Global Information Assurance Certification Paper

Detecting and Preventing Anonymous Proxy Usage

GCIH Gold Certification

Author: John Brozycki, john@trueinsecurity.com

Advisor: Dr. Eric Cole

Accepted: 9/13/2008

Detecting and Preventing Anonymous Proxy Usage

Abstract

Many organizations filter the Internet sites that their users may view. They do this for legitimate reasons that include preventing hostile work environments for their users, protecting network assets and data from malicious code or theft, and complying with regulations and company policies. Anonymous proxy services allow users to bypass most filtering systems. This paper explores methods organizations may use to detect and prevent anonymous proxy usage.

Introduction

Today's incident handlers and IT/Security professionals face many challenges in securing their networks and enforcing company policies that protect those networks. A growing concern is the use of anonymous proxy services [1]. With anonymous proxies, a user accesses an anonymous proxy website and selects an intended website (such as one that is blocked for the user) that the proxy will retrieve and display within its own page. More frequently, obfuscation techniques are used to hide the destination website, transmitted to the proxy server from the user via a parameter in the URL, making detection extremely difficult. New proxy sites popping up every day make the problem even more challenging. Users are able to circumvent policies and most of the technical measures put in place to enforce them, putting companies at risk of malware, illegal content, data loss, and hostile work environments if inappropriate content is accessed. Can access to these proxy services be detected and prevented? Further, are there instances when users should be allowed to use anonymous proxy services?

Many existing solutions rely almost entirely on blacklisting undesirable web sites, with the end result being that many users learn that anonymous proxies allow them to easily bypass this filtering. While blacklists serve a purpose, how do you know if users are circumventing your policies and your blacklists? One answer is to focus on detecting access to anonymous proxies.

This paper will focus on techniques for detecting access to anonymous proxy services. The first section, Detecting Known Proxies, will focus on known proxy services that can be blacklisted as well as TOR (The Onion Router). The second section, Identifying and Detecting Popular Proxy Systems, will focus on the popular proxy packages such as PHPPRoxy, CGIProxy, and Glype. The third section,

Other Ways to Proxy, will focus on translators, home network proxies, SSH and rogue VPNs. The fourth section, Detecting Proxied Access to Your Website, will discuss the challenges of detecting proxied access by Internet users to web sites that you host. Within the sections I will examine search strings you can use to review your web access logs, look at regular expressions you can use to search for whole types of anonymous proxies in logs or incorporate into a Snort IDS rule, and suggest a way to use anonymous proxy advertisement sites to update your own blacklists. Finally, in section five, Advanced Proxy Detection, some advanced proxy detection techniques will be discussed including some Perl scripts to help detect dynamic DNS usage and Base64 encoding within URLs, which is heavily used by today's anonymous proxy servers to conceal destination web addresses.

Legitimate uses of anonymous proxies

It's worth noting that there can be legitimate uses of proxy services. Proxy servers can allow access to sites that might normally restrict you based on IP address range. If your organization engages in research this may be important. Anonymous proxy servers also mask the user's origin, which may also be important if your organization wishes not to advertise certain access, such as to competitor's web sites. Anonymous proxies can also generally be set to filter certain content, such as Java script, flash, and other components that can contain malicious content. (It's probably a better idea for an organization to do this filtering itself rather than relying on an anonymous proxy for this protection!) Anonymous proxies can also pose security and privacy risks to users. Anonymous proxies come and go every day, established by people driven by varying motives, thus opening the possibility that at least some will siphon passwords or push malware. In early 2007 Microsoft warned about a weakness in the Web Proxy Automatic Discovery (WPAD) protocol that could cause user systems to download configuration settings that would make the systems use malicious proxy servers [2]. It's just as easy to create a few dozen anonymous proxy servers (or a few dozen names which you change or recycle over time and point to the same machines) and wait for users in schools and businesses desperate to bypass Internet filters to come to you. Although there may be some legitimate circumstances for some users to access anonymous proxy servers, in most cases you will not want your users accessing them and you will want to know about it when it happens.

Section I: Detecting Known Proxies

If you know something is a proxy, you can block it with a blacklist. This can be a lot of work to keep up. One possibility is to make use of proxy advertisement sites to update your blacklist for you.

1. Blacklists

Blacklists are not generally effective at preventing new or previously unknown proxy sites because the proxy sites have to be known before they can be added to a blacklist. However, as part of a larger solution, performing blacklisting of anonymous proxies is generally worth the effort. Blacklists can prevent already known proxies from being reused, leaving only newly created or renamed sites accessible. Blacklists are relatively "low cost" in that using them in blocking rules or log searches doesn't severely impact system performance. In some cases you may know something is a proxy but not have a way to readily detect it, so including it on a list of blocked sites gets the job done. Most anonymous proxy servers want to attract users (especially if they are hosting ads to generate revenue) and so are advertised by their owners. Just as proxy users can find these lists, so too can security analysts who want to block these proxies. With a little bit of scripting, sites that host fresh lists of new anonymous proxy servers can be mined and the sites blocked.

One such site that hosts proxy lists is mwolk.com. It tracks new proxy servers and also allows owners of new proxy sites to submit URLs for inclusion on the list, which is updated daily. The data contained in this list can be used to create an automatic feed for your blacklist. On the mwolk.com site, the proxy list resides at mwolk.com/proxy_list.html. It can be easily retrieved by several methods and utilities including CURL. To use the CURL command in Unix/Linux/OS X, the syntax would look like this:

```
curl http://mwolk.com/proxy_list.html > proxy_list.html
```

Once the html is retrieved to a local file, GREP can be used to select only the lines that contain the URLs for newly listed proxies and CUT can be used to filter the actual URL from the rest of the line in the HTML. A command to do this is:

```
grep url\"\>\<a\ href=\"http proxy_list.html | cut -d \" -f 6 > blacklist.txt
```

Another site offering proxy lists is Proxy4free at http://www.proxy4free.com. This site lists the IP (Internet Protocol) addresses and ports of anonymous proxies around the world, and appears to be updated about once a week. This site breaks its proxy listing down into five separate pages. Using the same technique, the IP addresses can be extracted and incorporated into a black list.

```
curl http://www.proxy4free.com/page1.html > proxy1.html
curl http://www.proxy4free.com/page2.html > proxy2.html
curl http://www.proxy4free.com/page3.html > proxy3.html
curl http://www.proxy4free.com/page4.html > proxy4.html
curl http://www.proxy4free.com/page5.html > proxy5.html
grep whois\.cgi\?domain\= proxy1.html | cut -d \= -f 3 | cut -d \" -f 1 | sort | uniq > proxy.txt
grep whois\.cgi\?domain\= proxy2.html | cut -d \= -f 3 | cut -d \" -f 1 | sort | uniq >> proxy.txt
grep whois\.cgi\?domain\= proxy3.html | cut -d \= -f 3 | cut -d \" -f 1 | sort | uniq >> proxy.txt
grep whois\.cgi\?domain\= proxy4.html | cut -d \= -f 3 | cut -d \" -f 1 | sort | uniq >> proxy.txt
grep whois\.cgi\?domain\= proxy5.html | cut -d \= -f 3 | cut -d \" -f 1 | sort | uniq >> proxy.txt
```

By running these commands in a scheduled process, this information can be automatically updated every day. Newer proxy listing sites can also be added as they are discovered and, if the sites change their output, the GREP and CUT commands can be adapted to continue retrieving proxy URLs.

There are some free proxy lists that people have compiled, but all seem to suffer from lack of attention; someone started the blacklist with good intentions and then ran out of time to keep it updated. If you run across a good blacklist it may prove a valuable resource, but keep in mind that it is likely not the owner's top priority. Tapping sites that advertise proxies, such as the examples just discussed, can ensure that you're able to blacklist many of the new proxies out there more effectively.

2. TOR

TOR, or The Onion Router, is a project to provide free software and volunteered network infrastructure (TOR nodes) to provide anonymity online. It's goal, as described on the TOR Project website is to protect you by:

> "…bouncing your communications around a distributed network of relays run by volunteers all around the world: it prevents somebody watching

> your Internet connection from learning what sites you visit, and it prevents the sites you visit from learning your physical location." [3]

While preventing someone who is monitoring your Internet connection from learning what sites you visit has positive connotations for free speech in many parts of the world, it presents a problem for corporate and educational environments where identification and control of sites accessed is necessary to enforce acceptable use policies, minimize data leakage, and stop users from accessing harmful content.

To test for TOR detection methods I first installed the Vidalia package, a free multi-platform suite of TOR tools containing TOR, a GUI for TOR called Vidalia, Privoxy (a TOR add-on which filters out ads, banners, and pop-ups), and Torbutton, the Firefox add-on that allows one button enabling and disabling of TOR sessions within Firefox. I then used Wireshark to capture network traffic as I initiated a TOR session.

Wireshark's ability to reconstitute a TCP stream was used to observe the content being sent and received. I noticed a string that the client sends out each time it establishes a connection with TOR. The string is as follows:

```
Tor1.0 U client <identity>0
```

TOR clients will attempt connections on the following ports: 9001-9004, 9030-9033, and 9100. According to the TOR FAQ the TOR clients will use ports 80, 443, 9001, or 9030 outbound [4]. If you can't block installation of Vidalia/TOR client on PCs in your network, you can write a Snort rule to detect them. The next section will discuss adapting regular expressions to Snort. Depending on what you are logging, you may not capture TOR client access in your web access logs, so an IDS is one way to catch TOR usage. Here is one rule for detecting TOR client usage in Snort:

```
alert tcp $HOME_NET any -> $EXTERNAL_NET 80,443,9001,9030 (msg: "TOR client access detected"; pcre:"/.*(Tor).+(client <identity>).*/i"; classtype:policy-violation; sid:50009;
```

Others have also found methods of detecting TOR usage. David Bianco posted a TOR detection Snort rule on his blog on 1/25/2005 [5].

David's rule is as follows:

```
alert tcp any any -> $HOME_NET any (msg: "TOR 1.0 Proxy Connection Attempt"; content: "TOR"; content: "<identity>"; within:30; classtype:policy-violation; resp:rst_all; sid:5000030; rev:1;)
```

Interlude: Adapting Regular Expressions to Snort Rules

Snort is a popular and widely used open source Intrusion Detection System. Snort supports the ability to incorporate Perl Compatible Regular Expressions or PCRE. Perl encloses regular expressions within a pair of "/" instead of the single quotes that were used with GREP. Given the regular expression above for detecting access to a CGIProxy proxy server, a generic Snort rule could be created to alert in real time rather than waiting for a log review.

There is also another option available with Snort. If you read David Bianco's rule, above, you may have noticed the "resp:rst_all" parameter. Snort can be compiled with the ability to do flexible response. This can allow it to send out TCP reset packages to attempt to reset the connection. It can also function in inline mode, where all Internet traffic must pass through the Snort device, giving it the ability to drop traffic that matches specific rules. Regular expressions to find proxy access can be used to create Snort rules that would prevent the proxy access in real time. Configuration of Snort goes well beyond this paper. However, if you have the Snort expertise (or know someone who does) you could opt to use rules to prevent in addition to detect proxy access.

In addition to Snort rules, I'll also use regular expressions to try to detect proxies while scanning web access log files with the GREP command (built into most Unix-like operating systems and available as a download for Windows operating systems) and in Perl programs. Regular expressions can be both complex and challenging. A good way to test or develop a regular expression is with a regular expression testing program or online utility. One such utility is Quanetic Software's online RegEx Test at http://www.quanetic.com/regex.php. However, if you really want to understand regular expressions the best resource I've come across is the O'Reilly *Mastering Regular Expressions* book written by Jeffrey Friedl.

Section II: Identifying and Detecting Popular Proxy Systems

Anonymous proxies are often built from a common code base or project. Understanding how the most popular anonymous proxy projects work can allow you to detect all proxies based on a particular project.

Overview

Most anonymous proxy servers are created using popular CGI and PHP based scripts that provide proxy capabilities and run on both Unix-like and Windows hosts. In both types, a client connects to the server and a CGI program (frequently written in PERL) or PHP script takes the client request and issues a request to the destination site, and then sends that data back to the client. Popularity of proxy scripts is often tied to a particular script's ability to support the full functionality of the destination web site, which grows more complicated as sites include embedded video and JavaScript functionality as well as flash and other media types. Of concern to users of anonymous proxy services is content that is downloaded through the proxy to the browser, where it then executes and tries to connect back directly to the website without going through the proxy. In these instances, your Internet log of blocked sites can come in handy so don't underestimate the value of reviewing what you have successfully blocked. If you see something blocked it doesn't necessarily mean the user didn't get to it. It may indicate the user went through an anonymous proxy but some component that was accessed tried to connect back to the site directly. Some anonymous proxies can check for content such as JavaScript and alter it accordingly, but most recommend that you disable JavaScript and not view other media types. While most proxy users are only concerned with the ability to completely experience the desired web site, the savvier users will also be concerned about getting detected. In the earlier section discussing blacklists, we examined blocking access to a sites that we *know* are proxies. However, if we can discern characteristics or signatures of the systems these proxies are built on, we can block entire classes of proxies instead of trying to blacklist all of the individual anonymous proxies that are created every day. Most proxy scripts are easy to customize and if this is done the altered script could bypass a signature. In practice, at least as observed at the present time, this isn't commonly done. The vast majority of proxies that I have experienced rolled out on a given script are identical to others running the same script with customization only occurring for the page theme and

advertisements. In looking at the most popular scripts I'll examine how each can be detected or blocked, allowing all non-customized proxies built on the same script to also be blocked. Techniques and rules can also be utilized to adapt to variations and new proxy scripts that come out in the future.

In the introduction, I mention that when a user connects to an anonymous proxy web server the user still needs to pass the URL of the restricted web site he or she is trying to reach. As with any typical website, this is usually passed as a parameter within the URL string. Often, this means that even if the proxy website is not a blocked site, the blocked site can still be read within the URL. This would mean that it could be detected through a simple string search of a web log. It is becoming increasingly common for the destination parameters to be obfuscated through code run in client so that the destination URL cannot be easily read.

3. PHPProxy

PHPProxy is a PHP-based proxy server that works on Unix variants as well as Solaris and Windows. The project is maintained at https://sourceforge.net/projects/phpproxy/ although, at the time of this writing, a new version hasn't been released since 2004. According to the SourceForge.net statistics page for PHPProxy, in mid-June 2008 it still averages 25-50 downloads per day [6].

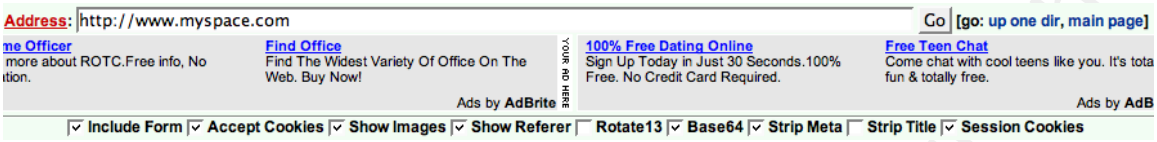
An example of an anonymous proxy that runs on PHPProxy is schoolsnooper.com (www.schoolsnooper.com). To test the PHPProxy server, a destination of www.myspace.com is entered and the resulting URI created is analyzed. The URI is as follows:

```
http://www.schoolsnooper.com/index.php?q=d3d3Lm15c3BhY2UuY29t&hl=2ed
```

PHPProxy follows this pattern:

{*hostname***}/index.php?q={***obfuscatedURL***}&hl={***identifier***}**

Once you get to the destination site, the proxy places a control frame at the top of the browser window with some interesting options along the bottom. Of particular interest are "Rotate 13" and "Base64", of which the latter is selected by default.



Unchecking the Base64 option and loading the page again yields the following URI:

http://www.schoolsnooper.com/index.php?q=http%3A%2F%2Fwww.myspace.com&hl=2e5

Checking the Rotate 13 option yields the following:

http://www.schoolsnooper.com/index.php?q=uggc%3A%2F%2Fjjj.zlfcnpr.pbz&hl=2f5

This confirms that Base64 and ROT13 (taking the real alphabet character and substituting it with what comes, or rotates, 13 positions after it, such that "w" becomes "j") are the two obfuscation methods used to hide the destination site name. Base64, also used to encode email attachments as a binary to text encoding scheme, is built into PHP with the function base64_encode(), so it's not surprising that this would be used for obfuscation. More information on Base64 encoding is available in RFC 3548 available at http://tools.ietf.org/html/rfc3548. An online Base64 encoding/decoding utility available at http://www.motobit.com/util/base64-decoder-encoder.asp confirms that "d3d3Lm15c3BhY2UuY29t" decodes to "www.myspace.com".

To detect any usage of a PHPProxy anonymous proxy server, use the following regular expression:

(index\.php\?q=).+(&hl).*

For example, if your web log file is named weblog.txt then PHPProxy proxies can be detected by using the GREP command as follows:

grep -Ei '(index\.php\?q=).+(&hl).*' weblog.txt

The 'E' parameter tells GREP the search field is a regular expression and the 'i' parameter tells GREP to be case insensitive. A Snort rule to detect this proxy would be:

alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg: "PHP Proxy detected"; pcre:"/(index\.php\?q=).+(&hl).*/i"; classtype:policy-violation; sid:50010;)

4. Glype

As described in the about section of the www.glype.com web page:

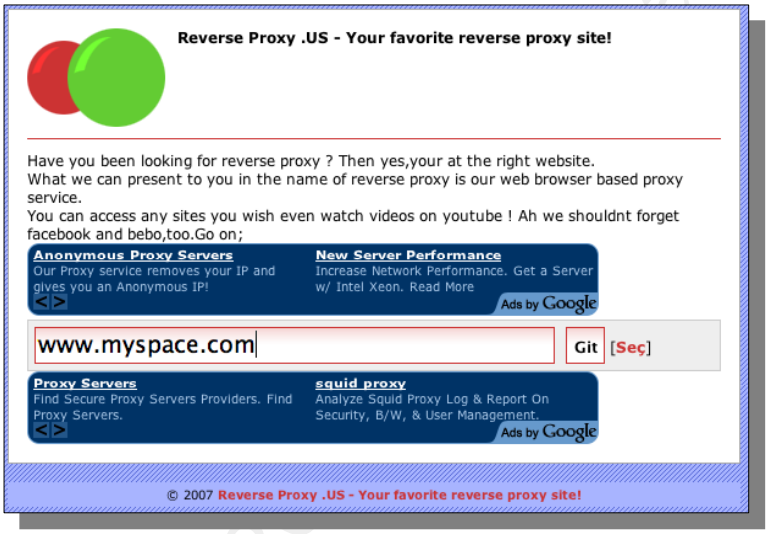
> "Glype Proxy is a free web-based proxy script written in PHP. It allows webmasters to quickly and easily set up their own proxy site. There is a huge market for these proxy websites that allow both anonymous browsing and bypassing network filters at school, college or work." [7]

Key to its popularity is the fact that current versions fully support the most popular sites and services including MySpace, You Tube, orkut, and others. It is also actively developed. As of May 31, 2008, there were more than 4,500 sites containing the "powered by Glype" tag as detected through a Google search. Glype stipulates that you must include this text on the page of systems running Glype, but I observed many implementations that did not comply, including sites listed on Glype's own featured proxy page. At a minimum of several thousand systems, Glype must be taken into consideration for detecting anonymous proxies.

Earlier versions of Glype gave the user the option of including the destination URL in the clear within the Glype URI or selecting an option to obfuscate it. In not obfuscating the destination URL, it becomes easy to determine the destination. You can simply read the destination URL out of the whole URL string being sent to the Glype server. Perhaps as a consequence, all of the Glype proxies that I've checked as of May 2008 were enabling obfuscation as the default, without an option to turn this off. Pictured below is a screenshot of the www.reverseproxy.us proxy server that was available at the time I observed it in May 2008. The site allows for a URL entry. The option button doesn't work, but could allow disabling of cookies, flash, images, and scripts if it functioned. The proxy is hosting Google ads to pay for expenses. When the destination site is reached, there is an additional Google ad at the top of the page.

www.reverseproxy.us is Glype powered but doesn't include the required tag line

The URI that Glype creates follows a set pattern just like PHPProxy does. Although this pattern may change in the future, at the current time it is consistent across all versions of Glype that I've observed. The URI for accessing MySpace from www.reverseproxy.us is:



http://www.reverseproxy.us/browse.php?u=Oi8vd3d3Lm15c3BhY2UuY29t&b=143

Going to another Glype server (www.proxyboxonline.com) and typing in MySpace as the destination, the URI was as follows:

http://proxyboxonline.com/browse.php?u=Oi8vd3d3Lm15c3BhY2UuY29t&b=29

Note that the "u=" value is exactly the same. This is the obfuscation for www.myspace.com. In fact, it is Base64 encoded, just like PHPProxy. From this, we can see that Glype servers will use the following pattern:

{*hostname***}/browse.php?u={***obfuscatedURL***}&b={***identifier***}**

Further, copying the URL starting at the "/browse.php?u=" all the way to the end and pasting it after

the hostname of another Glype server retrieves the same destination URL.

To detect specific sites being accessed through Glype, enter the site in a Glype server and copy the obfuscated site from the "u=" value. For example, to only detect when MySpace is being accessed through Glype, use the string:

```
/browse.php?u=Oi8vd3d3Lm15c3BhY2UuY29t
```

The Base64 obfuscated URL for www.myspace.com looks similar to the encoding utilized by PHPProxy but yet the string is not identical. The "u=" value of "Oi8vd3d3Lm15c3BhY2UuY29t" was run through the Base64 decoder available at http://www.motobit.com/util/base64-decoder-encoder.asp and confirmed to be Base64 encoded. The string decodes to "://www.myspace.com". Glype adds a leading "://" to all addresses before encoding, likely to keep string searches of Base64 URLs from finding a hit. (An additional base64 encoding and decoding site can be found at http://makcoder.sourceforge.net/demo/base64.php.)

To block or detect any usage of a Glype anonymous proxy server, use the following regular expression:

(browse\.php\?u=).+(&b).*

To search with GREP as in the previous example:

grep -Ei '(browse\.php\?u=).+(&b).*' weblog.txt

A Snort rule to detect Glype would be:

```
alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg: "Glype Proxy detected";
pcre:"/(browse\.php\?u=).+(&b).*/I"; classtype:policy-violation; sid:50015;)
```

5. CGIProxy

Another popular proxy is James Marshall's CGIProxy, which is an actively developed project available at the author's site and is described as a CGI script that acts as an HTTP or FTP proxy [8]. CGIProxy is written in the PERL language instead of PHP. By default, no obfuscation is done to the destination URL. ROT13 support can be enabled if you remove the line comments in the proxy_encode() and proxy_decode() routines. Custom encoding routines may also be created. As the

author warns, the routines need to be processor-efficient or it will become a performance bottleneck. An example of a site using CGIProxy that utilizes URL encoding is http://www.daveproxy.co.uk. Using this site to access MySpace produces the following URL:

http://www.daveproxy.co.uk/browse.php/Oi8vd3d3Lm15/c3BhY2UuY29t/Lw_3D_3D/b29/

The obfuscated text looks like Base64. After removing the forward slashes and extraneous data, starting with the "_" character, a Base64 decode yields the following:
Oi8vd3d3Lm15c3BhY2UuY29tLw = ://www.myspace.com/

The addition of the forward slash characters is an obvious attempt at thwarting automated parsing; forward slashes usually represent separation of values. Like the other proxy scripts examined it shows that variations of Base64 are predominant due to its built-in function support, the lack of ability or desire by most users to alter the source routines, and the performance penalty that custom encryption or obfuscation could introduce. To detect a CGIProxy proxy use the following regular expression:

(/browse\.php/).+/.+/.+(/b).+/

A GREP statement to detect this would be:

grep -Ei '(/browse\.php/).+/.+/.+(/b).+/' weblog.txt

A Snort rule for CGI Proxy would be:

```
alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg: "CGIProxy detected"; pcre:"/
(/browse\.php/).+/.+/.+(/b).+//i"; classtype:policy-violation; sid:50020;)
```

6. SSL Proxies

Anonymous proxies can employ digital certificates to make detection more difficult as all traffic passed will be encrypted. Except where self-issued (in which case they're not trusted by default browsers) SSL certificates are expensive from both a monetary and CPU (as the server handles the encryption) standpoint. The free, advertised anonymous proxies discussed here are unlikely to pay these premiums, especially when digital certificates get tied to a fully qualified domain name and many

anonymous proxies come and go or change names week by week. For these reasons, SSL proxies are generally represented by commercial offerings. Examples of SSL proxies include Slicky Proxy (http://www.slickyproxy.com/) and Anonymizer (www.anonymizer.com). Because they are commercial entities advertising their names and paying for commercial digital certificates, these proxies are easy to find and easy to block; they will not be changing their domain names frequently, if at all. This is not surprising since their model is to provide anonymous browsing services to customers looking for anonymous browsing, not trying to circumvent filtering. These services can be blocked with a blacklist or a filtering rule as described in the blacklisting section.

Section III: Other Ways To Proxy

There are other ways to gain proxy services including language translators and SSH and VPN access to home or remote networks. This section will discuss them.

7. Translators

Sometimes tools are used for purposes beyond their original intent. Language translation services are a good example. Google, Yahoo, and Microsoft all offer free translation services that take an entered URL and retrieve, translate, and display it within the translation service's browser window. Take away the translation and what you have is basically a proxy. There are some significant disadvantages to using translators as an anonymous proxy. These include the inability in some translators to navigate to additional pages, the inability to run JavaScript, the inability to access media in the retrieved page, and the inability to completely shield the user's activity if JavaScript, flash, or other downloaded elements attempt to reference the destination site directly. Also, since their purpose isn't to hide the translated site, I am unaware of any translators that obfuscate the destination URL, so the destination site will appear within the URL and be detectable if monitored. Still, for some content, translators can be highly effective for bypassing filters.

If you can't block these sites outright because they are needed for legitimate use, you'll probably want to monitor them and check on their usage. Frequency and site destination will immediately tip you off if translators are being abused as will monitoring with keywords, such as looking in web logs for sites that you've blocked, like myspace.com.

Google Translator

Google Translator can be accessed at http://www.google.com/language_tools?hl=en under the section “Translate A Page” from www.google.com. Incredibly, the FROM and TO languages can be set to the same values! (You’ve got to wonder if someone didn’t design it like that on purpose.) To view MySpace using Google Translator as a proxy, one would only have to enter the URL and set both the FROM and TO languages to be English. During the time of this writing, I was sometimes able to get Google Translator to go from English to English and at other times I was not, so Google may be starting to restrict this. Regardless, by selecting a different FROM language the proxy effect works just fine. For example, selecting to translate MySpace from Dutch to English yields the MySpace page with this URL:

http://translate.google.com/translate?u=http%3A%2F%2Fwww.myspace.com&sl=nl&tl=en&hl=en&ie=UTF-8

The “&sl” variable indicates the source language, the “&tl” variable indicates the target language, and the “&ie” variable indicates the character set to use. As previously noted the destination URL is not obfuscated. The text “www.myspace.com” is clearly present so string searches for restricted sites will catch this. If Google’s translation services are not required then blacklist the following site: translate.google.com

If translation services are required, string comparisons will be necessary to block or detect access to unauthorized sites. Since it isn’t the intent of a translator to provide anonymous proxy capabilities it doesn’t obfuscate destination URLs, so the destination site can be easily found in a normal text search of a web access log. This holds true for the remaining translation sites as well.

Yahoo Babel Fish

Yahoo Babel Fish (http://babelfish.yahoo.com/) works much like Google Translator. Its FROM/TO language drop down menu doesn’t allow you to pair the same FROM and TO languages. However, if MySpace is entered as the translation site and a language pairing such as Dutch to English

is selected, the retrieved page is in unaltered English. The URL Yahoo creates for doing this is as follows:
http://babelfish.yahoo.com/translate_url?doit=done&tt=url&intl=1&fr=bf-home&trurl=http%3A%2F%2Fwww.myspace.com&lp=nl_en&btnTrUrl=Translate

Babel Fish uses the variable "&lp" to denote the language pair (in this case Dutch to English, or "nl_en") and the destination website is clearly visible in the URL. Since there are multiple language pair combinations that can yield unaltered pages and English to English isn't allowed, there really isn't a specific combination to focus on to block certain usage. The best bet is to block it outright or to search for unapproved sites within the entire URL. To block all use of Yahoo Babel Fish block access to the site:
bablefish.yahoo.com

Windows Live Translator

Live Translator works much like Yahoo Babel Fish. However, its default view is a side-by-side, framed comparison of the original site and the translated site. View option icons in the upper right of the top control frame allow toggling the window view, allowing only the source (non-translated) page to be viewed. Using this option, the selected destination language for translation doesn't make any difference because the original version is always available. The URL is as follows:

http://www.windowslivetranslator.com/BV.aspx?ref=Internal&a=http%3A%2F%2Fwww.myspace.com

As with the previous translators, there isn't a string to search on to indicate abuse. Options are to block access outright or to scan web access logs to see if unauthorized sites were accessed. To block all use of Windows Live Translator use:
windowslivetranslator.com

InterTran

Translation Expert's InterTran is another translation service capable of translating (and proxying) web pages. Its URL is http://www.tranexp.com:2000/InterTran. Aside from the use of a

non-standard port (port 2000) that many organizations may already block outbound for http traffic, my experience is that the site is difficult to bring up. Nevertheless, the site should probably be blocked unless you have an explicit need to access it. To block all use of InterTran block the site: www.tranexp.com

8. Home Network Proxies

Recognizing that anonymous proxies can be detected and that some can be set up for malicious purposes (such as capturing passwords) there have been efforts to allow you to use your home broadband connection as an anonymous proxy that you could then reach from work, school, or any Internet connection where filtering was employed. If the proxy is known only to the owner and isn't advertised, it will not be known to any proxy blacklists.

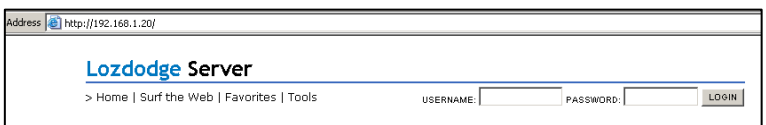
LozDodge

LozDodge (available at http://www.proxy-avoidance.com/) is software for Windows operating systems that users install on their home PCs (or on any other unfiltered PCs that they may access) to turn these PCs into personal proxy servers for use when they are at a location where site filtering is enabled. Once installed, a system service named LDG_Service is created. Accessing the LozDodge Server brings up a logon screen as pictured below.

After logging in, a URL field is provided. Entering www.myspace.com as the destination URL yields the following browser URL, shown here with the non-routable private IP address of the machine I was testing it on:

http://192.168.1.20/http/com/myspace/www/.prx/

LozDodge provides minimal obfuscation; the destination is immediately evident to the human eye but it may be enough to bypass filters looking for the fully qualified name. The URL is separated at the



"dots", reversed, and forward slashes are also used for good measure. Regardless, a rule looking for "myspace" would catch it. The goal and capability of this product, at least at the present, is to get you past filters that are only checking the destination site name. One possible filter for LozDodge would be:

(\/http\/).+\/.+(.prx)

A GREP command to detect this in your web access log:

grep -Ei '(\/http\/).+\/.+(.prx)' weblog.txt

A Snort rule to detect LozDodge:

```
alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg: "LozDodge Proxy detected";
pcre:" /(\/http\/).+\/.+(.prx)/i"; classtype:policy-violation; sid:50025;)
```

While LozDodge appears easy to catch, personal proxy solutions in general present a difficult problem for detection. There is also more incentive for the owner to add customization and obfuscation (or to look for a solution that is easily customized) to protect their personal resource. There are some techniques to detect the access to a home connection even if it cannot easily be determined to be a proxy service.

Detecting Access to Dynamic IP Addresses

One way to detect access to dynamic IP (consumer) addresses is by utilizing Real time Black List (RBL) resources which have been created and used primarily to aid in blocking spam. There likely isn't any reason a corporate or educational institution should allow access to a broadband user's IP address, so why not alert on it or even block it? (As an added bonus, this can detect when users or compromised systems access a phish, botnet, or other malware hosted on a consumer IP address.) Several dynamic blocklists/blacklists exist. One example blocklist is SORBS (http://www.dnsbl.us.sorbs.net), which also maintains an individual listing of dynamic IP addresses. Not limited to only spam IPs. SORBS adds in dynamic IP blocks as people report them and as it finds them itself.

Most RBLs work just like a DNS server. As an example, say you wanted to check the IP address 1.2.3.4 to see if it was a known dynamic address. You could manually use the HOST

command as follows:

host 4.3.2.1.dnsbl.sorbs.net {DNS SERVER IP}

The IP Address is reversed and the RBL DNS server name is appended and then a DNS server to run the query for your client is added. A successful lookup, meaning the IP address was on the list, would look like this:

```
host 4.3.2.1dnsbl.sorbs.net 4.2.2.1
Using domain server:
Name: 4.2.2.1
Address: 4.2.2.1#53
Aliases:
4.3.2.1.dnsbl.sorbs.net has address 127.0.0.10
```

Note that a non-routable IP address is returned. It is the existence of the record that matters. (There is a significance to the last octet of the private IP address returned: ".10" as it indicates that the record is associated with a dynamic/user account. For more information, see the return codes at: http://www.us.sorbs.net/using.shtml.) An unsuccessful lookup, meaning the IP address was not on the list, would look like this:

```
host 4.3.2.1.dnsbl.sorbs.net 4.2.2.1
Using domain server:
Name: 4.2.2.1
Address: 4.2.2.1#53
Aliases:
Host 4.3.2.1.dnsbl.sorbs.net not found: 3(NXDOMAIN)
```

The first query result would indicate that the IP address was in the blacklist as a dynamic IP address. The question now becomes how to make this actionable. For the purposes of this paper, I'm going to stick to detection, but the concepts could be applied to blocking as well. Using techniques similar to those used to extract proxy server names for blacklists, the same could be done to extract a

list of URLs accessed through your corporate proxy server, web filter, firewall, etc. Given a list of sites accessed, both by URI and IP addresses, we'll start with a text file that's formatted as follows:

www.servername.com
1.2.3.4
aserver.somewhere.com/default.htm

From this file, we want to extract any IPs to another file, reverse the order of the four octets, and do a lookup to see if the IP is a known dynamic IP address. The following four line script will extract entries from "list.txt" that are IP addresses, reverse the octet order, create and run a script file that checks each entry against the SORBS database, and puts any hits (i.e.: where a private IP was returned) in a file called homeusers.txt. To add or replace with another RBL, simply replace ".dul.dnsbl.sorbs.net" with the correct information for the other RBL. Note that 4.2.2.1 is a fast DNS server that I'm using for this example, but you can (and probably should) use others in place of it. Also note that the second, indented line is a continuation of the first line.

```
grep '^[0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}$' list.txt | awk -F . '{print "host " $4."." $3."." $2."."
        $1.".dul.dnsbl.sorbs.net 4.2.2.1 >> temp2.txt"}' > temp1.bat
chmod +x temp1.bat
temp1.bat
grep 127.0.0 temp2.txt | awk -F . '{print $4."." $3."." $2."." $1}' > homeusers.txt
```

The GREP command searches for IP addresses in the input file 'lists.txt' and then pipes the output to awk. Awk is used to output a command starting with the HOST command, then to separate the four octets of the IP address by using the "." as the delimiter, then output the fields backwards (1.2.3.4 becomes 4.3.2.1) and finally to append the sorbs.net domain, IP of the DNS server being used, and syntax to append the destination file 'temp2.txt' with the results. The CHMOD command is used to make the 'temp1.bat' output file executable under Unix-like operating systems (it's not necessary if using Windows). On the third line the batch/script file (temp1.bat) is run, with output going to the file 'temp2.txt'. Finally, in the last line GREP is used to pull out any 'hits' with the RBL, undoing the reversing of the IP address before outputting the entry to the file 'homeusers.txt.' All of these commands are supported natively in most Unix-like operating systems. GNU Awk for Windows can

be found at http://gnuwin32.sourceforge.net/packages/gawk.htm and GNU Grep for Windows can be found at: http://gnuwin32.sourceforge.net/packages/grep.htm.

Reviewing the file "homeusers.txt" should give you the destinations that you want to take a closer look at. Spamhaus has a similar Policy Block List (pbl.spamhaus.org) but their service is not free for business/professional use. To see if you qualify or to get rates go to http://www.spamhaus.org/organization/dnsblusage.html.

One problem with relying on an RBL is that there are none that I am aware of that cover the Internet Service Provider IP address space 100%. If you are a company or a school that has users who are likely to be using a few common providers, it is worthwhile to do an online check on some of the IP addresses used locally by each provider to see if the associated address block is in their database. In the case of SORBS, the page is at www.de.sorbs.net/lookup.shtml. If not found, you can contribute it yourself (for SORBS that would be done via the email form at www.de.sorbs.net/cgi-bin/mail), which simultaneously helps your company, as well as any other that uses the service.

In the next section a Perl script will be introduced that can help strip out web log entries where IP addresses were used. While IP addresses are sometimes used to access legitimate web sites (especially in the web developer carelessly left a direct IP address link or reference on a page) this is often suspect and worth reviewing.

Detecting Dynamic DNS Hosts

A myriad of dynamic DNS services exist with most offering at least some services for free. Examples include; www.no-ip.com, www.dyndns.com, and freedns.afraid.org, all available as of June 2008, and there are many others. These services allow you to run a client or access a web page where you can update your IP address to an entry within one of the dynamic DNS domains (or your own domain if you've paid for one) whenever your ISP changes your IP address. In so doing, it allows people to access their home systems with a name, such as BobsPC.dynamicdnsprovider.com, instead of remembering their currently assigned IP address. Dynamic DNS providers are numerous and often add new domain names, making blacklisting a mostly fruitless exercise. However, there is a way to detect them.

Most ISPs have created reverse lookup records for all of their dynamic IP addresses. For example, take the IP address I have as I type this at a public library, which is 24.xx.107.227. I'm sanitizing the real address by replacing the second octet with "xx". If I do a reverse lookup (in Windows: ping -a 24.xx.107.227, in Unix/Linux/OS X: host 24.xx.107.227) I get results as follows: rrcs-24-xx-107-227.nyc.biz.rr.com

This is because that address belongs to the ISP; they are authoritative for the address block and they provide the answer when a reverse lookup request is made for an IP address. Although the dynamic DNS providers can provide forward DNS resolution as owners of the domain, their DNS servers are not used for reverse lookups. Therefore, one technique to detect the use of dynamic DNS is to take the domain part of a web access log entry, resolve the IP address, do a reverse lookup of the IP address, and compare the two names. If the domain names don't match, it's probably someone using a dynamic DNS provider.

As an example, I have set up a dynamic account with no-ip.com. The dynamic name johntest.zapto.org will now resolve to my home IP, currently 24.xx.173.237. The name that my ISP provides for reverse lookups of my IP address is cpe-24-xx-173-237.xxx.res.rr.com. If I attempted to access a proxy running on my home system from a work or school location, the web access log might contain an entry like this: "http://jontest.zapto.org/ http/com/myspace/www/.prx/"

To strip out the domain name, we'll look for the "://" from the left and the next "/" after that, and grab everything in between. The CUT command can only accept a single character as a delimiter, which might make it a complex, multiple pass process to strip out. In actuality, we can specify the field number as well and, in doing so, it becomes pretty easy. Given a file of URLs only (you'll need to add further commands to get down to this if your log file contains more info) the following command will extract the domains:

cut -d "/" -f 3 weblog.txt | sort | uniq

The results are used as the input for the HOST command. However, due to the need for logic and file control, a batch command script was abandoned for a Perl script. The Perl script is findddns.pl, which is in Appendix A. The script works as follows. The CUT command is still utilized via an external call, which includes piping the output into SORT and UNIQ to remove duplicate domains to

minimize lookups. The CUT external call here assumes a file with only one field, the URL field, for example:

http://www.google.com
http://192.168.0.1/myhomeproxy.php
http://mydynamicdnsservice.com/myhomeproxy

If your web log file has multiple fields, use CUT similarly to extract just the URL field and pipe that into the existing CUT command. For example, if you had ten fields separated by commas and the URL field was the fourth field, you might wan to use: **cut -d "\t" -f 16 weblog.txt | cut -d "/" -f 3 | sort | uniq**. (However, the CUT command only works on a single character. One way to get around this is to type the first quote, then press CONTROL and V at the same time and, while still holding them down, press the TAB key , then end quote it. It will look like you've got about five blank spaces between the quotes, and it takes a little practice to do. If you can't get it to work, there's always the AWK solution, which will be discussed in a moment.) The first cut looks for tab-delimited fields in a file named 'webog.txt' and takes the sixteenth field (which is the URL field in Microsoft ISA logs) and pipes that into a second CUT command. The second CUT command uses the '/' as a delimiter and grabs the third field. Since the output from the first CUT command should be something like 'http://www.website.com/default.htm' the third field will be everything after the "//" and up to the first "/" after the fully qualified name. (Note: sometimes the CUT command doesn't parse properly and crashes processing the input. I've observed it crashing often when processing log files from Microsoft's ISA Server. When it crashes, you will still have the data that it had processed so far, but not the rest of the file, so it's important to monitor any errors generated when CUT is run. You may also have trouble entering a Tab or other value that fields are separated on. The AWK language can help out here. To replace the above CUT commands with AWK it would be: awk -F "\t" '{print $16}' weblog.txt | awk -F "/" '{print $3}'. Once the URLs have been extracted via CUT or AWK, they are ready to be processed by the Perl script.

The script reads in the output of the CUT commands, which are "domains" such as "www.google.com", "192.168.0.1" and "mydynamicdnsservice.com." The HOST command is called to retrieve the IP address from the name. The HOST command is then called again to perform a

reverse DNS lookup of the IP address. The script attempts to grab the simplified name, which is the information needed to distinguish the domain without any specific host information, and compare the original to the one retrieved from the reverse lookup. Using my "jontest.zapto.org" dynamic DNS example above, the name I set up resolves to the IP 24.xx.173.237, and a reverse DNS lookup of the IP yields cpe-24-xx-173-237.xxx.res.rr.com. Comparing the "simplified" name we see that .zapto.org doesn't equal .rr.com. This also works for many anonymous proxies that have multiple names or aliases or resolving to the same IP. For example, the "www.schoolsnooper.com" proxy mentioned earlier resolves to an IP that has a reverse DNS value of ".techentrance.com." There are legitimate sites that can be set up this way, but you can frequently tell by looking at the name if it is likely a proxy or not. This didn't work out as well as I had hoped. I found it surprising how many sites don't have a reverse DNS record set up. Further, many sites are convoluted with the name resolving to many IPs (for load balancing or redundancy) and/or one or more aliases, and even lookups on the IP address can return multiple names. This means there are a lot of false positives as so many legitimate domains aren't configured correctly or resolve to many different values. The script grabs the first name or IP address retrieved for simplicity, which may cause it not to match a legitimate entry even if one of its values is a match. Although HOST has been ported to Windows (http://www.itsamples.com/software/host.html) the parameters and response are not quite the same and will not work with this script without modification. Another limitation is that it is not country domain aware, so while it will strip out ".domain.com" successfully when identifying a domain, if the original domain is "domain.co.uk" it will strip it out to "co.uk." Also, I found that Akamai distributed caching servers cause a lot of false positives. I added a section for ignoring certain hits from the output. Instructions are in the comments for adding more and, as another "to do" this could be an external file that the script reads to make it easier to edit. Finally, a Perl module to handle the DNS calls (perhaps Net:DNS) could improve upon the external HOST calls and make the script more universal. Even without these improvements and the problem of false positives, the script is still useful for reviewing IP access and domain mismatches. It should make reviewing easier even if there are a lot of false positives. The script output contains a header row and each value is tab delimited, so it can be easily imported into Excel or another spreadsheet for easy sorting and review. Directions and comments are included in the script.

9. SSH Tunneling and Rogue VPNs

The SSH protocol enables encrypted connections as well as the ability to tunnel various traffic over it via port redirections. It can be used to remotely connect to a home system and web traffic tunneled through that connection. To address this, it is a good idea to monitor SSH traffic destinations and limit its usage where possible, such as a white list of acceptable destination if SSH is used. Since access will require an IP address or domain name (likely dynamic) previous detection techniques should help here, too.

VPNs may be used in similar fashion. Individuals could establish connections to VPN services set up on a home network or to a rogue or compromised VPN site to access filtered sites. For VPNs on home networks you can use the techniques for detecting access to dynamic DNS hosts or consumer IP address ranges. Monitoring outbound VPN usage, if allowed, is a good idea to determine what is normal and what is not. If your organization doesn't have any reason to allow outbound VPN access you can block the ports numbers used for IPSec. A listing of the ports can be found on the VPNTools web site (http://www.vpntools.com/vpntools_articles/port-for-vpn.htm).

Section IV: Detecting Proxied Access To Your Websites

Many websites wish to control the regions where users can access their web resources. One example is the British Broadcasting Company or BBC, which tries to restrict usage of video content available through its iPlayer to British IP addresses only. Many people have figured out that if they can access a UK-based proxy server that supports proxying of the media and features they need to view the content, they can easily get around this limitation [9].

There are two ways where detection can occur. The first is when the proxy server acts as a client to request the data from your server on behalf of the user. The second is by "interrogating" clients or scanning back as they make requests. Unfortunately, both have severe disadvantages.

User-Agents

When a browser client connects to a web server, it identifies itself with a string called the User-Agent. For example, Firefox on my Mac produces the following User-Agent when I access websites

(as captured in Wireshark):

User-Agent: Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10.5; en-US; rv:1.9) Gecko/2008061004 Firefox/3.0

In order to detect access to a website through a proxy, the proxy would need to produce a unique "signature" User-Agent by altering the original. Using the website http://www.ioerror.us/ip/, which displays User-Agent values, I confirmed that directly accessing it with my Mac yielded the same results as obtained through the Wireshark capture. In trying proxy sites already examined in each of the categories within this paper, it's apparent that they all do a good job of passing the browser User-Agent through unaltered. Interestingly, when Google Translator was tried it did modify the User-Agent, but then its purpose is not to be an anonymous proxy. The agent that Google Translator provided is as follows:

User Agent: Mozilla/5.0 (Macintosh; D; Intel Mac OS X 10.5; en-US; rv: 1.9) Gecko/2008061004 Firefox/3.0, gzip (gfe) (via translate.google.com)

In short, the User-Agent isn't going to give away an anonymous proxy server.

Scanning Back

The other method is to scan back the address of each client that accesses your web site. To do this, you send out a website request, such as for www.google.com, to the address of the client connecting to you. If you get an answer with the data for www.google.com, then the site is likely a proxy that is being used to access your site. For many organizations performing active scanning is not acceptable. For others, the volumes of connection attempts they experience make this impractical. However, if you have the desire and the resources, this may be something for you to consider.

One free project is the Frost Jedi Open Proxy Detector, a PHP-based proxy detector, available at the Frost Jedi Group's website (http://frostjedi.com/). You can see an example of it at http://www.monster-submit.com. In my testing of it, I found that results were not always accurate from one scan to another, and often took a considerable amount of time. This isn't necessarily the fault of the detector; proxy sites can frequently be utilized to the point of exhausting their resources and not show up as a proxy simply because they do not respond in time. Additionally, some proxies may not

respond to requests from certain IP ranges, making it more difficult to test without a second system in a different IP range. One technique might be to initially allow the connection and farm off the proxy detection to another system. If the system determines the client is a proxy, a rule could be automatically added to a router or firewall to prevent access and hopefully disrupt the current connection, if it is still up. This is a risky proposition and best left to situations where preventing proxy access is crucial and the risk of false positives or abuse (attempts by malicious individuals to get legitimate sites blocked by finding a way to get the proxy detector to register a false negative) is acceptable. It should also be noted that this can help in reducing proxy access but will never be able to block all proxy access. As one example of why this is the case, a proxy could allow inbound connectivity to a non-standard port. Making a proxy detector scan all possible ports for all connecting hosts is not practical and could likely be viewed as a hostile action by legitimate networks that connect to you, possibly even getting your address range blocked. Since you can't scan every port value there is always the chance that the client connecting to you is a proxy but you won't detect it. As cool as scanning back may seem, so far the associated problems really keep it from being practical. Utilizing black lists currently appears to be the safest, least processing intensive method to use, although it will not catch all attempts.

Section V: Advanced Proxy Detection

In the pre-modern coal mining days, canaries were used to detect the build up of poisonous gases in the mine [10]. In detecting anonymous proxy usage, there are some things we can look for in network traffic that could be indicative of a proxy (and without having to sacrifice any small, yellow song birds!)

Myspace is once again an example of a site that users want to go to that administrators might want to block. (If some other prohibited site is more appropriate for your instance, use it here instead.) Knowing some of the popular obfuscation methods, we can create the Base64 and ROT13 representations of myspace (leaving off the "www." and the ".com") and alert or log on URLs that include literal string matches. In the case of Base64, every 3 characters of the original string are converted to 4 characters in the encoded string, so adding or changing characters at or near the

beginning of the string completely alters the results. In the case of rotational encoding, such as ROT13, there's a one for one substitution so we don't need to create multiple search values to catch the search string within a larger string.

Literal Searches

The following literal search values could be used to review the daily Internet access logs to detect obfuscated instances of MySpace or other prohibited sites. The point would not be to detect all anonymous proxy usage, but to pick a popular site you think users might try to access and search specifically for encodings of it to detect a new proxy. Using Base64 encoding sites as mentioned earlier, you may create the encoding for any sites that you want to check for. Examples are given below for MySpace.

Base64 of "myspace" is bXlzcGFjZQ==

Base64 of "www.myspace.com" is d3d3Lm15c3BhY2UuY29t

Base64 of "://www.myspace.com" is Oi8vd3d3Lm15c3BhY2UuY29tIA0K

ROT13 of "myspace" is zlfcnpr

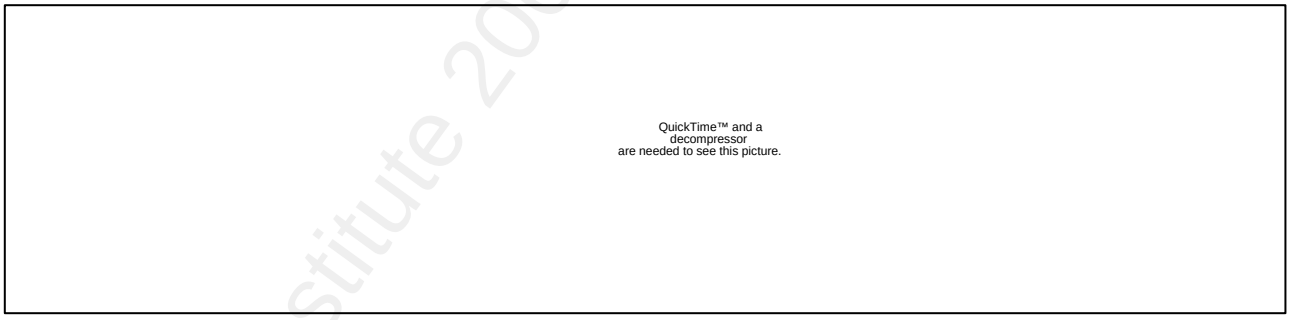
Web Log Base64 Searches

There are legitimate uses for Base64 encoding in URLs. However, since this is a predominant means of obfuscation for anonymous proxy services, it's worth reviewing logs for Base64 usage to detect anonymous proxy usage. There is no specific utility for detecting Base64 that I am aware of, so I set to the task of making one. Several problems had to be overcome. Base64 takes each character, translates it to it's ASCII value, stores the ASCII value in binary, 8-bit format as 1's and 0s, and then grabs 6 bit groups (since in binary 6 bits are needed to represent 64 possibilities) to represent the Base64 values. Because 6 bits and 8 bits take 4 bytes to "align" (6*4=24 and 8*3=24) Base64 values are always expressed in 4-byte groupings (three ASCII characters become four Base64 characters) so a Base64 value should always be divisible by four. If the original string doesn't take up the entire four bytes, the Base 64 value is supposed to be padded with the equal sign character, which represents a null value [11]. For more details as well as an online encoder and decoder visit

http://www.hcidata.info/base64.htm.

To detect Base64 usage, I wrote a Perl script, findbase64.pl, which is in Appendix B. The script takes as input a file containing a single URL per line. Once again, the CUT command (or AWK) can be used to extract the URL field from your web access logs. For example, Microsoft ISA server log files are tab delimited and hold the URL in the sixteenth field, so use the CUT trick to get the Tab character using CONTROL, V, and the Tab key or, better yet, use AWK: awk '{print $16}' weblog.txt | awk -F "/" '{print $3}'.

The script parses each URL, grabbing each section between the "/"s. It looks for characters that aren't valid Base64 characters (such as ., %, !, _,], or +) to exclude some entries. Next it checks the length to see if it's divisible by four. Finally, it decodes the string and checks to see if any of the ASCII values of the characters are not expected regular characters (values 33-128.) Code values below 32 are not printable characters and code values above 128 are extended characters, which would not be valid in a URL. Even after these tests, there are still strings that are not Base64 encoded but are divisible by four that will decode to strings that are normal ASCII characters. In fact, I was deeply disappointed in just how many there were. In a log file with about 1.5 million records, I was getting



tens of thousands of false positive hits. I observed the contents of these false decodes and found several combinations that weren't valid in URLs, such as "\^" and "~\", as well as several others. Having the program filter out decoded strings that contained those values greatly reduces the number of false positives. In fact, on the log file with 1.5 million records, I got 133 hits, only four of which were false positives. Most of the rest were legitimate sites that made use of Base64 encoding within the URL string. I'm happy to report that the first time I ran the script on a file containing real data it caught a user accessing MySpace through an anonymous proxy at roflrofl.com.

As with the other Perl script, output is to a tab-delimited file that can be easily loaded into a spreadsheet for review or sorting. A "finding number" is listed first (if more than one Base64 encoding is found in a single URL, each is listed in a new row with the same finding number so if you change the sorting in the spreadsheet you can tell which entries were from the same record) then the Base64 decoding, then the original field it was "decoded" from, and then finally the original URL. In the first few lines of the program is a variable called $Strict which defaults to "1". This means use strict Base64 checking and ignore strings that aren't divisible by four. However, I have found anonymous proxies that simply drop off the equal sign (or equivalent URL encoding.) This doesn't effect decoding it, but it makes the script miss it since the length is too short to conform to the standard. If you change the value of $Strict to 0 the script will not use the "divisible by four" check. As a result, you'll catch Base64 encodings that don't adhere to the standard, but you'll also get more false positives of strings that aren't Base64 but decode to valid Base64 values. One improvement would be to add additional checking within a single string, as I've also found sites where two or more encoded values are within one set of "/"s. Still, I've found it to work extremely well and even the non-proxy results are interesting. A warning is that some sites actually use Base64 encoding to carry userids and passwords and this script can reveal them. The script runs very fast on my Core 2 Duo laptop, running through a file with about 1.5 million records in roughly 30 seconds.

Conclusions

Although there is no single, easy way to detect anonymous proxy usage, I've reviewed a number of techniques that can detect or block many proxies. Blacklists are a good starting point, especially if you can automate the updating of them by utilizing proxy-advertising sites. Snort rules can give instant alerts or, for the advanced Snort users, may be used to block proxy access. Finally, GREP searches through web access logs and a couple of Perl programs to look for dynamic DNS usage and Base64 usage can help detect anonymous proxy usage. Combined, these tools and techniques can help you stay on top of anonymous proxy usage in your environment.

References

[1] Sahasrabudhe, Shailendra. Risks Posed By Anonymous Proxies. 17 March 2008. Accessed 1 May 2008 at <http://www.expresscomputeronline.com/20080317/technology02.shtml>

[2] Zoica, Remus. *New Threat That Can Be Used to Divert Web Traffic Through a Malicious Proxy Server*. 31 March 2007. Accessed 28 August 2008 at <http://www.securitysoftwarezone.com/new-threat-that-web-traffic-review350-add-comment.html>

[3] Tor project staff. *Tor: anonymity online*. Unknown. Accessed 30 May 2008 at <www.torproject.org>

[4] Tor project staff. *TorFAQ*. Accessed 30 May 2008 at <http://wiki.noreply.org/noreply/TheOnionRouter/TorFAQ#head-c7d58fb7afe0df2e76a50a288f3c6777d6a4adda>

[5] Bianco, David. *Detecting Tor on your network*. 25 January 2005. Accessed 30 May 2008 at <http://blog.vorant.com/2005/01/detecting-tor-on-your-network.html>

[6] Anonymous. Usage Statistics for phpproxy. 11 June 2008. Accessed on 11 June 2008 at <https://sourceforge.net/project/stats/?group_id=109342&ugn=phpproxy>

[7] Anonymous. About Glype. Accessed 31 May 2008 at <www.glype.com>

[8] Marshall, James. HTTP/FTP Proxy in a CGI Script. What it is, what it is. 2008. Accessed on 19 June 2008 at <http://www.jmarshall.com/tools/cgiproxy/, 6/19/2008>

[9] "Oldfogey". Question: *Proxy sites: I want the BBC site to think I am in the UK.?*. June 2008. Accessed on 8 July 2008 at <http://answers.yahoo.com/question/index?qid=20080515103028AApoBSI>

[10] Pollick, Michael. *What does It Mean to be a Canary in a Coal Mine?* Accessed 8 July 2008 at <http://www.wisegeek.com/what-does-it-mean-to-be-a-canary-in-a-coal-mine.htm>

[11] Anonymous. *How does Base 64 Encoding Work?* 12 September 2007. Accessed on 20 June 2008 at <http://www.hcidata.info/base64.htm>

Appendix A: findddns.pl

(Find Dynamic DNS usage and direct IP address usage within a web log. latest version at http://www.trueinsecurity.com/proxy.htm)

```
# ================================================================
# TITLE: findddns.pl
# AUTHOR: John Brozycki
# DATE  : 7/15/2008
# LAST MODIFIED: 7/27/2008
# PURPOSE: Logs domain mismatches between the domain used in the URL
#       and the domain returned from a reverse DNS lookup performed
#       on the IP address the original domain resolved to.  Only the
#       last two name sections, ie: domain.com, are compared to prevent
#       false positives for sites that have aliases (such as
#       www.bigsearchengine.com and www.l.bigsearchengine.com) which
#       are noted as a "Domain mismatch."  Domains that don't have a
#       reverse DNS record are noted as "Unresolved rDNS"
#       Sites accessed by IP address are noted as "IP Address Access."
#       Final output file DYNDNS.TXT can be imported into Excel and
#       sorted by these categories.
#       This will help detect: users accessing home systems via Dynamic
#       DNS services, direct access to IP addresses, as well as many
#       anonymous proxy services that are aliased to the same IP and
#       thus don't have a corresponding rDNS entry.
# DEPENDENCIES: Requires the underlying operating system to support
#        CUT, SORT, UNIQ, and HOST.
#
# INPUT/OUTPUT: Input is a text file named 'weblog.txt' which is formatted
#        as follows:
#         http://www.google.com
#         http://mydynamicdnssite.com/proxy
#         http://192.168.0.1/myhomeproxy.htm
#       An intermediary file, websites.txt, is created by the
#        external CUT command.
```

```
#          Final output is to a file named DYNDNS.TXT
# ================================================================
#


#
# Perl trim function to remove whitespace from the start and end of the string.
sub trim($);
#
sub trim($) # Trim function from http://www.somacon.com/p114.php
{
        my $string = shift;
        $string =~ s/^\s+//;
        $string =~ s/\s+$//;
        return $string;
}
# weblog.txt is the input file of websites accessed, with "website URL" as the
# only field present.  The CUT utility will be run externally to strip out
# the site domain from the full URL.
# The stripped site domain wil be sorted and then duplicates removed to minimize
# redundant lookups.
# NOTE: If your web log file contains additional fields, you can modify the CUT
# command to reduce it to a single field with the website URL.
#`cut -d "/" -f 3 weblog.txt | sort | uniq > websites.txt`; #External command
open (OUTFILE, ">dyndns.txt") or die "Cannot open output file: $!";
open (WEBFILE, "websites.txt") or die "Cannot open input file: $!";
$URLisIP = 0;
# Print a header file for the output file
print OUTFILE ("Category\tLogfile Domain\tRDNS Domain\tLogfile URL\tIP Address\n");
# Process all rows in log file
while (<WEBFILE>)
{
  $weburl = $_;   #Load $line with the line read in from the input file
  chomp($weburl); #Remove trailing newline character
  $weburl =~ s/\.$//; #Sanitize a trailing dot by removing it.
  # Clean off any trailing port assignment, like ":443"
```

```
 if ($weburl =~ m/.*:.*/)
   {
     $weburl = substr($weburl,0,index($weburl,":"));
   };
 $URLisIP = 0;
 # Determine is site is an IP address or name
 if ($weburl =~ /^\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}$/)
   {
     $URLisIP = 1;
   };
$origURL = $weburl; # Save a copy of the original URL read from the log file without an port info
if ($URLisIP != 1)
   {
     # Call the HOST -t A command to retrieve IP address.
     chomp($IPaddr = `host -t A -W 2 $weburl`);
     #Remove all newline chars if query reply had multiple lines
     $IPaddr =~ s/\n/ /g;
     if ($IPaddr =~ /.*(not found).*/i)
        {
         $IPextract = "Not Found";
        }
     else
        {
         # Extract the IP address from the HOST command output. It starts eight
         # characters after the beginning of the word "address" and is at most
         # 15 characters long (i.e. aaa.bbb.ccc.ddd).
         $IPextract = substr($IPaddr, index($IPaddr, "address ") + 8, 15);
         chomp($IPextract);
         $IPHasASpace = index($IPextract," ",5);
         if ($IPHasASpace != -1)
           {
             $IPextract = substr($IPextract,0,$IPHasASpace);
           };
        };
   }
```

```
else
   {
    $IPextract = $weburl;
   };
# Call the HOST -t A command against the IP address to get the RevDNS name.
chomp($Reversed = `host -t A -W 2 $IPextract`);
# If query response came back with multiple lines get rid of all but the first.
if (index($Reversed,"\n") != -1)
  {
   $Reversed = substr($Reversed,0,index($Reversed,"\n"));
  };
if ($URLisIP != 1)
  {
    # Remove trailing "." that HOST output creates if querying an IP, using CHOP.
    chop($Reversed);
  };
#Remove all newline chars if query reply had multiple lines
$Reversed =~ s/\n/ /g;
# Extract the reverse DNS from the command output
$Reverse = substr($Reversed, index($Reversed, "pointer ") + 8);
# Pull out the Top Level Domain by finding last "."
$TLD = substr($Reverse, rindex($Reverse,"."));
# Get domains for RevDNS lookup and original entry from file.
$DomName = substr($Reverse,rindex($Reverse,".",rindex($Reverse,$TLD)-1));
$TLDurl = substr($weburl, rindex($weburl,"."));
$DomNameurl = substr($weburl,rindex($weburl,".",rindex($weburl,$TLDurl)-1));
# If name is only two parts, ie: domain.com, use the original value read from
# the log file.
if (not $weburl =~ m/.+\..+\..+/)
   {
    # Domain has only two parts, ie: domain.com and we munged it so
    # set it to the original value from the log file and set the reverse
    # DNSed name to the original value, before stripping out the TLD.
    $DomNameurl = $origURL;
   };
```

```
 if (not $Reverse =~ m/.+\..+\..+/)
   {
    # Domain name retrieved from reverse DNS has only two parts, ie: domain.com
    # and we took too much out of it (stripping wasn't needed) so set $DomName
    # to the value before it was stripped, which was $Reverse.
    $DomName = $Reverse;
   };
# Use included TRIM function to remove any spaces.
 $DomName = trim($DomName);
 $DomNameurl = trim($DomNameurl);
 $NotFound = 0;
 $weburl = $origURL;
 $OutputMsg = "Unknown Error";
# If the names don't match, we want to write the entry to a log.
# "Unresolved Mismatch" means the reverse DNS didn't get an answer.
# "Domain Mismatch" means the domains didn't match up.  "IP access" means
# an IP address was used in the URL, not a name.  "No rDNS exists" means
# that a reverse lookup value doesn't exist for the record searched.
# Make the comparison all lowercase, as some misconfigured rDNS records
# actually differ by case, i.e. 2o7.net and 2O7.net (letter O not 0.)
  if (lc($DomName) ne lc($DomNameurl))
    {
     if (index($Reversed,"not found") != -1)
       {
         $DomName = "Not found";
         $NotFound = 1;
       };
     if ($NotFound == 1)
       {
        $OutputMsg = "Unresolved rDNS";
       }
      else
       {
        $OutputMsg = "Domain mismatch";
       };
```

```
    if (index($Reversed,"no PTR") != -1)
      {
        $DomName = "No record";
        $OutputMsg = "No rDNS record exists";
      };
    if (index($Reversed,"timed out") != -1)
      {
        $DomName = "timed out";
        $OutputMsg = "Timed out";
      };
# If original log entry was an IP address, the HOST lookup and
# string manipulation will yield invalid results, so set them
# here.
    if ($origURL =~ /^\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}(:[0-9]{1,5})?$/)
      {
       $DomNameurl = $origURL;
       if (index($DomName,"not found") != -1)
         {
          $DomName = ("Not found");
         };
       $IPextract = $origURL;
       $OutputMsg = "IP Address access";
       chop($Reverse);
       $DomName = $Reverse;
      };
    $DoNotPrint = 0;
    if (not $origURL =~ m/\./) # If no "dot" then Intranet or alias - don't bother printing
      {
       $DoNotPrint = 1;
      };
    if ($Reverse =~ /.*(Not found).*/i)
      {
       $DomName = "Not Found";
      };
    # Put sites you don't want to 'hit' and output on here by adding additional "elsif" statements.
```

```
    # For values that appear in "Log Domain File" column in the results use:
    #    elsif ($DomNameurl =~ m/.*VALUE.*/i)  { $DoNotPrint = 1}
    # For values that appear in "RDNS Domain" column in the results use:
    #    elsif ($DomName =~ m/.*VALUE.*/i)  { $DoNotPrint = 1}
    if    ($DomName =~ m/.*akamai.*/i)        { $DoNotPrint = 1 } # akamai.net or akamaitechnologies.com
    elsif ($DomName =~ m/^.*aka\..*/i)        { $DoNotPrint = 1} # aka.{dest_domain}
    elsif ($DomNameurl =~ m/^.*akamai.*/i)    { $DoNotPrint = 1} # aka.{dest_domain}
    elsif ($DomName =~ m/.*\.google\.com.*/i)  { $DoNotPrint = 1} # Google site
    $OutputMsg = $OutputMsg."\t".$DomNameurl."\t".$DomName."\t".$origURL."\t".$IPextract."\n";
    if ($DoNotPrint == 0)
      {
       print OUTFILE ($OutputMsg);
       print ($OutputMsg."\n");  #Comment out this line if you don't want output to screen, too
      };
   };
};
close WEBFILE;
close OUTFILE;
```

Appendix B: findbase64.pl

(Find and decode Base64 usage within a web log. latest version at http://www.trueinsecurity.com/proxy.htm)

```
# ====================================================================
# TITLE: findbase64.pl
# AUTHOR: John Brozycki johnb@trueinsecurity.com
# DATE  : 7/15/2008
# LAST MODIFIED: 7/31/2008
# PURPOSE: Logs occurrences of Base64 encoding within a URL string.
#       Each URL that contains a Base64 encoded value is given a
#       sequential finding number, and each encoded value is decoded
#       and output.  Output is as follows:
#       Finding#, Decoded value, Found in, original URL.
#       Values are Tab delineated and a header row is provided so
#       load up the output file in your favorite spreadsheet app.
#       A caveat with Base64 checking is that some anonymous
#       proxies drop the padding instead of URL encoding it. The
#       result is that it still decodes correctly, but it's not
#       "proper" Base64.  I set up a strict variable, $Strict, so
#       it can be set to '0' if you want strict Base64 checking or
#       '1' if you want loose checking.  Loose checking will result
#       in more false positives being output.
# DEPENDENCIES: Requires the MIME::BASE64 module, which is built in to
#        most modern versions.  If your Perl implementation
#        doesn't already support this, go to search.cpan.org to get it.
# INPUT/OUTPUT: Input is a text file named 'weblog.txt' which is formatted
#        as follows:
#         http://www.google.com
#         http://somegenericproxy/proxy/Oi8vd3d3Lm15c3BhY2UuY29
#         http://192.168.0.1/myhomeproxy.htm
#
#      Final output is to a file named BASE64CHECK.TXT
# ====================================================================
```

```
#
#
# Use the MIME::Base64 module to decrypt Base64 obfuscated text.
#use MIME::Base64;
use MIME::Base64 ();
#
open (OUTFILE, ">base64check.txt") or die "Cannot open output file: $!";
open (CHECKFILE, "<weblog.txt") or die "Cannot open input file: $!";
# Print a header file for the output file
print OUTFILE ("Finding\tBase64\tFound In\tURL\n");
$FindingNumber = 0;
$LineCt = 0;
$Matches = 0;
$Strict = 1; # Set to 0 for loose Base64 checking
$previousurl = "";
# Process all rows in log file
while (<CHECKFILE>)
{
 $weburl = $_;   #Load $line with the line read in from the input file
 chomp($weburl); #Remove trailing newline character
 $CurrentPos = 0;
 $Length = length($weburl);
 $LineCt = $LineCt + 1;
 # Look for initial "//" and set current position to two characters beyond it
 $DoubleSlash = index($weburl,"//");
 if ($DoubleSlash ge 0) # Found
   {
    $CurrentPos = $DoubleSlash + 2;
   };
#
# Main loop through the URL
#
 while ($CurrentPos < $Length)
 {
  $Slash = index($weburl,"/",$CurrentPos);
```

```
#$CheckForB64 = 0;
if ($Slash == -1)
  {
   # no more slashes found
   $TempElement = substr($weburl,$CurrentPos);
   $CurrentPos = $Length;
  }
else
  {
   # process the next URL element section
   $ElementLength = $Slash - $CurrentPos;
   $TempElement = substr($weburl,$CurrentPos,$ElementLength);
   $CurrentPos = $Slash + 1;
  };
$ElementLength = length($TempElement);
if ($ElementLength >= 4) # Base64 needs to be at least 4 chars
  {
  $EqualPos = index($TempElement,"=");
  # If element contains a var like "?var=" then strip it out.
  if (($TempElement =~ m/.*\?*=/) && ($EqualPos < $ElementLength - 4))
    {
     #Take the string after the = character.
     $TempElement = substr($TempElement,index($TempElement,"=") + 1);
     $ElementLength = length($TempElement);
    };
  # If Base64 encoding ended with an "=" sign at the end, it will likely be "URL" encoded
  # as the "=" sign has meaning within a URL.  The value it has is "%3D".
  if ($TempElement =~ m/(%3D)+/)
    {
     $TempElement =~ s/%3D/=/;
    };
  # Clean off any trailing variables like "&b=52"
  if ($TempElement =~ m/.*&.*/)
    {
     $TempElement = substr($TempElement,0,index($TempElement,"&"));
```

```
      $ElementLength = length($TempElement);
    };
  # Base64 is always on a 4 byte boundary (padded with "=" if needed.)  If we get a 0 from Length MOD 4
  # then we know the length is right for a Base64 string.
  $DivByFour = $ElementLength%4;
  # It's not Base64 if it has a: .%!_ in it.  Also, Base64 is in 4 byte blocks so if
  # the string is not divisible by 4, it's not Base64.  A caveat is that some anonymous
  # proxies drop the padding instead of URL encoding it. The result is that it still decodes
  # correctly, but it's not "proper" Base64.  I set up a strict variable so it can be set to
  # '0' if you want strict Base64 checking or '1' if you want loose checking.  Loose checking
  # will result in more false positives output.
  if ( (not $TempElement =~ m/[.%!_]+/) && (($DivByFour == 0) || ($Strict == 0)) )
    {
#  print "got here\n";
      $DecodedValue = MIME::Base64::decode($TempElement);
      # If we've decoded a non-Base64 value, the ASCII value will probably not be
      # one of the first 128 regular characters and give a value > 128.  This extra
      # check will prevent some false positives.
      $Len = length($DecodedValue);
      $PosInString = 0;
      $NotB64 = 0;
   #    print "PosInString: ".$PosInString." Length: ".$Len."\n";
      while ($PosInString < $Len)
        {
          $CurChar = substr($DecodedValue,$PosInString,1);
          # Get current character's ASCII value
          $ASCIIval = ord($CurChar);
          # We don't want non printable, space, or extended ASCII codes
          if (($ASCIIval > 128) || ($ASCIIval < 33))
            {
              $NotB64 = 1;
            };
          $PosInString = $PosInString + 1;
        };
      # There are several invalid character combinations that can result if decoding a string
```

```
        # that wasn't really Base64 encoded.  Look for them here and, if found, flag as notBase64.
        # These include "+-", "'^", "~)", "+^", ”~*”, ”*^”
        if ($DecodedValue =~ m/.*((\+-)|(\'\^)|(~\))|(\+\^)|({)|(~\*))|(\*\^).*/)
          {
            $NotB64 = 1;
          };

        # If not detected to be notBase64 and the string is atleast 4 characters...
        if ((not $NotB64 == 1) && ($PosInString >= 4))
          {
           #     print "Str: ".$TempElement." Decode: ".$DecodedValue." Ordinal: ".$CHRChar."\n";
            if ($weburl ne $previousurl)
              {
                $FindingNumber +=1;
                $previousurl = $weburl;
              };
            print ($FindingNumber."\t".$DecodedValue."\t".$TempElement."\t".$weburl."\n");
            $Matches = $Matches + 1;
            print OUTFILE ($FindingNumber."\t".$DecodedValue."\t".$TempElement."\t".$weburl."\n");
          };
      }; # end TempElement
    }; # end Element Length
 }; #end Current pos
}; # EOF
print $Matches." matches on ".$FindingNumber." URLs out of ".$LineCt." rows processed.\n";
close CHECKFILE;
close OUTFILE;
```