A Robust Link-Translating Proxy Server Mirroring the Whole Web

Ziqing Mao
Purdue University
West Lafayette, IN
zmao@purdue.edu

Cormac Herley
Microsoft Research
Redmond, WA, 98052
cormac@microsoft.com

Abstract

Link-translating proxies are widely used for anonymous browsing, policy circumvention and WebVPN functions. These are implemented by encoding the destination URL in the path of the URL submitted to the proxy and rewriting all links before returning the document to the client. Commonly these are based on the CGIProxy implementation or a variant. While popular, broken links and very slow load times are common. This is so, since the use of scripting languages makes finding and translating links (the essential task of such a proxy) very difficult. Some web-sites become entirely non-functional when loaded.

This paper proposes a novel architecture for a link-translating proxy. Using a sub-domain mapping technique we entirely eliminate the need to translate (or even find) relative links in content. Then, using robust absolute link rewriting, cookie re-assignment, and wildcard certificates we achieve an extremely robust and performant proxy. Our architecture preserves the same-origin policy separation of sites, and thus entirely eliminates cookie-stealing and other security concerns of CGIProxy. In measurements our proxy is far more robust, loads pages more quickly, and is more scalable than CGIProxy. We call our system ABOProxy, since it involves the Address Bar Only. It is invoked by appending `.aboproxy.com` *to the hostname of any URL.*

1. Introduction

Proxy servers perform a critical rôle on the web. They provide caching, acceleration, content filtering and isolation functions. Most of this is invisible to users who are unaware that proxies intervene between them and the end server. Commonly there is a web proxy that the browser uses for outbound web traffic; this is set and controlled by the ISP or home network, and is used for caching, policy enforcement, auditing and so on. This is often known as a *forward proxy*. In contrast, a *reverse proxy* generally sits close to a server and dispatches in-bound web traffic to a set of backend servers, presenting a single interface to the clients [18]. It is generally used to pass requests from the Internet, through a firewall to an isolated, private network, preventing Internet clients having direct, un-monitored access to the sensitive servers. Besides forward and reverse proxies, sometimes an *intermediate proxy* sits between them. This is typically provided by a third-party in the Internet; (sometimes this is unintentional where a mis-configured server acts as an *open relay* [1, 2]). In each of these three cases the proxy intercepts the traffic at the HTTP layer or lower. Forward proxy settings are often auto-detected by the browser (*e.g.* using the Web Auto Detect Proxy (WPAD) service in Windows [22]). When using a reverse proxy the backend servers are configured to accept and send all traffic through the reverse proxy. An intermediate proxy is often reached by changing the proxy settings on the browser manually. Figure 1 summarizes the different kinds of proxy servers.

The proxy architecture discussed in this paper differs from these three kinds of proxy. We call it a *web-service based or link-translating proxy*, because the proxy server is running and is reachable as a standard web-service. In terms of network location, the link-translating proxy is similar to an intermediate proxy. However, its main advantage is that no alteration of browser or other settings is required. To use the proxy, the user submits an initial desired URL in a form or in the address bar *without changing proxy settings*. The proxy returns the webpage of the destination URL and all communication for the subsequent clicks and actions performed in the webpage will also go through the proxy. Such proxies are widely used. The most popular link-translating proxy is an open-source implementation called CGIProxy, which is designed to be an anonymous proxy. However, broken links and slow load times are

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SAC'10 March 22-26, 2010, Sierre, Switzerland.
Copyright 2010 ACM 978-1-60558-638-0/10/03 ...$5.00.

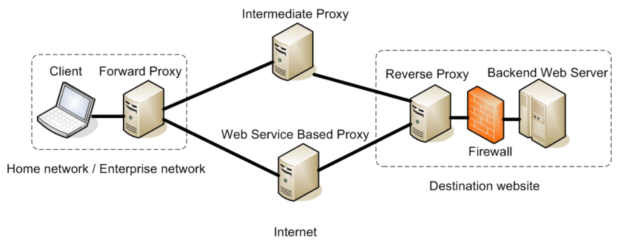


Figure 1. Forward proxy, reverse proxy, intermediate proxy and web-service based (link-translating) proxy

common with CGIProxy.

In this paper, we propose a novel web-service based link-translating proxy. We call it *Address-Bar-Only Proxy (ABO-Proxy)*, because the user simply uses the address-bar to direct traffic through the proxy. We borrow an elegant idea first used in the design of the Coral Content Delivery Network [19] and adapt it to the problem addressed by CGIProxy [3]. Our design outperforms the CGIProxy in design simplicity, robustness, security, load time and scalability. To use ABOProxy, the user simply appends `.aboproxy.com` to the end of the URL hostname. For example, to visit `www.google.com` through the proxy, the user navigates to `www.google.com.aboproxy.com`. All the communication for the subsequent clicks and actions performed in the webpage will also go through the proxy (whether fetched from the same domain or other domains). The proxy is currently deployed and readers are encouraged to test. We describe our method in Section 3. The implementation and evaluation are presented in Section 4 and Section 5, respectively.

2. Related Work

An excellent text on web proxies is [18].

Coral. An important building block of our system is borrowed from the design of the Coral [19]. Coral is a peer-to-peer distribution network that enables web developers to offer high performance and meet huge demand without large infrastructure investment. To reach content via the network a user appends `.nyud.net:8090` to an address. The request is then directed to a nearby Coral cache. Relative links within the response will automatically point back to Coral (just as with our subdomain mapping described in Section 3.2.1). However Coral makes no effort to find or translate absolute links. Thus absolute links, and content that Coral decides not to cache will be fetched from the origin server. While we borrow a very important design element from Coral our goal is very different. While Coral seeks to improve the performance in loading content for relative links, we seek to provide a robust proxy for all links, and entire sessions, including SSL sessions. Thus the major differences between our proxy and Coral are as follows. We translate all links absolute as well as relative, whether statically or dynamically encoded. Since we point at a single proxy rather than many caches our DNS configuration is simpler. We preserve session information as represented by cookies. Since, in Coral, relative links are fetched from `.nyud.net`, and absolute ones from the origin server, a cookie set in one will not be available to the other *even though they belong to the same domain*. Thus an artificial security boundary is constructed between content from the same origin site, which can break the experience. We handle both SSL and regular content. Thus the user applications that our proxy handles that Coral cannot include secure browsing, anonymous browsing, logging in to websites and any browsing that relies on cookies and dynamic content.

CGIProxy. The most popular general link-translating proxy implementation is CGIProxy [3]. This a generic link-translating proxy with implementations available for most major platforms. It's primary uses seem to be to provide anonymous web-browsing and to circumvent network policies (*e.g.* surf to forbidden sites). Many of the popular circumventors are based on instances of this code (see *e.g.* [4]). There is an enormous number of instances of CGIProxy running at any given time. For example [5] lists 5200 of what it claims are more than 26000 instances running. The major differences between our proxy and CGIProxy are as follows. By using sub-domain mapping relative links resolve automatically. We experience far fewer broken links, load times are far faster, and the proxy is more scalable. We eliminate several security concerns such as cookie stealing and cross-site scripting (see Section 5.3).

The Tor network [21, 6] also offers a means for Internet users to conceal their browsing patterns even from traffic analysis.

Open Relays. An open relay is a proxy that will accept traffic from any client and relay it to any server. Often this occurs when an administrator mis-configures a machine. For example many firewall products, such as ISA [7] or UserGate [8] can be configured to proxy from anywhere to anywhere (though this is seldom what a system administrator would intend). There appears to be enormous demand for open relays. There are several sites [1, 2] that are dedicated to providing up to date lists of IP addresses and port numbers where open relays can be found. Thus users can manually configure their browser to use one of these open relays as proxy and have all their traffic appear to come from the relay rather than their own IP address. For example [2] lists what it claims are about 2000 new open relays per day.

Web VPN. Virtual Private Networks (VPNs) allow for secure communication to internal resources from external locations. Traditional VPNs require software to be installed on the client machine, while WebVPN is a VPN connection that is established using only the web browser.

Many universities uses WebVPN to enable the students to access restricted websites outside the campus. The WebVPN is achieved using a link-translating proxy which runs a web-service that fetches the internal webpages for the clients. However, dynamic web content does not work well when being accessed through a WebVPN. For example, on the CMU WebVPN website, users are warned that 11 databases cannot be accessed with WebVPN. In addition, there are known cross-site scripting vulnerabilities [9] with the popular WebVPN products from Cisco. The details of the security problem is discussed in Section 5.3. Using the ABOProxy for WebVPN can robustly serve any web content, and eliminates the security problem.

Debugging Proxies. There are a number of lightweight proxy applications that are suitable for use by a single or small number of users. Generally these are not intended to scale to supporting a large number of users. For example Charles [10] and Fiddler [11] are debugging proxies mainly intended to enable developers to examine the request/responses stream that flows between the browser and web-site. They provide powerful tools to examine, modify and replay headers, content and cookies in great detail. Both provide interfaces to allow modification of the request/response stream, and handle SSL as well as regular HTTP traffic. It is thus possible using either of these proxies to customize the browsing experience. Burp [12] and Paros [13] provide similar functionality.

Personal proxy. The personal proxy has been widely used for debugging, form-filling, content-filtering, archiving the browsing history, ad-removal and so on [14]. There are a great number of open-source and product personal proxies. All of them require the user to manually change the browser's proxy settings in order to use the proxy from other machines. Using the ABOProxy eliminates this requirement. An advantage of ABOProxy is that a user might get all of the advantages of a personal proxy *from any location*. That is, even when alteration of the browser proxy settings is not possible or convenient the functionality would still be reachable. Various filtering functions suitable for personal proxies are suggested by the Internet Content Adaptation Forum [15].

Other Uses of Proxies Gabber *et al.* [17] describe a proxy system, known as the Lucent Personalized Web Assistant, which acts as a credential intermediary and allows anonymous access to various web-services. The user's browser is configured to use the proxy, which might reside on their own machine or in the cloud.

Provos and Holz [20] describe a creative use of a proxy which monitors the loading of blacklisted sites. The idea is that vulnerable webservers can be made to serve content that loads from malicious sites. SpyBye [20] acts as a proxy and monitors all of the requests originating from the browser. By maintaining a blacklist and a set of URL classification rules it can determine when fetches are sent to malicious or unapproved servers.

3. Method

3.1 Background and Existing Approaches

The most popular and reliable link-translating proxy implementation currently available is CGIProxy [3], which is a server-side script in Perl and has been developed and actively updated since 1998. The general idea of CGIProxy is to *encode the destination URL in the path*. The same idea is used in existing implementations of WebVPN (*e.g.* from Cisco). The idea can be illustrated with an example. Suppose the CGIProxy's script is running at the address `http://cgiproxy.com/nph-proxy.cgi`. If the user wants to visit `http://www.google.com` through the CGIProxy, then the URL shown in the browser's address-bar is `http://cgiproxy.com/nph-proxy.cgi/http/www.google.com`. In this way, when the perl script receives the HTTP request, it is able to decode the destination URL from the path of the submitted URL. More specifically, the script does the following upon receiving each request.

1. Decode the destination URL from the URL path.
2. Retrieve the content from the destination website.
3. Modify the response by rewriting all links, pointing them back to the proxy script.
4. Send the modified response to the user.

The key operation of the proxy script is the link rewriting in Step 3. The purpose of which is to point all links contained in the server's response back to the proxy, so that when those links are triggered in the user's browser, the requests will be also routed through the proxy. For example, in Google's homepage, there is a logo image file embed by the following tag ⟨img src="/intl/en_ALL/images/logo.gif"⟩. The proxy script has to modify the link of the logo image to be `/nph-proxy.cgi/http/www.google.com/intl/en_ALL/images/logo.gif`. The design is conceptually simple and clear. However, it is actually very difficult to perform link rewriting robustly, as explained next.

3.1.1 Understanding Links

A link can be either absolute or relative. An absolute link defines the location absolutely including the protocol, the host-name, and the path of the resource. For example, `http://www.cnn.com/linkto/ticker.html` is an absolute link. A relative link takes advantage of the fact that the browser knows the location of the current document. Thus, if we want to link to another document from the same host as the current document, we just need to specify the path of the file. In the previous example, the link to Google logo image (`/intl/en_ALL/images/logo.gif`) is a relative link. The protocol and the host-name of the link are the same as the current document located at `http://www.google.com`. A link in a HTML document can also be either statically encoded or dynamically generated. The link to the Google image logo in the previous example is a statically encoded link. With the wide use of client-side scripting and AJAX technology, many links are dynamically generated by client-side scripts in the browser.

The difficulty of link rewriting mainly comes from the difficulty of finding all links contained in the document. Once identified, modifying links becomes a trivial task, but the complexity of finding them varies according to the type. Links that are statically encoded can be easily identified by parsing the HTML document, because they can only appear as certain attributes of certain HTML tags, such as ⟨img⟩, ⟨a⟩, and ⟨frame⟩. However, the links that are dynamically generated are difficult to identify, because scripting languages are very flexible and powerful. For example, JavaScript supports all structured programming syntax in C and many high-level programming language features such as objects, run-time evaluation and so on. One possible way to robustly identify all dynamically generated links would be for the proxy to contain a complete JavaScript engine and render the page just as a browser would do. While conceptually simple this is infeasible. Rendering the page is expensive computationally: the overhead would be too large to make it scalable.

Hence, CGIProxy and WebVPNs handle dynamically generated links in an ad-hoc way. Instead of implementing a complete JavaScript engine, the CGIProxy uses heuristics to identify the constants and variables that possibly store links, and modify those links by invoking a function. For example, the function call window.open opens a webpage in a new or an existing window. The first argument of the function call specifies the URL, *e.g.*, u="next.html"; window.open(u). The CGIProxy identifies the variable u stores a URL and modifies the JavaScript to u="next.html"; window.open(proxify(u)), where proxify is a function provided by the CGIProxy to rewrite an individual link. Consider the example where the website has the following JavaScript: u="next.html"; w=document.getElementById("sub_frame"); w.contentWindow.open(u);. In order to capture the link stored in u, the proxy has to identify the DOM element with ID sub_frame has an attribute named contentWindow that implements the interface window. This can not be done without a complete HTML and JavaScript engine. This example merely shows the difficulty of finding dynamically generated links, and explains why CGIProxy often breaks; there certainly exist many other cases making the problem very complicated.

3.2 A Novel Approach

3.2.1 Dynamic Links: Subdomain-Mapping

A key part of our approach comes from Coral [19]: map each destination domain to a sub-domain of the proxy server by appending the proxy domain to the destination hostname. For example, the proxy server is running at `aboproxy.com`, so the domain `www.google.com` is mapped to `www.google.com.aboproxy.com`. By navigating to `www.google.com.aboproxy.com`, the user visits the Google homepage through the proxy. Similarly, the logo image would be retrieved from `http://www.google.com.aboproxy.com/intl/en_ALL/images/logo.gif`. We call this technique *subdomain-mapping*, which simply appends `.aboproxy.com` to the end of the original URL hostname.

3.2.2 DNS Configuration

In the DNS account for the proxy domain, we use a wildcard DNS record to map the domain `*.aboproxy.com` to the IP address of the server machine. As a result, all requests for a sub-domain within `aboproxy.com` will be delivered to the server machine of the proxy. Note that we have a far simpler DNS configuration than Coral. Coral seeks to find one of many caches in the CDN that will give best performance to the client, while we point all requests at a single proxy.

Our proxy runs a script. On receiving each request, the script performs the following operations.

1. Decode the destination URL by removing `.aboproxy.com` from the request URL.
2. Retrieve the contents from the destination website.
3. Modify all *absolute* links in the response, by appending `.aboproxy.com` to the URL hostname, pointing them back to the proxy.
4. Send the modified response to the user.

Again, the key step for the proxy script is to do the link rewriting in server's responses. Observe that only absolute links need be translated. This is a main point of contrast with other web-service proxies. The path component of the URL remains unchanged after the subdomain-mapping. Therefore, all relative links, whether statically or dynamically generated, do not need modification.

3.2.3 Absolute Links: Robust Rewriting

The subdomain-mapping also makes rewriting absolute links easy. We merely append `.aboproxy.com` to the end of the URL hostname. As discussed before, the real difficulty of link rewriting comes from finding links that are dynamically generated by client-side scripts. We have eliminated the need to modify relative links. For absolute links, instead of trying to find all variables and constants that store links, we just need to find all positions that are the end of hostnames. We use pattern matching to do that.

Recall that hostnames end with a fixed set of tails, such as `.com`, `.edu`, and so on. They are called *top-level domains*, including 33 generic top-level domains and 253 country-code top-level domains. A complete list of top-level domains is available officially at `http://www.iana.org/domains/root/db/`. With the set of all top-level domains, by searching those patterns in the server's response, we can locate a set of candidate positions that may be the end of hostnames. However, certainly not all of them are really the end of hostnames. We need to do a further selection by examining the characters follows the appearances of top-level domain patterns. There are four cases that follow a hostname.

1. The hostname is followed by the port number, connected by the character '`:`'. For example, `https://www.vanguard.com:443`.
2. The hostname is followed by the path, connected by the character '`/`'. For example, `http://www.google.com/services/`.
3. The hostname is followed by an encoded colon `#58`, or an encoded slash `#47`.
4. The hostname is the end of a URL, or a string constant, which are identified by the characters '`'`' and '`"`'. The string constant may be further used in constructing URLs.

Therefore, only when the character immediately follows the appearance of top-level domain pattern is one of the four characters (in plain or encoded form): `: / ' "` , is it identified as the end of a hostname. Table 1 gives some examples on when to and when not to identify `.com` as the end of a hostname. Last, we append `.aboproxy.com` at all the positions that are identified as the end of hostnames in the server's response.

Limitations There are two limitations of the link rewriting algorithm described above. First, it is certainly possible for a web-site to break the algorithm (*i.e.* conceal links from the proxy script) intentionally. There exist two ways, (1) hide the top-level-domain pattern; (2) add a character other than those `: / ' "` to the end of a hostname. The examples include:

```
u1="http://www.example." + "c" + "o" + "m";
u2="http://www.example.cpp"; u2=u2.replace(/pp/, "om");
u3="http://www.eg.comp"; u3=u3.substr(0, u3.length-1);
```

However, these techniques are adversarial rather than normal: they are essentially unknown in benign real-world web applications (see Section 5.1). However, this does imply that ABOProxy will not necessarily function well in adversarial settings (a good example of a proxy in an adversarial setting is SpyBye [20]).

Second, the algorithm may append `.aboproxy.com` to a hostname that will be displayed in the webpage. For example, the algorithm will append `.aboproxy.com` in the following JavaScript, document.write("Thank you for visiting example.com"). We don't try to distinguish between a hostname in a link and a hostname displayed in a webpage, because it would again require a complete JavaScript and HTML engine. The only effect is that the user will see a superfluous `.aboproxy.com` on the webpage and it doesn't affect the functionality.

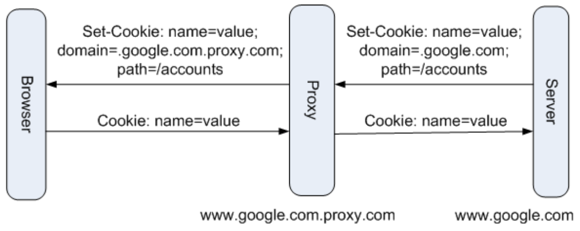

Browser
Proxy
Server
Set-Cookie: name=value; domain=.google.com.proxy.com; path=/accounts
Set-Cookie: name=value; domain=.google.com; path=/accounts
Cookie: name=value
Cookie: name=value
www.google.com.proxy.com
www.google.com

Figure 2. Modifying cookie attributes

Since they do not end with generic or country code top-level domains, our implementation makes no attempt to translate numeric IP address links. This has minor impact since such links are now rare. A user who loads a page from the Google search cache (which loads from numeric addresses) will go directly to the cache rather than the proxy.

3.2.4 *HTTP Cookies*

HTTP Cookies are pieces of text data sent by a web server to a browser to maintain state. Each cookie is a name/value pair with four optional fields: expires, domain, path, and secure. The domain and path fields indicate with which HTTP requests the cookie should be sent back. For example, if the user requests the URL `http://www.bank.com/credit/index.html`, then a cookie with domain=`.bank.com` and path=`/credit` would be attached to the request, but a cookie with domain=`script.bank.com` would not. With the introduction of the proxy, from the browser's point of view, the cookies are set by the proxy server returned to it. Therefore, the proxy needs to modify the attribute values. The *subdomain-mapping* technique makes the modification very simple. We append `.aboproxy.com` to the end of the original value of the attribute domain. In this way, as shown in Figure 2, the browser will send back the appropriate set of cookies for each request. We don't need to modify the value of the attribute path because the path components in the URLs remain unchanged.

4. Implementation and Applications

We have implemented and deployed a prototype of ABOProxy. We report the experiences in the implementation and deployment and discuss interesting design issues.

4.1 Domain Registration and DNS Setup

We acquired the second-level domain `aboproxy.com` and custom DNS hosting service from DynDNS.com. In the DNS settings, we added a wildcard host record that maps `*.aboproxy.com` to the IP address of our server machine (this means that all requests ending in `.aboproxy.com` will be referred to that server). The total cost for the domain registration and DNS service is $42.5 per year. Actually this can even be done without cost. Among others,

Examples	The end of a hostname?	Explanation
src='http://www.example**.com**'	Yes	A statically encoded link
window.open('http://www.example**.com/**index.html')	Yes	A dynamically generated link
sUrl = "http://collect.myspace**.com/**index.cfm?MyToken=" + myToken;	Yes	A dynamically generated link
var Hna="mail.google**.com**", Ina="https", Jna="?logout&hl=en"; ... if (b.xk==Hna) {b.Dt(Ina);a=b[o];return a+Jna}	Yes	Use JavaScript to compute links (from GMail)
Sys**.com**ponent=function() {...}	No	A method starting with ".com"
u="http://www.sina**.com**.cn"	No	A second-level-domain

Table 1. Examples on when to and when not to append ".aboproxy.com".

DynDNS.com provides free dynamic DNS service. Instead of a paid-for second-level domain, a free third-level hostname can be obtained, e.g., `aboproxy.dontexist.com`.

4.2 Web Server Setup

Our implementation is based on ASP.NET 2.0 and Microsoft IIS 6.0 running on Windows 2003 SP2. We implemented the proxy script as an HttpHandler in C#, which acts as a target for all incoming HTTP requests. The script is about 3000 lines of C# code. The proxy script can be implemented using other server-side scripting languages, such as Perl and Python. Besides IIS, the server can use other web server applications that supports server-side scripting. For example, in the Apache web server, one can use the directive SetHandler to make all requests to be delivered to the scripting handler cgi-script, which will invoke the corresponding script interpreter.

4.3 SSL and Certificates

ABOProxy handles SSL as well as regular traffic. If the user visits a HTTPS website through the proxy, e.g., `https://www.paypal.com.aboproxy.com`, the proxy will maintain one SSL connection with the client browser and another SSL connection with the end server. The client browser sees a certificate from the proxy rather than the origin server. To proxy any HTTPS website, we use a wildcard certificate for `*.aboproxy.com`. However, the handling wildcard certificates (as governed by RFC 2818) is browser dependent. All versions of Firefox and Opera that we tested (*i.e.* Firefox 1.0 and 2.0 and Opera 9.25) allow the wildcard to match multiple fields in the hostname. So, the certificate for `*.aboproxy.com` matches `mail.google.com.aboproxy.com`, `www.paypal.com.aboproxy.com`, etc. However, IE, Safari only allow using the wildcard to match the leftmost field in the hostname. In this way, the wildcard certificate for `*.aboproxy.com` would match `a.aboproxy.com`, or `b.aboproxy.com`, etc., but would not match `a.b.aboproxy.com`. In addition, they do not accept certificates with multiple wildcards, such as `*.*.aboproxy.com`. In conclusion, as currently implemented, users can visit any HTTPS website through the proxy without certificate warnings in Firefox and Opera. But with IE, Safari and Chrome, there will be one warning per new certificate encountered. We point out that these occasional certificate warnings are specific to our prototype implementation and can be rectified. This can be done by obtaining a separate certificate for each domain to be reached. Alternatively, if the user accepts the proxy as a Trusted CA all subsequent browsing via the proxy is warning-free.

4.4 Applications and Extensions

Anonymous Proxy. We extended the ABOProxy implementation as an anonymous proxy. Besides normal anonymization steps, such as removing third-party cookies and web bugs, the key step is to encrypt the communication between the client and the proxy server using HTTPS *even when loading non-SSL sites*. In ABOProxy, the protocol between the client and the proxy is the same as that between the proxy and the website. To work around this restriction, we extend the link rewriting of ABOProxy by modifying all appearances of `http://` in the server's response to `https://non-secure.`. In this way, all communication between the client and the proxy is over HTTPS, and the proxy is able to tell whether the original URL is using HTTP or HTTPS by examining the first filed of the hostname. If the first field of the hostname is `non-secure`, the proxy changes the protocol to HTTP and remove the first filed of the hostname. For example to navigate to `www.google.com` the user would load `https://non-secure.www.google.com.aboproxy.com`. Traffic to the proxy is encrypted, even though the end server does not support SSL.

WebVPN. We built a WebVPN service using ABOProxy to enable access to an internal enterprise network from outside locations. The enterprise network uses Microsoft Active Directory Domain Services. Each internal website is an active directory, such as `team`, `paystub`, `hrweb`, *etc.*. The VPN server machine connects to the internal network, and at the same time has an external IP address that is accessible from the Internet. The communication between the clients and the VPN server is using HTTPS. For example, the active directory `hrweb` can be accessed using the WebVPN at `https://hrweb.aboproxy.com`. A single wildcard SSL certificate for `*.aboproxy.com` is enough to cover arbitrary active directories. The VPN server rejects the requests targeting a website that does not exist in the

Websites	Failed operations
Gmail	Send an email; View/edit settings/contacts/labels
Microsoft Exchange Email	Every link broke after login
Facebook	Search friends; Add a friend; Confirm a friend request; Join a group; Send a message
Hotmail	Manage calendar; Manage contacts; Write a new message; Manage folders; Add a blog entry
Yahoo	Check emails; After login, when performing personal operations, it asked for a login again.

Table 2. Popular websites failed in the CGIProxy implementation.

internal network.

One-time-password System. We have integrated the ABOProxy implementation with the one-time-password (OTP) system proposed in [16]. The service enables the user to safely log into sensitive websites from public computers, and is running at `www.urrsa.com`.

5. Evaluation and Status

5.1 Efficacy

We tested the ABOProxy implementation by using it for everyday browsing. In four months' testing with two users (*i.e.* the authors) the ABOProxy provided the same experience as that without using the proxy. We tested using a variety of browsers and from locations in the US, Canada, Brazil, Taiwan, Ireland, Peru, Ecuador and Australia. The proxy is currently running and to test readers need merely append `.aboproxy.com` to any URL hostname, so that `www.google.com` becomes `www.google.com.aboproxy.com` *etc.*.

We observed only one set of cases where ABOProxy implementation failed. That is due to the extra restrictions that the IIS and .NET framework place on URL's besides those specified in RFC 1738. The IIS and .NET framework have the following extra restrictions: (1) The total length of the path in the URL should not exceed 160. (2) The path in the URL should not contain characters that are not allowed to be used in the NTFS file names, such as `:`, `*`, `|`, and so on. Both are un-configurable. We circumvented the second restriction by adding an extension before the .NET in the stack. The extension maps each of those special characters to a specific sequence of normal characters. The .NET extension can use the mapping rules to change those characters back when processing the requests. However, we are unable to work around the first restriction in any IIS based implementation. The websites hosted using other web servers, such as Apache, may use a very long URL whose path is longer than 160. When visiting such a URL through the ABOProxy, the IIS server will respond a 404 (bad request) error. In our testing, we observed very few websites use URLs that have a path longer than 160. Only two exceptions were observed: `doubleclick.com` and `yahoo.com`. URLs of length greater than 160 even from these sites are extremely rare.

CGIProxy For comparison, we also tested the efficacy of the CGIProxy implementation. We set up the CGIProxy version 2.1beta18 (released on Aug 10, 2008)[1] with a Apache web server 2.2, and visited some popular websites through the proxy using Firefox 2.0. We observed many basic operations failed when performed through the proxy. A summary of the failure cases of CGIProxy on some popular sites is given in Table 2.

5.2 Performance Evaluation

The following experiments were performed on a Server machine with Intel XEON Processor 2.40GHz, 1GB RAM and 90GB hard disk. For comparison, we conducted the same experiments against the CGIProxy implementation on the same server machine and report the results.

5.2.1 Page load time and page size

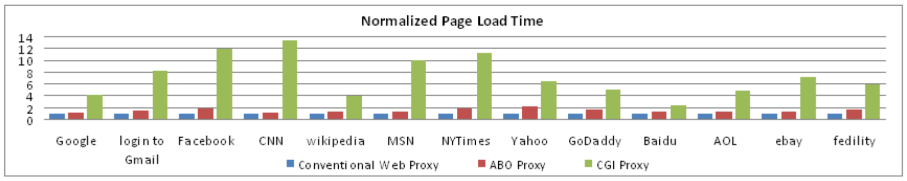
We visited a set of popular web site through the proxies using Firefox. We measured the time used to load the web pages and the total amount of HTTP data received by the browser to load the pages. Before each visit, we removed all cache files in both the server machine and the client machine. The evaluation results is shown in Figure 3. The results for "conventional web proxy" are obtained by configuring the browser to use a standard web proxy within the client's Intranet. The results for "ABO Proxy" and "CGI Proxy" are obtained by connecting through our ABOProxy implementation and the CGIProxy implementation, respectively. The results are normalized to make the measurements for "conventional web proxy" to be 1.

In terms of the page load time, the ABOProxy has 47% overhead on average compared with the conventional web proxy, while the CGIProxy has 620% overhead. In terms of the page size, the ABOProxy has 2.6% overhead on average, while the CGIProxy has 27% overhead.

5.2.2 Server throughput and responsiveness

The server scalability is typically estimated by measuring the responsiveness and throughput of the server when gradually increasing the number of concurrent clients. The server responsiveness is measured by the average Time To First Byte (TTFB) on the clients. The TTFB is the duration from the client making an HTTP request to the first byte of the page being received by the client. The server throughput is measured by the maximum number of requests that are processed by the server every second. We simulated the concurrent clients using the Web Application Stress Tool from Microsoft. The evaluation results are shown in the Figure 4. The throughput of the ABOProxy is about 4 times

[1]As stated in the website of CGIProxy, though this is a beta release, it is stabler than the old production release



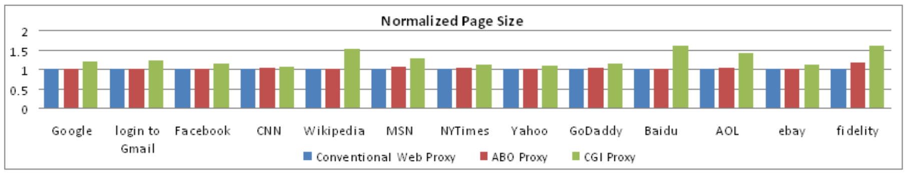
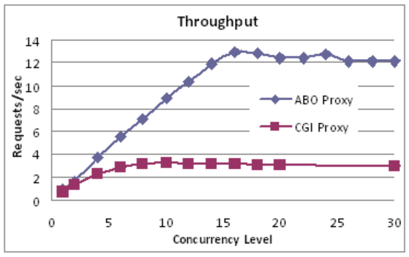


Figure 3. The performance evaluation on page load time and page size



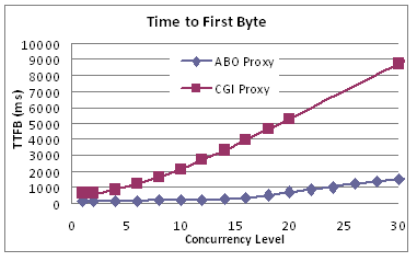


Figure 4. The performance evaluation on the throughput and TTFB under heavy load

larger than that of the CGIProxy, and the TTFB of the ABOProxy is 4-6 times less than that of the CGIProxy.

5.3 Security: Same-Origin Policy

The browser same-origin policy is enforced by isolating webpages according to the security context of a three-tuple <protocol,hostname,port>. For example, the script from `http://www.evil.org` cannot access the contents downloaded from `https://bank.com`. In the design of the CGIProxy, the original URL is encoded in the path of the new URL. In other words, every destination domain is mapped to a directory in the host of `www.cgiproxy.com`. From the browser's point of view, all websites the user visits through the proxy are hosted at `http://www.cgiproxy.com` (or `https://www.cgiproxy.com` if the proxy server is using HTTPS), so that they have the same security context. As a result, a malicious website can launch cross-site scripting attacks and cookie-stealing attacks against a sensitive website if the user visits both through the proxy. Thus, CGIProxy completely destroys the domain boundary enforced by the same-origin policy in the browser. In contrast, in ABOProxy, each destination domain is mapped to a subdomain of the proxy. Different websites visited through the proxy have different security contexts and are isolated by the same-origin policy. Therefore, our approach preserves the domain boundary among different websites and eliminates the security concern that CGIProxy introduces.

6. Conclusion

We present a novel link-translating proxy. We implemented ABOProxy using IIS and ASP.NET, and tested by using for everyday browsing. Both theoretical and experimental evaluation show that the ABOProxy is significantly better than existing web-service based proxies and WebVPN's in terms of robustness, performance, and security.

7. REFERENCES

[1] `http://www.proxy4free.com`.
[2] `http://www.freeproxy.ru`.
[3] `http://www.jmarshall.com/tools/cgiproxy`.
[4] `http://circumventor.net`.
[5] `http://proxy.org`.
[6] `http://www.torproject.org`.
[7] `http://msdn2.microsoft.com/en-us/library/ms828058.aspx`.
[8] `http://www.entensys.com`.
[9] `http://www.securityfocus.com/bid/18419`.
[10] `http://www.xk72.com/charles/`.
[11] `http://www.fiddlertool.com`.
[12] `http://www.portswigger.net/proxy`.
[13] `http://www.parosproxy.org`.
[14] `http://www.privoxy.org`.
[15] `http://www.icap-forum.org`.
[16] D. Florêncio and C. Herley. One-Time Password Access to Any Server Without Changing the Server. *ISC 2008, Taipei.*
[17] E. Gaber, P. Gibbons, Y. Matyas, and A. Mayer. How to make personalized web browsing simple, secure and anonymous. *Proc. Finan. Crypto '97*.
[18] A. Luotonen. Web Proxy Servers. *Prentice Hall*, 1998.
[19] M. J. Freedman and E. Freuenthal and D. Mazières. Democratizing Content Publication with Coral. *NSDI*, 2004.
[20] N. Provos and T. Holz. *Virtual Honeypots*. Addison Wesley, 2007.
[21] R. Dingledine and N. Mathewson and P. Syverson. Tor: the Second-generation Onion Router. In *Usenix Security*, 2004.
[22] M. E. Russinovich and D. A. Solomon. *Microsoft Windows Internals*. Microsoft Press, fourth edition, 2005.