Global Information Assurance Certification Paper

Visualizing the Hosting Patterns of Modern Cybercriminals

GIAC GCIH Gold Certification

Author: Andrew Hunt, sans.ahunt@gmail.com
Advisor: Rick Wanner

Accepted: 12 September 2010

Abstract

The Domain Name Service (DNS) forms the basis of all Internet hosting for companies, individuals and criminals alike. Passive DNS logging provides a domain history, linking it not only to Internet Protocol (IP) addresses, but to domain registrars, ISPs and geographic locations. This paper will demonstrate the applied utility of passive DNS records through pivots, relationships to Internet Service Providers (ISPs), and the power of link-nodal visualization. It will also show how 'bullet-proof' hosters layer their products from their legitimate bases of operations, package them, and provide resiliency to illegitimate purposes. The ultimate goal of the analysis, beyond education of how illicit hosting works, is to provide techniques for incident responders to employ in making intelligent decisions when selecting the most useful combination of layered defense techniques, either for efficiency or completeness, against an identified, mapped threat.

Introduction

The Domain Name Service (DNS) is critically important to translating human readable domain names into Internet Protocol (IP) addresses. Take Google, for example. Without DNS, users would find it extremely difficult to connect to one of the hundreds of IP addresses providing Google's services. Resolving 'google.com' via DNS provides the user almost instantaneous access to Google content from the closest four of its thousands of servers. This speed of association and resiliency has been the backbone of the Internet's success. This holds true for those serving legitimate content and services, and for those meaning to do harm. This paper will demonstrate how historical DNS resolution data can be used to identify patterns of malicious domain registrations. It will also show how to identify weak points, assisting security analysts in the defense of their networks.

DNS Record Types

To understand what DNS records are related to each other and which are important, one must first know something about how DNS resolves a queried domain name to an IP. DNS queries are prompted from the client machine when the user or application requests a named domain. The domain resolution client software initiates a query record over User Datagram Protocol (UDP) port 53 to the domain's authoritative domain server (Mockapetris, 1987). The server responds with a resource record (RR) object, which is sent back to the client's originating port. The RR response contains information linking the request to resolved data through a variety of types.

There are a variety of DNS record types provided by the authoritative nameserver that correspond to the kind of data requested (Mockapetris, 1987). 'A' records are the most common, providing an IP address or list of IP addresses for a given domain query. Having multiple servers represented by a domain name is a form of service distribution, which provides resiliency to the client connections. If the client cannot connect to the first IP address provided, it can move the second, third, and so on until it receives the desired response. A domain having many A records to different IP addresses could be a fast-flux domain supporting botnet command-control. An IP address with many domain names would likely be a virtual hoster, a server shared among many sites. The following shows a typical example of an A record return.

```
google.com.   172  IN   A    72.14.204.99
```

There are other forms of resiliency in the DNS system. Nameservice (NS) records provide the client information about the authoritative name servers that are responsible for providing the information for the requested domain. These servers themselves have authoritative name servers, and so on until reaching the root domain servers. NS queries can also return multiple and disparate responses, indicating the level of resiliency, planning, and investment a provider has provisioned to support one or a set of domains. NS records directly indicate the level of investment that has been made to support a domain. Most domains have no more than a few redundant authoritative nameservers to ensure resolution of the domain name is available. A base domain having many more supporting nameservers or criss-crossing redundancies are unusual, providing a great deal of redundancy at this weak point for the malicious domain owner.

```
google.com. 315013 IN NS ns3.google.com.
```

Pointer (PTR) queries, commonly called 'reverse lookups', provide a return record with the domain for a requested IP address. The Internet Service Provider (ISP) that controls the IP address in question determines the domain returned in the RR response. At times, this can yield valuable information about an IP addresses' class or general use. For example, if an email were delivered, purportedly from Amazon.com, and the email gateway performed a reverse lookup on the delivering IP address, a resource record returned with the domain of a Polish consumer-grade Internet connection would seem suspicious. Further scrutiny might reveal that the email was not legitimate.

```
13.148.125.74.in-addr.arpa. 14400 IN PTR s9b1.psmtp.com.
```

A domain name can refer to another domain name through the use of a 'CNAME' resource record. This is typically used as a shorthand to direct one domain to another, such as directing a commonly used subdomain to the actual web host to assist users following old habits. Google is a prime example, returning a CNAME record when requesting 'www.google.com' directing the user to the actual service hosted at 'www.l.google.com'.

```
www.google.com. 572415 IN  CNAME   www.l.google.com.
```

There are a few other kinds of records that might be observed, but are much less frequent. Mail exchange (MX) records indicate servers providing email services for a domain. Source of Authority (SOA) records provide source of authority data for a domain query. Text (TXT) records provide free form text information that might be interesting to some applications, but is less useful for the link analysis presented in this paper. IPv6 and DNSSEC domain resolutions were not available in the datasets, therefore were not covered in this paper.

Andrew Hunt, sans.ahunt@gmail.com

```
google.com.  506 IN MX 200 google.com.s9a2.psmtp.com.
```

These basic return record types provide the building blocks for directing Internet traffic and for more advanced analysis of these directions. However, this distributed architecture is also easily changed, making it ethereal and time-sensitive. The resolution data must be captured into a historical body of data to be analyzed. Passive DNS monitoring infrastructures enable this by providing standard distributed architectures to extract, sort, and retain DNS data from raw traffic feeds.

The Basics of Passive DNS Monitoring

The nature of DNS resolution presents a couple of problems for those looking to collect data about domain hosting structures (Zdrnja, 2009). First, it is easily changeable, as often as every couple of minutes. This gives a malicious actor great agility, making it difficult to follow and map their infrastructures. An example of this is fast-flux hosting, which involves a domain resolving to many compromised IP addresses that are rapidly circulated to prevent investigators from effectively mapping and isolating the malicious networks. Second, an actor can use DNS diversity to amplify their base servers, providing a much larger DNS structure and layers that make it hard for investigators to peel back and engage the base infrastructure. As with the common defender's moniker, 'defense in depth', the attacker also utilizes the idea of the onion by layering their attack structure in DNS to make tracking the attack difficult. The cost of investing in DNS diversity is far less than that of compromising servers or buying infrastructure from which to base operations. This provides an incentive to use the technique as cover. The massive diversity means that, while an organization might be affected by one aspect of a malicious compromise, ten others could exist that it cannot detect or defend.

Attackers specialize in using different ruses and attacks against different targets. An infrastructure supporting an attack against many financial firms may be completely different than that used against a single targeted firm or organization. Due to this variety, the collection must cover a diverse set of organizations, types of traffic, and activities. Otherwise, the data will cover too narrow a slice of activity to reveal linkages between seemingly diverse activities (Zdrnja, 2009). Internet service providers are excellent candidates for collection as they have many customers with different activities and pass lots of queries. Universities are also great candidates as their constituents perform a broad base of activities, from business to research to personal uses. Students also make up a large, vulnerable population frequently targeted by aggressors. However, any one of these types of hosters alone might still not have enough queries to be representative. Pulling the collections together into a central collection provides the most comprehensive resource to determine what something resolved at a point in time.

There are two ways to collect historical DNS resolution data: actively and passively. 'Active' collection involves using customized resolvers to accept domain or IP address inputs for suspect domains and issue live resolution queries for them (Zdrnja, et al, 2007). This provides accurate resolution information for that moment in time, but must be repeated regularly to develop a body of data from which to determine patterns and relations. This kind of resolution is also noisy and may provide an aggressor with a tip-off that they are being investigated. It may also provide them some data about the investigator, which is undesirable. Knowing that an investigative sweep will follow malfeasance allows an aggressor to prepare cover, or bad data, that they will return to distract and hide from those investigators. The required initial investment is minimal, but the return on investment is quickly diluted as the attacker adapts, providing false data, sometimes called ‘poisoning the returns’, and reducing confidence in the accuracy of the collected data.

Passive DNS monitoring is a distributed collection and analysis platform designed to capture, retain, and query historical DNS requests and resolutions by real hosts (Zdrnja, 2006). It is more reliable than an active query collection system because the data comes from real hosts doing real things. There is no false data, just real recorded data with nothing to undermine as the attacker has to use a valid resolution path to control their targets. That is not to say that the aggressor cannot find ways to have hosts resolve bogus queries, but it is much more costly as they have to manage that activity concurrent to their attack. Having a body of data, it is also easier for the investigator to filter out bogus resolutions.

The greatest strength of the passive method is that it yields nothing to the attacker. They have no indicator of what the investigator is analyzing, whether they are on any particular track or otherwise. This changes the balance of the advantage to the investigator, allowing them to query for relationships at will, throughout the historical data available to them, without revealing anything to the aggressor. Because passive methods are based on real queries, they could also be configured to reveal the initial host making the query, allowing an investigator to locate and address an affected host. The recording of client data would have to withstand privacy review and restrictions according to the laws and policies that apply to the jurisdiction of the collector.

Unfortunately, to be consistently useful, a passive DNS collection repository must cover enough of the Internet to gather large swaths of the geographic, utility, and categorical diversity inherent to such a large network (Zdrnja, et al, 2007). The vast nature of this collection scope presents several requirements. First, the collection must be distributed, lightweight and easy to deploy. A stable, single-function distribution allows the collectors to operate at a variety of organizations with disparate functions and needs, but still allows each organization to participate in useful collection. Second, a large body of organizations and ISPs must provide these collections, or at least distributed query access to portions of their datasets, to a central repository for processing, providing a large dataset from which to query relationships across actors' many targets. Selecting and provisioning a trusted central processing agent to serve the requirements of this diverse set of multi-industry, multi-national contributors presents its own challenges, but is beyond the scope of this paper (Mitchell, et al, 2007). For providing useful, accurate data that can be utilized by researchers and investigators to profile normal and anomalous hosting structures and patterns, passive monitoring provides a clear advantage.

Harnessing Passive DNS

Having this repository of data, where does one start looking? Usually the investigator has an artifact from an investigation indicating a domain or IP address of interest. Wanting to profile that artifact to determine if it is hostile by intent, or perhaps a compromised waypoint acting as the infection vector, the investigator might turn to passive DNS. For the purposes of this paper, Malware Domains was consulted to provide a list of known hostile domains to query and generate results for demonstration (Malware Domains, 2010, May 03). Malware Domains List (http://www.malwaredomainslist.com/) is another excellent source for initial domains to query.

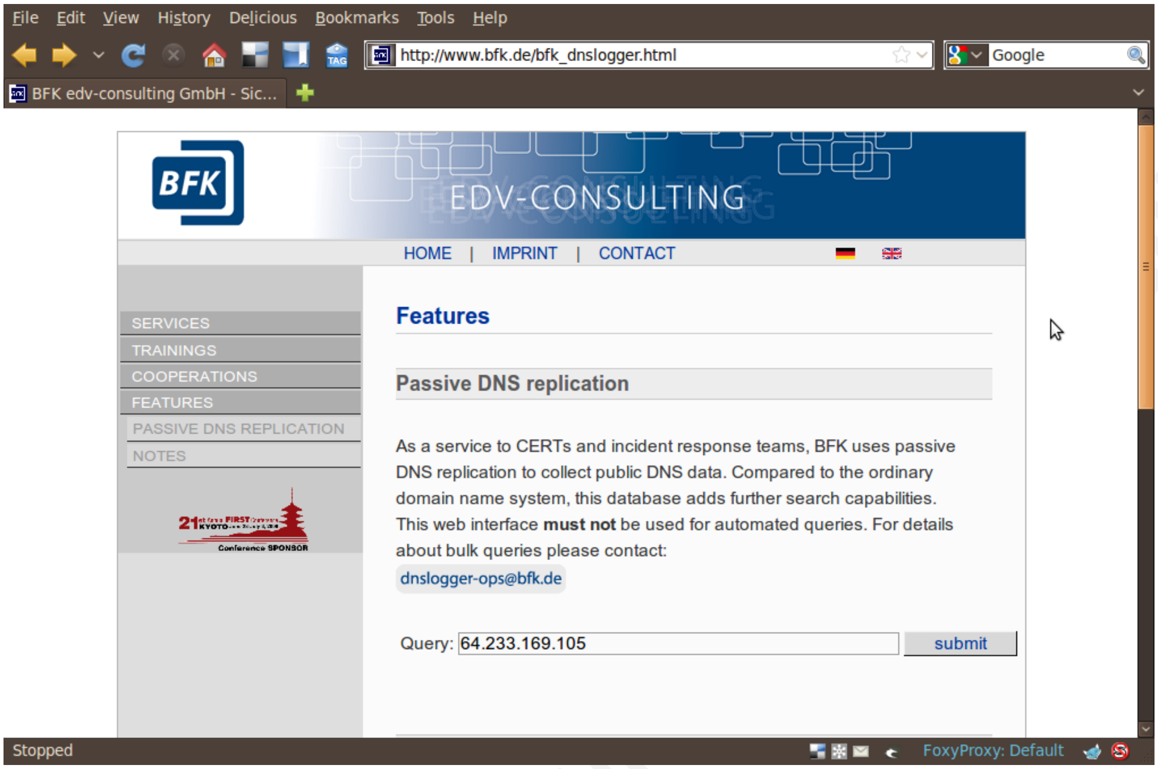
When querying a passive DNS repository, a single query will generate many results. A single pass may reveal some name service provisioners and maybe a couple of ISPs, but it will likely lack the depth needed to reveal how diverse domains known to be related to an attack might be related. To gather useful data about relations, the investigator must also query the results to find several recursive layers of related name services.

It is also important to scope how many recursions are necessary. Like the Oracle of Bacon, if the investigator performs too inclusive a recursion, they could end up introducing too many legitimate relations into the data result to make finding the interesting relations more difficult (Reynolds, 1999). Everything in the Internet is ultimately linked to everything else. That's how DNS works. Large recursive queries also impose a heavy burden on the repository and may incur the wrath of the provider. Don't abuse the privilege. It is recommended to never use more than three recursions for a run, and then review the returns to weed out uninteresting returns before continuing.

Several passive DNS repositories have been established, but only a few are available for public query. Most have some kind of restriction that limits the scope of their utility. The repositories utilized for the analysis supporting this paper will be discussed.

BFK's DNSLogger Project

Created by Florian Weimer in 2005, originally under the University of Stuttgart (Weimer, 2010, March 15, RUS-CERT), DNSLogger is the oldest and most available passive DNS project (Lorenzen, 2006). A public interface allows anyone to enter an IP address or domain and query the database for other records previously resolving to the same (Weimer, 2010, March 15, BFK).



http://www.bfk.de/bfk_dnslogger.html

Features

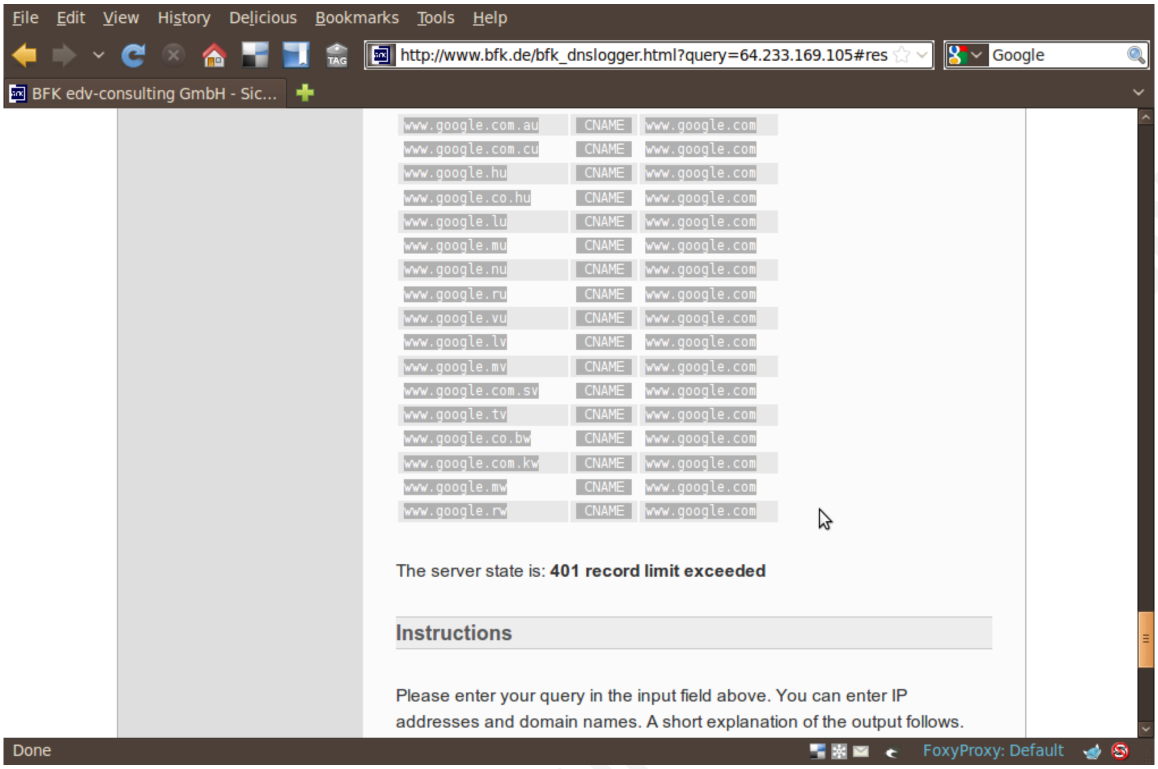
Passive DNS replication

As a service to CERTs and incident response teams, BFK uses passive DNS replication to collect public DNS data. Compared to the ordinary domain name system, this database adds further search capabilities. This web interface **must not** be used for automated queries. For details about bulk queries please contact:
dnslogger-ops@bfk.de

Query: 64.233.169.105

The records returned are simple and easy to understand, only including a unique list of the matching resolutions and record type (Weimer, 2005). Intended for incident responders, the BFK repository enjoys one of the largest deployments of collection sensors at ISPs in Germany, providing a healthy mix of consumer and commercial query data. However, when presenting queries for investigations of more targeted attacks, the geographic locus of the data to its German participants becomes apparent, lacking the scope to provide data from other locales. The web front end also imposes a limitation on the number of records returned, which can diminish the available scope when investigating a large front of malicious domains. Finally, the posted usage guidelines discourage automated query tools, making collection of the needed mass of data surrounding an incident as large as an organized aggressor a daunting manual task. Regardless of these detractors, BFK's product provides essential insights and has provided much of the data used for these case studies.

Andrew Hunt, sans.ahunt@gmail.com


File Edit View History Delicious Bookmarks Tools Help
http://www.bfk.de/bfk_dnslogger.html?query=64.233.169.105#res
Google
BFK edv-consulting GmbH - Sic...
Done
FoxyProxy: Default

www.google.com.au	CNAME	www.google.com
www.google.com.cu	CNAME	www.google.com
www.google.hu	CNAME	www.google.com
www.google.co.hu	CNAME	www.google.com
www.google.lu	CNAME	www.google.com
www.google.mu	CNAME	www.google.com
www.google.nu	CNAME	www.google.com
www.google.ru	CNAME	www.google.com
www.google.vu	CNAME	www.google.com
www.google.lv	CNAME	www.google.com
www.google.mv	CNAME	www.google.com
www.google.com.sv	CNAME	www.google.com
www.google.tv	CNAME	www.google.com
www.google.co.bw	CNAME	www.google.com
www.google.com.kw	CNAME	www.google.com
www.google.mw	CNAME	www.google.com
www.google.rw	CNAME	www.google.com

The server state is: **401 record limit exceeded**

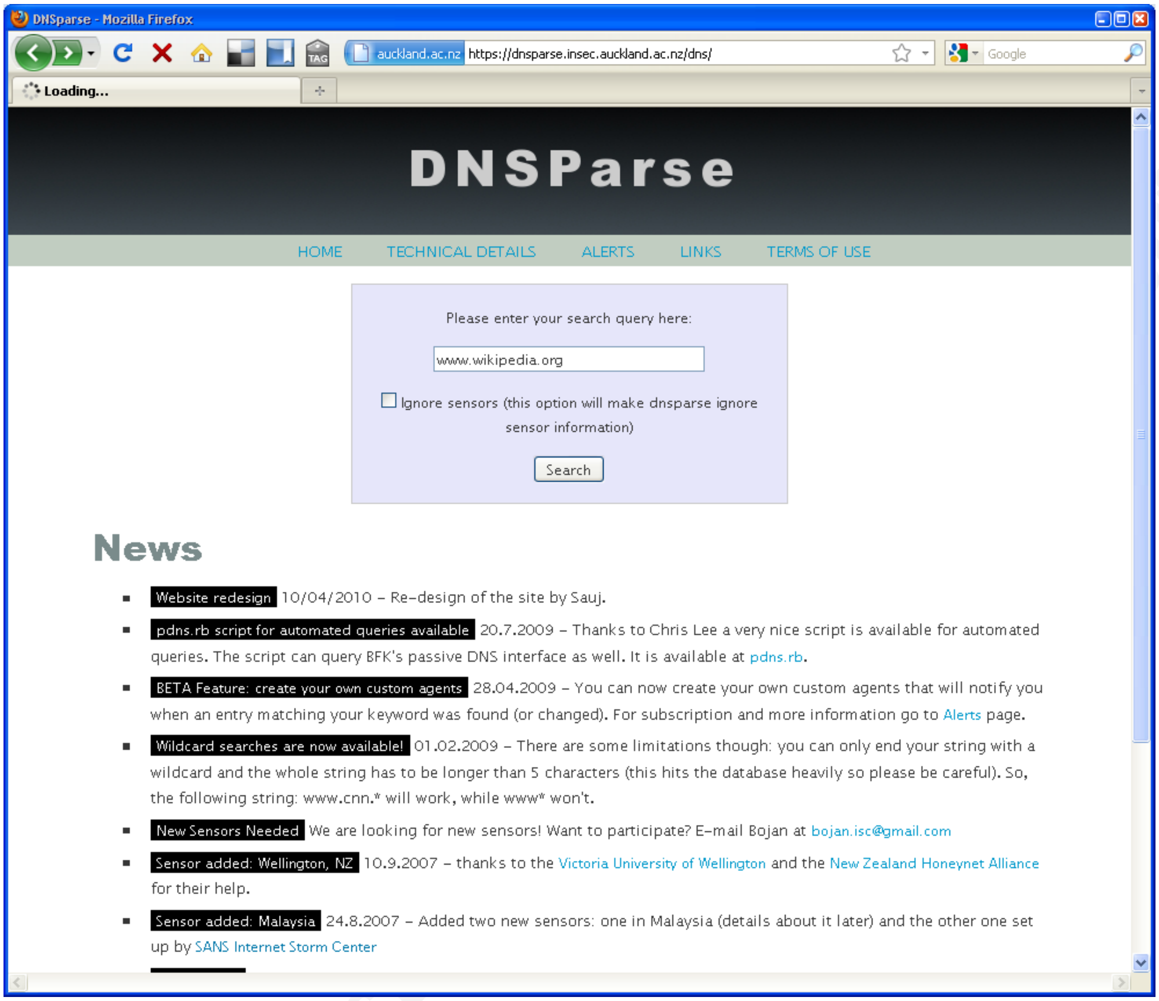
Instructions

Please enter your query in the input field above. You can enter IP addresses and domain names. A short explanation of the output follows.

University of Auckland's DNS Parse

The brainchild of Bojan Zdrnja, DNS Parse is hosted by the University of Auckland (Zdrnja, 2010). While BFK's repository returns simple resolution returns, DNS Parse provides more record types, including TXT and SOA, along with information about how long and when the query provided a resolution return. While this temporal data is ignored for the link analysis in this paper, it would prove useful for deeper forensic investigation against known malicious domains.

Andrew Hunt, sans.ahunt@gmail.com


DNSparse - Mozilla Firefox
auckland.ac.nz https://dnsparse.insec.auckland.ac.nz/dns/
Google
Loading...
DNSParse
HOME
TECHNICAL DETAILS
ALERTS
LINKS
TERMS OF USE
Search

Please enter your search query here:

www.wikipedia.org

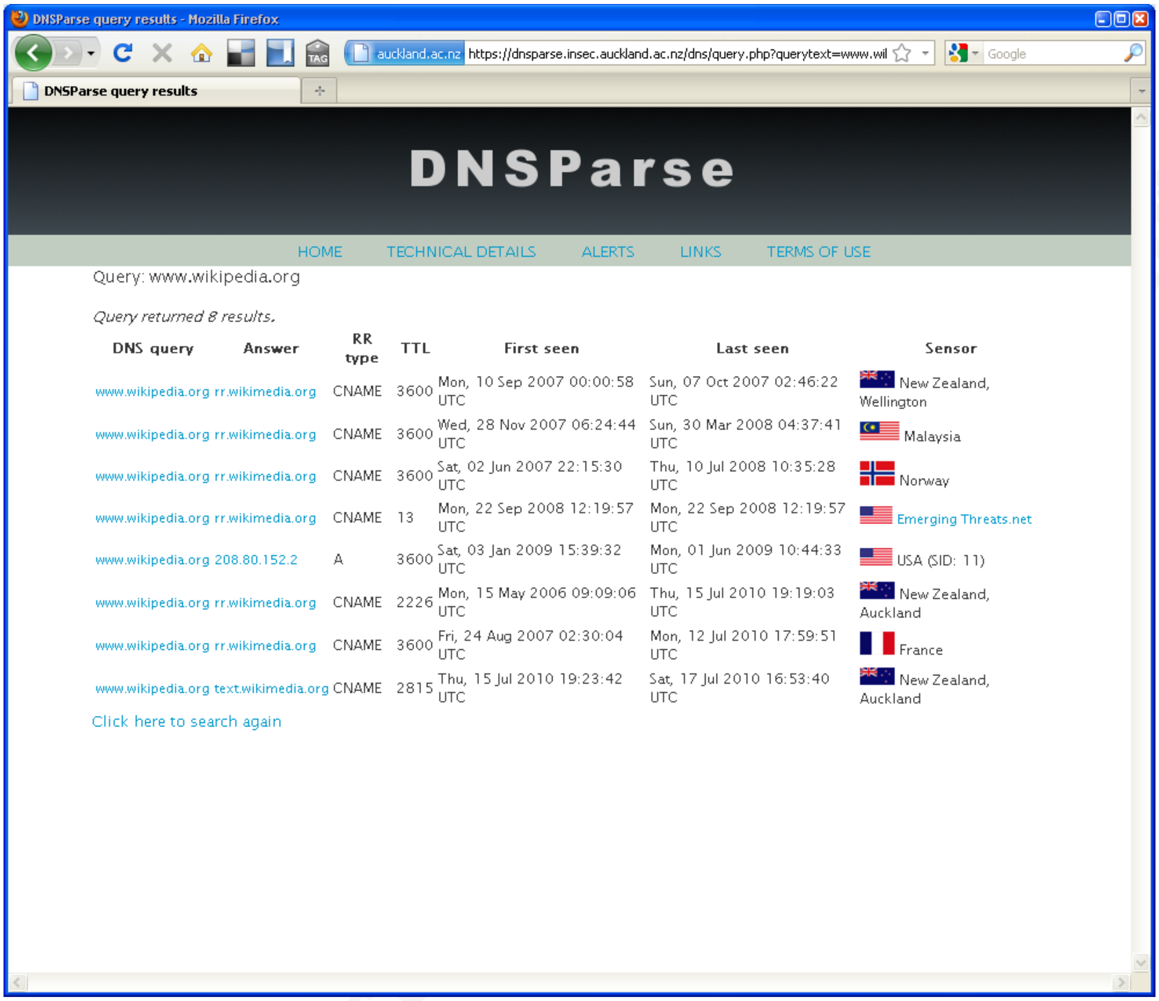
☐ Ignore sensors (this option will make dnsparse ignore sensor information)

News

- **Website redesign** 10/04/2010 – Re-design of the site by Sauj.
- **pdns.rb script for automated queries available** 20.7.2009 – Thanks to Chris Lee a very nice script is available for automated queries. The script can query BFK's passive DNS interface as well. It is available at pdns.rb.
- **BETA Feature: create your own custom agents** 28.04.2009 – You can now create your own custom agents that will notify you when an entry matching your keyword was found (or changed). For subscription and more information go to Alerts page.
- **Wildcard searches are now available!** 01.02.2009 – There are some limitations though: you can only end your string with a wildcard and the whole string has to be longer than 5 characters (this hits the database heavily so please be careful). So, the following string: www.cnn.* will work, while www* won't.
- **New Sensors Needed** We are looking for new sensors! Want to participate? E-mail Bojan at bojan.isc@gmail.com
- **Sensor added: Wellington, NZ** 10.9.2007 – thanks to the Victoria University of Wellington and the New Zealand Honeynet Alliance for their help.
- **Sensor added: Malaysia** 24.8.2007 – Added two new sensors: one in Malaysia (details about it later) and the other one set up by SANS Internet Storm Center

Figure 3: DNSParse Query Page

Bojan encourages flexible use of the product, working with Chris Lee to provide the initial automated query script upon which the modified version presented in Appendix A was based (Lee, 2008). The scope of the sensors extends far beyond New Zealand, making the quality of the returned records superior to the other tested sources. Access to the DNS Parse project is closely monitored, and is only available to security researchers that are vetted through the project. DNS Parse was extremely helpful to the success of the analyses presented in this project.


DNSParse query results - Mozilla Firefox
auckland.ac.nz
https://dnsparse.insec.auckland.ac.nz/dns/query.php?querytext=www.wil
Google
DNSParse query results
DNSParse
HOME
TECHNICAL DETAILS
ALERTS
LINKS
TERMS OF USE

Query: www.wikipedia.org

Query returned 8 results.

DNS query	Answer	RR type	TTL	First seen	Last seen	Sensor
www.wikipedia.org	rr.wikimedia.org	CNAME	3600	Mon, 10 Sep 2007 00:00:58 UTC	Sun, 07 Oct 2007 02:46:22 UTC	New Zealand, Wellington
www.wikipedia.org	rr.wikimedia.org	CNAME	3600	Wed, 28 Nov 2007 06:24:44 UTC	Sun, 30 Mar 2008 04:37:41 UTC	Malaysia
www.wikipedia.org	rr.wikimedia.org	CNAME	3600	Sat, 02 Jun 2007 22:15:30 UTC	Thu, 10 Jul 2008 10:35:28 UTC	Norway
www.wikipedia.org	rr.wikimedia.org	CNAME	13	Mon, 22 Sep 2008 12:19:57 UTC	Mon, 22 Sep 2008 12:19:57 UTC	Emerging Threats.net
www.wikipedia.org	208.80.152.2	A	3600	Sat, 03 Jan 2009 15:39:32 UTC	Mon, 01 Jun 2009 10:44:33 UTC	USA (SID: 11)
www.wikipedia.org	rr.wikimedia.org	CNAME	2226	Mon, 15 May 2006 09:09:06 UTC	Thu, 15 Jul 2010 19:19:03 UTC	New Zealand, Auckland
www.wikipedia.org	rr.wikimedia.org	CNAME	3600	Fri, 24 Aug 2007 02:30:04 UTC	Mon, 12 Jul 2010 17:59:51 UTC	France
www.wikipedia.org	text.wikimedia.org	CNAME	2815	Thu, 15 Jul 2010 19:23:42 UTC	Sat, 17 Jul 2010 16:53:40 UTC	New Zealand, Auckland

Click here to search again

Figure 4: DNSParse Results Page

OARC's Passive DNS Repository

Internet Systems Consortium (ISC)'s Operations Analysis and Research Center (OARC), under Keith Mitchell and Paul Vixie, has been developing a passive DNS repository of its own (Mitchell, et al, 2007). While its development has been well documented in public, it is only available to authenticated users that have access through OARC's contribution or vetted research programs. This repository was not used for this project, but deserves note for its development in the field.

Other Sources

Other passive DNS repositories are available by commercial vendors for a fee. An organization might also install its own DNS sensor, or build a passive DNS database from its own historical log data. Common sources to extract historical name resolutions include proxy, Windows DHCP server, DNS, and individual browser history logs. The creative use of a centralized log store yields a trove of data that can be useful in building a searchable archive of an organization's resolution history.

Combining Passive DNS with IP Data

DNS data on its own will provide some interesting intelligence about how domains are related to provisioners. It leads to some other interesting questions. How are provisioners related themselves? Does an adversary prefer particular ISPs? Are there geographic jurisdictions they prefer to operate in? The data to start addressing these questions is easily attained by adding Autonomous System Number (ASN) query results provided by Team Cymru's IP-to-ASN Mapping service for each passive DNS returned IP address, which includes network, country code, ASN number, and ASN description (Team Cymru, 2010). An Autonomous System (AS) is a group of IP networks run by a network operator, more commonly called an ISP, with a single clearly defined routing policy. The Autonomous System Number is used as an identifier to allow the AS to exchange dynamic routing information with other network providers. This allows packets to route from one provider to another throughout the Internet. An example return from a scripted ASN query combining the domain and the IP2ASN queries is below.

```
google.com |    72.14.204.99 |     72.14.204.0/23 | US | 15169 | GOOGLE - Google Inc.
google.com |    72.14.204.99 |     72.14.192.0/18 | US | 15169 | GOOGLE - Google Inc.
google.com |   72.14.204.103 |     72.14.204.0/23 | US | 15169 | GOOGLE - Google Inc.
google.com |   72.14.204.103 |     72.14.192.0/18 | US | 15169 | GOOGLE - Google Inc.
google.com |   72.14.204.147 |     72.14.204.0/23 | US | 15169 | GOOGLE - Google Inc.
google.com |   72.14.204.147 |     72.14.192.0/18 | US | 15169 | GOOGLE - Google Inc.
google.com |   72.14.204.104 |     72.14.204.0/23 | US | 15169 | GOOGLE - Google Inc.
google.com |   72.14.204.104 |     72.14.192.0/18 | US | 15169 | GOOGLE - Google Inc.
```

The response records are sometimes duplicated due to overlapping network mappings. These are due to subleased networks from larger providers. See Appendix B for a script to automate the collection of the ASN data. Appendix C contains a script to convert the ASN output into Graphics Definitions Format (GDF), a representation suitable for visualization.

Visualization Primer

Bringing passive DNS and ASN data together provides a large pool of linked data, so large that it can be difficult to differentiate between legitimate and malicious relationships. The analyst needs a representation of the data that draws out the high-confidence links. Visualizing the data allows a more natural analysis for the analyst. The chosen tool for this research, GUESS, provides the analyst the flexibility to highlight artifacts of interest, allowing them to more easily draw conclusions from large conglomerate data sets.

GUESS

The Graph Exploration System (GUESS) is an open-source visualization tool provided by Eytan Adar. It uses a domain-specific language, Gython, that marries Java, Python, and Jython libraries into a slick visual objects field with a python interpreter command line (Adar, 2005). This allows maximum flexibility for an analyst to manipulate graph objects on-the-fly, make notations and highlights, and automate routine visual processing into modules. GUESS is a powerful visualization tool and was used to generate the graphs used in this paper.

Graphics Definition Format

The default input format for GUESS, the Graphics Definition Format (GDF), allows for the standard definition of nodes (dots) and edges (lines linking the dots). It consists of two blocks, each starting with a definition statement followed by the entries.

Definitions

Nodes and edges must be defined consistently so that the importing parser understands how to interpret the fields. This enables the translation of the entries into viewable objects. The definition includes two blocks, one for nodes, the other for edges between the nodes. The following are examples of node and edge definitions used in this demonstration.

Nodes

Nodes are an atomic entry representing a tangible artifact. IP address, country, ASN number, domain name, and ISP name are all individual artifacts. Think of a node as a thing, or noun, to be described.

```
nodedef> name,description VARCHAR(12),color,style
```

The node definition designed for this exercise is fairly simple. It includes the name of the object created, a brief optional description, the color of the node, and a style format. The styles are interpreted and rendered by the GUESS engine. Here are some examples of node entries. The following node is derived from the ASN Lookup data, providing ASN network and ISP data. It is represented by a white square labeled '47142'.

```
"47142","STEEPHOST-AS SteepHost DC-UA-SteepHost.COM
```

Andrew Hunt, sans.ahunt@gmail.com

```
Datacentre Allocation",white,1
```

This node is a network range represented by a small gray circle with the name of the network.

```
188.124.0.0/19,,gray,2
```

The following node is a domain name, represented by a small gray circle.

```
binglbalts.com,,gray,2
```

The last node is an IP address, represented by a small white square.

```
188.124.16.104,,white,1
```

Edges

If nodes are the nouns, edges represent the verbs and adjectives of the graphical vocabulary. With arrows, they can indicate direction, providing action or causality among objects in the graph. Through color and weight, they can describe the relationship between nodes. Because the direction of the relations is not readily apparent with DNS and IP resources, this research only represents through color the DNS record or geolocational relationships between the nodes.

```
edgedef> node1,node2,color
```

The edges in this demonstration are very simple, with the connecting nodes defined and the representation of that relationship only being color. With the following example, you see the domain and its resolved IP address related by a blue color, indicating an 'A' record resolution.

```
networkads.net,95.143.193.60,blue
```

Nameservice records are represented by the color red.

```
networkads.net,ns3.everydns.net,red
```

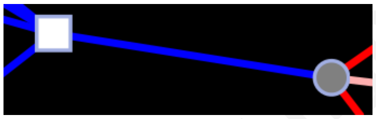
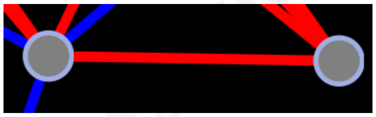
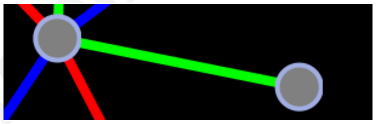
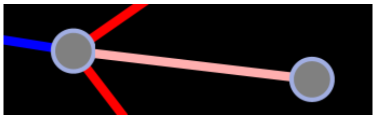
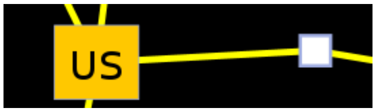
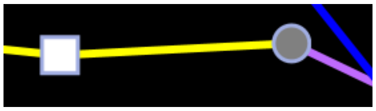
ASN to country location relationships are represented in yellow.

```
"15135",US,yellow
```

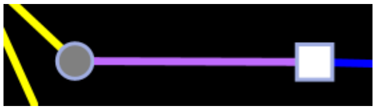
Andrew Hunt, sans.ahunt@gmail.com

Relationships Table

The following table illustrates the relationships and their colors.

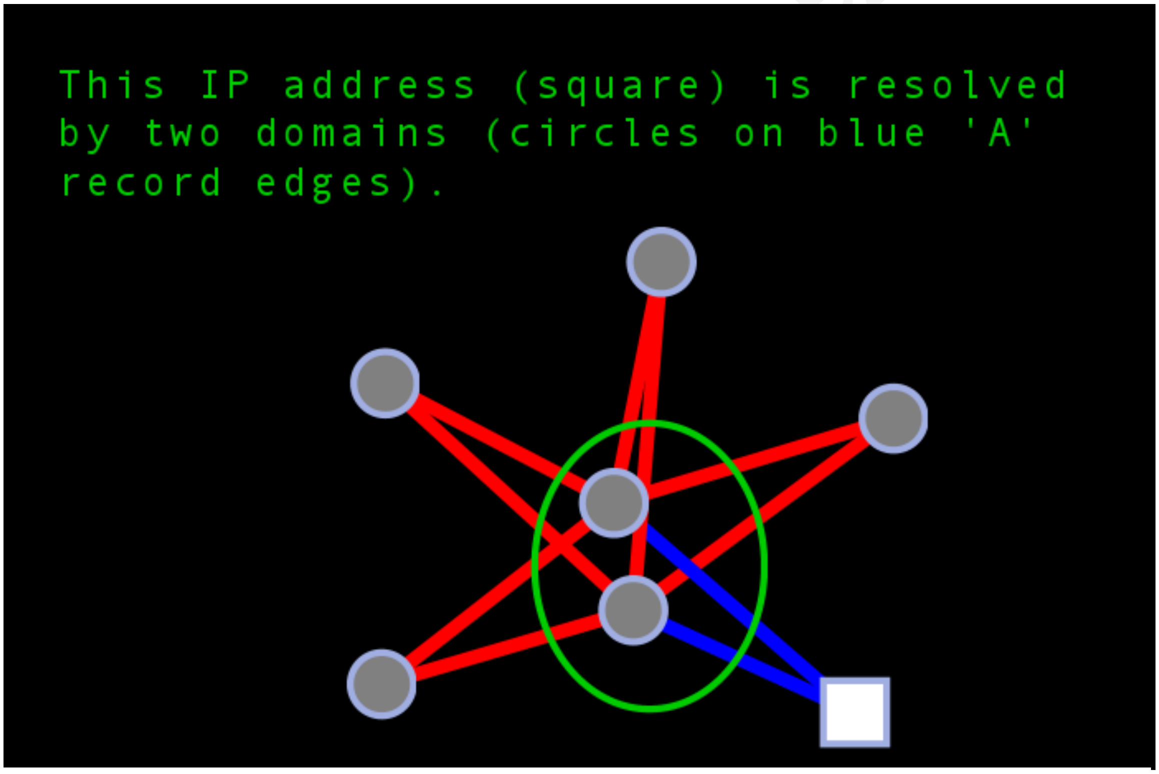
Relationship	Color	Example
DNS 'A' record	Blue	
DNS 'NS' record	Red	
DNS 'MX' record	Green	
DNS 'CNAME' record	Pink	
DNS 'PTR' record	Purple	Available in DNSParse, but not represented in the example data sets
DNS 'SOA' record	White	Available in DNSParse, but not represented in the example data sets
DNS 'TXT' record	Brown	Available in DNSParse, but not represented in the example data sets
ASN to Country	Yellow	
ASN to Network	Yellow	

Andrew Hunt, sans.ahunt@gmail.com

IP to Network	Purple	

Basic Analysis with GUESS

After the passive DNS and ASN data is collected and groomed into GDF format, it can be imported into GUESS. Upon initial rendering, apply the Generalized Expectation Maximization(GEM) algorithm (Borman, 2004). This applies a clustering algorithm that returns intuitive rendering of the relationships with a relatively quick response. The analyst should be able to recognize basic resolution patterns. The first example demonstrates following an IP address back to its resolved domains. The white square has blue 'A' record resolutions to two domain name nodes.

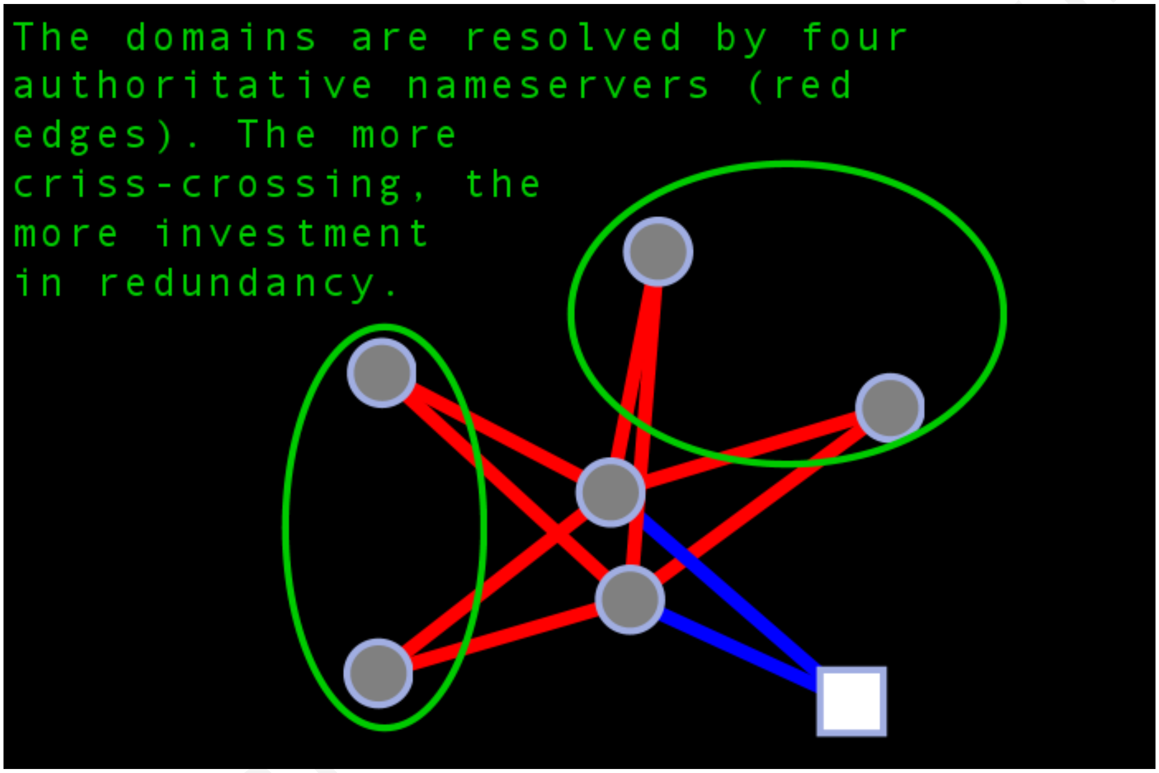


This IP address (square) is resolved by two domains (circles on blue 'A' record edges).

Figure 5: Resolution of two domains to a common IP.

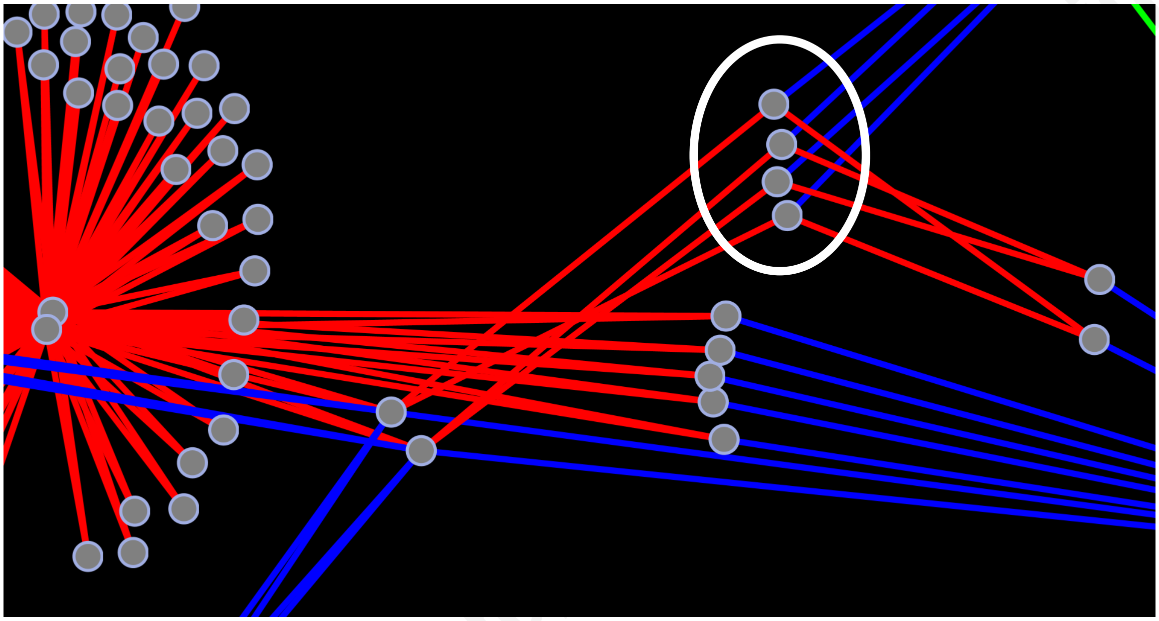
Andrew Hunt, sans.ahunt@gmail.com

The same example shows how to trace the domains to their authoritative nameservers. Each domain may have several supporting NS records. In this case, the two domain names are each supported by the same four authoritative nameservice providers. Red 'NS' edges indicate investment as a monetary transaction almost always occurs to lease support for each authoritative nameservice. More red links indicate a greater investment in support of the domain.


The domains are resolved by four authoritative nameservers (red edges). The more criss-crossing, the more investment in redundancy.

Andrew Hunt, sans.ahunt@gmail.com

Several hosting behaviors lead to interesting relationships. Criss-crossed red nameservice records reveal nameservice redundancies. Heavily redundant structures require management, indicating a greater monetary investment or importance in that linkage. These cross-linkages may also indicate points of weakness that required additional investment to maintain access.



Andrew Hunt, sans.ahunt@gmail.com

Also indicative of nameservice redundancy are singular domains supported by more than a few authoritative nameservers. As is demonstrated with 'treatsudden.com' below, the owner clearly wants that domain to resolve no matter what. It is interesting as most domain names only have two or three supporting authoritative nameservers. This domain has six.

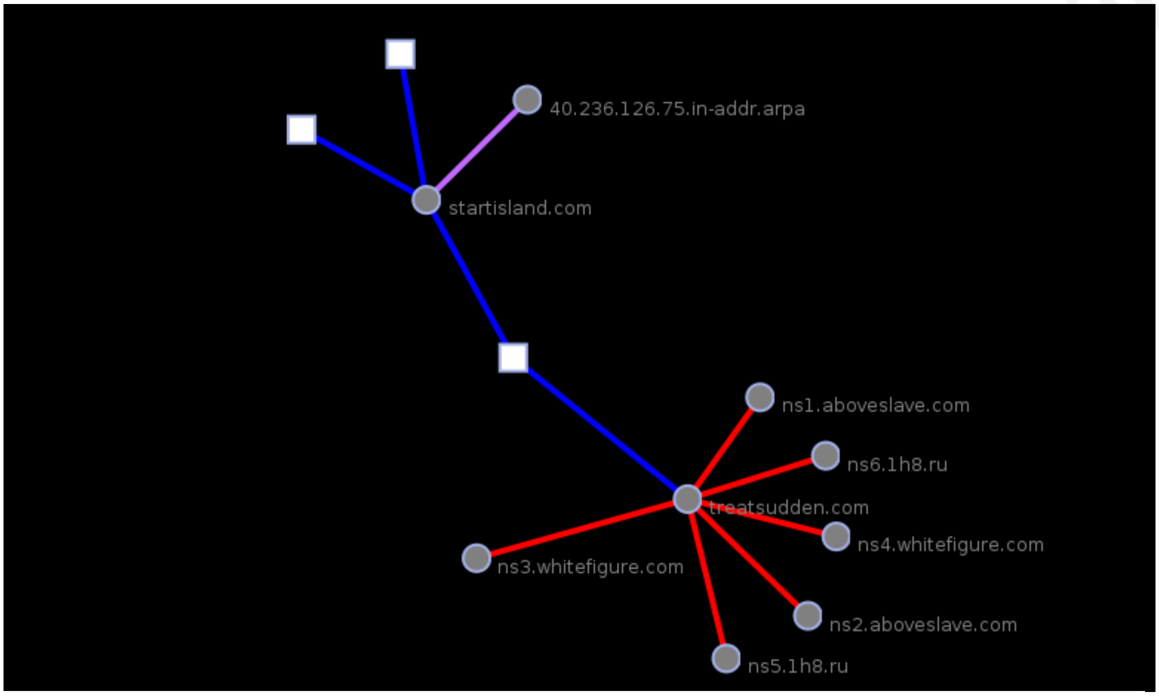
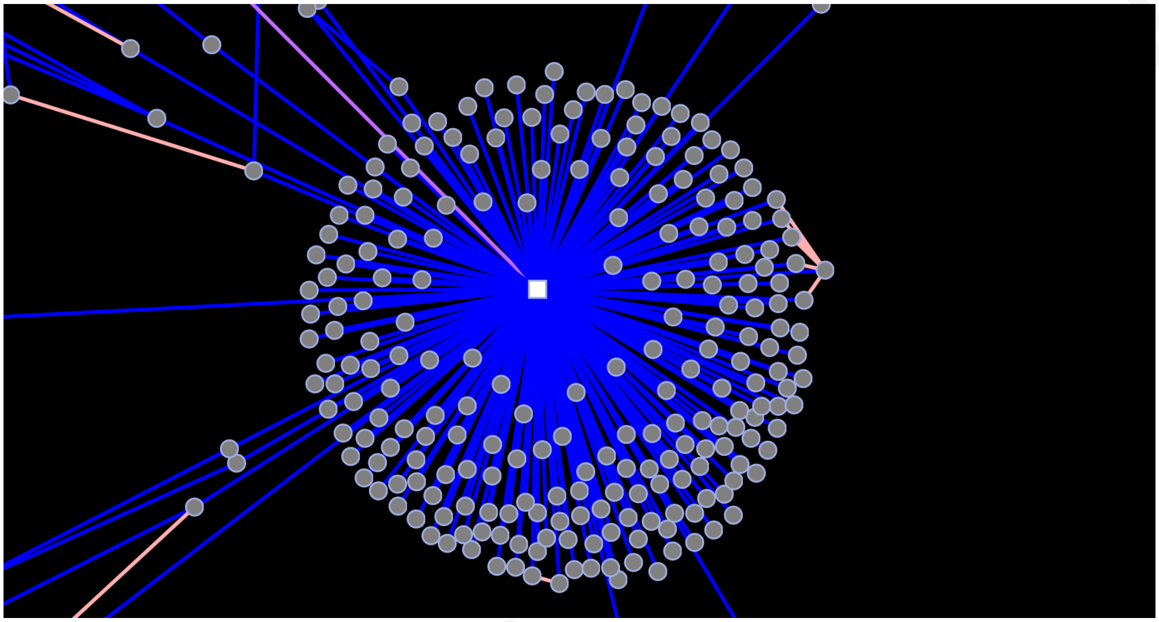


Figure 8: An example of excessive NS support for a domain.

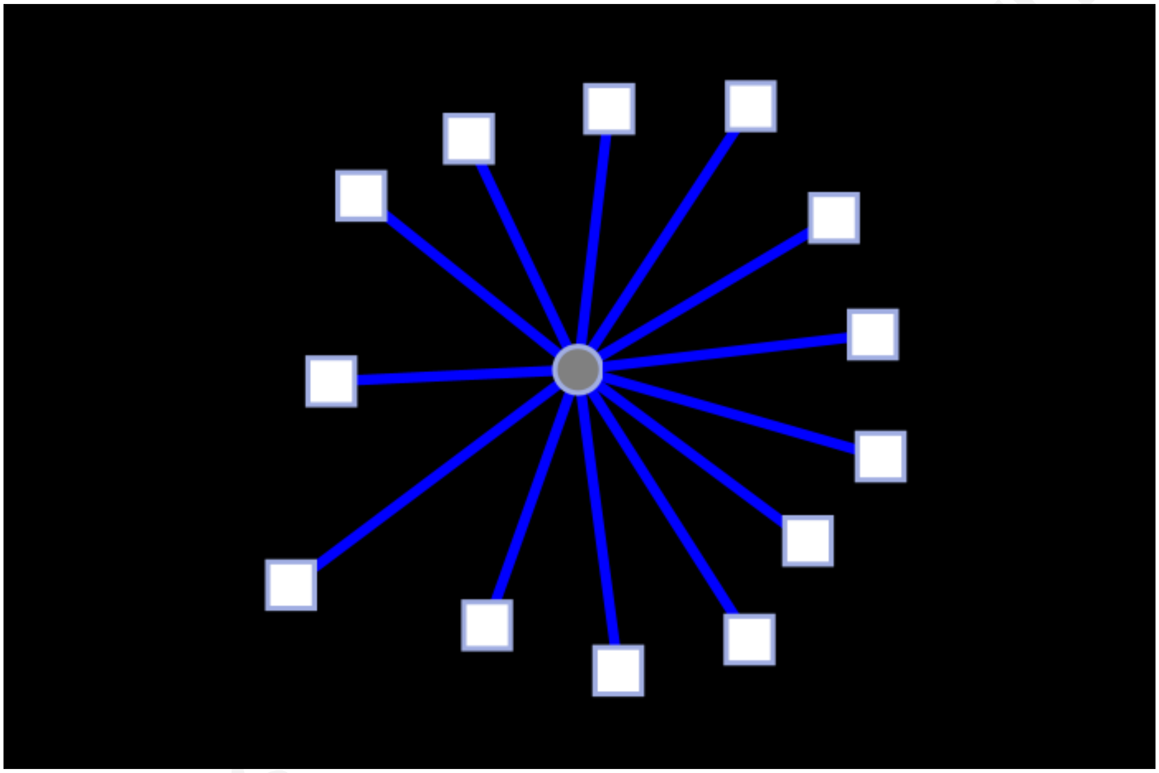
Andrew Hunt, sans.ahunt@gmail.com

A blue 'A' record bloom around a square IP address node indicates a virtual hosting service supporting many domains from the same or proximately close IP addresses. Dynamic domain hosters, such as DynDNS and ChangeIP, can appear like this. Large ISPs, such as Cogent or Saavis, also demonstrate this pattern.



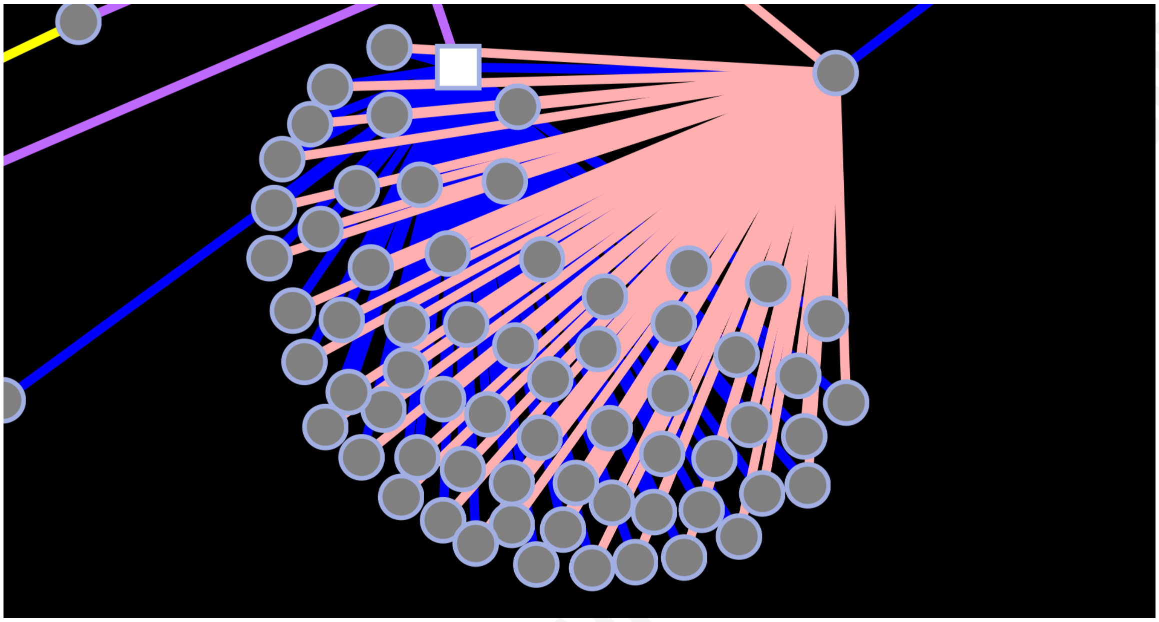
Andrew Hunt, sans.ahunt@gmail.com

Alternatively, a blue bloom around a round domain node may demonstrate hosting provided by fast-flux domain resolution, a critical diversity component for malicious botnets. This representation shows how a domain is supported by many resolved IP addresses. Passive DNS collection, as a historical record of real resolutions on the network, is not affected by the rapid cycling of these IP address as all are recorded as they resolve. Note the domain circle surrounded by IP squares.



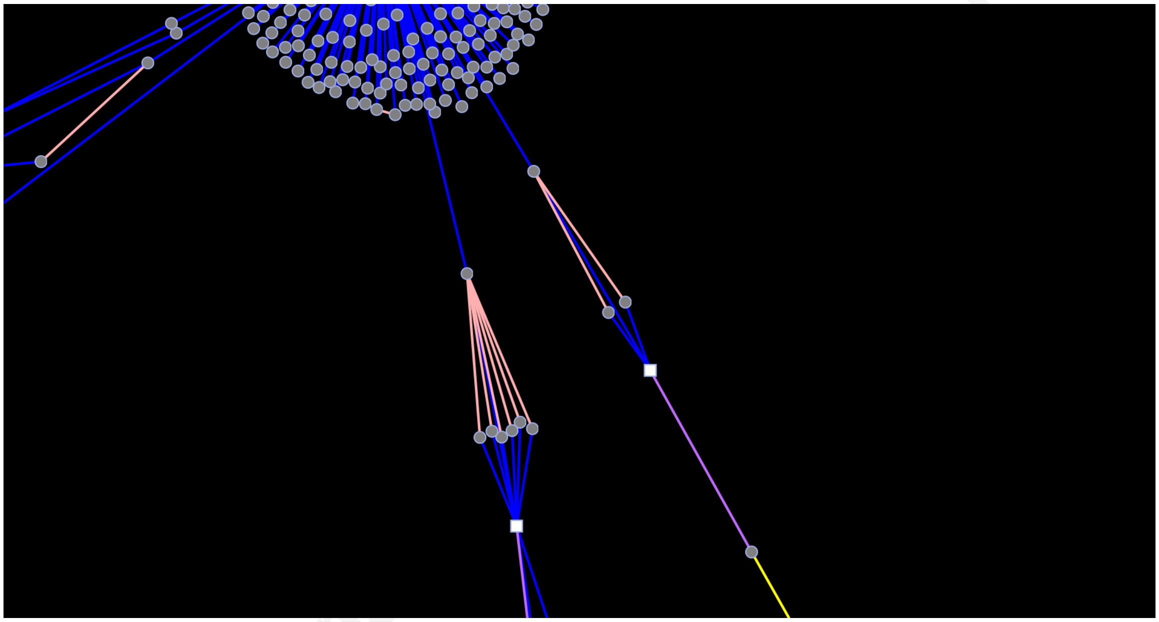
Andrew Hunt, sans.ahunt@gmail.com

CNAME record edges can form pink sprays. These indicate a collection of CNAME aliases to nearby domain names.



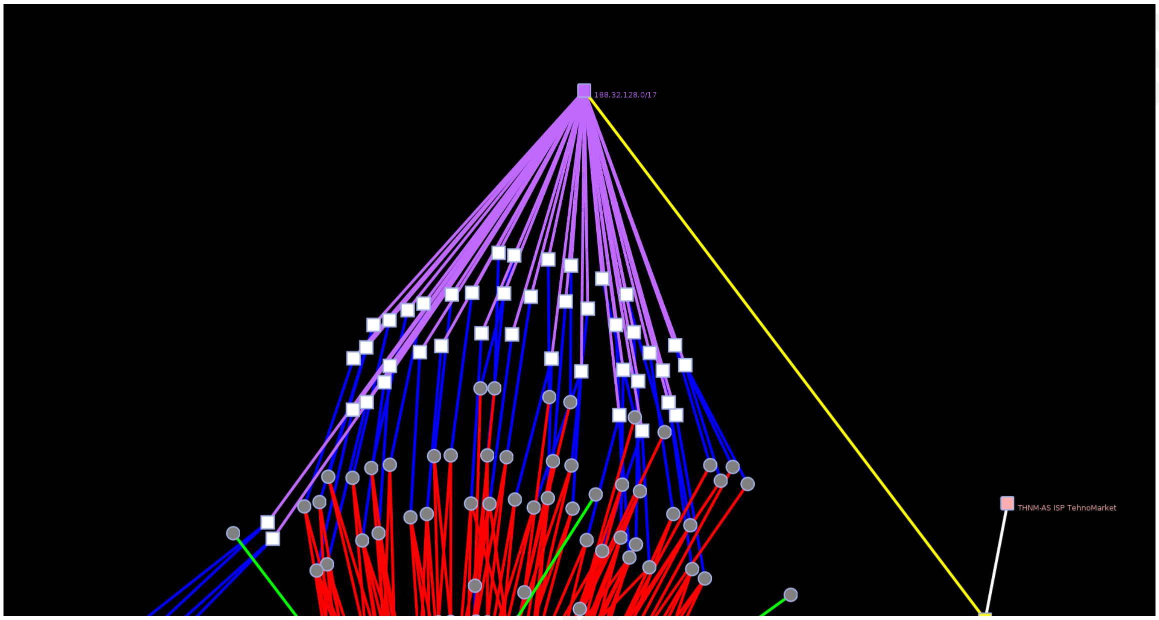
Andrew Hunt, sans.ahunt@gmail.com

Another indication of support across domains are blue A records to pink CNAME records leading to other blue A records. This demonstrates traversal across IP addresses, usually across ISPs. Typical examples of these traversals are domains supported by Google hosting services. A domain occurring at a third-party registrar and using the third-party's nameservices to redirect traffic to the services hosted at Google Apps (http://www.google.com/apps/intl/en/business/index.html) virtual hosting space will appear like this in the graph. For a malicious hoster, this may indicate domain support diversion.



Andrew Hunt, sans.ahunt@gmail.com

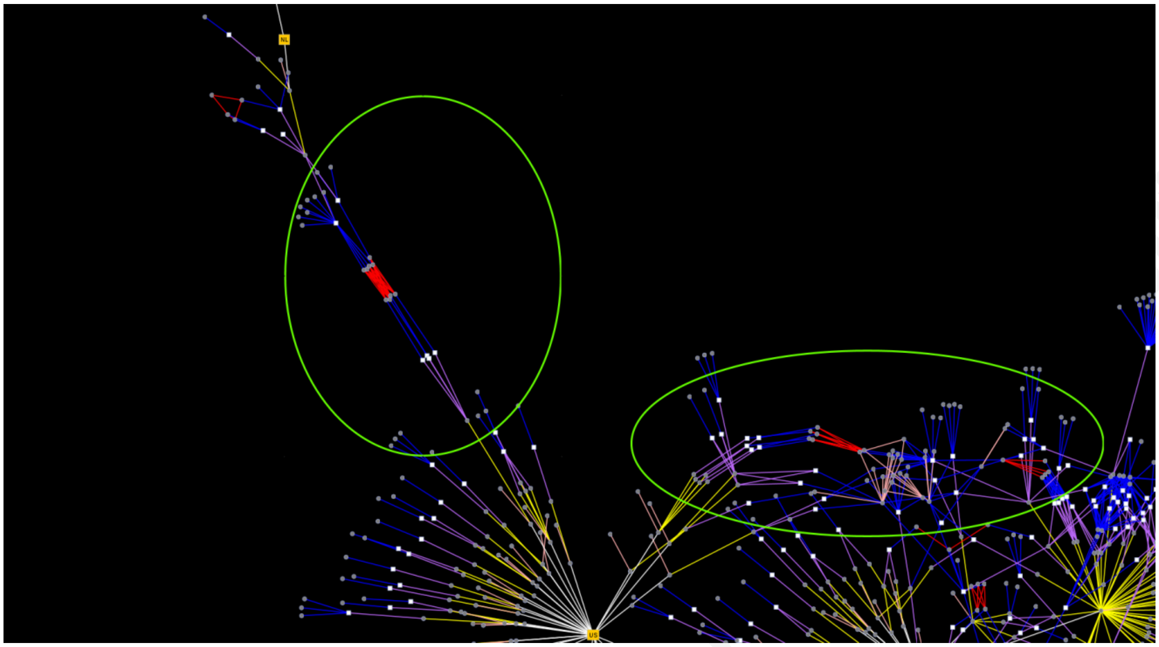
Purple edges clustered around a network block node, as below, may indicate leased space by an aggressor. If the netblock edges cluster around a yellow star with a round ASN node, it may indicate their favoring of a particular ISP for hosting.



Andrew Hunt, sans.ahunt@gmail.com

Long connecting paths with criss-crossing redundancies at multiple levels, red, yellow, and blue, indicate important linkages between assets across national boundaries. As seen in the following example in the left circle, the assets in the Netherlands have a high amount of nameservice redundancy to fortify the resolution capability with its assets in the United States. Another branch highlighted on the right shows the same type of high-redundancy links pulling together resources between the United States and China.

Andrew Hunt, sans.ahunt@gmail.com



Single linkages from the country level all the way out to the domain level are typically not interesting. These usually indicate singularly compromised IP addresses or domains that redirect to some other structure at the application layer, e.g. a malicious IFRAME. This will not stand out in this analysis.

Key Considerations

With this rich data source, many interesting facets begin to emerge. However, there are some important points and caveats an analyst must remember when working with passive data.

- Take the results here with a grain of salt. You know what you know, the rest is based on the assumption of relation due to hosting proximity. These suppositions are only that until substantiated and supported by outside evidence.

- Be mindful of scope creep. Sometimes nominal data gets pulled into the results when malfeasance is hosted in the same places as legitimate domains. This will become evident in the case study with Cogent, a large ISP hosting many domains. This can also happen when the too many recursions are taken beyond the scope of a malicious domain. As was mentioned earlier, eventually everything on the Internet relates to everything.

- ISP location is not necessarily an indication of malicious activity. The Internet is truly a global hosting environment where data can be provided, moved, or redirected to and from anywhere. Grouping by geolocation is a convenient way to tease out cross-jurisdictional activity. It in no way infers national policy or intent. However, it may indicate places where weak laws or enforcement provide safe havens for malfeasance based on demonstrated behavior.

- The impact of privacy laws may be a consideration depending on the organization and investigator's station in pursuing this data. It is unlikely that the collection of authoritative DNS responses without client request data would run afoul of privacy laws as the resolution data is published publicly to the Internet in order to route services. However, care should be taken when collecting client information for requested queries as this could fall under privacy protections for personal information. Even if organizations determine they have the authority to collect their client query data, which provides many useful artifacts when tracing infection, should they choose to engage in data sharing agreements with other organizations to access a more robust passive DNS dataset, they may need to limit access to client data.

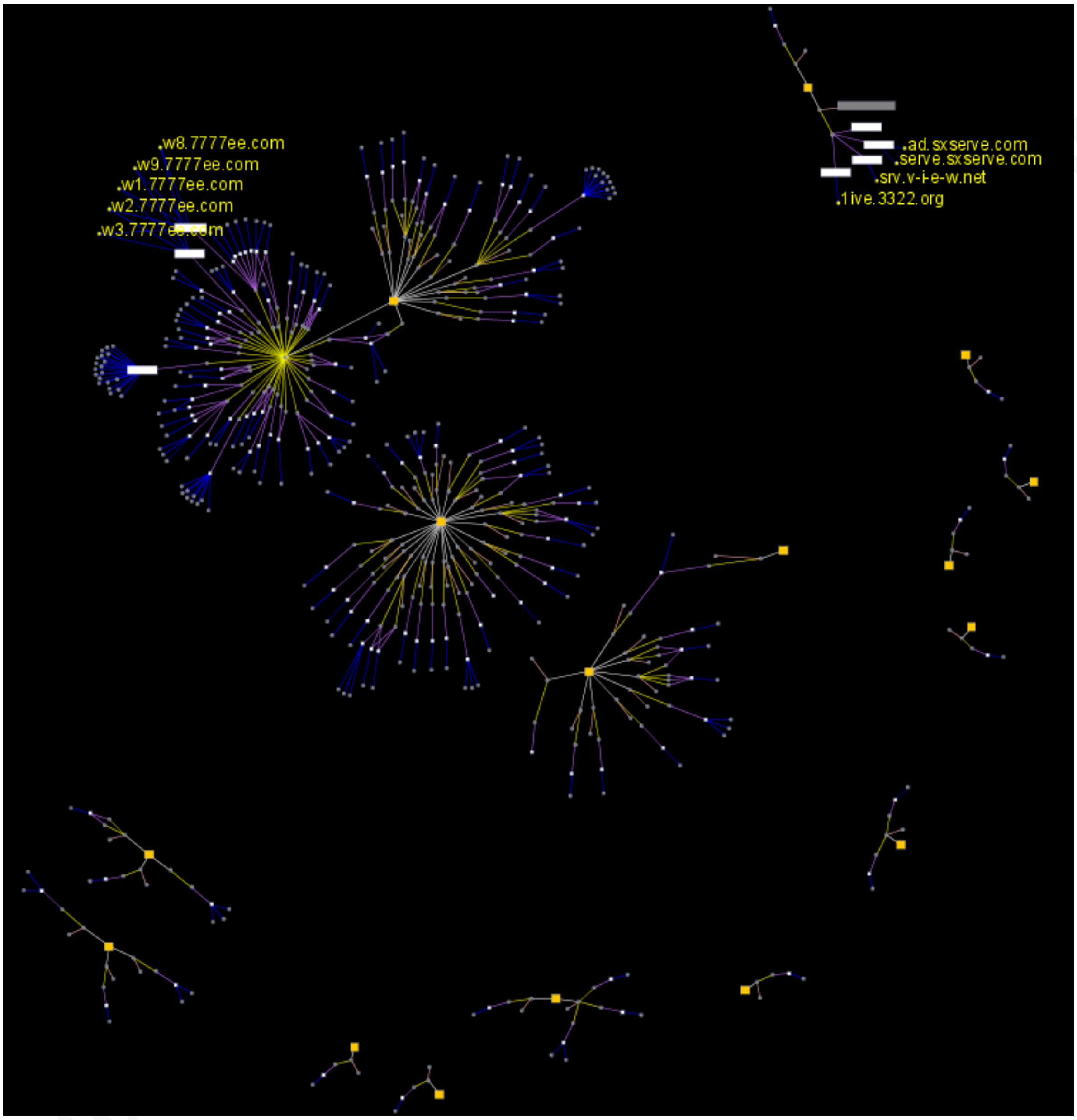
After considering these points, some very interesting indicators begin to emerge.

Case Study: ActiveX Zero Day, July 2009

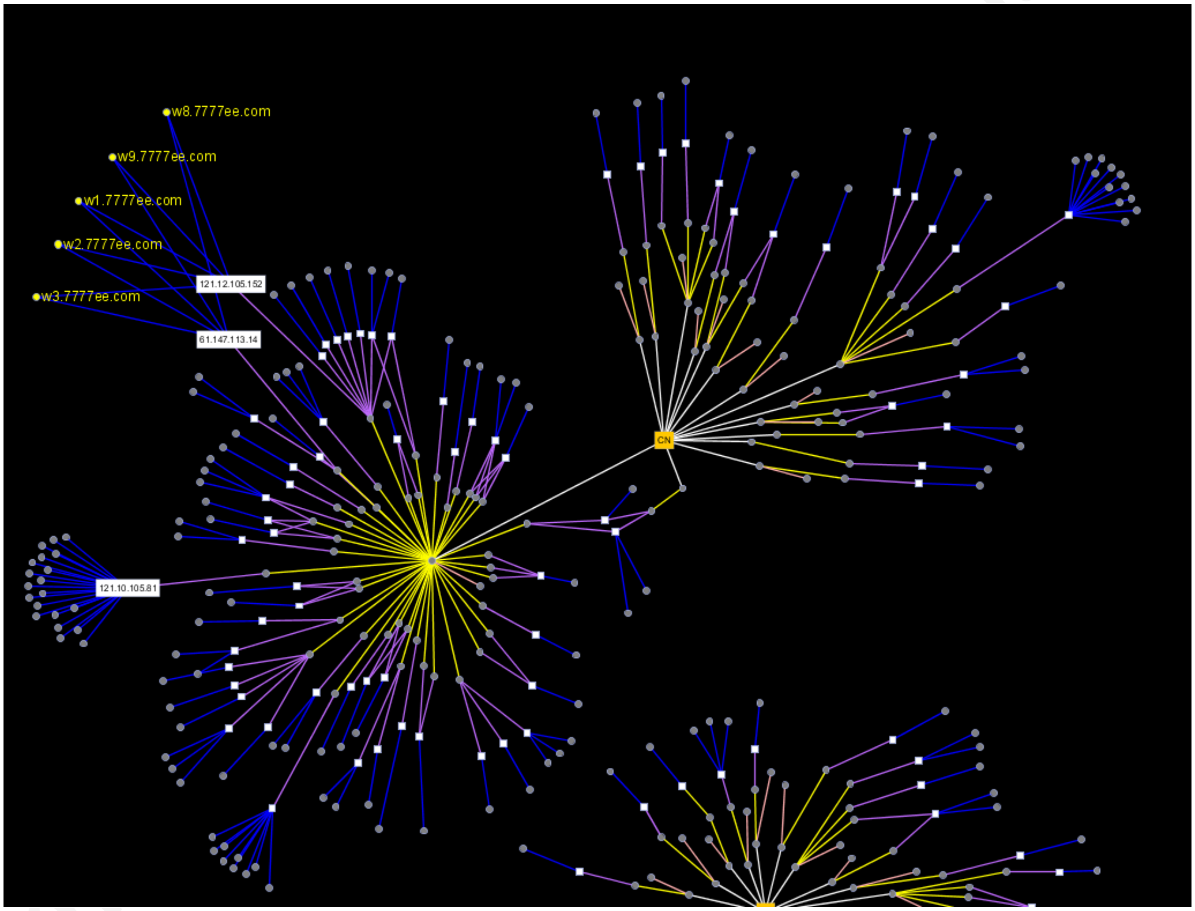
The original impetus for this research developed around the Office Web Component ActiveX zero day attacks in July 2009 (de Beaupre, 2009; Ludwig, 2009). So many hosts were compromised; the question came up "Who is making money supporting all of this infrastructure?" Having used BFK's DNSLogger for historical DNS data in the past, it was the logical place to begin searching with the known domain artifacts (Weimer, 2010, March 15, BFK).

First Pass

Starting with a list of compromised and malicious control domains provided by SANS (http://isc.sans.edu/diary.html?storyid=6739), DNSLogger returned hundreds of similarly named and seemingly nefarious sites hosted at several groups of IP addresses.


w8.7777ee.com
w9.7777ee.com
w1.7777ee.com
w2.7777ee.com
w3.7777ee.com
ad.sxserve.com
serve.sxserve.com
srv.v-i-e-w.net
1ive.3322.org

Within this initial pass from the known list, the compromised sites were fairly diverse which left few relations to draw. The domains and IP addresses themselves were not indicating many relationships, with the exception of the 7777ee.com group, sxserve.com, and v-i-e-w.net via close-kit network groupings. There were two interesting redundancies with the 7777ee.com domains: each resolved to two IP addresses and both were hosted at the same ISP through different network blocks. The IP address 121.10.105.81 also stood out as many of the compromised domains resolved to it via the same ISP. It seemed the aggressor favored compromising hosts on this ISP's networks, perhaps indicating a weak security posture during the prestaging activities that preceded this campaign.



The nodes around the Netherlands ISPs were also interesting as their mappings diverged from the typical one-off compromises per country. The network block cluster around the 85.17.0.0/16 block indicated something more concentrated was going on. Each of the service domains were resolving to separate IP addresses around this netblock. This netblock spread was marked of interest, deserving further scrutiny.

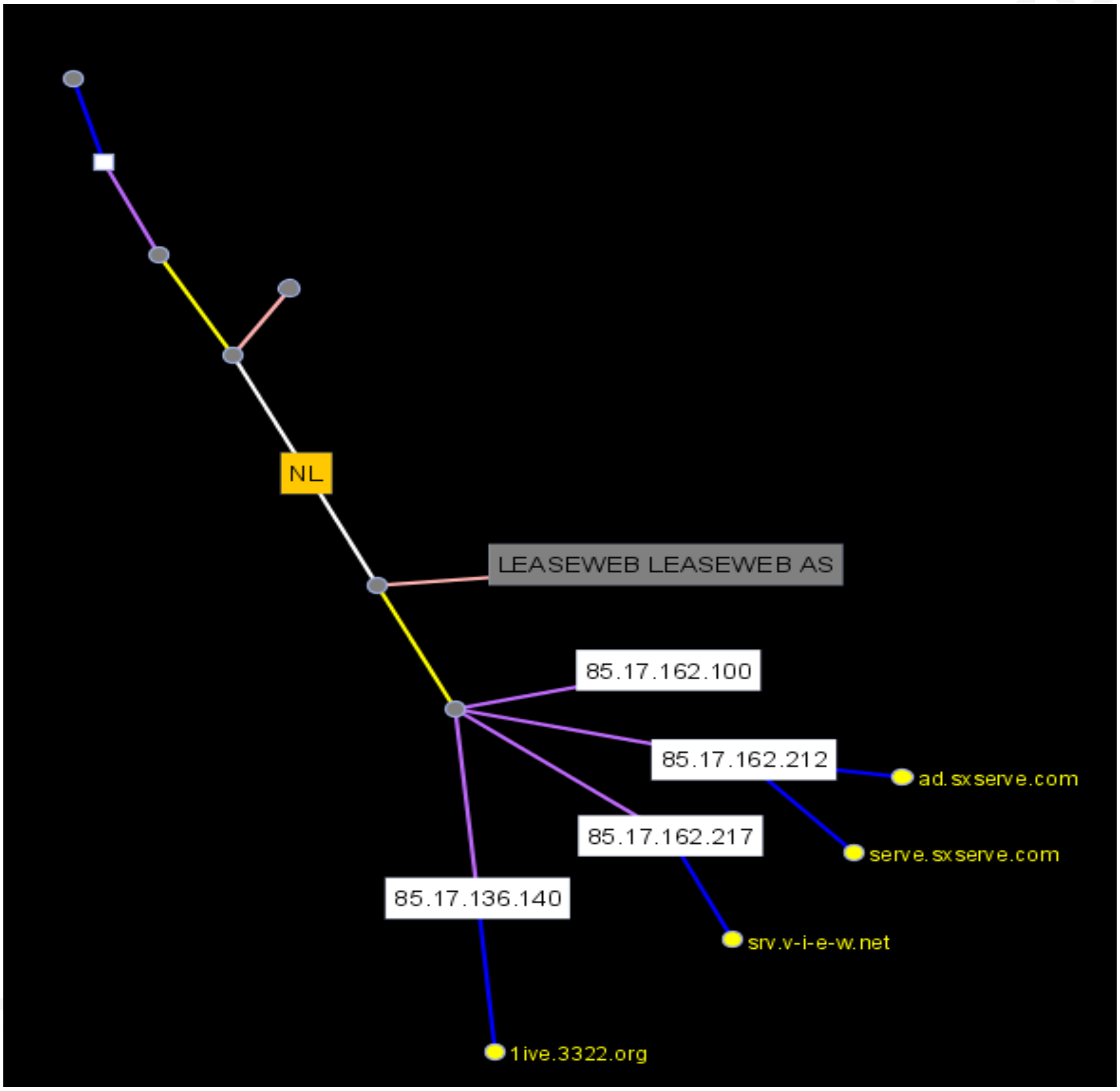
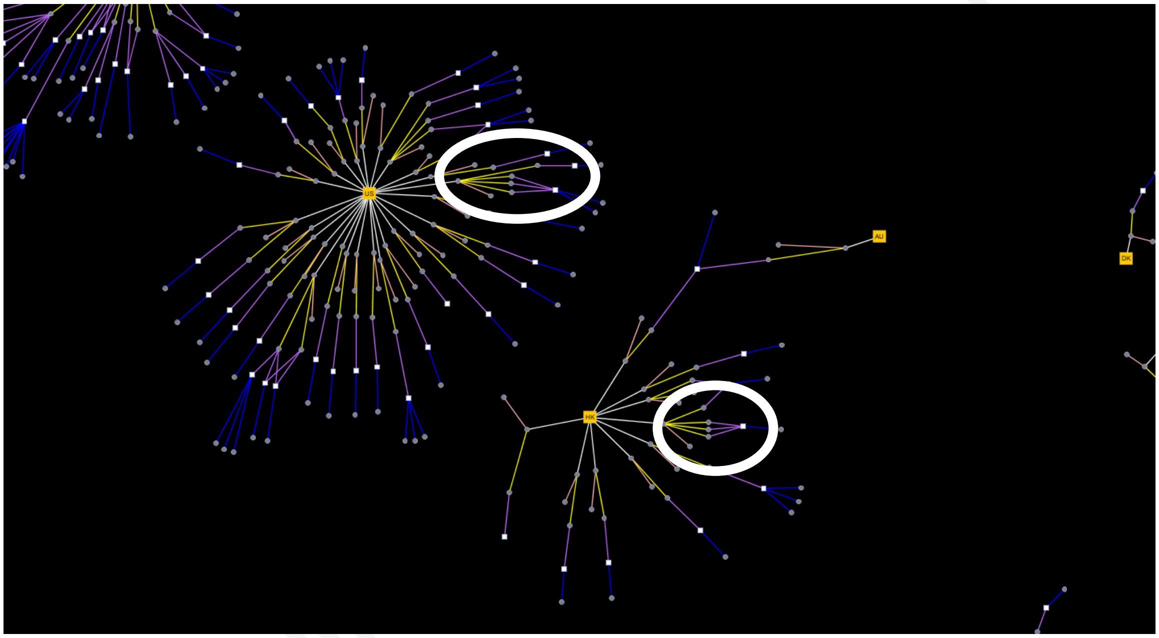


Figure 17: ISP that is apparently favored with this particular actor.

Andrew Hunt, sans.ahunt@gmail.com

Another interesting anomaly was the netblock-to-ASN spreads that occurred in the United States and Hong Kong clusters. However, it was determined that the ASN queries returned overlapping network block information for the queried IP address. This duplicate entry lead to the overlapping blocks mapping back to the same ISP. These network-to-ASN point-to-purple spray-to-yellow spray-to-point structures happen occasionally and may be ignored. The individual points themselves did not reveal any relations of interest. In the absence of other indicators, 'domains hosted from ISPs in a common country' was a tenuous relationship for drawing conclusions. Better conclusions would require more data.



Andrew Hunt, sans.ahunt@gmail.com

Second Pass

To get more data, another recursive passive DNS pass was conducted on the next level of immediate relations to the noted artifacts of interest. As soon as they were mapped, interesting links began to emerge.

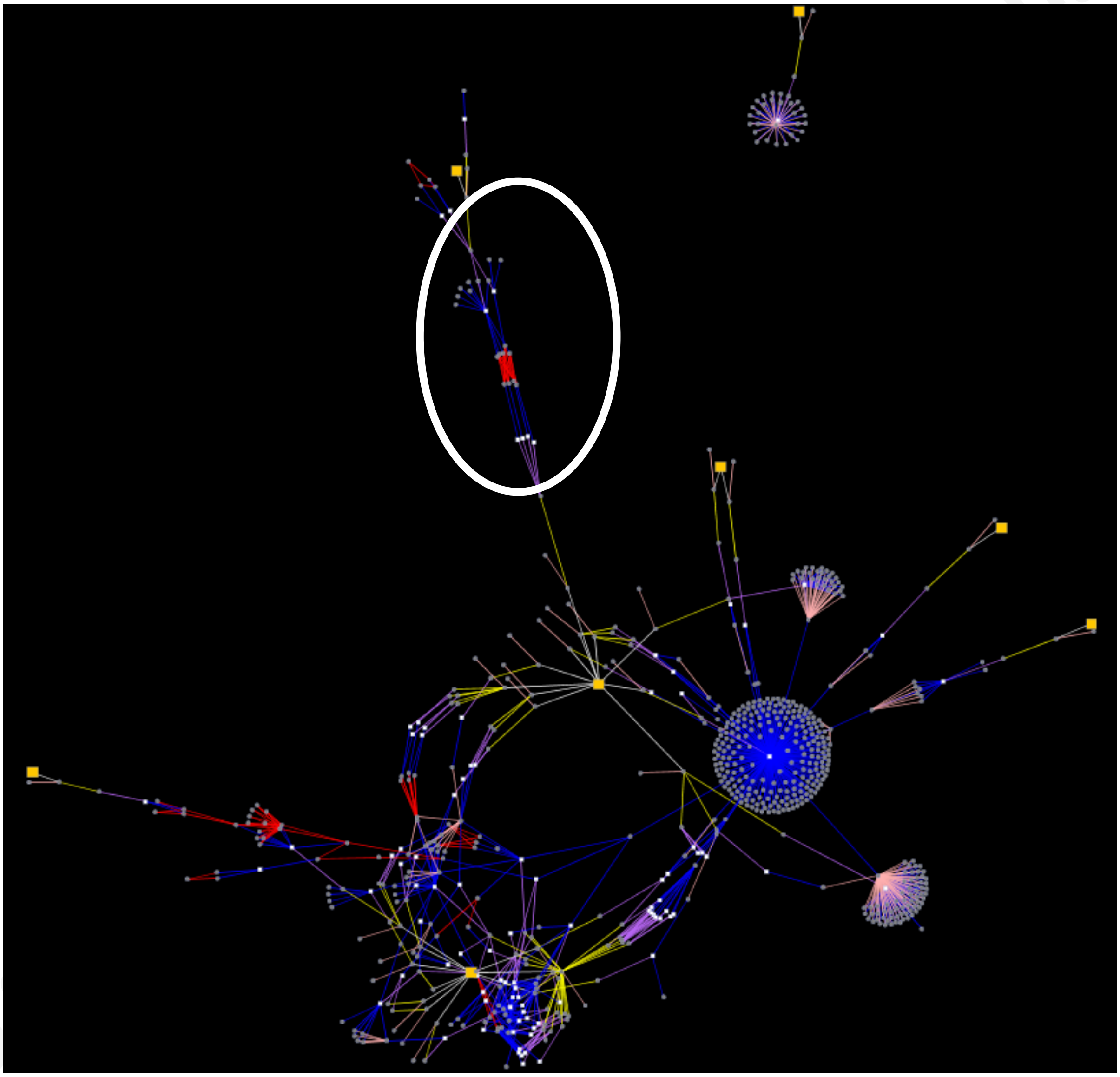


Figure 19: Second pass of passive DNS data with ASN query results.

Andrew Hunt, sans.ahunt@gmail.com

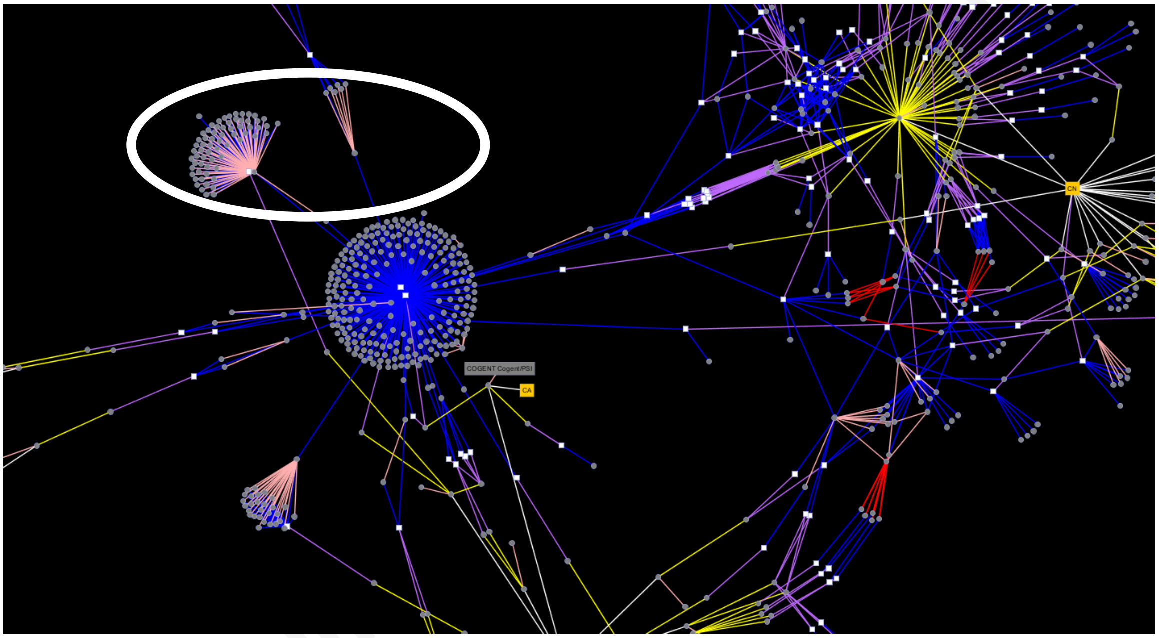
One immediately saw interesting relationships emerging between the known compromised hosts across national boundaries. This was an important distinction, as it demonstrated how the aggressors got users from their locally compromised redirect servers to the aggressors' established hosting and command-control services in jurisdictions of their choice. This was demonstrated through the long purple-to-blue-to-red criss-crossed-to-blue-to-purple linkages in the center and left side of the graph. There were also a few interesting clusters of pink CNAME references that were marked for further exploration.

Third Pass

This targeted dataset supported continuing to a complete set of data for all of the original first pass data. Based on the original dataset results, each of the passive returns was explored until the branches came full circle, diverged to clearly legitimate sites, or reached a dead end.

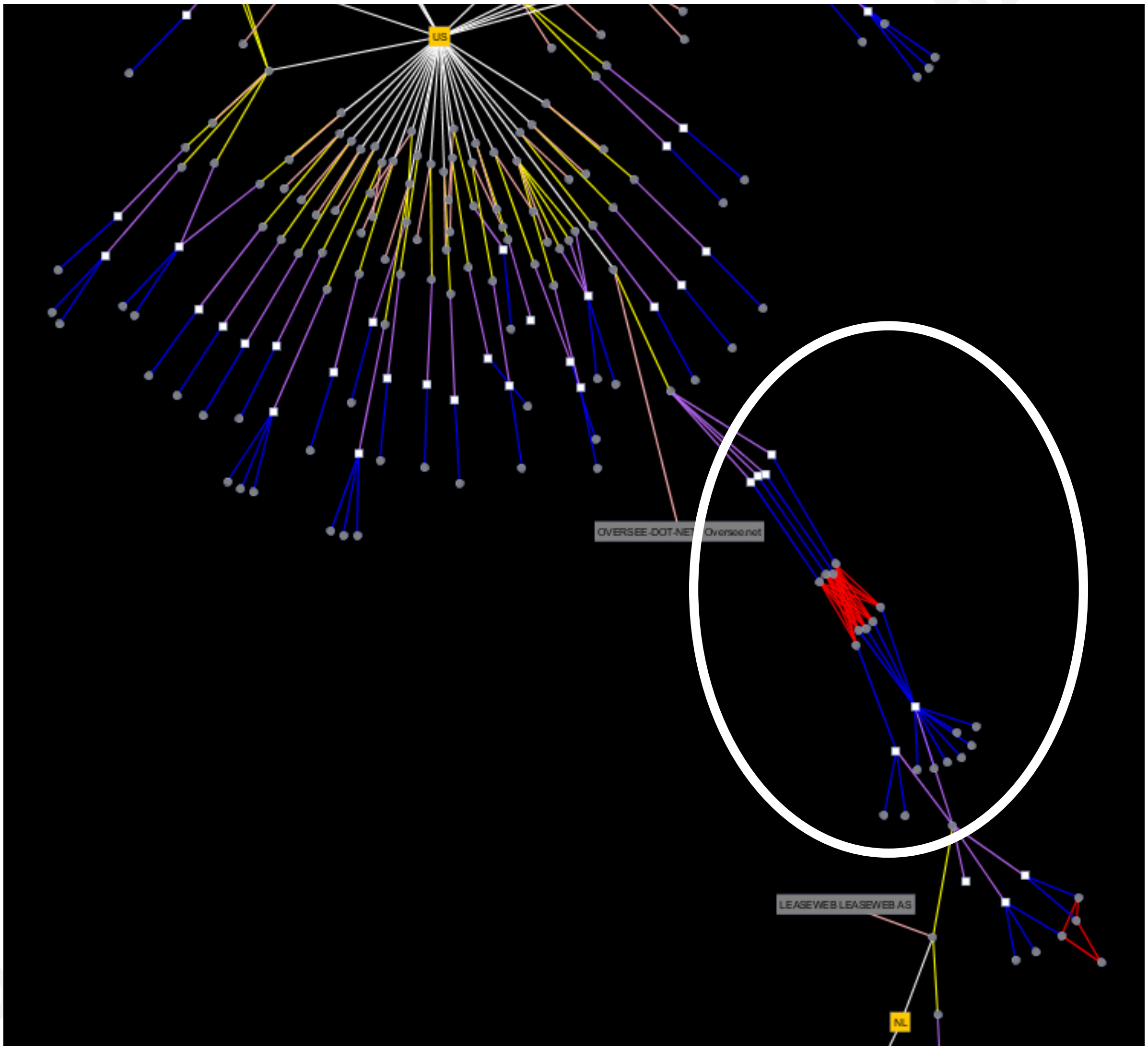
Circular patterns seemed to indicate hosting enclaves, which would be set up and leased out to clients. This would be the basis for bullet-proof hosting, where legitimate space was leased from ISPs, then repackaged into smaller parcels through redundant authoritative nameservers. These nameservers were reused for hundreds of domains, so domains featuring the same supporting malicious NS records could be considered related.

Diverging branches indicated either a branching to another part of the redundant infrastructure, usually a backup link that supported another circular grouping, or a point where the legitimate ISP wandered off to other unrelated, legitimate site services. You can see this legitimate divergence in the upper-left of the below illustration, where the pink CNAME bloom diverges from the blue A record bloom. This is where Cogent, the base ISP in this case, had services to legitimate clients that diverged from the compromised IP address of interest. It is an excellent representation of what the analyst must consider when bringing in data that may indicate legitimate activity.



Andrew Hunt, sans.ahunt@gmail.com

As indicated in the second pass, the most interesting linkages occurred when highly redundant name services bridged domains together across jurisdictional boundaries. These occurred at weak points for the aggressor, where common monitoring and blocking techniques would easily expose and potentially prevent traffic from successfully resolving. In order to overcome the possibility of being shut down, the aggressor invested in massive redundancy to ensure that as one resolution path was blackholed, another was prestaged to take its place. Rotating through these upstream links had made taking down the provision extremely difficult.



This aggressor had invested in infrastructure at LeaseWeb, located in the

Netherlands, which they wanted to ensure resolution through assets in the United States. Their network, domain, and nameservice redundancy through Oversee Dot Net indicated their preference to use that ISP for this conduit.

Extrapolating the Business Model for Defense

Now that some common means of malfeasance have been identified, they will be applied to another example of malicious hosting. This section uses visualizations generated from an identified set of malicious domains from July 2009 (Malware Domains, 2010). The analysis will reveal targets of opportunity for defensive measures.

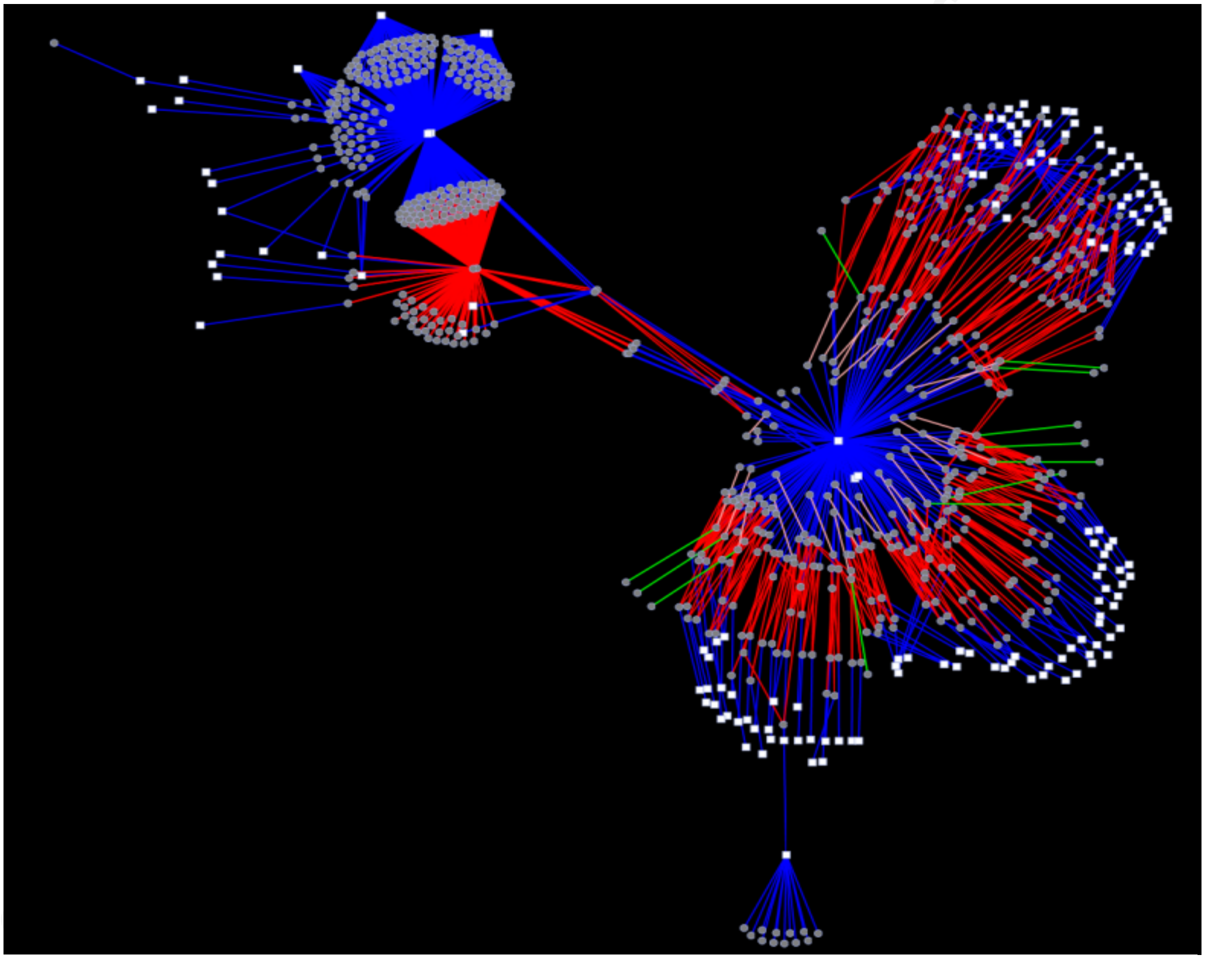
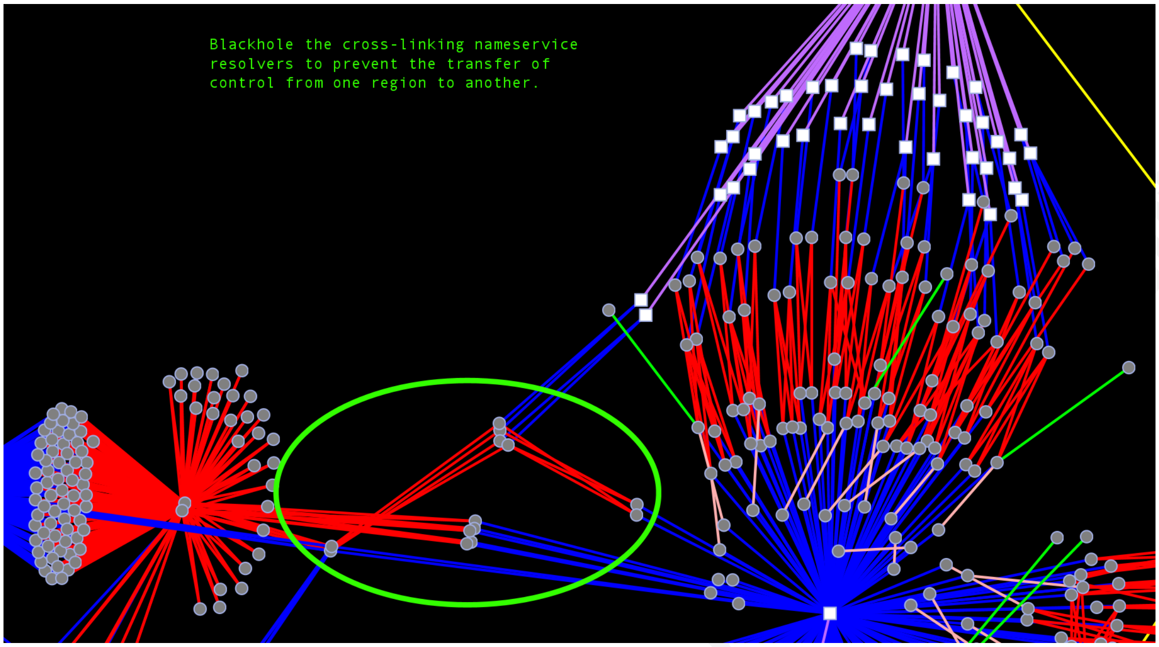


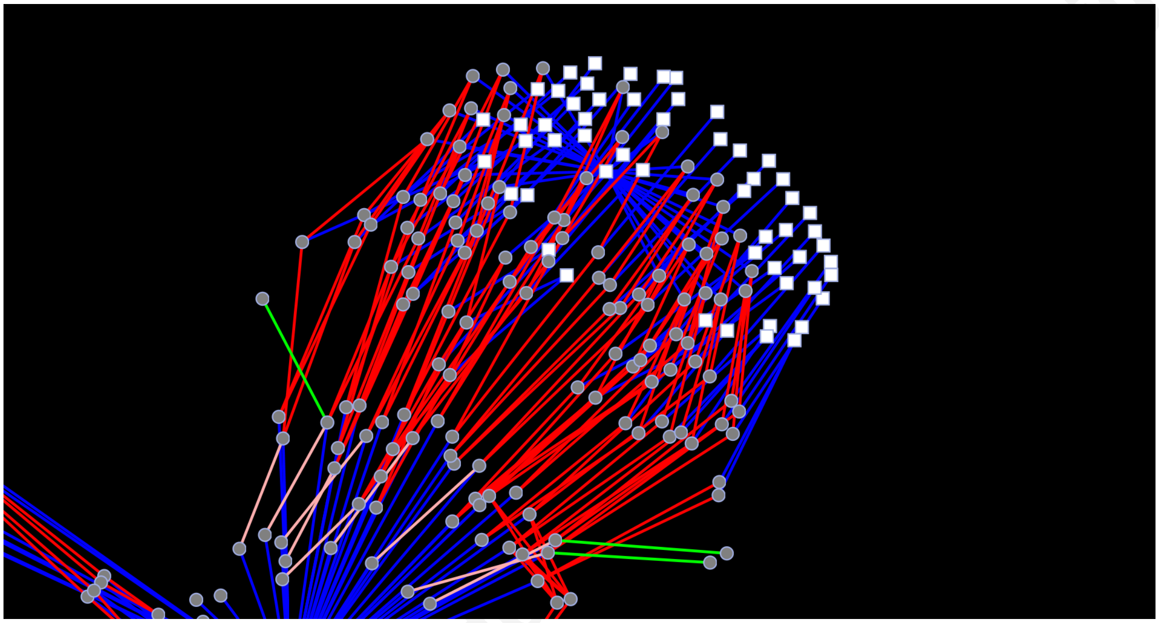
Figure 22: Zoomed out view of malfeasance.

Andrew Hunt, sans.ahunt@gmail.com

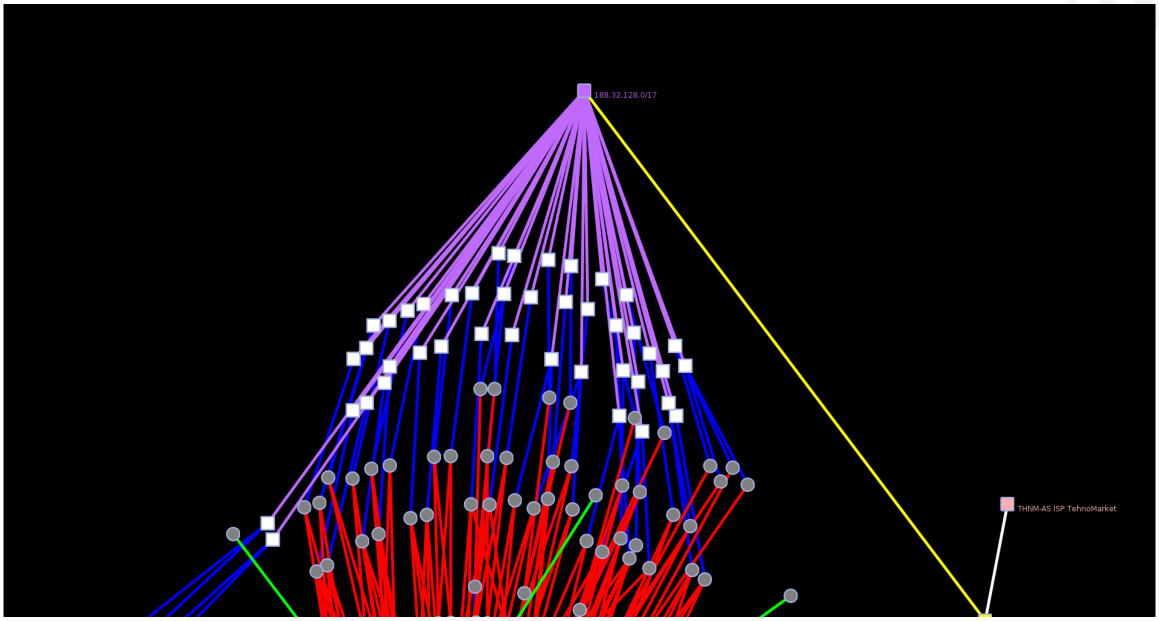
The first step is to identify patterns unique to malicious hosting. As was demonstrated with the ActiveX exploitation case study, ultra-redundancy between authoritative nameservers and their supported domains is a key tipoff to suspicious hosting behavior. These redundancies typically taper into a few IP addresses or authoritative nameservers that become excellent targets for an effective block affecting all of the dependent resolutions. This is demonstrated below where the aggressor is attempting to link their assets between the United States and Russia.


Blackhole the cross-linking nameservice
resolvers to prevent the transfer of
control from one region to another.

Another pattern to look for is aggregation around a particular ISP, network block, or ASN. This sample shows how the aggressor is attempting to diversify the authoritative nameservice support to reduce the dependency on a single authoritative resolver. This kind of diversification makes identification of individual target NS servers for blocking more difficult.

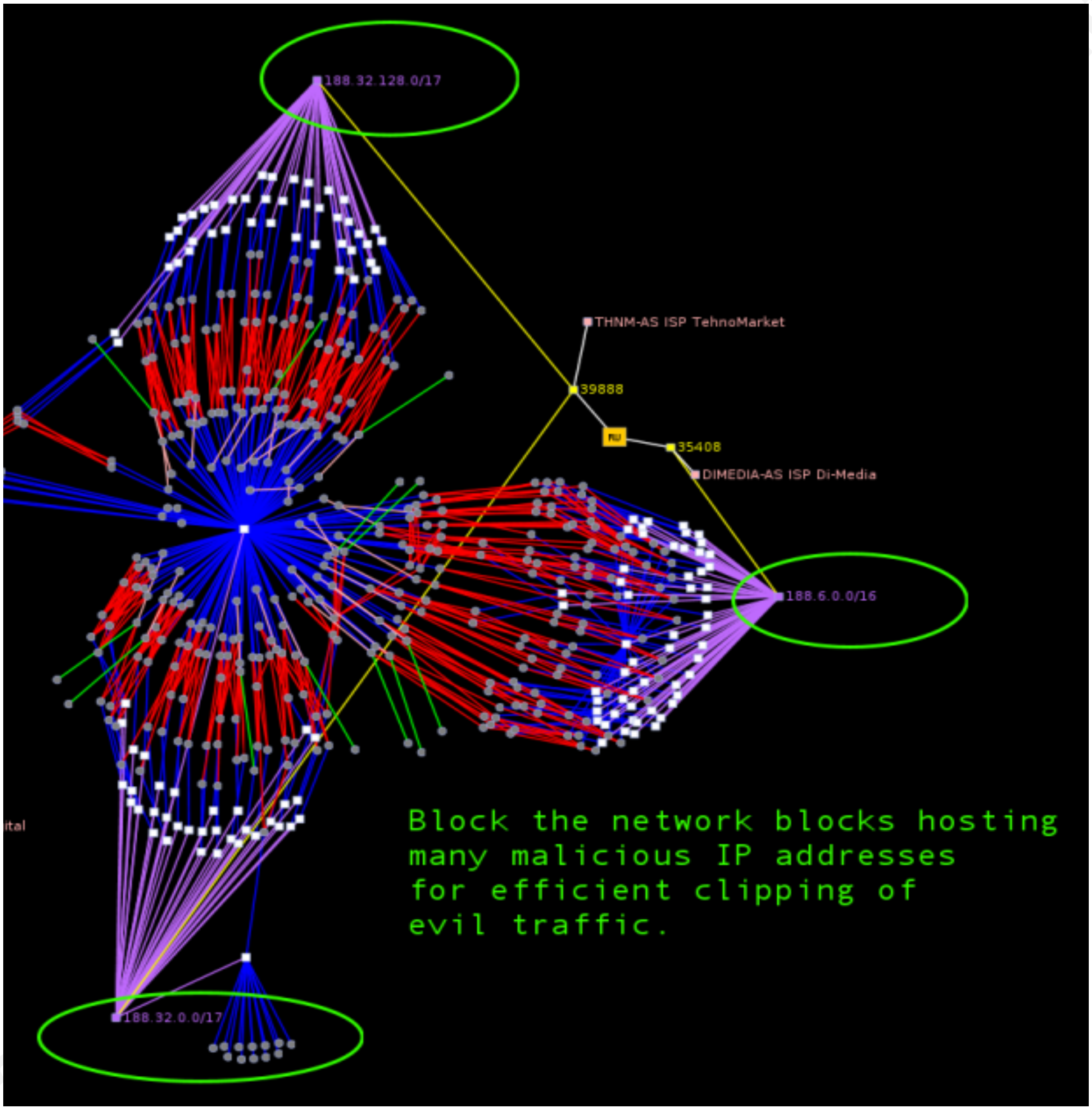


However, all of the IP addresses for the authoritative nameservers are hosted in the same network block. This dependence indicates an investment by the aggressor into a particular hosting company and can provide an effective network-level block at relatively low cost. As always, be aware of potential collateral damage when blocking a network portion that may also contain legitimate IP hosting space.



Andrew Hunt, sans.ahunt@gmail.com

When taken back out to an overall view, the analyst may note that three network blocks on the right would exterminate the resolution of the malicious IP address serving malware in the middle. The block would also preempt a simple change of IP address by the aggressor to another IP address within their leased network blocks.

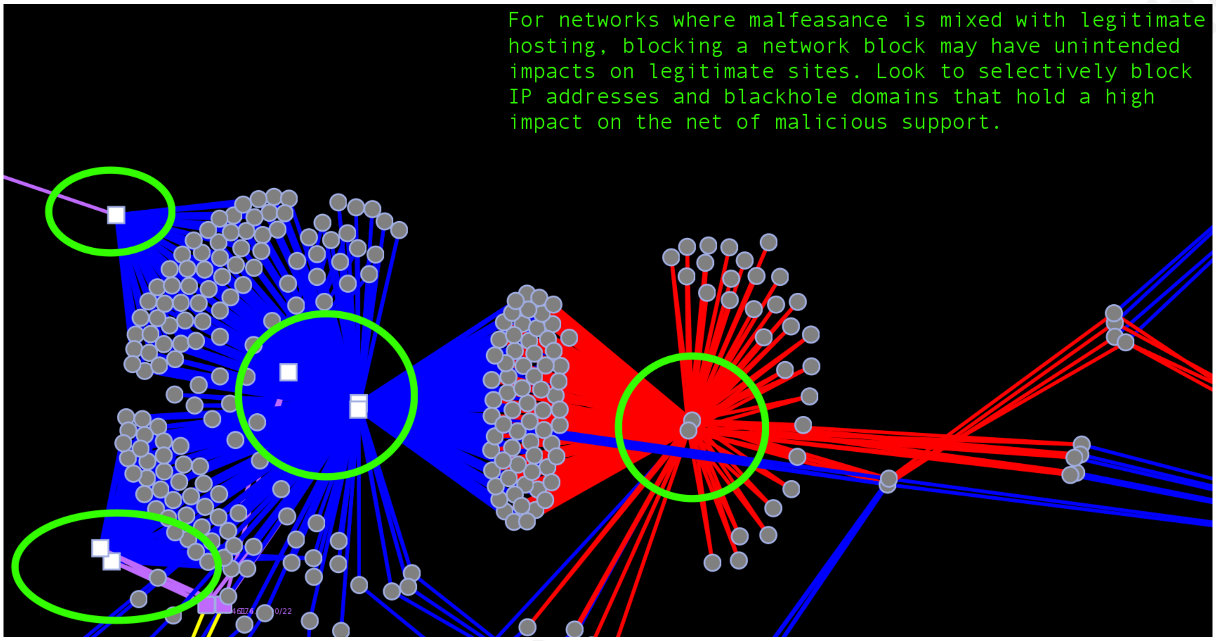


Block the network blocks hosting many malicious IP addresses for efficient clipping of evil traffic.

Figure 26: Three network blocks stand out in the graph. Discovering if there is no legitimate traffic at these networks from your organization could lead to fewer, more efficient network blocks of nefarious areas.

Andrew Hunt, sans.ahunt@gmail.com

Finally, look for finely-attuned targets of opportunity. As seen in this example, identifying the hosts of dependence provides selective kills among ISPs that provide many other legitimate services, limiting collateral damage from defensive measures.



For networks where malfeasance is mixed with legitimate hosting, blocking a network block may have unintended impacts on legitimate sites. Look to selectively block IP addresses and blackhole domains that hold a high impact on the net of malicious support.

While IP addresses, network blocks, and domains have been the focus of these defenses as they are most common, ASN-level blocking using Border Gateway Protocol (BGP) blackholing of routing advertisements from identified rogue ISPs can also be effective (McPherson, 2005).

Conclusion

As demonstrated in the previous sections, visualizing the passive DNS historical resolutions combined with ASN and geolocation information provides an analyst with powerful tools to understand the business of malicious hosting. Harnessing that knowledge can lead to effective assessment of the most productive targets for defense, leveraging the adversaries' reliance on common bottlenecks to effectively quash an attack.

While the usefulness of these applied techniques has been demonstrated, no large publicly available repository of production-grade passive DNS data yet exists. In order to be most effective, a large repository of provided data from many collection points across national, government, and business sector boundaries must be established. Echoing Keith Mitchell (2007), a trusted entity must manage the equities among these different groups to build a repository of wide enough scope to be effective for almost any domain or IP address query. With a repository of data in place, research into new kinds of visual, statistical, and behavioral analyses would become possible.

References

Adar, Eytan. (2005). GUESS Visualization Tool. Retrieved from http://graphexploration.cond.org/

Borman, Sean. (2004, July 18). The Expectation Maximization Algorithm: A Short Tutorial. Available as PDF from http://www.isi.edu/natural-language/teaching/cs562/2009/readings/B06.pdf

Dagon, David; Lee, Wenke. (2009). Global Internet Monitoring Using Passive DNS. Cybersecurity Applications & Technology Conference for Homeland Security (CATCH). pp.163-168. Available as PDF from http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4804440

Dagon, David; Provos, Niel;, Lee, Christopher P., Lee, Wenke. (2008, February). Corrupted DNS Resolution Paths: The Rise of a Malicious Resolution Authority. Proceedings of the 15th Annual Network & Distributed System Security Symposium (NDSS '08). San Diego, CA. Available as PDF from http://www.citi.umich.edu/u/provos/papers/ndss08_dns.pdf

de Beaupre, Adrien. (2009, July 13). Vulnerability in Microsoft Office Web Components Control Could Allow Remote Code Execution. SANS Internet Storm Center. Retrieved from http://isc.sans.edu/diary.html?storyid=6778

Lee, Chris. (2008, December 06). pdns.rb. Retrieved from http://dnsparse.insec.auckland.ac.nz/dns/files/pdns.rb

Lorenzen, April. (2006). Passive DNS Evolution. Available as PDF from https://www.dns-oarc.net/files/workshop-2006/Lorenzen-PassiveDNS.pdf

Ludwig, Andre. (2009, July 06). IE 0day Exploit Domains. SANS Internet Storm Center. Retrieved from http://isc.sans.edu/diary.html?storyid=6739

Malware Domains. (2010, May 03). DNS-BH Update for May 2. Retrieved from http://www.malwaredomains.com/wordpress/?p=952

Malware Domains. (2010, May 05). Important Additions: grepad-dot-com, ginopostdot-com, thejustb-dot-com. Retrieved from http://www.malwaredomains.com/wordpress/?p=955

McPherson, Danny. (2005, February). BGP Security Techniques. Available as PDF from http://www.arbornetworks.com/index.php?option=com_docman &task=doc_download&gid=112

Mitchell, Keith; Vixie, Paul. (2007, July). Passive DNS at OARC. DNS Operations Meeting, Chicago. Available as PDF from https://www.dns-oarc.net/files/dnsops-2007/Mitchell-PassiveDNS.pdf

Mockapetris, Paul. (1987, November). Domain Names - Implementation and Specification. Retrieved from http://www.ietf.org/rfc/rfc1035.txt

Reynolds, Patrick. (1999). How the Oracle of Bacon Works. The Oracle of Bacon. Retrieved from http://oracleofbacon.org/how.php

Team Cymru. (2010, March 15). IP to ASN Mapping. Retrieved from http://www.team-cymru.org/Services/ip-to-asn.html

Weimer, Florian. (2005, June 29). Passive DNSLogger Project. FIRST 2005. Retrieved from http://www.enyo.de/fw/software/dnslogger/

Weimer, Florian. (2010, March 15). BFK Passive DNS Replication. Retrieved on March 15, 2010 from http://www.bfk.de/bfk_dnslogger.html

Weimer, Florian. (2010, March 15). RUS-CERT Passive DNS Replication. Retrieved March 15, 2010 from http://cert.uni-stuttgart.de/stats/dns-replication.php

Zdrnja, Bojan. (2006, May). Security Monitoring of DNS Traffic. Available as PDF. Retrieved from http://www.caida.org/~nevil/Bojan_Zdrnja_CompSci780_Project.pdf

Zdrnja, Bojan. (2009, July). Using Passive DNS Data to Track Malicious Activities. SANSFire 2009. Available as webcast at https://www.sans.org/webcasts/sansfire-2009-passive-dns-data-track-malicious-activities-92528&ei=lGgtTJnhAsOB8gbxxf2qAw&usg =AFQjCNHY2Ys4NjJCWjLxfIYhpGVdQ7C2kA [Requires SANS login account]

Andrew Hunt, sans.ahunt@gmail.com

Zdrnja, Bojan. (2010). DNSParse. Retrieved March 15, 2010 from https://dnsparse.insec.auckland.ac.nz/dns

Zdrnja, Bojan; Brownlee, Nevil; Wessels, Duane. (2007, July). Passive Monitoring of DNS Anomalies (Extended Abstract). Proceedings of the 4th International Conference, DIMVA 2007. 129-139. Available as PDF. Retrieved from http://bojan.isc.googlepages.com/PassiveMonitoringOfDNSAnomalies.pdf

Appendix A: pdns-v2.rb

The following is a ruby script to automate the collection of passive DNS data using known repositories. The original script was written by Chris Lee (Lee, 2008) and posted to Bojan Zdrnja's DNS Parse website (Zdrnja, 2010). This updated version is provided with this paper to demonstrate the value of passive DNS query results and advance research on the topic. Please read the release notes before using this tool. Report bugs to Andrew Hunt at sans.ahunt@gmail.com or Chris Lee at chrislee@gatech.edu.

```
#!/usr/bin/env ruby
# DESCRIPTION: queries passive DNS databases
# Written by Chris Lee (chrislee@gatech.edu) on 12/6/2008
# Modified by Andrew Hunt (sans.ahunt@gmail.com) 03/15/2010
# Release version provided by Chris Lee (chrislee@gatech.edu) on 07/20/2010
# This code is released under the LGPL: http://www.gnu.org/licenses/lgpl-3.0.txt
# Please note that use of any passive dns database is subject to the terms of use of that
passive dns database.  Use of this script in violation of their terms is not encouraged
in any way.  Also, please do not add any obfuscation to try to work around their terms of
service.  If you need special services, ask the providers for help/permission.

require 'open-uri'
require 'openssl'
require 'timeout'
require 'yaml'
require 'structformatter'
require 'getoptlong'

$debug = false

module PassiveDNS
        class PDNSResult < Struct.new(:query, :answer, :rrtype, :ttl, :firstseen,
:lastseen)
                def to_s(sep="|")

        [self.query,self.answer,self.rrtype,self.ttl,self.firstseen,self.lastseen].join(se
p)
                end
                def PDNSResult::results_to_gdf(res)
                        output = "nodedef> name,description VARCHAR(12),color,style\n"
                        # IP "$node2,,white,1"
                        # domain "$node2,,gray,2"
                        # Struct.new(:query, :answer, :rrtype, :ttl, :firstseen, :lastseen)
                        colors = {"MX" => "green", "A" => "blue", "CNAME" => "pink", "NS"
=> "red", "SOA" => "white", "PTR" => "purple", "TXT" => "brown"}
                        nodes = {}
                        edges = {}
                        res.each do |i|
                                if i
                                        nodes[i.query + ",,gray,2"] = true
                                        if i.answer =~ /[0-9]{1,3}\.[0-9]{1,3}\.[0-
9]{1,3}\.[0-9]{1,3}/ then
                                                nodes[i.answer + ",,white,1"] = true
                                        else
                                                nodes[i.answer + ",,gray,2"] = true
                                        end
```

Andrew Hunt, sans.ahunt@gmail.com

```
                                end
                        end
                        nodes.each do |i,j|
                                output += i+"\n"
                        end
                        output += "edgedef> node1,node2,color\n"
                        res.each do |i|
                                if i
                                        color = colors[i.rrtype]
                                        color ||= "blue"
                                        edges[i.query + "," + i.answer + "," + color] = true
                                end
                        end
                        edges.each do |i,j|
                                output += i+"\n"
                        end
                        output
                end
                def PDNSResult::results_to_graphviz(res)
                        colors = {"MX" => "green", "A" => "blue", "CNAME" => "pink", "NS"
=> "red", "SOA" => "white", "PTR" => "purple", "TXT" => "brown"}
                        output = "digraph pdns {\n"
                        nodes = {}
                        res.each do |l|
                                if l
                                        unless nodes[l.query]
                                                output += "  \"#{l.query}\" [shape=ellipse,
style=filled, color=gray];\n"
                                                if l.answer =~
/^\d{3}\.\d{3}\.\d{3}\.\d{3}$/
                                                        output += "  \"#{l.answer}\"
[shape=box, style=filled, color=white];\n"
                                                else
                                                        output += "  \"#{l.answer}\"
[shape=ellipse, style=filled, color=gray];\n"
                                                end
                                                nodes[l.query] = true
                                        end
                                        output += "  \"#{l.query}\" -> \"#{l.answer}\"
[color=#{colors[l.rrtype]}];\n"
                                end
                        end
                        output += "}\n"
                end
                def PDNSResult::results_to_graphml(res)
                        output = '<?xml version="1.0" encoding="UTF-8"?>
<graphml xmlns="http://graphml.graphdrawing.org/xmlns"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://graphml.graphdrawing.org/xmlns
     http://graphml.graphdrawing.org/xmlns/1.0/graphml.xsd">
  <graph id="G" edgedefault="directed">
'
                        nodes = {}
                        edges = {}
                        res.each do |r|
                                if r
                                        output += "    <node id='#{r.query}'/>\n" unless
nodes["#{r.query}"]
                                        nodes[r.query] = true
                                        output += "    <node id='#{r.answer}'/>\n" unless
nodes["#{r.answer}"]
                                        nodes[r.answer] = true
                                        output += "    <edge source='#{r.query}'
target='#{r.answer}'/>\n" unless edges["#{r.query}|#{r.answer}"]
                                end
                        end
                        output += '</graph></graphml>'+"\n"
                end
        end

        class DNSParse
                @@dns_rtypes = {
                        1 => 'A',
                        2 => 'NS',
                        3 => 'MD',
                        4 => 'MF',
                        5 => 'CNAME',
                        6 => 'SOA',
                        7 => 'MB',
                        8 => 'MG',
                        9 => 'MR',
                        10 => 'NULL',
                        11 => 'WKS',
                        12 => 'PTR',
                        13 => 'HINFO',
                        14 => 'MINFO',
                        15 => 'MX',
                        16 => 'TXT',
                        17 => 'RP',
                        18 => 'AFSDB',
                        19 => 'X25',
                        20 => 'ISDN',
                        21 => 'RT',
                        22 => 'NSAP',
                        23 => 'NSAP-PTR',
                        24 => 'SIG',
                        25 => 'KEY',
                        26 => 'PX',
                        27 => 'GPOS',
                        28 => 'AAAA',
                        29 => 'LOC',
                        30 => 'NXT',
                        31 => 'EID',
                        32 => 'NIMLOC',
                        33 => 'SRV',
                        34 => 'ATMA',
                        35 => 'NAPTR',
                        36 => 'KX',
                        37 => 'CERT',
                        38 => 'A6',
                        39 => 'DNAME',
                        40 => 'SINK',
                        41 => 'OPT',
                        42 => 'APL',
                        43 => 'DS',
                        44 => 'SSHFP',
                        45 => 'IPSECKEY',
                        46 => 'RRSIG',
```

Andrew Hunt, sans.ahunt@gmail.com

```
                        47 => 'NSEC',
                        48 => 'DNSKEY',
                        49 => 'DHCID',
                        55 => 'HIP',
                        99 => 'SPF',
                        100 => 'UINFO',
                        101 => 'UID',
                        102 => 'GID',
                        103 => 'UNSPEC',
                        249 => 'TKEY',
                        250 => 'TSIG',
                        251 => 'IXFR',
                        252 => 'AXFR',
                        253 => 'MAILB',
                        254 => 'MAILA',
                        255 => 'ALL',
                }
                def initialize(config="#{ENV['HOME']}/.dnsparse")
                        if File.exist?(config)
                                @base,@user,@pass = File.open(config).read.split(/\n/)
                        else
                                raise "Configuration file for DNSParse is required for
intialization\nFormat of configuration file (default: #{ENV['HOME']}/.dnsparse)
is:\n<url>\n<username>\n<password>\n"
                        end
                end

                def parse(page)
                        res = []
                        # need to remove the json_class tag or the parser will crap itself
trying to find a class to align it to
                        page = page.gsub(/\"json_class\"\:\"PDNSResult\"\,/,'')
                        recs = JSON.parse(page)
                        recs.each do |row|
                                res <<
PDNSResult.new(row["query"],row["answer"],@@dns_rtypes[row["rrtype"].to_i],row["ttl"],row
["firstseen"],row["lastseen"])
                        end
                        res
                rescue Exception => e
                        $stderr.puts "DNSParse Exception: #{e}"
                        raise e
                end

                def lookup(label)
                        puts "DEBUG: dnsparse:"+label if $debug
                        puts "DEBUG: dnsparse proxy: " + $proxy if $debug & $proxy
                        Timeout::timeout(240) {
                                open(
                                        @base+label,
                                        "User-Agent" => "Ruby/#{RUBY_VERSION} DNSPARSE
passive dns script",
                                        #"From" => "#{ENV['USER']}@#{ENV['HOSTNAME']}",
                                        :http_basic_authentication => [@user,@pass]
                                ) do |f|
                                        parse(f.read)
                                end
                        }
                rescue Timeout::Error => e
```

Andrew Hunt, sans.ahunt@gmail.com

```
                        $stderr.puts "DNSParse lookup timed out: #{label}"
                end
        end
end

if __FILE__ == $0
        def usage
                puts "Usage: #{$0} [-a|-b|-d|-i|-e|-r#] [-c|-x|-y|-j|-t] <ip or domain>"
                puts "  -a uses all of the available passive dns databases"
                puts "  -g outputs a link-nodal GDF visualization definition"
                puts "  -v outputs a link-nodal graphviz visualization definition"
                puts "  -m output a link-nodal graphml visualization definition"
                puts "  -c outputs CSV"
                puts "  -x outputs XML"
                puts "  -y outputs YAML"
                puts "  -j outputs JSON"
                puts "  -t outputs ASCII text (default)"
                puts "  -f[file] specifies a YAML dump input to work on instead of calling
to the pDNS servers."
                puts "  -r# specifies the levels of recursion to pull. **WARNING** This is
quite taxing on the pDNS servers, so use judiciously (never more than 3 or so) or find
yourself blocked!"
                puts "  -w# specifies the amount of time to wait, in seconds, between
queries (Default: 0)"
                exit
        end

        opts = GetoptLong.new(
                [ '--help', '-h', GetoptLong::NO_ARGUMENT ],
                [ '--gdf', '-g', GetoptLong::NO_ARGUMENT ],
                [ '--graphviz', '-v', GetoptLong::NO_ARGUMENT ],
                [ '--graphml', '-m', GetoptLong::NO_ARGUMENT ],
                [ '--csv', '-c', GetoptLong::NO_ARGUMENT ],
                [ '--xml', '-x', GetoptLong::NO_ARGUMENT ],
                [ '--yaml', '-y', GetoptLong::NO_ARGUMENT ],
                [ '--json', '-j', GetoptLong::NO_ARGUMENT ],
                [ '--text', '-t', GetoptLong::NO_ARGUMENT ],
                [ '--sep', '-s', GetoptLong::REQUIRED_ARGUMENT ],
                [ '--file', '-f', GetoptLong::REQUIRED_ARGUMENT ],
                [ '--recurse', '-r', GetoptLong::REQUIRED_ARGUMENT ],
                [ '--wait', '-w', GetoptLong::REQUIRED_ARGUMENT ]
        )

        # sets the default search methods
        pdnsdb = "dnsparse"
        format = "text"
        sep = "\t"
        recursedepth = 1
        wait = 0
        res = nil

        opts.each do |opt, arg|
                case opt
                when '--help'
                        usage
                when '--gdf'
                        format = 'gdf'
                when '--graphviz'
                        format = 'graphviz'
```

Andrew Hunt, sans.ahunt@gmail.com

```
                when '--graphml'
                        format = 'graphml'
                when '--csv'
                        format = 'text'
                        sep = ','
                when '--xml'
                        format = 'xml'
                when '--json'
                        format = 'json'
                when '--text'
                        format = 'text'
                when '--sep'
                        sep = arg
                when '--file'
                        res = YAML::load( File.open( arg ) )
                when '--recurse'
                        recursedepth = arg.to_i
                        if recursedepth > 3
                                puts "The recursion depth can be a maximum of 3"
                                exit
                        end
                when '--wait'
                        wait = arg.to_i
                else
                        usage
                end
        end

        if pdnsdb == 'dnsparse' or pdnsdb == 'all'
                # We have to turn off OpenSSL verification for DNSParse
                # Beware how you use this code with other security related code
                $stderr.puts "pdns.rb has turned off SSL verification for use with
DNSParse"
                OpenSSL::SSL::VERIFY_PEER = OpenSSL::SSL::VERIFY_NONE
        end

        def
pdnslookup(query,pdnsdb='parsedns',format='text',sep="\t",recursedepth=1,wait=0)
                res = []
                queried = {}
                pending = [query]
                reclevel = 0
                if recursedepth > 1
                        scratchfile = File.new("scratch.tmp","w")
                end
                while reclevel < recursedepth
                        pending_this_loop = pending
                        pending = []
                        while q = pending_this_loop.shift
                                next if queried[q]
                                case pdnsdb
                                when 'dnsparse'
                                        rv = PassiveDNS::DNSParse.new.lookup(q)
                                        res += rv if rv
                                end
                                queried[q] = true
                                pending = []
                                res.each do |r|
                                        pending << r.answer
```

Andrew Hunt, sans.ahunt@gmail.com

```
                                        scratchfile.puts(r.to_json) if recursedepth > 1
                                end
                                reclevel += 1
                                sleep wait
                        end
                end
                scratchfile.close if recursedepth > 1
                res
        end

        def printresults(res,format,sep="\t")
                case format
                when 'text'
                        puts res[0].to_s_header(sep) if res[0]
                        res.each do |rec|
                                puts rec.to_s(sep)
                        end
                when 'yaml'
                        puts YAML.dump(res)
                when 'xml'
                        puts '<?xml version="1.0" encoding="UTF-8" ?>'
                        puts "<report search='#{ip}'>"
                        puts "  <results>"
                        res.each do |rec|
                                puts "         "+rec.to_xml
                        end
                        puts "  </results>"
                        puts "</report>"
                when 'json'
                        puts JSON.pretty_generate(res)
                when 'gdf'
                        puts PassiveDNS::PDNSResult::results_to_gdf(res)
                when 'graphviz'
                        puts PassiveDNS::PDNSResult::results_to_graphviz(res)
                when 'graphml'
                        puts PassiveDNS::PDNSResult::results_to_graphml(res)
                end
        end

        needsleep = false
        if ARGV.length > 0
                ARGV.each do |query|
                        sleep wait if needsleep
                        res = pdnslookup(query,pdnsdb,format,sep,recursedepth,wait)
                        printresults(res,format,sep) if res
                        needsleep = true
                end
        else
                $stdin.each_line do |query|
                        sleep wait if needsleep
                        res = pdnslookup(query.chomp,pdnsdb,format,sep,recursedepth,wait)
                        printresults(res,format,sep) if res
                        needsleep = true
                end
        end
end
```

Andrew Hunt, sans.ahunt@gmail.com

structformatter.rb

```
#!/usr/bin/env ruby
# DESCRIPTION: extends ruby Structs to be outputted as yaml, xml, and json. This is meant
to be used as a mixin
require 'json'

class Struct
        def render_xml(element_name, element)
                str = ""
                if element.class == Date
                        str = "<#{element_name}>#{element.strftime("%Y-%m-
%d")}</#{element_name}>"
                elsif element.class == Time or element.class == DateTime
                        str = "<#{element_name}>#{element.strftime("%Y-%m-
%dT%H:%M:%SZ")}</#{element_name}>"
                elsif element.kind_of? Struct
                        str = element.to_xml
                else
                        str = "<#{element_name}>#{element}</#{element_name}>"
                end
        end
        def to_xml
                children = []
                str = "<#{self.class}"
                self.members.each do |member|
                        if self[member].class == Array or self[member].class == Hash or
self[member].kind_of? Struct
                                children << member
                        elsif self[member].class == Date
                                str += " #{member}='#{self[member].strftime("%Y-%m-%d")}'"
                        elsif self[member].class == Time
                                str += " #{member}='#{self[member].strftime("%Y-%m-
%dT%H:%M:%SZ")}'"
                        else
                                str += " #{member}='#{self[member]}'"
                        end
                end
                if children.length == 0
                        str += ' />'
                else
                        str += '>'
                        children.each do |member|
                                if self[member].class == Array
                                        str += "<#{member}s>"
                                        self[member].each do |item|
                                                str += render_xml(member,item)
                                        end
                                        str += "</#{member}s>"
                                elsif self[member].class == Hash
                                        str += "<#{member}s>"
                                        self[member].each do |key,value|
                                                str +=
"<HashElement><key>#{key}</key><value>"
                                                str += render_xml(member,value)
                                                str += "</value></HashElement>"
                                        end
                                        str += "<#{member}s>"
                                elsif self[member].kind_of? Struct
                                        str += self[member].to_xml
                                end
                        end
                        str += "</#{self.class}>"
                end
        end

        def to_json(*a)
                hash = { 'class' => self.class }
                self.members.each do |member|
                        hash[member] = self[member]
                end
                hash.to_json(*a)
        end

        def to_s(sep = " ")
                self.members.map{ |x| self[x].to_s }.join(sep)
        end

        def to_s_header(sep = " ")
                self.members.map{ |x| x.to_s }.join(sep)
        end

end
```

Appendix B: finger.pl

This script automates the query of Team Cymru's IP-to-ASN Mapping Project to provide ISP and country level data about a given domain and its resolved IP address (Team Cymru, 2010). It is provided to augment passive DNS query results by providing additional data, but can be used as a fast and powerful profiler of IP addresses in its own right. It was written and is maintained by Andrew Hunt at sans.ahunt@gmail.com.

```
# Filename: finger.pl

# Description: Queries Team Cymru's ASN Lookup
# service for a given file containing a list of
# domains
or IP addresss, one per line. Results
# are returned to STDOUT.
# Author: Andrew Hunt, sans.ahunt@gmail.com
# License: Creative Commons Share Alike (http://creativecommons.org/licenses/by-sa/2.5/)

if (-e "$ARGV[0]") {
        open (IFILE, "$ARGV[0]");
        while (<IFILE>) {
                chomp;
                undef $ipaddr;
                undef @whois_results;
                undef @resolve_results;
                undef $domainname;
                undef $a;
                undef @results;
                undef @resultr;
                if (/^\s*$/) {
                        next;
                } elsif (/^\s*(\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})\s*$/) {
                        $ipaddr = $_;
                        @whois_results = &whois($ipaddr);
                        foreach $a (@whois_results) {
                                print " | $a\n";
                        }
                } elsif (/.+?\...?$/) {
                        $domainname = $_;
                        @resolve_results = &resolve($domainname);
                        foreach $a (@resolve_results) {
                                print "$a\n";
                        }
                 } elsif (/.+?\....?$/) {
                          $domainname = $_;
                          @resolve_results = &resolve($domainname);
                          foreach $a (@resolve_results) {
                                  print "$a\n";
                          }
                 } elsif (/.+?\.....?$/) {
                          $domainname = $_;
                          @resolve_results = &resolve($domainname);
                          foreach $a (@resolve_results) {
                                print "$a\n";
                        }

                } else { print "BAD INPUT LINE: $_\n"; }
        }
} elsif ($ARGV[0] =~ /^\s*(\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})\s*$/) {
        $ipaddr = $ARGV[0];
        @whois_results = &resolve($ipaddr);
        foreach $a (@whois_results) {
                print "$a\n";
        }
} elsif ($ARGV[0] =~ /.+?\...?$/) {
        $domainname = $ARGV[0];
        @resolve_results = &resolve($domainname);
        foreach $a (@resolve_results) {
                print "$a\n";
        }
} elsif ($ARGV[0] =~ /.+?\....?$/) {
        $domainname = $ARGV[0];
        @resolve_results = &resolve($domainname);
        foreach $a (@resolve_results) {
                print "$a\n";
        }
} elsif ($ARGV[0] =~ /.+?\.....?$/) {
        $domainname = $ARGV[0];
        @resolve_results = &resolve($domainname);
        foreach $a (@resolve_results) {
                print "$a\n";
        }

} else { print "BAD INPUT: $ARGV[0]\n"; }


sub resolve {
        undef $domain; undef @answersr; undef $answerr; undef @reresolve; undef @resultr;
undef $infor;
        my $domain = shift;
        if ($domain =~ /^\s*(\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})\s*$/) {
                my @answersr = `dig +short -x $domain`;
                @resultr;
                foreach my $answerr (@answersr) {
                        my @whois_resultr = &whois($domain);
                        foreach my $whois_answerr (@whois_resultr) {
                                if ($answerr =~ /^\s*$/) {
                                        $infor = join(' | ', "NO RDNS", $whois_answerr);
                                } else {
                                        $infor = join(' | ', substr($answerr,0,$answerr-1),
$whois_answerr);
                                }
                                @resultr = (@resultr,$infor);
                        }
                }
                return @resultr;
        } else {
                my @answersr = `dig +short $domain`;
                chomp(@answersr);
                @resultr;
                foreach my $answerr (@answersr) {
                        if ($answerr =~ /^\s*(\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})\s*$/) {
                                my @whois_resultr = &whois($answerr);
                                foreach my $whois_answerr (@whois_resultr) {
                                        $infor = join(' | ', $domain, $whois_answerr);
                                        @resultr = (@resultr,$infor);
                                }
                        } else {
                                my @reresolve = &resolve(substr($answerr,0,$answerr-1));
                                foreach $reresolve (@reresolve) {
                                        @resultr = (@resultr,$reresolve);
                                }
                        }
                }
                return @resultr;
        }
}

sub whois {
        undef $octet1; undef $octet2; undef $octet3; undef $octet4;
        undef @answers; undef @results;
        my $ip = shift;
        if ($ip =~ /^(\d{1,3})\.(\d{1,3})\.(\d{1,3})\.(\d{1,3})$/) {
                my $octet1 = $1;
                my $octet2 = $2;
                my $octet3 = $3;
                my $octet4 = $4;
                # Perform the IP WHOIS lookup and parse the result
                my @answers = `dig +short -t TXT
$octet4\.$octet3\.$octet2\.$octet1\.origin\.asn\.cymru\.com `;
                chomp(@answers);
                foreach my $answer (@answers) {
                        undef @afields;
                        undef $ip_as;
                        undef $ip_netblock;
                        undef $ip_cc;
                        undef $ip_as_source;
                        undef $ip_as_date;
                        undef $as_num;
                        undef $as_cc;
                        undef $as_source;
                        undef $as_date;
                        undef $as_desc;
                        undef $info;
                        $answer =~ s/\t//g;
                        $answer =~ s/\"//g;
                        $answer =~ s/\s\|\s/\|/g;
                        my @afields = (split/\|/,$answer);
                        my $ip_as = $afields[0];
                        my $ip_netblock = $afields[1];
                        my $ip_cc = $afields[2];
                        my $ip_as_source = $afields[3];
                        my $ip_as_date = $afields[4];
                        # Perform the AS WHOIS lookup and parse the result
$answer = `dig +short -t TXT AS$ip_as\.asn\.cymru\.com`;
                        chomp($answer);
                        $answer =~ s/\t//g;
                        $answer =~ s/\"//g;
                        $answer =~ s/\s\|\s/\|/g;
                        my @afields = (split/\|/,$answer);
                        my $as_num = $afields[0];
```

Andrew Hunt, sans.ahunt@gmail.com

```
                        my $as_cc = $afields[1];
                        my $as_source = $afields[2];
                        my $as_date = $afields[3];
                        my $as_desc = $afields[4];
                        my $info = join(' |
',sprintf("%15.15s",$ip),sprintf("%18.18s",$ip_netblock),sprintf("%2.2s",$ip_cc),sprintf(
"%5.5s",$ip_as),$as_desc);
                        @results = (@results,$info);
                }
                return(@results);
        } else {
                print "BAD IP ADDRESS: $ip\n";
                return("$ip \| UNKNOWN");
        }
}
```

Andrew Hunt, sans.ahunt@gmail.com

Appendix C: enum_asn.pl

This script converts finger.pl ASN returns into GDF format as described in this paper.

```
#!/usr/bin/perl -w
# converts ASN output from finger.pl to GDF format

open OUTFILE, ">out_asn.gdf";
my (%nodes, %edges);

READ: while(<>) {
    @line = split /\|/;
    for (@line) {
        next READ if (/^\s*$/);
        $_ =~ s/^\s*(\S.*\S)\s*$/$1/;
    }

    my $domain = $line[0];
    my $ipaddr = $line[1];
    my $net = $line[2];
    my $country = $line[3];
    my $asn = "\"$line[4]\"";
    my $asn_desc = "\"$line[5]\"";

    $nodes{"$ipaddr,,white,1"}++;
    $nodes{"$domain,,gray,2"}++;
    $nodes{"$net,,gray,2"}++;
    $nodes{"$country,,orange,4"}++;
    $nodes{"$asn,$asn_desc,white,1"}++;

    $edges{"$domain,$ipaddr,blue"}++;
    $edges{"$ipaddr,$net,purple"}++;
    $edges{"$net,$asn,yellow"}++;
    $edges{"$asn,$country,yellow"}++;
}

print OUTFILE "nodedef> name,description VARCHAR(12),color,style\n";
print OUTFILE "$_\n" for sort keys %nodes;
print OUTFILE "edgedef> node1,node2,color\n";
print OUTFILE "$_\n" for sort keys %edges;
close OUTFILE;
```

Andrew Hunt, sans.ahunt@gmail.com