Rogue Proxy PAC Example

Martin Zinaich *www.martinz.net* *2014*

A PAC file or Proxy Auto-Configuration file defines how web browsers can automatically select an appropriate proxy for accessing a given URL. The file is based on rules defined using Javascript, providing a scalable solution, which can be powerful enough to meet the demands of almost every situation it may face.

Introduction

Proxy PAC files are experiencing resurgence due to numerous cloud vendors supporting the tunneling of traffic up to their sites for traffic management or Information Security. Of course, on-premise proxy firewalls still provide one of the best protections and traffic monitoring tools.

This document presents some alternatives in building a robust proxy pac file that accomplishes the following:

- *Pseudo Load-Balancing between two proxy servers*
- *Fail-over between two proxy servers*
- *Traffic redirection for hosted and internal services sharing the same domain*
- *Enhanced performance via rogue IP address capture*

Background

This example pac file is for a company that has web services that reside in the cloud and on- premise, sharing the same domain and utilizing two proxy servers for load balancing and fail-over.

It is not an uncommon to desire that internal users hit web services directly, verses going outside only to come back in to get to a web services that has been NATed outside. There can be additional advantages for this type of configuration such as knowing the traffic is coming from an inside PC verses an external PC – thus additional protections, authentication or content may be employed as needed. However, some corporate web services may be hosted in the cloud, while some remain on-prem. This common situation can be easily handled in a pac file.

It is also common to have two proxy servers for load balancing and fail-over. Many times there needs to be additional pieces of equipment (aka: Load Balancers) to handle this function, but a specially constructed pac file can actually do this and eliminate additional hardware and thus additional expense and additional points of failure.

The File

The complete file is listed at the end of this document, but each section will be shown and explained to help you build your own file as needed.

The Basic PAC file

This is a simple PAC file – where if the browser is looking up http://webserver/login" then the browser will go local (bypassing the proxy) to retrieve the content. Everything else will go out to the proxy server "myproxy" on port 8080:

```
function FindProxyForURL (url, host)

{

if shExpMatch (url, "http://webserver/login") return "DIRECT";

else return "PROXY myproxy:8080";

}
```

This, as noted, is a basic proxy PAC file and references to some good sites on PAC commands are listed at the end of this document.

Building your own

Since PAC files are java script, I like to add a debug feature in the file for trouble shooting:

```
function FindProxyForURL(url, host)

{

  if (dnsDomainIs(host, ".pactest.xyz")) return alert("MyCompany\nPAC v03\nO=A E=B\n"+myIpAddress());
```

The above will pop-up a window if the browser goes to www.pactest.xyz – this is useful because the pop-up will validate that the PAC file is loading, it will return the version of your PAC file to make sure it is the one you think you are deploying and it will return the IP address of the computer. More on the IP address in the load balance section. NOTE: you will only see the pop-up window in IE – in Firefox it will show in the debug window – in Chrome, nothing.

Inside verses outside

You often will want to direct traffic either out to the Internet (and over the proxy) or you will want to bypass the proxy server and get the connection local (say an intranet). However, what if you have some of your sites in a cloud hosted service and some local – yet they all use the same domain. No problem:

```
(dnsDomainIs(host, "mydomain.com") &&

            !localHostOrDomainIs(host, "www.mydomain.com") &&

            !localHostOrDomainIs(host, "email.mydomain.com") &&

            !localHostOrDomainIs(host, "data.mydopmain.com") ||
```

This is sending all traffic for mydomain.com local (bypassing the proxy) unless the domain is www.mydomain.com or email.mydomain.com or data.mydomain.com. It first checks to see if the domain being called is mydomain.com. Next, it checks to see if it IS NOT one of the listed hosts. This first section we are building (and we have not seen all of it yet) is to list all hosts and domains where we want traffic to stay inside. Anything else will go out the proxy.

Of course, we could never list all of the places we want to send traffic out to the proxy, so we list the exceptions we want to keep inside the network. By doing the (dnsDomainIs(host, "mydomain.com") we are sending all traffic for mydomain.com local (bypassing the proxy). But we add the !localHostOrDomainIs for mydomain.com traffic we do want to send traffic out to the Internet (over the proxy). The "!" is a Boolean NOT and the "&&" is a Boolean AND. Thus, we are saying if the traffic is mydomain.com AND NOT www.mydomain.com AND NOT email.mydomain.com AND NOT data.mydomain.com then send the traffic local. The "||" is a Boolean OR and we will get to that next.

All Local Domains

It is a good idea to make the first check to see if the host being contacted is "domain-less", that is to say the user just typed in a host name only. This usually indicates a local host and thus we want to keep that traffic local. We do this by checking for domains:

```
if ((dnsDomainLevels(host) == 0) ||
```

The above command is checking if there are any domains listed – if not the traffic will go local

So at this point our PAC file looks like:

```
function FindProxyForURL(url, host)

{

if (dnsDomainIs(host, ".pactest.xyz")) return alert("MyCompany\nPAC v03\nO=A E=B\n"+myIpAddress());

if ((dnsDomainLevels(host) == 0) ||

//except hosted hosts

(dnsDomainIs(host, "mydomain.com") &&

        !localHostOrDomainIs(host, "www.mydomain.com") &&

        !localHostOrDomainIs(host, "email.mydomain.com") &&

        !localHostOrDomainIs(host, "data.mydopmain.com") ||

//end except
```

Next, we can add any other local domains:

```
dnsDomainIs(host, ".my.ads") ||

dnsDomainIs(host, "company.local") ||
```

IP Addresses and Rogue IP Detection

Rogue IP Detection is my own made-up term, to describe an "interesting" way to deal with IP addresses in a PAC file. Often developers and support staff need to type the IP address of an internal server in the browser, so a method is needed to deal with IP addresses in addition to host names. An often used method is the isInNet() command. This is used often to determine if a host name or an IP address is in a certain IP range. The problem with isInNet is that it relies on the OS to do a lookup and can cause a delay. This seems to happen even with an IP address. Some versions of browser do not deal with isInNet very well either, so if you do not need to use it - do not. There is an alternative option, if you are really just concerned with IP addresses being entered in the browser.

We can treat IP addresses as text and not IP numbers. By doing this there is no need to have the OS resolve and we eliminate a performance hit. Additionally if you assume that users typing in an IP are more the exception than the rule, you can take a bit of liberty with RFC1918 or the private IP address space. The private IP address space (the traffic you want to flow inside) is as follows:

10.0.0.0 - 10.255.255.255 (10/8 prefix)

172.16.0.0 - 172.31.255.255 (172.16/12 prefix)

192.168.0.0 - 192.168.255.255 (192.168/16 prefix)

The problem with the 172 range is it is not one contiguous block. To include exceptions for all the possible IP addresses will add many more lines. This is where the "Rogue" part comes into play – just consider all of the 172 space as internal (of course if you are not using 172 address space you can skip this completely). While people on the Internet are using the non RFC1918 172 numbers, it is highly unlikely someone will be typing a 172 address in to get to a web site. Therefore, if we want to treat IP addresses as text and we do not care to define out the 172 address space, we can use a few simple lines to route all private IP address space to the internal network:

```
(url.substring(7, 11) == "172.")||

(url.substring(8, 12) == "172.")||

(url.substring(7, 10) == "10.")||

(url.substring(8, 11) == "10.")||

(url.substring(7, 15) == "192.168.")||

(url.substring(8, 16) == "192.168.")||
```

The above will take the URL string, look for the "characters" of the IP address, and use that to determine where to send traffic. You will note two lines for each address space, one with a substring of 7 and one with 8. The command url.substring is looking at the URL, starting in position 7 or 8 and for the length of the IP address prefix, to see if it matches one of the private IP addresses. Starting at the 7th position accounts for "http://" and starting at the 8th position accounts for "https://". Therefore, using text commands verses IP commands and embracing a bit of rogue-ness, we can cover all internal IP space with a few lines and improve performance.

Send them Direct

The only thing left, for this part, is to send then "Direct" – that is bypassing the proxy server. It is also a good idea to add a bypass for the loopback IP:

```
(host == "127.0.0.1"))

    return "DIRECT";
```

At this point we have the following:

```
function FindProxyForURL(url, host)

{

if (dnsDomainIs(host, ".pactest.xyz")) return alert("MyCompany\nPAC v03\nO=A E=B\n"+myIpAddress());

if ((dnsDomainLevels(host) == 0) ||

            //except hosted hosts

            (dnsDomainIs(host, "mydomain.com") &&

                        !localHostOrDomainIs(host, "www.mydomain.com") &&

                        !localHostOrDomainIs(host, "email.mydomain.com") &&

                        !localHostOrDomainIs(host, "data.mydopmain.com") ||

            //end except

            dnsDomainIs(host, ".my.ads") ||

            dnsDomainIs(host, "company.local") ||

            (url.substring(7, 11) == "172.")||

            (url.substring(8, 12) == "172.")||

            (url.substring(7, 10) == "10.")||

            (url.substring(8, 11) == "10.")||

            (url.substring(7, 15) == "192.168.")||

            (url.substring(8, 16) == "192.168.")||

            (host == "127.0.0.1"))

return "DIRECT";
```

Load Balancing and Fail-over

If you have two proxy servers, you can do a pseudo load-balancing act with a PAC file and eliminate the need for load balancers. In my experience, this has been a good leveler of traffic. In addition, the fail-over works just fine.

First, if we do not match any of the local traffic rules, we have to do something "else":

```
else

   {
```

Next, it is very helpful to "stick" users to a single proxy. This helps with session state issues and is similar to "sticky sessions" in a load balancer. One easy way to do this is to send all users with an even IP address to one proxy and all users with an odd IP address to the other proxy:

```
// Return a static selected proxy list by even or odd IP address

    var myIp = myIpAddress();

    var ipBits = myIp.split(".");

    var mySeg = parseInt(ipBits[3]);


        if((mySeg % 2) == 0) //EVEN

        {

         return "PROXY 10.0.0.3:8080; PROXY 10.0.0.6:8080; DIRECT";

        }

        else  //ODD

        {

         return "PROXY 10.0.0.6:8080; PROXY 10.0.0.3:8080; DIRECT";

        }
```

Assuming that proxy A is 10.0.0.3 and proxy B is 10.0.0.6 – the above code will give someone with an even IP address proxy A as his or her primary proxy server and proxy B as the secondary proxy server. A user with an odd IP address will get proxy B as his or her primary and proxy A as the secondary.

That pretty much completes the code and we now have the complete PAC file as follows:

```
function FindProxyForURL(url, host)
{
if (dnsDomainIs(host, ".pactest.xyz")) return alert("MyCompany\nPAC v03\nO=A E=B\n"+myIpAddress());
if ((dnsDomainLevels(host) == 0) ||
            //except hosted hosts
            (dnsDomainIs(host, "mydomain.com") &&
                        !localHostOrDomainIs(host, "www.mydomain.com") &&
                        !localHostOrDomainIs(host, "email.mydomain.com") &&
                        !localHostOrDomainIs(host, "data.mydopmain.com") ||
            //end except
            dnsDomainIs(host, ".my.ads") ||
            dnsDomainIs(host, "company.local") ||
            (url.substring(7, 11) == "172.")||
            (url.substring(8, 12) == "172.")||
            (url.substring(7, 10) == "10.")||
            (url.substring(8, 11) == "10.")||
            (url.substring(7, 15) == "192.168.")||
            (url.substring(8, 16) == "192.168.")||
            (host == "127.0.0.1"))
  return "DIRECT";
else
{
      // Return a static selected proxy list by even or odd IP address
      var myIp = myIpAddress();
      var ipBits = myIp.split(".");
      var mySeg = parseInt(ipBits[3]);
            if((mySeg % 2) == 0) //EVEN
            {
             return "PROXY 10.0.0.3:8080; PROXY 10.0.0.6:8080; DIRECT";
            }
            else  //ODD
            {
             return "PROXY 10.0.0.6:8080; PROXY 10.0.0.3:8080; DIRECT";
            }
     }
}
```

References

http://findproxyforurl.com/

http://www.proxypacfiles.com

http://goo.gl/q16NYA

http://technet.microsoft.com/en-us/library/dd361918.aspx