Open Proxy Honeypots

If you build it, they will come…

Ryan C. Barnett
http://honeypots.sourceforge.net
Last Modified: 30 March, 2004

Abstract

This paper will provide instructions for deploying an Open Proxy Honeypot or Proxypot, by using an Apache web server compiled with additional security modules. 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 my most web servers.

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 - 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

- **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

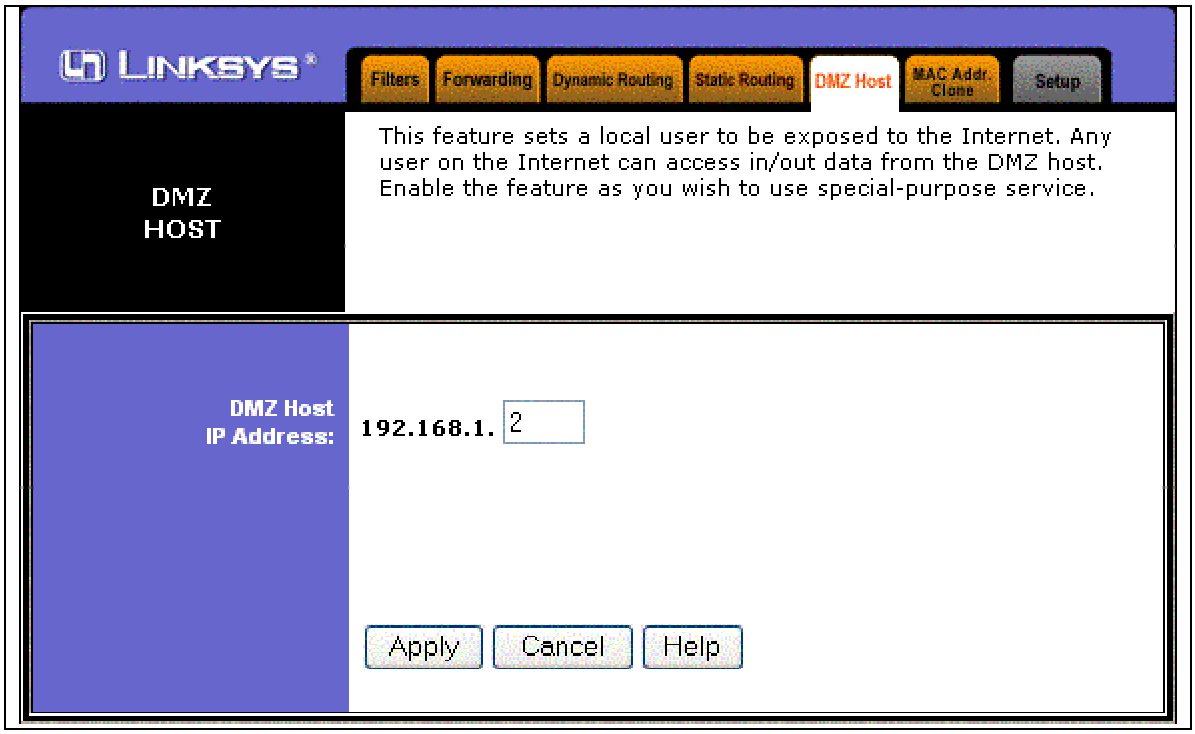
The LURHQ Intelligence Group has a fantastic write-up on open proxies background –

> *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 (Slide below is an example screenshot, the proxypot IP address was 192.168.1.103) –



This would restrict all Internet traffic to only communicate with the honeypot host.

Turn Off 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) –

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 snews 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 –

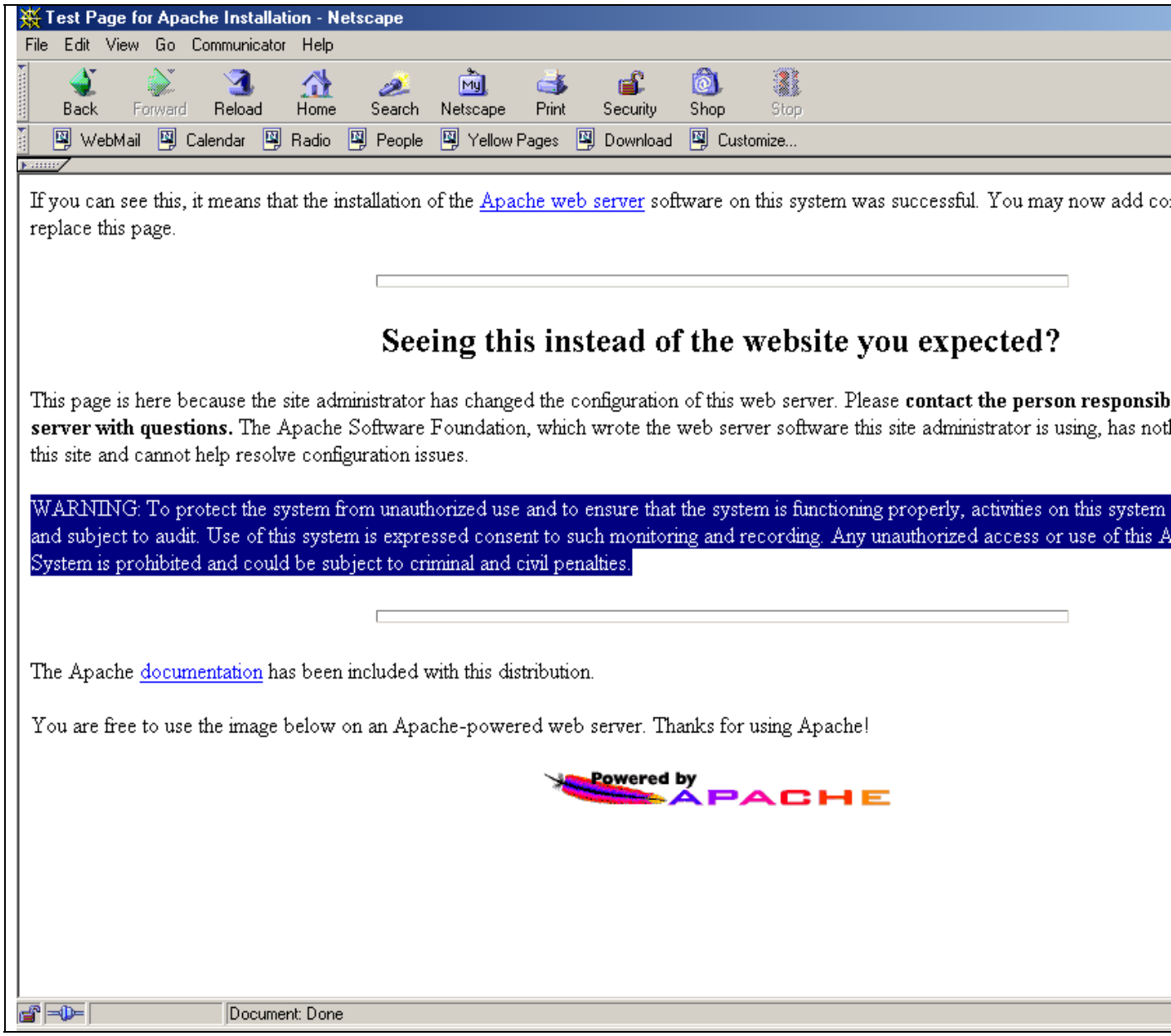
- **ProxyBlock –** We do not implement this directive. The ProxyBlock directive specifies a list of words, hosts and/or domains, separated by spaces. HTTP, HTTPS, and FTP document requests to sites whose names contain matched words, hosts or domains are *blocked* by the proxy server. The proxy module 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. Oreilly 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 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)

This method has worked well in both single-server script attacks as well as distributed attacks, but just like other evasive tools, is only as useful to the point of bandwidth and processor consumption (e.g. the amount of bandwidth and processor required to receive/process/respond to invalid requests), which is why it's a good idea to integrate this with your firewalls and routers.

This module instantiates for each listener individually, and therefore has a built-in cleanup mechanism and scaling capabilities. Because of this, legitimate requests are never compromised but only scripted attacks. Even a user repeatedly clicking on 'reload' should not be affected unless they do it maliciously.

Two different module sources have been provided:

Apache v1.3 API: mod_dosevasive.c
Apache v2.0 API: mod_dosevasive20.c

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. Additionally, a system command and/or an email notification can also be triggered to block all the originating addresses of a DDoS attack.

Once a single 403 incident occurs, mod_dosevasive now blocks the entire IP address for a period of 10 seconds (configurable). If the host requests a page within this period, it is forced to wait even longer. Since this is triggered from requesting the same URL multiple times per second, this again does not affect legitimate users.

The blacklist can/should be configured to talk to your network's firewalls and/or routers to push the attack out to the front lines, but this is not required.

mod_dosevasive also performs syslog reporting using daemon.alert.

```
Messages will look like this:

Aug  6 17:41:49 elijah mod_dosevasive[23184]: [ID 801097 daemon.alert]
Blacklisting address x.x.x.x: possible DoS attack.

WHAT IS THIS TOOL USEFUL FOR?

This tool is *excellent* at fending off small to medium-sized request-
based DoS attacks or script attacks and brute force attacks.  Its
features will prevent you from wasting bandwidth or having a few
thousand CGI scripts running as a result of an attack.  When used in
conjunction with other preventative measures such as router
blackholing, this tool is very
effective against larger DDoS attacks as well.

If you do not have an infrastructure capable of fending off any other
types of DoS attacks, chances are this tool will only help you to the
point of your total bandwidth or server capacity for sending 403's.
Without a solid infrastructure and DoS evasion plan in place, a heavy
distributed DoS will most likely still take you offline.
```

HOW TO INSTALL

APACHE v1.3

Without DSO Support:

1. Extract this archive into src/modules in the Apache source tree

2. Run ./configure --add-module=src/modules/dosevasive/mod_dosevasive.c

3. make, install

4. Restart Apache

With DSO Support, Ensim, or CPanel:

1. $APACHE_ROOT/bin/apxs -iac mod_dosevasive.c

2. Restart Apache

APACHE v2.0
---

1. Extract this archive

2. Run $APACHE_ROOT/bin/apxs -i -a -c mod_dosevasive20.c

3. The module will be built and installed into $APACHE_ROOT/modules, and loaded into your httpd.conf

4. Restart Apache

CONFIGURATION

mod_dosevasive has default options configured, but you may also add the following block to your httpd.conf:

APACHE v1.3

```
<IfModule mod_dosevasive.c>
    DOSHashTableSize    3097
    DOSPageCount        2
    DOSSiteCount        50
    DOSPageInterval     1
    DOSSiteInterval     1
    DOSBlockingPeriod   10
</IfModule>
```

APACHE v2.0

```
<IfModule mod_dosevasive20.c>
    DOSHashTableSize    3097
    DOSPageCount        2
    DOSSiteCount        50
    DOSPageInterval     1
    DOSSiteInterval     1
    DOSBlockingPeriod   10
</IfModule>
```

Optionally you can also add the following directives:

```
  DOSEmailNotify  you@yourdomain.com
    DOSSystemCommand    "su - someuser -c '/sbin/... %s ...'"
```

You will also need to add this line if you are building with dynamic support:

APACHE v1.3

```
AddModule       mod_dosevasive.c
```

APACHE v2.0

```
LoadModule dosevasive20_module modules/mod_dosevasive20.so
```

(This line is already added to your configuration by apxs)

DOSHashTableSize

The hash table size defines the number of top-level nodes for each child's hash table. Increasing this number will provide faster performance by decreasing the number of iterations required to get to the record, but consume more memory for table space. You should increase this if you have a busy web server. The value you specify will automatically be tiered up to the next prime number in the primes list (see mod_dosevasive.c for a list of primes used).

DOSPageCount

This is the threshold for the number of requests for the same page (or URI) per page interval. Once the threshold for that interval has been exceeded, the IP address of the client will be added to the blocking list.

DOSSiteCount

This is the threshold for the total number of requests for any object by the same client on the same listener per site interval. Once the threshold for that interval has been exceeded, the IP address of the client will be added to the blocking list.

DOSPageInterval

The interval for the page count threshold; defaults to 1 second intervals.

DOSSiteInterval

The interval for the site count threshold; defaults to 1 second intervals.

DOSBlockingPeriod

The blocking period is the amount of time (in seconds) that a client will be blocked for if they are added to the blocking list. During this time, all subsequent requests from the client will result in a 403 (Forbidden) and the timer being reset (e.g. another 10 seconds). Since the timer is reset for every subsequent request, it is not necessary to have a long blocking period; in the event of a DoS attack, this timer will keep getting reset.

DOSEmailNotify

If this value is set, an email will be sent to the address specified whenever an IP address becomes blacklisted. A locking mechanism using /tmp prevents continuous emails from being sent.

NOTE: Be sure MAILER is set correctly in mod_dosevasive.c (or mod_dosevasive20.c). The default is "/bin/mail -t %s" where %s is used to denote the destination email address set in the configuration. If you are running on linux or some other operating system with a different type of mailer, you'll need to change this.

DOSSystemCommand

If this value is set, the system command specified will be executed whenever an IP address becomes blacklisted. This is designed to enable system calls to ip filter or other tools. A locking mechanism using /tmp prevents continuous system calls. Use %s to denote the IP address of the blacklisted IP.

WHITELISTING IP ADDRESSES

As of version 1.8, IP addresses of trusted clients can be whitelisted to insure they are never denied. The purpose of whitelisting is to protect software, scripts, local searchbots, or other automated tools from being denied for requesting large amounts of data from the server.

Whitelisting should *not* be used to add customer lists or anything of the sort, as this will open the server to abuse. This module is very difficult to trigger without performing some type of malicious attack, and for that reason it is
more appropriate to allow the module to decide on its own whether or not an individual customer should be blocked.

To whitelist an address (or range) add an entry to the Apache configuration in the following fashion:

```
DOSWhitelist      127.0.0.1
DOSWhitelist      127.0.0.*
```

Wildcards can be used on up to the last 3 octets if necessary. Multiple DOSWhitelist commands may be used in the configuration.

TWEAKING APACHE

The keep-alive settings for your children should be reasonable enough to keep each child up long enough to resist a DOS attack (or at least part of one). For every child that exits, another 5-10 copies of the page may get through before putting the attacker back into '403 Land'. With this said, you should have a very high MaxRequestsPerChild, but not unlimited as this will prevent cleanup.

You'll want to have a MaxRequestsPerChild set to a non-zero value, as DosEvasive cleans up its internal hashes only on exit. The default MaxRequestsPerChild is usually 10000. This should suffice in only allowing a few requests per 10000 per child through in the event of an attack (although if you use DOSSystemCommand to firewall the IP address, a hole will no longer be open in between child cycles).

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/1.1 403 Forbidden
HTTP/1.1 403 Forbidden
HTTP/1.1 403 Forbidden
HTTP/1.1 403 Forbidden
HTTP/1.1 403 Forbidden
HTTP/1.1 403 Forbidden
HTTP/1.1 403 Forbidden
HTTP/1.1 403 Forbidden
HTTP/1.1 403 Forbidden
HTTP/1.1 403 Forbidden
HTTP/1.1 403 Forbidden
HTTP/1.1 403 Forbidden
HTTP/1.1 403 Forbidden
```

The following Syslog message was generated –

```
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

It has the following features:

- **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.

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
=======

DSO

Compiling the module as a dynamic library is easy. Go to the folder that contains the source code for your Apache branch, and type the following:

```
apxs -cia mod_security.c
apachectl stop
apachectl start
```

Of course, now you need to add mod_security specific directives to make it do something. Take a look at files httpd.conf.example-minimal or httpd.conf.example-full to
get some idea of that to do. Or even better, read the manual.

Apache 1.x static compilation

To compile the module into the body of the web server do the following:

1. Copy the file mod_security.c to /src/modules/extra

2. Configure Apache distribution with two additional configuration options:

```
--activate-module=src/modules/extra/mod_security
 --enable-module=security
```

3. Compile and install as usual

Apache 2.x static compilation

At the moment it seems that there is no definitive documentation on how to compile a module into the body of an Apache 2.x web server. The procedure I use is as follows (valid for Apache 2.0.45, the last version at the time of writing):

1. Configure Apache instructing it to include mod_security:

```
--with-module=mappers:security
```

2. Copy the file mod_security.c to modules/mappers/

3. Edit modules/mappers/modules.mk and add the following to it:

```
mod_security.la: mod_security.lo
       $(MOD_LINK) mod_security.lo
```

^ please note that this is a TAB character

Also, add "mod_security.la" to the "static = " line:

```
static = mod_negotiation.la mod_security.la mod_dir.la ...
```

4. You can proceed to compile the server:

```
make
 make install
 apachectl stop
 apachectl start
```

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 atacks (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\:"
```

```
    # These two entries will help to identify Brute Force Attacks
    SecFilterSelective OUTPUT "Authorization Required" pause:10000
    SecFilterSelective HTTP_AUTHORIZATION "Basic"

    # 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"
```

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". Here is an excerpt:

The Challenge
In order to discover and understand an attempted web application attack, we first need to gather all the clues from the crime scene. Collecting these "digital fingerprints" left by the reckless hacker requires that **all** of the following data fields are available, for every HTTP request:

- Date
- Time
- Client IP Address
- HTTP Method
- URI
- HTTP Query
- A Full Set of HTTP headers
- The full request body

Some of this data can be extracted from files such as the web server or application server log files, but unfortunately, the most crucial data is unavailable through these sources. Most web servers and application servers do not grant access to HTTP information such as the full set of HTTP headers and the request body. Without those fields many log entries look alike, and the person conducting the forensics will not be able to distinguish between valid requests and lethal web application attacks.

A simple example is the "invisible data in POST request" problem, mentioned in this paper's introduction. Let's take a look at a simple HTML form:

```
<FORM METHOD=POST ACTION="/cgi-bin/script.cgi">
<INPUT TYPE=HIDDEN NAME="template" VALUE="tempFile.html">
<INPUT TYPE=TEXT NAME="user" VALUE="enter username here">
<INPUT TYPE=PASSWORD NAME="pass" VALUE="">
<INPUT TYPE=SUBMIT VALUE="submit">
```

….
Now, let's take a look at the request made for this form through 3 different views. The first two are extracted from log files of Microsoft IIS/5.0 and Apache 1.3.24, and the last one will be the actual request:

[**Microsoft IIS/5.0 log file**]:
```
2002-05-05 13:24:45 192.168.1.1 – 192.168.1.2 80 POST /cgi-
bin/script.pl - 200 381 322 192.168.1.2 Mozilla/4.7+[en]+(WinNT;+I) - -
```

[**Apache log file**]:
```
192.168.1.1 - - [05/May/2002:16:21:53 +0300] "POST /cgi-bin/script.pl
HTTP/1.0" 200 150
```

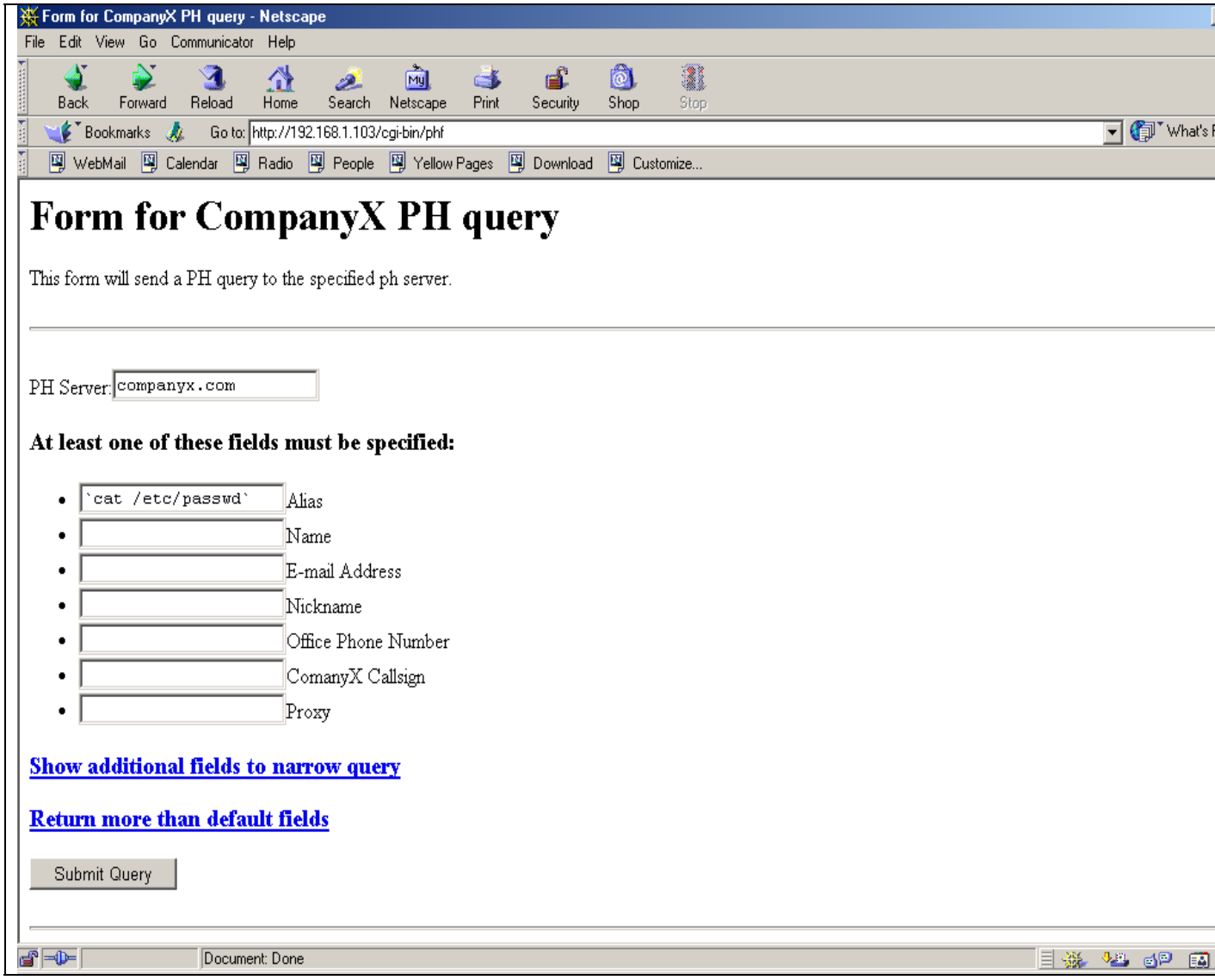
[**Actual request]:**
```
POST /cgi-bin/script.pl HTTP/1.0
Host: localhost
User-Agent: Mozilla/4.7 [en] (WinNT; I)
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png,
*/*
Accept-Encoding: gzip
Accept-Language: en
Accept-Charset: iso-8859-1,*,utf-8
Content-length: 60

template=../../../../../boot.ini%00.html&user=test&pass=test
```

As seen, the most important data does not exist in the web servers' logs. Any attempt to conduct a web application forensics investigation on this site, will surely fail to recognize that this request is a "Poison null byte" attack, which successfully retrieves the "boot.ini" file.

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



PHF Attack Audit_Log Entry –

```
========================================
Request: 192.168.1.102 - - [Mon Mar 29 17:02:32 2004] "GET http://191.168.1.103/cgi-
bin/phf?Jserver=companyx.
com&Qalias=%60%2Fbin%2Fcat+%2Fetc%2Fpasswd%60&Qname=&Qemail=&Qnickname=&Qoffice_phone=&
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
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_R
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. Here is some brief info from the dsniff README file and the webspy MAN page:

```
README –
webspy
        sends URLs sniffed from a client to your local Netscape
        browser for display, updated in real-time (as the target
        surfs, your browser surfs along with them, automagically).
        a fun party trick. :-)

MAN –
SYNOPSIS
       webspy [-i interface] host

DESCRIPTION
       webspy  sends  URLs  sniffed  from  a client to your local
       Netscape browser for display, updated in real-time (as the
```

```
       target surfs, your browser surfs along with them, automag-
       ically). Netscape must be running on your local X  display
       ahead of time.

OPTIONS
       -i interface
              Specify the interface to listen on.

       host   Specify the web client to spy on.

SEE ALSO
       dsniff(8)

AUTHOR
       Dug Song <dugsong@monkey.org>
```

Webspy had definite potential for spying on users of the proxypot. The only real limitations of webspy are that it is designed to spy on real users who are using web browsers. Unfortunately, this is 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 order to address some of these issues, 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.

Live Data – Scan of the Month for April 2004

Scan 31

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 Images
apache_logs.tar..gz
c36d39dfd5665a58d7cea06438ceb96d apache_logs.tar.gz

Conclusion

We have just completed a step-by-step on how to build and deploy an Apache Open Proxy Server. Please keep in mind that even if this proxypot is configure correctly, there is still a chance that some new (0-Day) web exploit could be launched through your server. If you don't have a Mod_Security filter for this attack, it will be sent on to the target destination. The end result being that you could possibly be participating in the successful compromise of a commercial target. This is a similar situation that exists with both signature based Network Intrusion Detection and Anti-Virus deployments – these technologies are always playing catch-up with the new tools that the Blackhats are creating. Even though there is some level of risk involved, when configured correctly, an open proxy honeypot provides us a backstage pass to the type of web attacks that are occurring on the Internet.