyond might include the following line in the HTTP Request:

Accept: application/x-ns-proxy-autoconfig

The client MUST follow HTTP redirect directives (response codes 3xx) returned by the server. The client SHOULD send a valid "User-Agent" header.

4.7. Resuming Discovery

If the HTTP request fails for any reason (fails to connect, server error response, etc) the client MUST resume the search for a successful CURL where it left off. It should continue attempting other sub-steps in a specific discovery mechanism, and then move on to the next mechanism or TGTDOM iteration, etc.

5. Client Implementation Considerations

The large number of discovery mechanisms specified in this document may raise concerns about network traffic and performance. The DHCP portion of the process will result in a single broadcast by the client, and perhaps a few replies by listening DHCP servers.

The remaining mechanisms are all DNS based. All DNS queries should have the QNAME terminated with a trailing '.' to indicate a FQDN and expedite the lookup. As such each TGTDOM iteration will cause 3 DNS

lookups, each a unicast UDP packet and a reply. Most clients will have fewer than 2TGTDOM iterations, limiting the total number of DNS request/replies to6.

All total, 7 UDP request/reply packets on client startup is quite a low overhead. The first web page downloaded by the client will likely dwarf that packet count. Each of the DNS lookups should stand a high chance of hitting the cache in the client's DNS server, since other clients will have likely looked them up recently, providing a low total elapsed time.

This is of course the worst case, where no CURLS are obtained, and assuming a long client FQDN. Often, a successful CURL will be found early in the protocol, reducing the total packet count.
Client implementations are encouraged to overlap this protocol work with other startup activities. Also, client implementers with concerns about performance can choose to implement only the discovery mechanisms listed as MUST in section 3.4.

A longer delay could occur if a CURL is obtained, but the hosting web server is down. The client could spend considerable time waiting for the TCP connect () call to fail. Luckily this is an extremely rare case where the web server hosting the CFILE has failed. See section 5, where proxy server implementers are encouraged to provide support for hosting CURLs on the proxy itself (acting as web server). Since proxy servers are often deployed with considerable attention to fault tolerance, this corner case can be further minimized.

6. Proxy Server Considerations

As mentioned in the previous section, it is suggested that proxy servers be capable of acting as a web server, so that they can host the CURL directly.

The implementers of proxy servers are most likely to understand the deployment situations of proxy caches, the formats of proxy configuration files, etc. They can also build in the ability select a CFILE based on all the various inputs at the time of the CURL request("User-Agent", "Accept", client IP address/subnet/hostname, topological distribution of nearby proxy servers, etc).

7. Administrator Considerations

Administrators should configure at least one of the DHCP or DNS A RR methods in their environment (since those are the only two all compatible clients MUST implement). Beyond that, configuring to support mechanisms earlier in the search order will improve client startup time.

One of the major motivations for this protocol structure was to support client location of "nearby" proxy servers. In many environments there may be a number of proxy servers (workgroup,

corporate gateway, ISP, backbone). There are a number of possible points at which "nearness" decisions can be made in this framework:

- DHCP servers for different subnets can return different answers. They can also base decisions on the client cipaddr field or the client identifier option.

- DNS servers can be configured to return different SRV/A/TXT RRs for Different domain suffixes (for example, QNAMEs wpad.marketing.bigcorp.com and wpad.development.bigcorp.com).

- The web server handling the CURL request can make decisions based on the "User-Agent", "Accept", client IP address/subnet/hostname, and the topological distribution of nearby proxy servers, etc. This can occur inside a CGI executable created to handle the CURL. As mentioned above it could be a proxy server itself handing the CURL request and making those decisions.

- The CFILE may be expressive enough to select from a set of alternatives at "runtime" on the client. CARP [6] is based on this premise for an array of caches. It is not inconceivable that the CFILE could compute some network distance or fitness metrics to a set of candidate proxy servers and then select the "closest" or "most responsive" server.

Note that it is valid to configure a DHCP daemon to respond only to INFORM option queries in static IP environments

Not all of the above mechanisms can be supported in all currently deployed vendor hardware and software. The hope is that enough flexibility is provided in this framework that administrators can select which mechanisms will work in their environments.

8. Conditional Compliance

In light of the fact that many of the discovery technologies described in this document are not well deployed or not available on all platforms, this specification permits conditional compliance. Conditional compliance is designated by three class identifications.

Additionally, due to the possible security implications of a DHCP broadcast request, it is onerous to REQUIRE an implementer to put himself or his implementation at undue risk. It is quite common to have rogue DHCP servers on a network which may fool a DHCP broadcast implementation into using a malicious configuration file. On platforms which do not support DHCP natively and cannot get the WPAD option along with its IP address, and which cannot support the DHCP INFORM unicast request, presumably to a known and trusted DHCP server, the likelihood of an undetected spoofing attack is increased. Having an individual program, such as a browser, trying to detect a DHCP server on a network is unreasonable, in the

authors' opinion. On platforms which use DHCP for their system IP address and have previously trusted a DHCP server, a unicast DHCP INFORM to that same trusted server does not introduce any additional trust to that server.

8.1. Class 0 - Minimally compliant

A WPAD implementation which implements only the following discovery mechanisms and interval schemes is considered class 0 compliant:

DNS A record queries
Browser or System session refresh intervals

Class 0 compliance is only applicable to systems or implementations which do not natively support DHCP and or cannot securely determine a trusted local DHCP server.

8.2. Class 1 - Compliant

A WPAD implementation which implements only the following discovery mechanisms and interval schemes is considered class 1 compliant:

DNS A record queries
DHCP INFORM Queries

Network stack change refresh intervals
CFILE expiration refresh intervals

8.3. Class 2 - Maximally compliant

A WPAD implementation which implements only the following discovery mechanisms and interval schemes is considered class 1 compliant:

DNS A record queries
DHCP INFORM Queries
DNS TXT service: queries
DNS SRV RR queries
SVRLOC Queries
Network stack change refresh intervals
CFILE expiration refresh intervals

To be considered compliant with a given class, an implementation MUST support the features listed above corresponding to that class.

9. Security Considerations

This document does not address security of the protocols involved. The WPAD protocol is vulnerable to existing identified weaknesses in DHCP and DNS. The groups driving those standards, as well as the SLP protocol standards, are addressing security.

When using DHCP discovery, clients are encouraged to use unicast DHCP INFORM queries instead of broadcast queries which are more easily spoofed in insecure networks.

Minimally, it can be said that the WPAD protocol does not create new security weaknesses.

10. Acknowledgements

The authors' work on this specification would be incomplete without the assistance of many people. Specifically, the authors would like the express their gratitude to the following people:

Chuck Neerdaels, Inktomi, for providing assistance in the design of the WPAD protocol as well as for providing reference implementations.

Arthur Bierer, Darren Mitchell, Sean Edmison, Mario Rodriguez, Danpo Zhang, and Yaron Goland, Microsoft, for providing implementation insights as well as testing and deployment.

Ari Luotonen, Netscape, for his role in designing the first web proxy.

In addition, the authors are grateful for the feedback provided by the following people:

Jeremy Worley - RealNetworks
Eric Twitchell - United Parcel Service

11. Copyright

12. References

[1] Moats, R., Hamilton, M., and P. Leach, "Finding Stuff (How to discover services)", Internet Draft, October 1997.

[2] Gulbrandsen, A., and P. Vixie, "A DNS RR for specifying the location of services (DNS SRV)", RFC 2052, October 1996

[3] Droms, R., "Dynamic Host Configuration Protocol", RFC 2131, March 1997.

[4] Veizades, J., Guttman, E., Perkins, C., and M. Day, "Service Location Protocol," Internet Draft, October 1997.

[5] Hamilton, M., and R. Wright, "Use of DNS Aliases for Network Services", RFC 2219, October 1997.

[6] Valloppillil, V., and K. Ross, "Cache Array Routing Protocol", Internet Draft, October 1997.

[7] Alexander, S., and R. Droms, "DHCP Options and BOOTP Vendor Extensions", RFC 2132, March 1997.

[8] Luotonen, A., "Navigator Proxy Auto-Config File Format", Netscape Corporation, http://home.netscape.com/eng/mozilla/2.0/relnotes/demo/proxy-live.html, March 1996.

[9] Mockapetris, P., "Domain Names - Concepts and Facilities", RFC 1034, November 1987.

[10] Perkins, C., Guttman, E., and J. Kempf, "Service Templates and service: Schemes", Internet Draft, December 1997.

[11] A Sample DHCP Implementation for WPAD , Inktomi Corporation, http://www.inktomi.com/TBD.html, February 1998.

[KEYWORDS] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

13. Author Information

Paul Gauthier
Inktomi Corporation
1900 South Norfolk Street Suite 310, San Mateo, CA 94403-1151
Phone: (650) 653-2800
Email: gauthier@inktomi.com

Josh Cohen
Microsoft Corporation
One Microsoft Way, Redmond, WA 98052
Phone: (425) 703-5812
Email: joshco@microsoft.com

Martin Dunsmuir

RealNetworks, Inc.
1111 3rd Ave, Suite 2900, Seattle, WA 98101
Phone: (206) 674-2237

Email: martind@real.com

Charles Perkins
Sun Microsystems, Inc.
15 Network Circle, Menlo Park, CA 94025
Phone: (650) 786-6464
Email: charles.perkins@Sun.COM