

# A Cautionary Note on Automatic Proxy Configuration

Andreas Pashalidis  
Information Security Group  
Royal Holloway, University of London  
e-mail: A.Pashalidis@rhul.ac.uk

## ABSTRACT

Web proxies can be used for a variety of services. Web browsers typically offer the option not only to statically configure a web proxy, but also to download proxy settings dynamically from the Internet. Unfortunately, the supporting infrastructure does not enable the browsers to properly authenticate the origin of these proxy settings. This inadequacy provides an opportunity for an attacker to interpose his own proxy between a client device and the web. The scope of potential harm includes wholesale or selective interception of web traffic, and web spoofing. In a practical setting the attack works even in the presence of SSL/TLS channels that are supposed to protect against interception and modification. Depending on the presence and configuration of a firewall, attacks can be launched by outsiders as well as by insiders. This paper examines various attack scenarios and proposes countermeasures to these attacks.

## KEY WORDS

web proxy, web security, proxy-based attack, authentication, web spoofing

## 1 Introduction

A web proxy is a machine that sits between a user's browser and the web. All requests from a user are sent to the proxy which eventually forwards them to the appropriate web server. The replies from the server also go through the proxy before finally reaching the user's browser.

Web proxies can be used for a wide variety of services, such as minimisation of download times through caching, reducing network load, intranet protection, content filtering, and privacy protection. As these services are often useful, many organisations have deployed web proxies. Because of their dependence on proxy services, some organisations have deployed multiple proxies (also called proxy farms), for purposes such as load balancing, request routing to an array of servers, and automated switching to backup proxies in case of a failure. Also, as network topologies change, so do the addresses of web proxies. In these situations, there must be a way to instruct web browsers to use particular proxies in a dynamic and real-time fashion; manual configuration of proxy settings would be infeasibly cumbersome.

For this reason, two enhancements to manual proxy configuration for web browsers have been introduced:

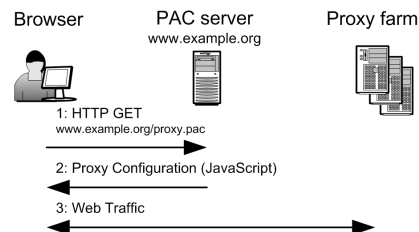


Figure 1. Basic proxy autoconfiguration mechanism.

the *Proxy Automatic Configuration* (PAC) mechanism [1] and the *Web Proxy Auto-Discovery* (WPAD) protocol [2], which we now briefly describe.

### 1.1 Proxy Automatic Configuration

The PAC mechanism was introduced by Netscape in 1996 [1]. Since then, it has become the *de facto* standard mechanism for web browsers to dynamically download proxy settings from the Internet. The idea is simple: the configuration settings exist as a small file, accessible via a published ‘Configuration Uniform Resource Locator’ (CURL); users need only configure their browsers to download the proxy settings from that CURL – this is done using the normal HyperText Transfer Protocol (HTTP) [3]. The fact that the configuration file actually contains a JavaScript function, enables organisations to flexibly assign different proxies based on properties of the browser and the sites that are being accessed.

The mechanism is illustrated in Figure 1. The browser is (manually) configured to download the proxy configuration from the CURL (`www.example.org/proxy.pac` in the figure). Whenever necessary, it contacts the specified web server (step 1) and downloads the proxy configuration in JavaScript, using normal HTTP (step 2). From that point on, all subsequent web traffic is routed through the proxy farm (or directly to the server), as dictated by the configuration JavaScript file (step 3).

### 1.2 Web Proxy Auto-Discovery Protocol

The PAC mechanism allows organisations to distribute web proxy configurations in a dynamic and real-time fashion. The fact, however, that users are still required to con-

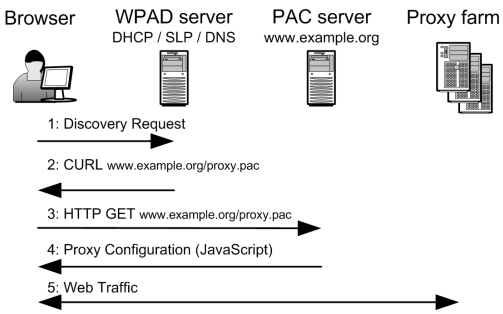


Figure 2. Web Proxy Auto-Discovery Protocol.

figure their browsers manually to set up the CURL is at best inconvenient and, in some cases, a prohibitive management overhead. It is for this reason that the WPAD protocol was introduced. WPAD is a mechanism for the automatic discovery of the CURL; it removes the need for users to manually configure their browser proxy settings. The WPAD mechanism is specified as an additional service of any of Dynamic Host Configuration Protocol (DHCP) [4], Domain Name Service (DNS), or a Service Location Protocol (SLP) [5, 6] server. The information flow that occurs when a browser configures its proxy settings is illustrated in Figure 2. The browser, by querying a DHCP, SLP or DNS server, first discovers the CURL (e.g. `www.example.org/proxy.pac`) of the PAC server (steps 1 and 2). The rest of the protocol is identical to the simple PAC scenario above; the proxy configuration JavaScript is downloaded using HTTP (step 3 and 4). Finally, subsequent web traffic is routed through the proxy farm, as dictated by the configuration file.

## 2 The attack

As pointed out in section 15.7 of [3], “by their very nature, HTTP [web] proxies are men-in-the-middle, and represent an opportunity for man-in-the-middle attacks. ... Proxies have access to security-related information, personal information about individual users and organizations, and proprietary information belonging to users and content providers”. Therefore, an attacker who wishes to intercept an organisation’s web traffic could achieve his aims by compromising its proxy server(s). However, if the organisation is using one of the automatic proxy configuration methods described above, the attacker has more possibilities at his disposal; all he has to do is to set up his own proxy and then try to make the target browsers use it. This is the attack considered in this paper. Its effectiveness is discussed in section 4. This section first examines how the attacker might inject a malicious proxy configuration into the target browsers.

### 2.1 PAC scenario

In the PAC scenario described in section 1.1, all it takes to launch the attack is to modify the proxy configuration file at the PAC server. Not only is this the most obvious and direct way, in practice it is likely to be the most effective as well because

- there is no way to prevent an insider with sufficient privileges modifying the file,
- web servers are vulnerable and often not adequately protected against outsider attacks, and
- the sensitivity of the proxy configuration file, compared with the rest of the web server’s contents, is often not appreciated.

Apart from compromising the PAC server, there are other techniques the attacker could use to inject a malicious proxy configuration. However, these typically require a presence at specific segments or subnetworks of the target organisation, making them less attractive for outsiders. They include the following.

- The attacker could intercept the HTTP response of the PAC server (step 2 in Figure 1) and replace (parts of) the proxy configuration. This could be done at the level of an internal router or hub. As proxy configuration files are typically small, operating at the packet level would suffice to mount the attack (i.e. there is no need for a complete TCP/IP stack).
- The attacker could set up a malicious PAC server and then employ DNS poisoning [7] in order to direct the target browser to use it when looking up the CURL.

### 2.2 WPAD scenario

In the WPAD scenario, described in section 1.2, several additional opportunities exist. First, the attacker could set up his own, malicious, PAC server and then try to compromise the automatic discovery mechanism of the CURL. According to the WPAD specification [2], the order in which browsers try to discover the CURL is as follows. First they try DHCP, then (optionally) SLP, and finally DNS. The most obvious attack route is, again, compromise of the corresponding WPAD server (which is then either a DHCP, an SLP or a DNS server). However, the following possibilities also exist.

#### 2.2.1 DHCP scenario

As “DHCP is built directly on UDP and IP which are as yet inherently insecure” [4], “unauthorized DHCP servers may be easily set up. Such servers can then send false and potentially disruptive information to clients such as incorrect or duplicate IP addresses, ..., incorrect domain name server

addresses, ... and so on.” [4]. In the WPAD scenario, the attacker only needs to send one spoofed UDP packet with the malicious proxy configuration to browsers that are trying to discover their proxy settings. It should be noted here that a properly configured firewall installation can protect the organisation from outsider attacks of this type.

Also, the DHCP-based attack prevents browsers from ever executing the SLP or DNS-based WPAD mechanisms. Thus, organisations that use SLP or DNS for WPAD are more vulnerable in this respect.

## 2.2.2 SLP scenario

SLP does provide the option for browsers to authenticate the origin of automatically discovered service URLs, including, of course, the CURL. Enabling this option is crucial, because “if Agents [SLP servers] are not configured to generate Authentication Blocks and Agents [user machines] are not configured to verify them, an adversary might easily use this protocol to advertise services on servers controlled by the adversary and thereby gain access to users’ private information” [5]. Setting up the system of servers and browser machines to perform authentication includes distributing keying material to SLP servers and end user machines (‘Agents’).

Unfortunately, SLP is not widely deployed. As a result, browsers that do not support it are still considered compliant with WPAD [2].

## 2.2.3 DNS scenario

Browsers are required to try the DNS-based discovery method only if the previous two methods have failed. This method requires browsers to perform DNS lookups for the ‘Well-known Alias’, ‘TXT’ and ‘SRV’ record types [8, 9] held by the nearest DNS servers. As DNS is built on UDP, the same considerations as with DHCP apply. Another issue of relevance is that, if the local DNS server is not responding, browsers query the DNS servers of superdomains as well. For instance, browsers belonging to the subdomain `a.b.c.org` first query the DNS server at `a.b.c.org`, then `b.c.org` and, if still unsuccessful, `c.org`. Thus, an organisation with a WPAD-enabled DNS server at its highest level (e.g. `c.org`) is vulnerable to attackers that set up malicious DNS servers in its subdomains. Further, querying a DNS server in a domain that lies outside the organisation’s trust boundaries also poses a threat. Microsoft Internet Explorer version 5.0 was shipped with a related vulnerability in its WPAD implementation — see [10].

## 3 Hiding the attack

It should be clear that, depending on the strategy used to inject a malicious proxy configuration, the attack can be

launched in a remarkably flexible fashion; it could massively target the web browsers of an entire organisation, only specific departments thereof, or just of selected hosts that may be ‘of particular interest’. But, unfortunately, things can get worse: the fact that the proxy configuration file is interpreted as JavaScript allows the attacker to further hide the fact that the attack is taking place by not significantly changing the usual web traffic patterns of the target organisation. The amount of flexibility at the attacker’s disposal is illustrated by means of the following example scripts. Consider an organisation, `someorg.org`, which uses two proxies using the aliases `proxy1` and `proxy2`. The authentic proxy configuration script would look similar to the following (for the syntax rules see [1, 11]).

```
function FindProxyForURL(url, host) {
  if (isPlainHostName(host) ||
      dnsDomainIs(host, ".someorg.org"))
    return "DIRECT"; else
    return "PROXY proxy1.someorg.com:8080;
           PROXY proxy2.someorg.com:8080";
}
```

This script causes browsers to bypass the proxies for all local requests (i.e. requests within `someorg.org`). All external requests go through `proxy1` or, if that is not available, `proxy2`. The following attack script causes browsers to behave exactly as in the original version, with the difference that all traffic is redirected to the attack proxy at weekends only.

```
function FindProxyForURL(url, host) {
  if (isPlainHostName(host) ||
      dnsDomainIs(host, ".someorg.org"))
    return "DIRECT"; else
    if (weekdayRange("SAT", "SUN"))
      return "PROXY proxy1.some0rg.org; " +
    else
      return "PROXY proxy1.someorg.com:8080;
             PROXY proxy2.someorg.com:8080";
}
```

Another variation, which allows the attacker to selectively intercept traffic to specific domains, is illustrated in the next example script.

```
function FindProxyForURL(url, host) {
  if (isPlainHostName(host) ||
      dnsDomainIs(host, ".someorg.org"))
    return "DIRECT";
  else
    if (dnsDomainIs(host, ".someBank.com))
      return "PROXY proxy1.some0rg.org; " +
    else
      return "PROXY proxy1.someorg.com:8080;
             PROXY proxy2.someorg.com:8080";
}
```

Here, browsers are instructed only to use the attacker’s proxy for requests to the `someBank.com` domain. Note that, in the example scripts, the attack proxy’s DNS name is `someOrg.org`, an intentionally misspelled version of `someorg.org`. By using this additional trick, the attacker can possibly hide the attack further. In fact, none of today’s popular web browsers can actually display a proxy configuration that was obtained using PAC or WPAD; as a result the probability that users will realise that something is wrong is extremely low.

## 4 Scope of the attack

Once the attacker has managed to cause target browsers to use his own proxy, he has effectively interposed himself between the user and the whole web. He can intercept, log and modify all browser requests and the responses of web sites. The scope of the attack is therefore twofold: on the one hand, all information users provide by means of their browsers is available to the attacker. On the other hand, the attacker can perform web spoofing. In contrast to other attacking techniques with potentially the same scope, such as URL rewriting [12] or interface spoofing [13], the proxy-based attack is more effective, as there are no indications in the browsers’ user interface that the attack is taking place — i.e. there is no deviation from the normal user browsing experience.

A technique is now presented that allows the attack to take place even if the browser requests a ‘secure HTTPS connection’, i.e. an SSL/TLS channel that is normally opaque to web proxies.

### 4.1 How to intercept SSL/TLS

Intercepting SSL/TLS traffic using a malicious web proxy is surprisingly simple. The technique is illustrated in Figure 3. Whenever the browser requests an SSL/TLS channel for a specific web site (e.g. `www.bank.com`, step 1), the proxy sets up its own SSL/TLS channel with the site (step 2). It then creates an SSL/TLS public key certificate with the ‘Common Name’ field equal to the web site in question (step 3). The certificate is signed using the attacker’s own private key. The proxy uses this spoofed certificate in order to set up a separate SSL/TLS channel with the browser (step 4). From this point on, the attack happens normally; all that happens in addition to normal operation is that the proxy decrypts and re-encrypts traffic between the two endpoints.

Of course, the user’s browser might complain in step 4, since the spoofed certificate will typically not be verifiable using a root key in the browser’s trusted key store. However, the spoofed certificate will not have expired, will contain the correct DNS name (these are parameters browsers typically check when validating SSL/TLS certificates), and may pose as trustworthy (for example, the ‘Issued By’ fields may contain the string `Verisign`,

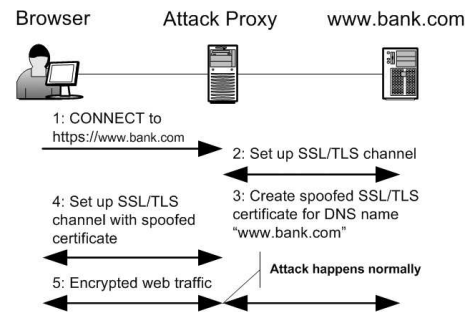


Figure 3. How to intercept SSL/TLS traffic

Inc). Thus, even users that rigorously check the certificate details, will find that the only problem is that they do not have an appropriate root key in their browser’s key store.<sup>1</sup> Anecdotal evidence suggests that not all users are likely to choose ‘No’ in a dialog box that asks whether or not to continue in what they are trying to do. Even worse, some users are likely to choose to permanently trust the certificate, an option typically offered by browsers in the same dialog box.

By default, the most widely used web browsers have well over 100 trusted root keys in their trusted store. It is therefore not inconceivable that a sophisticated attacker might manage to get his own root key into browsers’ key stores;<sup>2</sup> as a result even the dialog box mentioned above will not appear. ‘Root key injection attacks’ that achieve precisely this objective on Microsoft Internet Explorer have also been reported — see [14].

As a side comment, note that legitimate web proxies may ‘intercept’ SSL/TLS traffic in a manner similar to the attack proxy described above. The open-source Single Sign-On proxy available at `impostor.sf.net`, for example, behaves very much like the attack proxy described. As a result, it can be adapted to behave *exactly* like the attack proxy with only a few lines of extra code.

## 5 Countermeasures

The fundamental underlying problem that enables the attack is that the origin of the proxy settings is not properly authenticated. Without proper authentication it will always be possible to mount the attack in one way or another. Proper authentication, however, requires a key management infrastructure which is typically not in place. We therefore examine countermeasures that are readily applicable in today’s networks.

<sup>1</sup>Users that wish to validate the certificate’s fingerprint should do so in an out-of-band channel, i.e. *not* using their browser (since the attack proxy might also spoof the web). This appears the only way for an end-user to realise that something is wrong.

<sup>2</sup>Or certified by one of the sufficiently privileged — and trusted by default — certification authorities.

## 5.1 Simple countermeasures

The use of firewalls to protect against many types of outsider attacks is well-established. This also holds for most of the injection techniques mentioned in section 2. Therefore, a carefully configured firewall *must* be present everywhere where some form of automatic proxy configuration is in place. However, firewalls themselves must be protected and are not always impenetrable. Moreover, they do not provide the desired authentication; they merely make it more difficult for an outsider to inject malicious network traffic into sensitive links of the intranet.

Other countermeasures include the following.

- Do not use the PAC server's DNS name in the CURL (whether in the manual or the WPAD scenario); using its IP address instead prevents DNS poisoning attacks.
- In the WPAD scenario, DHCP should be used in preference to DNS and SLP, and SLP should be used in preference to DNS.
- If SLP is used for WPAD, the authentication option should be enabled.
- If DNS is used for WPAD, separate DNS servers should be used for every subdomain. Also, the DNS security extensions [15] should be employed.

## 5.2 Authentication through SSL/TLS

An additional countermeasure is to offer the proxy configuration for download via SSL/TLS (i.e. the CURL starts with `https` instead of `http`, in step 2 of Figure 1). An organisation could then issue its own certificates and pre-configure its browsers with a trusted root key.

Popular browsers support this already, although with different error handling procedures. Microsoft Internet Explorer 6.0, for example, silently ignores the CURL if the SSL/TLS certificate cannot be verified against a trusted root key. Mozilla 1.4, on the other hand, offers users the option to accept the certificate (temporarily or permanently) if it is not valid in some respect.

However, using this technique in its current form is not adequate; users are likely to accept 'invalid' certificates anyway and probably even choose to permanently trust them. What is required is a separate code path within browsers, such that certificates used to authenticate proxy settings are not treated like simple web site certificates. In fact, it appears that a separate *type* of certificate for downloading proxy settings via SSL/TLS makes sense. Further, it should not be possible for end users to trivially accept or enable trust for invalid certificates of this type. Instead, an invalid certificate should be treated as a fatal error and should generate appropriate error messages. These requirements are not met by any of today's popular web browsers. However, deploying them in future versions seems practical.

## 6 Conclusions

The attacks presented in this paper are the result of a trade-off between security and usability, in favour of the latter. In order to quickly deploy a solution to the automatic proxy configuration problem, mechanisms were built upon insecure protocols. The result is a multitude of entry points for an attacker to interpose his own proxy between users and the web. When combined with certain additional tricks, such as hiding the attack using sophisticated scripts or SSL/TLS-intercepting proxies, the attack can be mounted in a remarkably flexible fashion.

Although none of the techniques presented in this paper is, in isolation, new, it is the effect of combining these techniques that gives rise to particular concern. It is the intent of this paper to document and raise awareness of the potential scope of the attacks. It is hoped that this analysis will be helpful for organisations in making security-conscious decisions relating to automatic proxy configuration.

## Acknowledgements

The author would like to thank Chris J. Mitchell for enlightening discussions and comments on early versions of this paper, Dr. Ian Cooper, the editor of [2], for interesting discussions via e-mail, and Nessim Kisserli for setting up the environment needed to perform the necessary tests for this paper.

## References

- [1] Netscape, <http://wp.netscape.com/eng/mozilla/2.0/relnotes/demo/proxy-live.html>. *Navigator Proxy Auto-Config File Format*, March 1996.
- [2] Paul Gauthier, Josh Cohen, Martin Dunsmuir, and Charles Perkins. *Internet Draft: Web Proxy Auto-Discovery Protocol (work in progress)*. Internet Engineering Task Force, November 2000.
- [3] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. *RFC 2616: Hypertext Transfer Protocol – HTTP/1.1*. Internet Engineering Task Force, June 1999.
- [4] R. Droms. *RFC 2131: Dynamic Host Configuration Protocol*. Internet Engineering Task Force, March 1997.
- [5] E. Guttman, C. Perkins, J. Veizades, and M. Day. *RFC 2608: Service Location Protocol, Version 2*. Internet Engineering Task Force, June 1999.
- [6] J. Veizades, E. Guttman, C. Perkins, and S. Kaplan. *RFC 2165: Service Location Protocol*. Internet Engineering Task Force, June 1997.

- [7] CERT Coordination Center (CERT/CC). *Vulnerability Note VU#457875*, December 2002.
- [8] A. Gulbrandsen, P. Vixie, and L. Esibov. *RFC 2782: A DNS RR for specifying the location of services (DNS SRV)*. Internet Engineering Task Force, February 2000.
- [9] P. Mockapetris. *RFC 1035: Domain names — implementation and specification*. Internet Engineering Task Force, November 1987.
- [10] Microsoft. *Microsoft Security Bulletin (MS99-054)*, December 1999.
- [11] Stephen A. Thomas. *HTTP Essentials: Protocols for Secure, Scalable Web Sites*. John Wiley & Sons, 2001.
- [12] E. W. Felten, D. Balfanz, D. Dean, and D.S. Wallach. Web spoofing: An internet con game. In *Proceedings of the 20th National Information Systems Security Conference*, pages 95–103, Baltimore, Maryland, October 1997.
- [13] E. Ye, Y. Yuan, and S.W. Smith. *Web Spoofing Revisited: SSL and Beyond*. Technical Report TR2002-417, Dartmouth College, 2002.
- [14] S. McLeod and M. Cohen. SSL vulnerabilities. In *Australian Computer Emergency Response Team (AusCERT) conference 2002*, May 2002.
- [15] Donald Eastlake. *RFC 2535: Domain Name System Security Extensions*. IBM, March 1999.