

**The Role of  
Non Obvious Relationships  
in the Foot Printing Process**

**SensePost Research**

**BlackHat Windows Briefings  
February 2003  
Seattle USA**

# Index

<b>1</b>	<b>Introduction</b> .....	<b>3</b>
<b>2</b>	<b>Part I - Finding domains</b> .....	<b>3</b>
2.1	Finding non obvious relationships .....	3
2.1.1	BiLE .....	6
2.1.2	BiLE-weigh .....	9
2.2	Part 1.1 - Vetting domains .....	12
2.2.1	Vet-IPrange .....	12
2.2.2	Vet-MX .....	13
2.2.3	Vet-Whois .....	13
2.3	More expansion .....	14
2.3.1	Exp-whois .....	14
2.3.2	Exp-TLD .....	14
2.3.3	Baseline .....	15
2.3.4	Vet-TLD .....	17
<b>3</b>	<b>Part II - Finding IP numbers</b> .....	<b>20</b>
3.1	Forward.....	20
3.1.1	Qbrute .....	20
3.2	Reverse .....	21
3.2.1	Qreverse .....	21
3.3	Block issues .....	21
3.3.1	core-routes .....	22
3.3.2	routedornot .....	22
3.3.3	SubCBound / MaxiCBound.....	23
<b>4</b>	<b>Conclusion</b> .....	<b>24</b>

# 1 Introduction

Let us assume the position of chief system administrator / network manager / security officer for a major company. Let us assume that the company has many different electronic offerings/services in many different countries. In order to secure the network one has to know exactly how large the network is. With sales/marketing people running amok new domain name are registered on a daily basis. New services and web sites are created overnight. Network perimeters are expanded and additional Internet connections installed - often without the knowledge of system administrators. Without knowledge of the boundaries of one's network makes the task of securing it impossible.

Let us assume the position of a cyber terrorist. Part of the terrorist's game plan is to strike at certain industries/companies. The idea is to take out a major financial institution / government in a specific country. The problem is...where to begin? At the company's website? At the mail server? Eventually, in any type of cyber attack, IP numbers need to be fed to attack systems. How does one accurately find these IP numbers? Without knowing the location of one's target, attacking it is just a waste of time.

Technology has advanced to the point where network level filtering devices is pretty stable and secure. System administrators nowadays realize the importance of keeping up to date with patches and fixes. Where system administrators are lax, worms such as SQL Slammer and CodeRed forcefully "patched" hosts. Online 24x7 scanners such as X-Scanner, HackRack and Qualis keep admins on their toes. More and more it's finding the one vulnerable host in a list of class B networks and not breaking the state of the art firewall (which 1000s of other hackers have been banging their heads against), or trying exploits against a patched machine.

Many companies linked to the Internet are also connected internally - e.g. XYZ Motors have an internal link to their producers of car seats (SeatsRUs). The car seat producer also has a link to the Internet. The perimeter of XYZ Motors might be heavily secured - SeatsRUs however has no firewall in place. Breaking into SeatsRUs directly leads to the compromise of XYZ Motors perimeter. Knowing that XYZ Motors and SeatsRUs are linked in some way is the crucial bit of information that is keeping an attacker from breaking into XYZ Motors. Which of the following is easier - finding a 0-day in an application allowed through the firewall, or finding out that the two companies are related? Guess it depends on who you talking to...

## 2 Part I - Finding domains

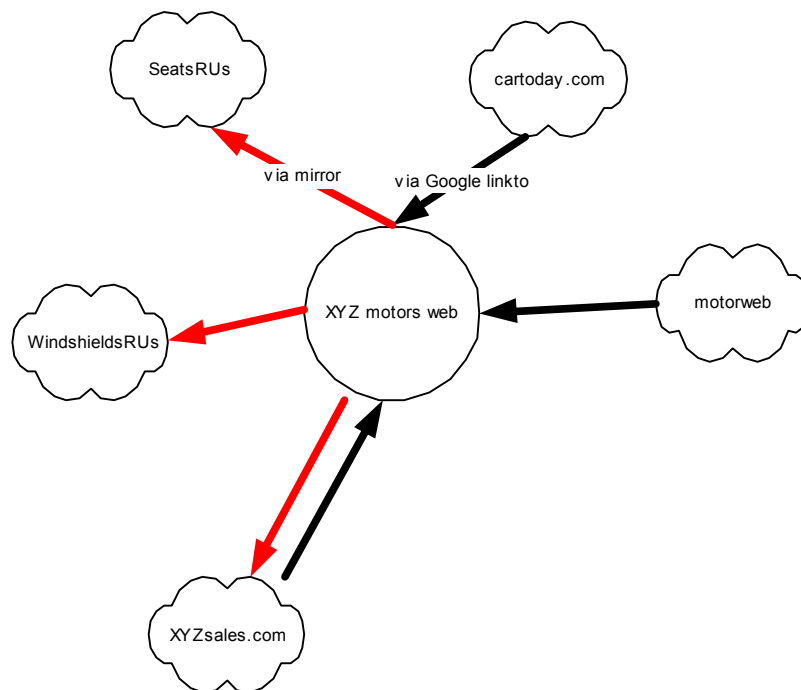
### 2.1 Finding non obvious relationships

The relationship between business people and the technical Internet world is probably the closest at the DNS domain name. Ask a CEO of a company what AS the company owns and you'll get a blank stare. Ask

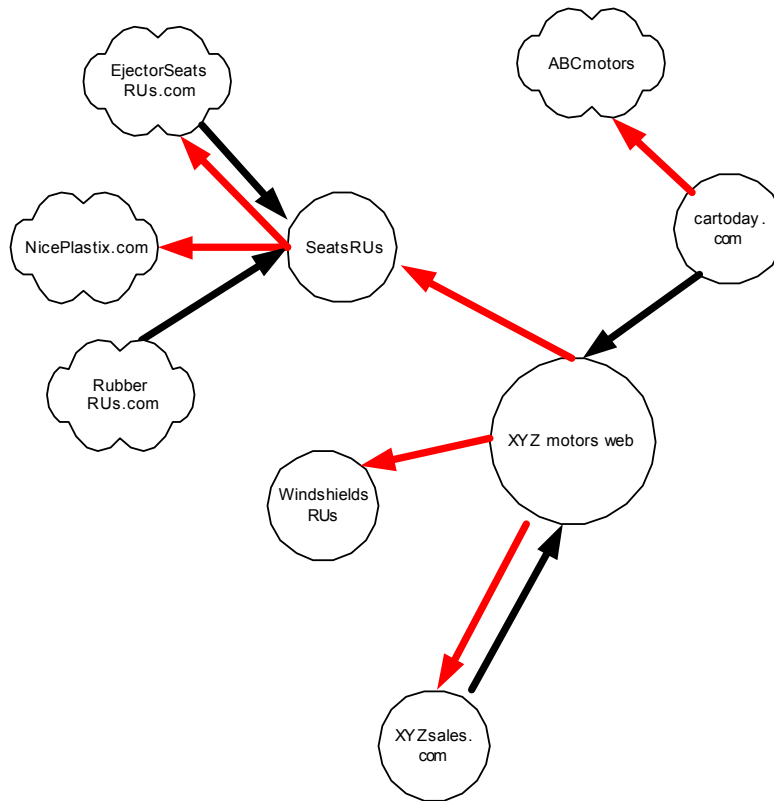
about the MX records and still you'll get a blank stare. But ask her/him about her/his email address (or website) and the domain name pops out easily. Everybody loves a domain name.

In order to map a company on the Internet we will start with the domains that a company owns. The focus at this stage is just to find domain names - we will later concentrate on IP addresses. Where do we start when getting domain names? A good first guess would be to try COMPANY.com/.net, or if they are located in a specific country .co.XX or .com.XX, where XX is the relevant country code. Once you have a web page you could start looking at links from the page to other domains - be that partners, other services, offices in other countries etc. etc. while keeping track of all the domains you find. You might also want to look at sites that link to the website. Basically what you will be doing is surfing...a lot. The list of new sites you'll find could also contain clues as to a new set of related domains - i.e. sites in the second degree of relevance.

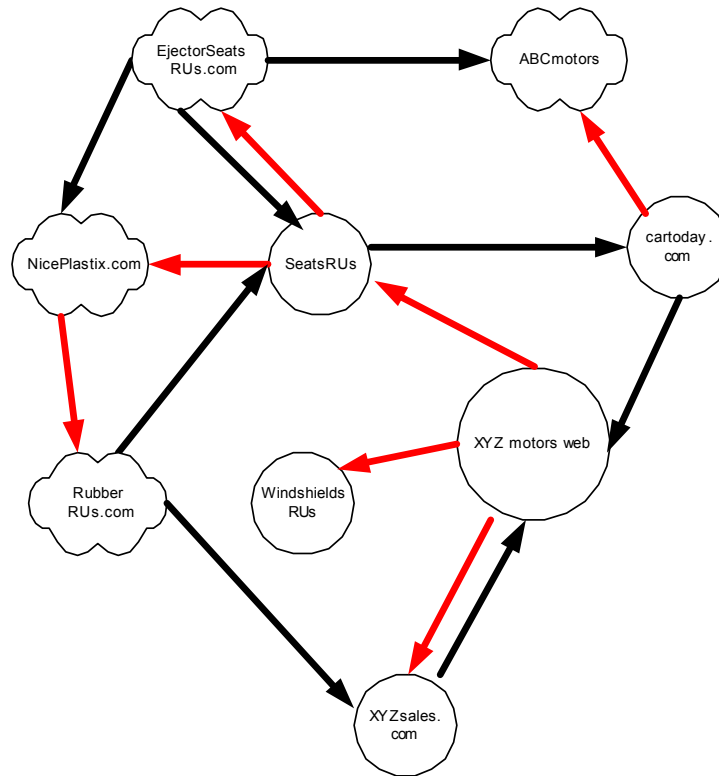
Doing this manually is entirely possible for a single web site. Do it for multiple sites and it gets tedious (and boring). Why not automate the process? Using a web site copier one can extract all the links from the site and isolate those that point outside the main site. Using Google's "linkto" function one can obtain all the sites that link to a site:



By looking at the second degree of relevance we start to see a picture that could look like this:



This is actually not a very good representation of what is really happening, as sites on the first layer of relevance (and on the second layer) are connected to each other. The picture could thus look like this:



One can quickly see that things become a little confusing and that a 3D graphics engine is needed to clearly see the relevance of some domains. Note that layer 2 sites (RubberRUs, NicePlastix, ABCMotors and EjectorseatsRUs) can never link to the core site (XYZ Motors) - if they are linked they must appear on layer 1.

One can take the processes a step further (e.g. 3 degrees), but the actual relevance to the core site rapidly deteriorates at that level. At this time we introduce our first tool called BiLE (Bi-directional Link Extractor)

### 2.1.1 BiLE

BiLE tries to do what is normally considered a manual process. It crawls a specified web site (mirrors the site) and extracts all links from the site. It then queries Google and obtains a list of sites that link to the target site. It now has a list of sites that are linked from the target site, and a list of sites that link to the target site.

It proceeds to perform the same function on all the sites found in the first round. The output of BiLE is a file that contains a list of source site names and destination site names.

#### How to use:

```
perl BiLE.pl [website] [project_name]
```

#### Input fields:

<website>	is a website name e.g. "www.sensepost.com"
project_name	name of project e.g. "sensepost"

#### Output:

Creates a file named <project\_name>.mine

**Output format:**

Source\_site:Destination\_site

**Typical output: (extract)**

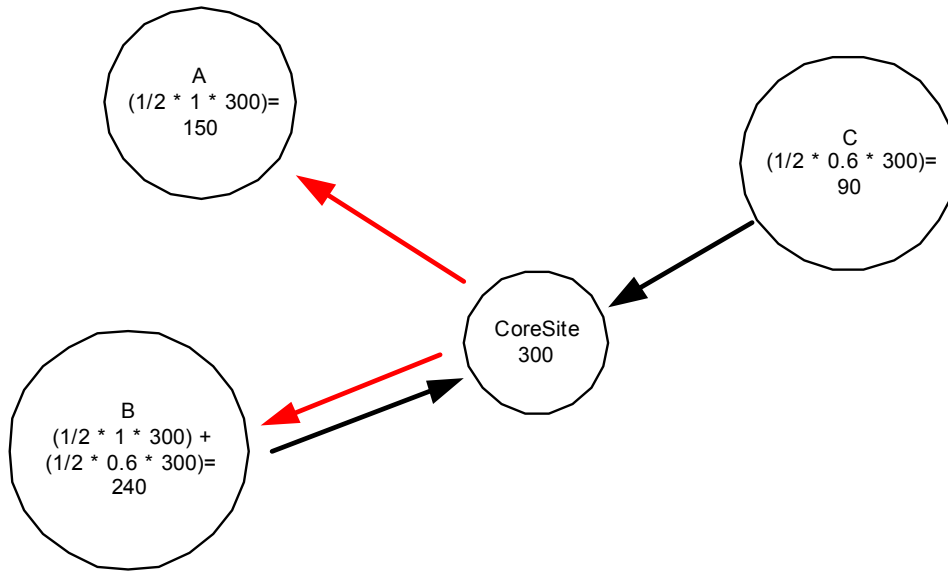
www.2can2.com:www.business.com  
www.2computerguys.com:www.business.com  
www.3g.cellular.phonecall.net:www.business.com  
www.4-webpromotion.com:www.business.com  
www.4investinginfo.com:www.business.com  
www.4therapist.com:www.business.com  
www.57threalestate.com:www.business.com  
www.5towns-business.com:www.business.com  
www.a2zofb2b.com:www.business.com  
www.4hb.com:www.amcity.com  
www.5starpr.com:www.amcity.com

BiLE produce output that only contains source site and destination site. It tells us nothing about the relevance of each site. The challenge now is to put human intuition in an algorithm.

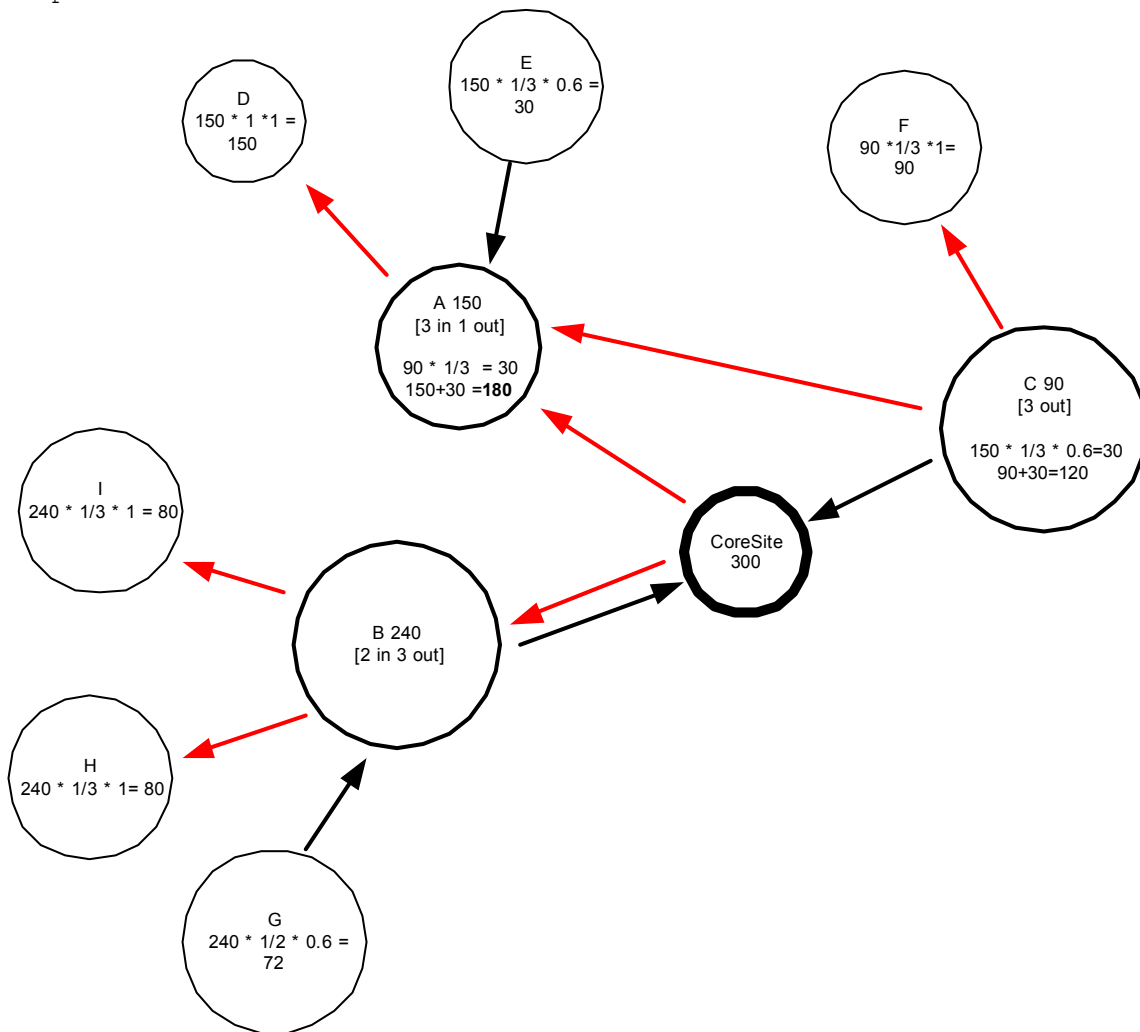
Let us first consider incoming links (sites linking to the core site). If you visit a site with only one link on it (to your core site) you would probably rate the site as important. If a site is an "Interesting Links"-type site with hundreds of links (with one to your core site), the site is probably not that relevant. The same applies to outgoing links. If your core site contains one link to a site that site is more relevant than one linked from 120 links. The next criteria is looking for links in and out of a site. If the core site links to site XX and site XX links back to the core site it means they are closely related. The last criteria is that links to a site is less relevant than links from a site (6:10 ratio). This makes perfect sense as a site owner cannot (although many would want to try) control who links to the site, but can control outgoing links (e.g. links on the site).

The core site is given a seed value and the rules listed above is applied to it. Lets see how this would work:

Step1:



Step 2



Of course real sites have much more than just three links - the process of computing the weights for every site is becomes complex and very much fit for a computer to do. The second tool listed does exactly that.

### 2.1.2 BiLE-weigh

BiLE-weigh takes the output of BiLE and calculates the relevance of each site found. The actual weighing algorithm is complex and not discussed in detail here. The following should be noted:

- A link from a site weighs more than a link to a site
- A link from a site with a lot of links weighs less that a link from a site with a small amount of links
- A link to a site with a lot of links to the site weighs less that a link to a site with a small amount of links to the site.
- The site that was given as input parameter need not end up with the highest weight – a good indication that the provided site is not the central site of the organization.

#### How to use:

```
perl BiLE-weigh.pl [website] [input file]
```

#### Input fields:

<website>	is a website name e.g. "www.sensepost.com"
input file	typically output from BiLE

#### Output:

Creates a file called <input file name>.sorted, sorted by weight with lower weights first.

#### Output format:

Site name: weight

#### Typical output:

```
www.openbsd.org:7.49212171782761
www.nextgenss.com:7.34483195478955
www.sys-security.com:7.25768324873614
www.checkpoint.com:7.0138611250576
www.linuxjournal.com:6.79452957233751
```

As an example I looked at blackhat.com. Here are the top 25 results:

Total number of nodes: 28193

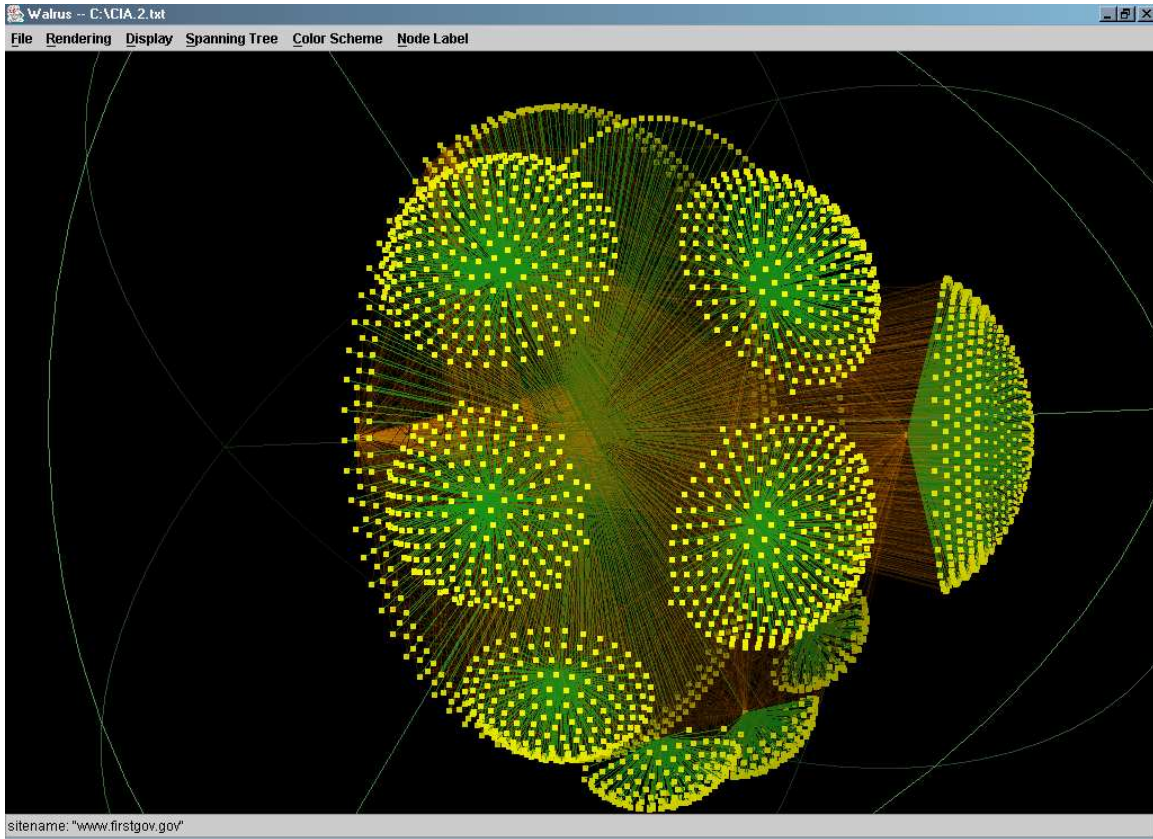
```
www.blackhat.com:50.3049528424648
www.securityfocus.com:11.3150081121516
www.defcon.org:11.2060624907226
www.securite.org:9.67638979330349
project.honeynet.org:9.65245677663783
www.attrition.org:8.46145320129212
www.nmrc.org:8.31004885760989
www.counterpane.com:8.21911119645071
www.scmagazine.com:8.16574297298595
www.infosecuritymag.com:7.92484572624653
www.convmgmt.com:7.89473684210526
www.argus-systems.com:7.66637407873695
www.eeye.com:7.54444401342593
www.whitehatsec.com:7.53535602958039
www.openbsd.org:7.49212171782761
www.nextgenss.com:7.34483195478955
```

www.sys-security.com:7.25768324873614  
www.checkpoint.com:7.0138611250576  
www.linuxjournal.com:6.79452957233751  
www.virusbtn.com:6.77886359051686  
www.sqlsecurity.com:6.6389999339814  
www.itsx.com:6.57476232632585  
www.jjbsec.com:6.5624504711275  
www.doxpara.com:6.52105355480091  
www.syngress.com:6.49850924368889  
www.sensepost.com:6.48120884564563

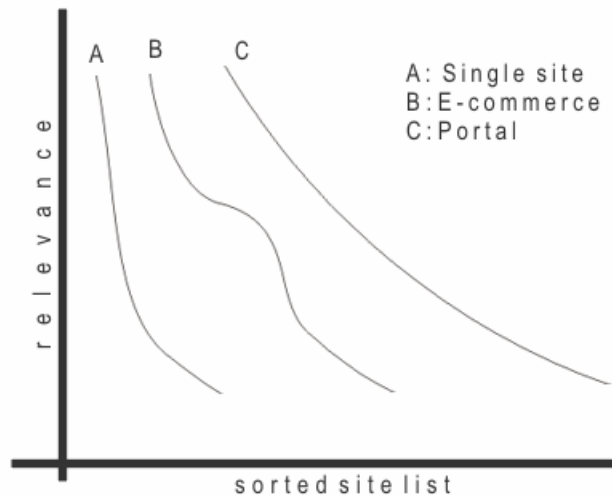
Interesting to see that a lot of the sites are those of the speakers, trainers and sponsors. Second to the top is DefCon - which is organized by the same crowd.

The list compiled here gives a good indication of how BlackHat is connected to other domains - it does not mean that BlackHat and SecurityFocus is owned (no pun intended) by the same people. In fact - the output of BiLE-weigh only gives the same type of output than someone who surfed a whole lot of websites would do. It tells us nothing about the ownership of the domain - it tells us it's closely linked to the core site, but nothing more. For a closer match we have to start looking at other forms of information such as whois info and DNS.

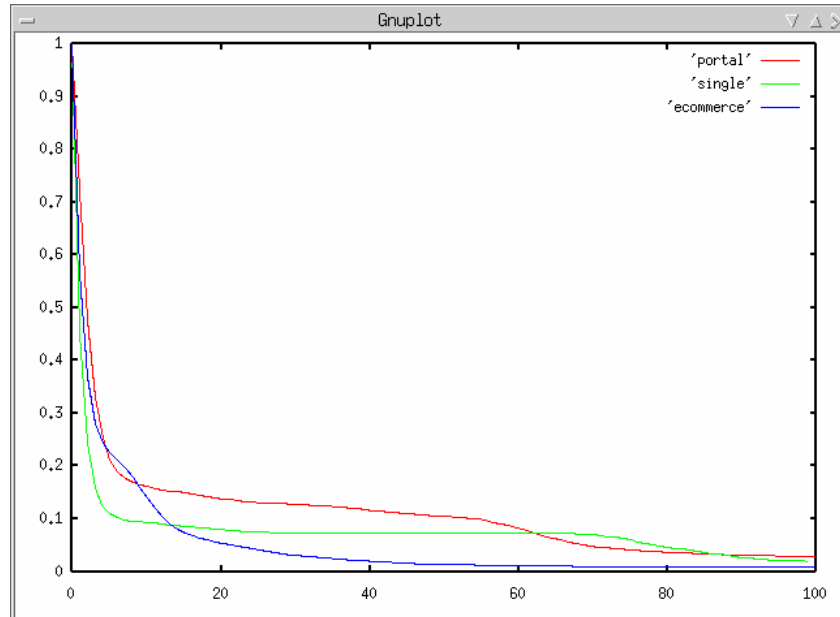
Something to consider is how to visually show the data. Clearly a 2D representation of the data is going to cause problems. A 3D view with translation and rotation is needed. After searching the web a tool called "Walrus" was found. After some email correspondence with the nice people at CAIDA.org a copy of the tool was obtained. Getting the data into Walrus was a very interesting exercise on its own. After a lot of huffing and puffing...:



An interesting thought is to try to identify the type of organization - be that an e-commerce provider, portal provider, stand-alone company, group of companies etc. One possible way of doing this is to look at the rate of relevance decay. If the rate of decay is slow it means that there are a lot of sites with a high relevance - an indication of wide spread cross-linking. This in turn could signify a portal type service. A bump early in the graph shows that there are a limited number of sites with a high relevance - this might be an indication that the organization has a few closely linked e-commerce offerings. A steep decent shows us that the site is fairly unknown and unconnected - this could be a standalone site.



Of course this method is not very scientific. In the few tests that we have done we have seen this type of behaviour - a lot more sites needs to be tested before real conclusions can be made...but this method looks promising. A real world example looks like this:



## 2.2 Part 1.1 - Vetting domains

The output of BiLE-weigh left us with a list of domains and their relevance index. We are not interested in sites that are way down in the list. We are only interested in sites with a high relevance. In practice it was found that getting all sites higher than 0.075% of the highest rating almost always return as good set of domains. In the case of our BlackHat set it resulted in 2883 entries (a bit more than 10%).

The simplest way of tying domains together is to look at their IP numbers. With virtual hosting (where a lot of web sites share the same IP address) this seems silly - but think of this - an ASP script on someone's site (that co-host with you) might be susceptible to SQL insertion. Their vulnerability might lead to the compromise of the complete platform (and your web site). Sites that are located close to the core (in terms of IP numbers) could well be related.

### 2.2.1 Vet-IPrange

This script performs DNS lookups for a set of DNS names. It stores each IP it gets and then performs DNS lookups on a second set of names. If the IP match with any of the IP numbers obtained in the first run (within a specified range) the script writes the DNS name to the output file.

**How to use:**

```
perl vet-IPrange.pl [input file] [true domain file] [output file] <range>
```

**Input fields:**

```
Input file          file containing list of domains
```

True domain file	contains list of domains to be compared to
Output file	a file containing matched domains
Range	(optional) Flexibility in IP number match (defaults to 32)

Another way of grouping domains together is to look at their MX records. If we have a list of domains that we know belong to an organisation we might want to find where their mail is going and see what other domains are sending their mail to the same group of IP numbers. The theory here is that a company might have different domains but that mail to these domains are almost always handled by only a few mail servers.

### 2.2.2 Vet-MX

This script performs MX lookups for a set of domains. It stores each IP it gets and then performs MX lookups on a second set of domains. If the IP of the MX record match with any of the IP numbers obtained in the first run the script writes the domain to the output file.

**How to use:**

```
perl vet-mx.pl [input file] [true domain file] [output file]
```

**Input fields:**

Input file	file containing list of domains
True domain file	contains list of domains to be compared to
Output file	a file containing matched domains

Another way of tying domains to each other is to look at the owners of the domains. I have to mention the Geekttools whois proxy by the CenterGate people here. Check out <http://www.geekttools.com>. The idea is simple - collect some search terms and obtain the whois information for each domain - where the returned data match any search term record the domain. The choice of search terms is very important. I usually choose something like the last 4 digits of the technical contact's fax number, the person who registered the domain's name, surname and all the related names of the core site.

### 2.2.3 Vet-Whois

This script performs a whois query on a domain. It uses the GeekTools proxy to do this. The output is compared to a list of search terms. These terms are each listed on a new line and may include spaces (although its better to have it as two terms). If the output contains any of the search terms in the file the domain is written to the output file.

Note that whois queries cannot be performed on all TLDs. The input file of this script could contain DNS names or domains - the script automatically computes the domain from a DNS name.

**How to use:**

```
perl vet-whois.pl [input file] [search terms file] [output file]
```

**Input fields:**

Input file	file containing list of domains or sites.
Search term file	contains search terms - each on a new line
Output file	a file containing matched domains

**Output:**

Creates a file containing only domains - not complete site name.

## 2.3 More expansion

One of the most common ways to expand on a domain is to use whois with wildcards -e.g. whois "sensep\*". The default behavior for the Unix whois command is to contact the whois server at whois.crsnic.net. There are many whois servers out there - and very little of them support wildcard searches (in fact its only .cz (Czech Republic) and .mil (US military). The whois server at whois.crsnic.net only replies with .com, .net and .org entries. Thus, in total we can only really expand on .com, .net, .org, .mil and .cz domains.

In some cases this is enough. The result to a whois wildcard search is usually limited to a specific amount of lines. This means you cant whois for "hacker\*" and get ALL the domains in a single go (whois.nic.mil is an interesting exception [note..I see some kid doing *whois -h whois.nic.mil "strategic"* and getting all worked up]). As such you will need to "brute force" it when the limit is exceeded -e.g. whois "hackera\*", "hackerb\*" etc etc.

### 2.3.1 Exp-whois

This script performs whois queries with wildcards. The script will automatically extract the domain name minus TLD and feed it to a wildcard whois. If the results exceed 50 entries (clips at 50) the script will automatically start a wildcard search with "a.z" appended to the domain. Added terms include: a-z, 0-9, "-" and "\_";

**How to use:**

```
perl exp-whois.pl [input file] [output file]
```

**Input fields:**

Input file	file containing list of domains
Output file	file containing domains expanded by wildcard whois entry

The other, more interesting expansion is TLD expansion. The idea here is to add all known TLDs to the main domain name. This needs to be combined with 'co', 'com', 'org' and 'ac'.

### 2.3.2 Exp-TLD

This script takes a list of domain and find out if the domain is valid in any of the other TLDs. It uses all of the TLD and combines them with a list of "middle terms". These are "co", "com", "ac" and "org". The script automatically removes the existing TLD and middle term from input file.

The script test if the domain exist by inspecting the output of a "nslookup -type=any <domain>".

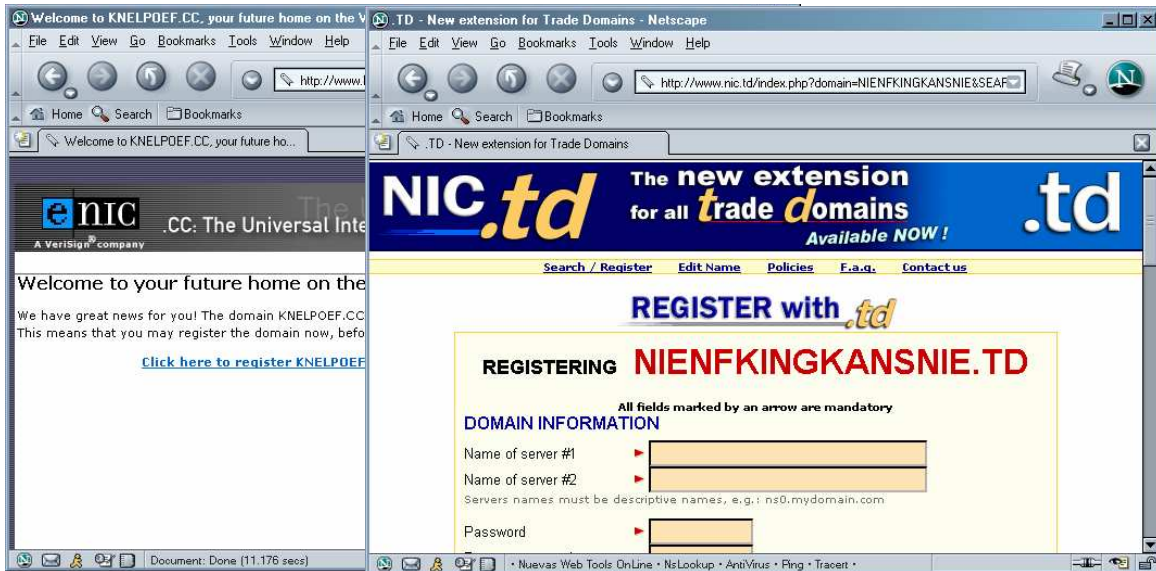
**How to use:**

```
perl exp-tld.pl [input file] [output file]
```

**Input fields:**

Input file	file containing list of domains
Output file	file containing domains expanded by TLD

The problem with TLD expansion is template sites. What's a template site? Well, it looks like this:



Anything.cc returns an actual IP address. Some TLD template sites even have MX records! As such one cannot simply decide if a TLD exists by looking at any DNS returned data. The other problem is that one cannot exclude all the TLDs where templates sites are involved - a lot of legitimate domains could be overlooked. Template TLDs are growing every day...soon *realllynotasite.com* could be redirecting to *register.com*... so how do we go about it?

The idea is to determine two things - 1) can a "fingerprint" of a template site be built? 2) what is the fingerprint? And what do I mean with a fingerprint anyhow? Here a fingerprint is nothing really exciting - it's just a MD5 hash of a sorted list of IP numbers. To start off with a baseline is created - the baseline is a file that contains the "fingerprint" of both MX and 'WWW' records for every co/com/ac/org/<blank> combination of every TLD. This file is then used later when vetting the domains.

### 2.3.3 Baseline

This script creates a baseline that is used with the Vet-TLD script. The script does the following. It adds any combination of TLD extension (with middle terms) to two non-existent domains (the domains are *bigred-control-domain* and *redbig-control-domain*). It then determines if a website for the domain exists. If so it determines the IP number(s) of the DNS name, sorts it and calculates the MD5 of it. The MD5 string is the "fingerprint" of the site.

The script also obtains all the IP number of MX records for the two domains and performs an MD5 hash thereof. The MD5 hash is referred to as the fingerprint.

The script stores the output in what is called a baseline file. Refer to the output format. The idea is not to run this script in every foot print process, but rather on a weekly basis.

#### How to use:

```
perl baseline.pl [baseline file]
```

**Input fields:**

Baseline file            where the baseline will be saved.

**Output format:**

<Type>:TLD:fingerprint1:fingerprint2

Type can be MXFP (MX Finger print) or SPFP (WWW Finger Print).

**Typical output:**

```
MXFP:.co.va:b36df02d6cd2e5b30c527ef5eb921182:b36df02d6cd2e5b30c527ef5eb921182
SPFP:.ac.va:0:0
MXFP:.ac.va:0a7b00fe0691d58b01e85b1cb3a04cd8:0a7b00fe0691d58b01e85b1cb3a04cd8
SPFP:.org.va:0:0
MXFP:.org.va:b36df02d6cd2e5b30c527ef5eb921182:b36df02d6cd2e5b30c527ef5eb921182
SPFP:.fr:0:0
MXFP:.fr:0:0
SPFP:.com.fr:0:0
MXFP:.com.fr:0:0
```

As of 3/2/2002 the following TLDs had templates sites (either on MX or on web address) associated with them:

- .cc
- .co.cc
- .co.cc
- .ac.cc
- .com.ki
- .org.ki
- .cx
- .co.cx
- .com.cz
- .ac.kz
- .co.dk
- .td
- .com.tj
- .tk
- .com.tk
- .co.tk
- .ac.tk
- .org.tk
- co.tv
- .co.nr
- .com.nu
- .com.vu
- .org.vu
- .ac.gs
- .ws
- .com.ws
- .org.ws
- .ph
- .com.ph
- .co.ph
- .ac.ph
- .org.ph
- .co.pl
- .org.com
- .co.pt
- .io
- .co.io
- .ac.io
- .co.is

The list is growing every month as ISP and companies shallow more DNS junk. Armed with the "baseline", we can now start to vet our TLD expanded domains. If we find any DNS information coming back (e.g. `nslookup -type=any domain.<middle>.<TLD>`) and the domain is not part of our baseline TLD "blacklist" we can safely assume the domain is registered by someone. If it falls within our "blacklist" we need to be more careful.

#### 2.3.4 Vet-TLD

This script performs vetting of TLD found by *exp-TLD*. It works different from *vet-MX* as it reports on non-false entries – this means that it's looking at all valid entries. It can be combined with something like *vet-MX* or *vet-whois* for more accurate results.

The script uses the baseline created by *baseline*. It scores each domain created by *exp-TLD* as a function of its true existence – i.e. it filters template sites. The script works by creating a MX and web site fingerprint (see *baseline*) for each domain in the input file. If the fingerprint for the two sites in the baseline file differs it means that fingerprinting is not possible. If it is the same fingerprinting is possible - it then compares the baseline fingerprint with the calculated fingerprints. If the fingerprint is different it means that a true site exists at the TLD. If the prints are the same it's a template site.

The following scoring is used:

TLD not in baseline	+10
Else: {	
No website:	-1
Fingerprint match (template site):	-2
Fingerprint mismatch (real site):	+2
Cannot fingerprint site:	0
No MX record:	-1
Fingerprint match (template MX):	-2
Fingerprint mismatch (real MX):	+3
Cannot fingerprint MX:	0
}	

The output is a list of domain and their scores.

#### How to use:

```
perl vet-tld.pl [input file] [baseline file] [output file]
```

#### Input fields:

Input file	file containing list of domains – typically from <i>exp-tld</i> .
Baseline file	file created by <i>baseline</i>
Output file	a file containing domains and scores

When we use *Exp-TLD* and *Vet-TLD* with *defcon* and *blackhat* as inputs we find this following (extract):

```
defcon.org:10
blackhat.com:10
defcon.org.ar:10
defcon.org.uk:10
defcon.co.uk:10
defcon.co.za:10
defcon.com.ar:10
blackhat.org.uk:10
```

defcon.com.au:10  
blackhat.org.org:10  
defcon.co.kr:10  
blackhat.co.uk:10

A quick visual inspection of the domains (by looking at their web sites) reveals that





The list of domains found here can now be fed into one of the previous scripts mentioned here - e.g. something like `vet-mx` or `vet-whois` in order to validate if they are really linked to the target/customer/core site.

At this stage it should be clear to the reader that there is no definitive way of linking the different tools together - it totally depends on the customer/target and the auditor/attacker's mood on the day.

## 3 Part II - Finding IP numbers

So you got all the domains - now what? Our end goal is to find targets - IP numbers. From the domains we are able to get the IP numbers. There are two types of DNS queries - forward and reverse. In a forward entry you provide a DNS name and get an IP number back. A single DNS name might return multiple IP addresses (such as `www.microsoft.com`). In a reverse entry you provide the IP address and get a DNS name. Neither forward nor reverse entries are guaranteed.

### 3.1 Forward

We start with all types of forward entries. The query with the biggest bang for your buck is of type AXFR (or zone transfer). Where possible this returns the whole zone file of a domain. The zone contains all the DNS names defined for the domain and their corresponding IP numbers. We'll record each IP number returned, chop the first 3 octets from it, and store this in a file (e.g we work on a 255 IP resolution). Let us look at an example:

```
> host -l whitehatsec.com
whitehatsec.com name server yns1.yahoo.com
whitehatsec.com name server yns2.yahoo.com
whitehatsec.com name server ns8.san.yahoo.com
whitehatsec.com name server ns9.san.yahoo.com
whitehatsec.com has address 209.132.84.90
smtp.whitehatsec.com has address 209.132.84.90
cvs2.whitehatsec.com has address 67.121.255.90
learn.whitehatsec.com has address 209.132.84.90
mysql.whitehatsec.com has address 67.121.255.90
disco.whitehatsec.com has address 67.121.255.90
mail.whitehatsec.com has address 209.132.84.90
www.whitehatsec.com has address 209.132.84.90
arsenal-old.whitehatsec.com has address 67.121.255.90
community.whitehatsec.com has address 209.132.84.90
```

In the case of `whitehatsec.com` we will store `209.132.84,67.121.255`. What if a zone transfer is not possible? We know that `www.<domain>` is a good start, as well as getting the MX (mail) records for the domain. How about `ns.<domain>`? How about doing a brute force - doing a lookup on anything from `aaaaaa` to `zzzzzz`? Its going to take a while - rather have a list of commonly used DNS names, mangle it a bit (to add `"1"`, `"-1"`, `"2"`, `"<domain>"` to it, and in combinations), make it multithreaded and see where you get responses.

#### 3.1.1 Qbrute

Script is used to brute force DNS forward entries where a zone transfer is not possible. This script will do a forward DNS lookup using the specified domain and each word contained in the specified domain. Output is sent to STDOUT. The script is multi threaded - firing up to 10 threads at a time.

##### How to use:

```
perl qbrute.pl [domain_name] [file_with_names] [nameserver]
```

##### Input fields:

Domain name	the domain name
File_with_name	the full path the file containing common DNS names
Nameserver	the nameserver to be used

**Typical use:**

```
perl qbrute.pl sensepost.com common 196.25.1.1
```

**Output format:**

DNS name ; IP number

Now - by means of zone transfer or using the brute forcer are left with a list of DNS names and IP numbers. Even if a zone transfer is not possible and the brute forcer returned no data we still have the MX records and the website's address (of course these might also not exist...but then the relevance of the domain should be questioned). The IP numbers have been "cropped" to its first 3 octets and saved. For each block (first 3 octets) we can again record the whois information (using the GeekTools whois proxy).

## 3.2 Reverse

Once all the blocks have been identified the whole block (from 1-255) should be reverse-walked. This mean every IP number in the block's reverse entry is obtained. If the reverse entry matched any term in a filter file (that was compiled earlier) the IP and the DNS name is recorded. IP numbers that does not match the filter file is kept in a separate file.

### 3.2.1 Qreverse

Script is used to reverse DNS walk a class C network. Output is sent to STDOUT. The script is multi threaded - firing 11 threads at a time (11 x 25 IPs per thread).

**How to use:**

```
perl qreverse.pl [subnetblock] [nameserver]
```

**Input fields:**

Subnetblock	first 3 octets of network address
Nameserver	the nameserver to be used

**Typical use:**

```
perl qreverse.pl 196.33.61 196.25.1.1
```

**Output format:**

DNS name ; IP number  
DNS name is blank if no reverse entry could be found.

## 3.3 Block issues

There are several interesting things we can do with each block that was found (besides finding the owner via GeekTool's whois proxy). The first is to look if the block is routed on the Internet. Normally we would look to see if a block in any of the unrouted blocks (e.g. 10, 172.16 or 192.168, >224). There are however many other networks that are not routed over the Internet. How do we find out if our netblock is routed or not?

### 3.3.1 core-routes

Script logs into a core router on the Internet, then queries the router to find matching subnets. The match is performed on the first part of the returned netblock.

**How to use:**

```
perl core-routes.pl [subnetblock]
```

**Input fields:**

Subnetblock                      any number of octets of network addresses

**Typical use:**

```
perl core-routes 196.8
perl core-routes 196.4.160
```

**Typical output**

```
196.8.100.0/24
196.8.101.0/24
196.8.103.0/24
196.8.104.0/24
```

Another way of determining if a netblock is routed or not is to contact one of the "looking glasses" on the Internet such as nitrous.digex.net. This method has the advantage that it provides the actual route. Typically this is what you want when you are scripting (you could be specifying the wrong boundary to *core-routes* - 196.30.67 wont return any results as the actual route is 196.30.0.0/16)

### 3.3.2 routedornot

Script queries MAE East looking glass via web interface for a specific IP. If the IP is routed it returns the block that is routed. If not it returns 0.

**How to use:**

```
perl routedornot.pl [IP]
```

**Input fields:**

IP                                  the IP number to the queried

**Typical use:**

```
perl routedornot.pl 196.4.160.2
```

**Output format:**

Routed : Route      Where Routed is 1 or 0 and Route is the actual route.

#### 3.3.2.1.1 Typical output

```
1;196.4.160.0/24
```

The next step is to find out if a Class C or a Class B is a continuous block of IP numbers or that the block is subnetted. Normally we would like to traceroute to each IP in the block and record the route - a change in the route followed indicates a boundary. Tracerouting to each IP in a class B network is not really an option - it takes forever. Load balancing and redundant network infrastructure also make it very difficult as packets could follow a different route for each IP number.

Let us quickly remind ourselves how traceroute works again. When a router routes a packet it decrease the TTL value in the IP header. If the value hits 0 the router or host sends back an ICMP TTL exceeded message. The packet sent could be any type of IP packet - TCP,UDP or ICMP. A Unix traceroute sends out a UDP packet (to the destination specified). It starts with a TTL of 1 and waits for an ICMP TTL exceeded. When it gets it an ICMP signal packet a packet with TTL of 2 is send...and so forth and so on.

If we want to trace to all IP numbers in a subnet we could start by tracing to the first IP in the subnet (increasing the TTL as a normal traceroute would do, waiting for the ICMP signal) and obtaining the TTL value. Let us assume that the TTL value where an ICMP signal is received is 15. We now trace to the next IP...but surely we don't have to start from TTL 0? We know that the previous IP's TTL was 15 - we thus send a packet with TTL 18 and see if we get something. If not we decrease to 17 then 16 and then 15. If the packet follows the same path to the IP we'll get an ICMP TTL exceeded at 15. If it follows a different route we'll get something else - perhaps 16 or 17 or 18. So what's so cool about it? With a backtrace number of 3 and a forward trace of 15 it means that we can trace 5 times faster than a normal traceroute.

We don't have to search for every IP in the block. Sub class C boundaries are likely to be on 4,8,16,32, or 64 boundaries. We can thus trace to the first IP in the block to obtain the correct TTL and then trace back to IPs that falls directly into the relevant blocks.

### 3.3.3 SubCBound / MaxiCBound

This script tries to identify network boundaries. It uses a custom traceroute engine (using modified hping) that traces from higher TTLs to lower TTLs. It performs the trace 3 times (for redundant / load balacing routing). If the any of the 3 next-hop IPs match a continuous block is assumed.

#### How to use:

```
perl subcbound.pl [classC] [grain] [backtrace] <start>
perl maxicbound.pl [classB] [grain] [backtrace] <start>
```

#### Input fields:

ClassC	The first 3 octets of the network
Grain	The resolution to find boundaries (4,8,16,32,64 IPs) In maxiCBound no limits are set -e.g. can go on 1 classC res.
Backtrace	How many TTL levels to add when reversing trace
<start>	IP where to start

#### Typical use:

```
perl subcbound.pl 196.20.10 32 3
perl maxicbound.pl 196.20 4 32 10
```

#### Typical output:

```
# perl subCbound.pl 196.15.241 32 2
Ramped till 7
Last hop for 196.15.241.1 is :168.209.13.218 168.209.13.218 168.209.13.218
Last hop for 196.15.241.33 is :168.209.13.218 168.209.13.218 168.209.13.218
Last hop for 196.15.241.65 is :168.209.13.218 168.209.13.218 168.209.13.218
Last hop for 196.15.241.97 is :168.209.13.218 168.209.13.218 168.209.13.218
Last hop for 196.15.241.129 is :168.209.13.218 168.209.13.218 168.209.13.218
```

Last hop for 196.15.241.161 is :196.15.241.76 196.15.241.76 196.15.241.76  
Last hop for 196.15.241.193 is :196.15.241.78 196.15.241.78 196.15.241.78  
Last hop for 196.15.241.225 is :168.209.13.218 168.209.13.218 168.209.13.218

Boundary at 196.15.241.160  
Boundary at 196.15.241.192  
Boundary at 196.15.241.224

## **4 Conclusion**

In the first section of the document we have seen how to collect domain names. In the second section we seen how to obtain the IP numbers associated with the domains, as well as how to find the boundaries of networks. By combining the different scripts listed here a very powerful automated foot printer can be constructed.