Merlin SRP

Michael Himbeault

April 4, 2011

Contents

1 Provided Proof-of-Concept Tool

The proof-of-concept tool provided along with this report is designed to demonstrate some of the analysis techniques proposed in sections 3.2 and 3.5.1 in form conducive to being applied in an operational context.

The tool is able to examine packet data, either off of a network interface or from a captured file, and determine the distribution and nature of DNS queries over time windows. The output contains the following information:

- The total number of *valid DNS queries and responses* over the window.
- The total number of packets to or from UDP port 53 that do not conform to the DNS RFC.[1]
- The entropy[2] of the queries over the window.
- The number of unique domains that are represented by some query (or response) over the window. A unique domain could be *example.com*, but will not be *ex.example.com* or *com*. As a rule, a domain name is one that is registered through a registrar.
- For each unique domain represented over the window, the following information is produced:
  - The total number of queries that are a part of this domain over the window. For example *sub.example.com* will count towards the *example.com* domain.
  - The entropy of the queries that are a part of this domain over the window.[3]

The tool is architected in an extensible fashion so as to allow further processing to be added to it as needs change. Unfortunately, it is not possible for this tool to directly block packets since it does not register with any firewall (such as Linux's iptables or the cross-platform ipfilter) hooks necessary for dropping packets.

The tool's current performance rate is approximately 500,000 packets per second.

2 Summary of Existing Solutions

2.1 Suricata

Homepage: http://www.openinfosecfoundation.org

Suricata is an open-source, free to use, community developed and maintained intrusion detection/protection framework that is designed to augment existing solutions. The primary maintainer of the project is the OISF (Open Information Security Foundation).

[1] There is no known source for these packets besides misbehaving NAT devices and malicious applications intentionally abusing the fact that UDP port 53 is usually left open on firewalls to allow DNS. Typically, NAT devices are supposed to reserve ports 0-1024 inclusive on the WAN side for direct forwarding to servers on the LAN side. Some NAT implementations disobey this recommendation and make use of this port range on the WAN side for additional mapping range.

[2] Entropy is a measure of the type of randomness that is present in a collection of items. Very high entropy indicates a very uniform distribution, and very low entropy indicate that a small number of items occur almost all the time.

[3] This is the primary statistic that is produced by this tool, and is tailored specifically to detect DNS tunnels (see section 3.5.1 for more information).

The goal of Suricata is not to become a replacement IDS or IPS, but rather to be a platform for research and deployment of new technologies. For this reason, it has value to anyone currently operating an IDS or IPS as it will provide the ability to test new or experimental methods of detecting or blocking unwanted traffic.

Similarly, because it is open source and community supported, it is very practical for network administrators to write new rules and filters for Suricata as new threats emerge, or to be used for diagnostics on their network. This is typically not the case for off-the-shelf IDS and IPS products which typically fall into one of three categories:

- Do not allow administrators to add new rules at all.
- Expose only a limited set of functionality to custom rules.
- Require writing complex code and interfacing with complex APIs in order to implement even the simples of custom rules.

Due to the ease with which it can be extended, and the flexibility it can provide, Suricata is an invaluable asset on networks of any size.

2.2 BotHunter

Homepage: http://www.bothunter.net

BotHunter is a research-focused project with the sole purpose of being the leading resource for identifying botnets. It makes use of a very general model of how botnets communicate and spread to be able to detect new and emerging threats. By applying this model to either live or archived packet data it is able to detect, and alert on, traffic that may be indicative of a botnet infection. BotHunter is able to identify inter-bot communication patterns, as well as patterns that correspond to new infections and command-channel messages.

BotHunter's model and method of detection is well suited to consumer-grade general-purpose hardware, and hence can be an inexpensive addition to existing infrastructures that are already making use of IDS and IPS technologies.

Since BotHunter makes use of a general model of communication it is not solely based on signatures. Thus it is possible for it to detect a wide range of infection types, as well as still be useful on zero-day infections. Further, because it is not dependent on signatures, it is far more difficult for existing malware to find ways of avoiding detection by BotHunter.

3 Summary of Detection Approaches

3.1 Identifying Network Topology Behind NAT Devices

It is possible for a great deal of network topology information to be hidden behind NAT devices at the internet-edge of a network. This can pose problems for identifying the precise source of strange network behaviour or infections, and it is even possible that this obfuscation of the source will cause some traffic models to fail altogether. Unfortunately while IPv4 is still the dominant network addressing scheme it is unlikely that this situation will see any resolution.

With the introduction of IPv6 addressing NAT no longer becomes necessary and hence it will become possible to completely map the topology of networks at all points on the internet. While it

is technically possible to perform IPv6 NAT, doing so is both unnecessary and strongly discouraged as it breaks end-to-end connectivity as well as impedes network visibility. There are still situations where IPv6 NAT is appropraite (that is where end-to-end connectivity is unimportant and network topology obfuscation is important), so the techniques presented here for Identifying network topology bbehind NAT devices will remain relevant.

3.1.1 Source Port Counting

NAT devices perform, in essence, a mapping from the source LAN IP address and the origin source port, to a single source port on the WAN side. Using this information, it is possible to estimate the number of devices behind a NAT device by counting the number of source ports of a window of time. Using this approach is relatively straight forward and does not require complicated analysis or maintenance infrastructure[4].

This however suffers from the drawback that, in its naivest form, it is unable to distinguish between two TCP streams from the same host with different source ports, and two TCP streams from different hosts. It is possible to somewhat patch this fault under some assumptions:

Consider a situation where the hosts of interest behind the NAT primarily use the internet for web browsing, resulting in large amounts (relative to the total traffic load) of HTTP traffic. Also assume that there is a window (say, one minute long) where it is reasonably guaranteed that all, or at least most, of the hosts of interest are actively producing or soliciting internet traffic. Consider, over this window, all TCP streams that are HTTP and contain HTML and count the number of unique source ports that occur on the WAN side of the NAT device. This count is an estimate of the number of hosts of interest behind the NAT.

This situation is tailored for a specific set of hosts of interest, but represents one likely situation. It specifically restricts consideration to those HTTP streams carrying HTML since it is reasonable to assume that no host is getting more than one HTML page at a time[5]. The window duration should be tuned carefully to keep the likelihood of hosts requesting multiple pages to a minimum, but long enough to approximately capture all of the hosts behind the NAT.

Regardless of the particular tuning of this approach, it is possible to repeat this approach over many windows, to obtain a relatively accurate approximation of the number of hosts behind the NAT device.

3.1.2 JavaScript Injection

The most accurate[6], but also the most intrusive, method for identifying the network topology behind a NAT device involves the use of client-side scripting techniques. Since JavaScript is a client-side scripting language that is prevalent on web pages, it can be reasonably assumed that JavaScript support will be ubiquitous amongst the hosts that are of interest. Note that this assumption restricts the set of hosts that can be enumerated

[4]For example, the approach described in 3.1.2 requires server infrastructure, and the ability to modify the contents of packets or streams on a live network connection.

[5]These situations do occur, but are rare enough. Further, for the small number of times that this does occur, the over-counting of that host can be thought of as filling the 'gap' left by those hosts that produced no traffic that fit the criteria over the window.

[6]The accuracy is subject to a caveat that it is not complete. Since it relies on JavaScript, this will be unable to enumerate hosts that either do not browse the web (such as servers or special-purpose hosts), or that do not support JavaScript (such as hosts running FireFox with the NoScript plugin enabled). Given the accuracy for all other hosts, and the fact that most hosts of interest do not fall into either of these categories this approach is still viable.

By making use of either a web-page proxy (such as Squid), or by editing TCP streams live on the wire using an inline packet modification tool, it is possible to inject client-side javascript into HTML pages that does not exist in the source web page[7]. This client-side script would fingerprint the host (by identifying its LAN IP, for example), and post this information back to a server operated by the interested parties. By correlating the public IP address that the post came from, with the content of the post, it becomes possible to build a map of the internal network.

In order to minimize the impact on network performance, it is reasonable to only apply this to pages named *index.html* or those produced by a server-side scripting page such as *index.php* or *index.aspx*. This should reduce the number of pages to which this scripting is applied, but should still ensure that all hosts browsing the insecure web internet still encounter this injected scripting.

3.2 DNS and non-DNS over port 53

Because DNS is such an integral part of internet communication, it is often the case that UDP (and now with the introduction of DNSSEC, TCP) port 53 is left unrestricted on firewalls and IPS equipment. Because of this oversight (intentional or otherwise), it is common for applications to move arbitrary internet traffic over these unrestricted ports to circumvent firewall restrictions that are effective on other ports.

The only acceptable traffic that should be moving across port 53 on UDP or TCP is RFC-conforming DNS, however this is often not the case. One way to mitigate this is to use a firewall to rewrite all packets moving in or out through port 53 to be directed to a recursive DNS server. This should result in all non-DNS traffic being dropped (as the DNS server software will recognize the packets at not conforming to DNS specifications and will hence drop it), while all valid DNS traffic should still be handled transparently from the client's perspective.

This does not handle the subject of covert channels that use DNS, however. See section 3.5.1 for more information on how to mitigate these situations.

3.3 BitTorrent

BitTorrent is a peer-to-peer file sharing protocol that has seen widespread deployment across a range of applications in recent years. The basic concepts of BitTorrent are as follows:

- Every *message* is broken up into a collection of *chunks*, with every chunk being the same size. These chunks are the atomic units[8] that are transferred between clients. Typically these chunks range from 1 megabyte to 16 megabytes (for very large messages). These messages can be a single file, or a collection of files.
- Each chunk is hashed for integrity verification when the torrent is initially created. This ensures that any chunks that are damaged in transit can be detected and retransmitted. Additionally, it allows someone to use the original torrent file to verify the download at a later date to prevent bit-rot.

[7] It is important to note that this cannot be done for HTTPS, or otherwise secured, HTTP sessions. The security of these streams secures the stream from all tampering rendering this approach impossible. However, since most of the web browsing is done over unencrypted HTTP, this approach is still viable for most web traffic.

[8] Transmission is still done using TCP to mitigate against lost packets, but at the application level, chunks are the atomic unit of data. Partial chunks are typically discarded or stored for the next time the application is run so that they can be finished.

- Clients connect to a *tracker* which maintains the database of which clients have which portions of which torrent.
- Clients obtain a list of *peers* from the tracker when they connect, and this determines who they begin to connect to. When a client polls a tracker for a list of peers, they obtain the addresses of each peer, as well as their status on how much (and which portions) of the message they can transmit.
- Peers that have the entire message are called *seeds*.
- BitTorrent uses an efficient mechanism of distributing the file by ensuring that chunks of the message are distributed as uniformly as possible.

  For example, if a torrent is initially created with ten chunks, and there are a total of nine additional peers that want the torrent (but currently have none of it), then the initial seed sends a different chunk to each peer. This means that each downloading peer can only download from one peer, and that seed's upload speed is split amongst nine peers. Once each client is finished downloading its chunk, it checks its database and decides who to get the next chunk from. If all of the other nine peers finish at around the same time, then each peer then has nine peers to download a different chunk from, thus increasing *everyone's* download speed by almost an order of magnitude.

  This is the advantage of BitTorrent.

Almost all BitTorrent clients support encryption, so it is not possible to rely on identification of the content of the message. Many BitTorrent clients support IPv6, and so can generate a significant amount of IPv6 traffic on IPv6 enabled hosts[9].

It is important to note that BitTorrent has seen adoption into commercial and community applications, and so has many 'legitimate' uses that do not involve the illegal piracy of intellectual property. For example, Blizzard Entertainment uses BitTorrent to distribute content updates and patches for all of their currently supported computer games. These games include StarCraft II, and World of Warcraft. This works by allowing every person with the game currently loaded and patched to the latest version to act as a seed for new clients that need to obtain patches. By distrbuting patches this way, Blizzard is able to greatly reduce their bandwidth costs for patch distribution, while at the same time ensuring that their customers get very good download speeds on patches and content update.

The Linux and Unix distribution community has also seen adoption of BitTorrent as the protocol of choice for distributing and obtaining distribution media. Most distributions offer both traditionally HTTP/FTP served versions, as well as BitTorrent served versions. Almost all locations that host the HTTP/FTP served versions also participate as peer for those wishing to obtain their media through BitTorrent.

It is possible to detect BitTorrent traffic by exploiting the fact that it operates with a chunk of data as atomic unit of transmission. BitTorrent traffic would tend to produce TCP[10] flows that all have approximately the same size. In the case that multiple chunks are bundled into a single flow, this will produce a single flow with a size that is approximately an integral multiple of the chunk

[9]This is important to note, as it can produce a highly unexpected load on an IPv6 network that wasn't designed for high-bandwidth applications.

[10]BitTorrent rarely operates over UDP due to the fact that it is a high-bandwidth application and is a prime candidate for lost packets.

size. So, when looking for flows, it is important to look for collections of flows whose sizes share a relatively large greatest common denominator.

Since BitTorrent does have legitimate applications, and is gaining more adoption as time progresses, a categorical blocking of BitTorrent traffic should be carefully considered.

3.4 Dark IP Space and Darknets

Dark IP space typically refers to publicly routable IP addresses that do not have any hosts connected to them and are *dark* in that they should not be the source, or destination, of any internet traffic[11]. In practice, however, dark IP space is rarely dark and often is the destination for a very interesting class of internet traffic. By its very nature, most of the traffic that is arriving at unallocated IP addresses is suspect since there are very few legitimate reasons for such traffic to exist.

In practice, large internet security vendors use dark IP space that is both logically and geographically distributed to collect information from what could be termed the *background radiation* of the internet. These collections of computers that are sitting on the unallocated IP addresses are collectively termed *darknets*.

For example, a large DDoS attack that makes use of spoofed source IP addresses in order to mask the true source will often register in a darknet. This is because it is likely that some of the random spoofed source addresses will fall into a darknet, simply due to probability. By analyzing the traffic that these security vendors see in their darknets, it becomes possible for them to estimate such things as the number of DDoS attacks that are happening at any given time on the internet.

Darknets do not need to be large to be useful, and can be employed by anyone that has unallocated (but publicly routable) IP addresses. By monitoring the traffic that is getting to these dark IP addresses, it becomes possible to detect IP-range scans, port scans, and other indiscriminate suspicious behaviour. One example of such behaviour is SIP scanning, as attackers search for an open SIP server that is on the public internet.

3.4.1 Use as Real-Time Signature Tracking

It is possible to use the information gathered in a darknet to produce real-time signatures for use in filtering traffic to the live addresses. By building signatures on the traffic that is being seen on the darknet, and feeding these signatures into an IPS device, it is possible to produce a form of learning system that is capable of adapting to new threats without any human interaction.

The trick becomes finding an IPS that will accept signatures on the fly like this, as well as deciding how to produce signatures, and what the action should be taken on the signatures. Taking the SIP scanning as an example, if the darknet sees that the same source IP address is sending the same payload to the same port to all of the dark IPs, then it could push out a signature for the IPS to block all packets from that source IP to that port. This would be a reactive measure that may stop attacks like these before they get too far into their scanning procedure.

3.4.2 Extension to Email

It is possible to extend the concept of real-time signature tracking to spam email as well. By collecting all email that comes to dark email addresses[12] and analyzing the *content* of these unsolicited

[11] An obvious exception is broadcast traffic.

[12] That is, email addresses that aren't registered as having a human recipient. Call the set of all dark email addresses a *darklist*.

emails, it becomes possible to produce signatures for spam email that can then be pushed to an email filter in real time.

It is important to note that the content of these emails is analyzed, not the destination or source IP or email addresses. If the source address is relied on, it is very possible that one may risk collateral damage by blocking legitimate mail. By analyzing the content however, it becomes possible to uniquely identify the same spam email message even if it is sent from multiple sources.

For example, consider the situation where the same email mesage is sent to several hundred email addresses in the darklist, it is reasonable to assume that this message may be spam. By producing a signature on the content of the message, this message can now be blocked at the mail filtering device or mail server.

Further, since email is very tolerant to delays (unlike network traffic), it is possible to obtain far better automatic-signature performance. By delaying all email[13] by a short time it is possible to produce and apply signatures for spam email before any of the mail gets to any legitimate user's inbox. This will reduce the amount of zero-day spam that gets received by end users without loading down other network systems[14].

3.4.3 Tiered Honeypots

Resources: List of honeypot applications[15]

Honeypots do not need to be a full-fledged computer to be useful, and there are applications that are able to emulate certain network stacks, and the behaviour of certain applications in order to act as an *emulated honeypot*. One such application is HoneyBOT[16] which mimics the behaviour of many different services on over a thousand different ports without actually running any of the services.

By making use of emulated services, it greatly reduces the cost of running honeypots and will scale to a far greater number of addresses that can be monitored. It is then possible to make use of an adaptive handoff system that will hand off from an emulated honeypot to a 'real'[17] honeypot when the attacker exceeds what the emulated environment can provide.

This tiered handoff style honeypot framework would allow for collecting data from hackers by baiting them with low-cost, low-risk, emulated environments, and escalating those attacks that are of interest to a virtual machine tailored to their attack.

3.5 Detecting Covert Communication Channels

It is possible for many types of covert communication channels to be established on the internet, using widely supported and generally unprotected protocols such as ICMP and DNS, among others. The software that implements these tunneling methods typically makes use of virtual network interfaces to provide the network traffic to applications and the operating system. Using this method, it is possible to route a complete internet connection over such a tunnel without producing any "real" packets except for RFC-conforming DNS, DHCP and ARP packets.

[13] One method would be by inserting all email into a mandatory thirty second or one minute first-in-first-out queue at the mail server.

[14] See section 3.5.1 for details on the load that some security vendors place on a DNS subsystem.

[15] http://www.honeypots.net/honeypots/products

[16] http://www.atomicsoftwaresolutions.com/honeybot.php

[17] This would likely be a virtual environment that mimics the environment the attacker was attempting to compromise. If the attacker was attempting to compromise an emulated apache with a Linux-style networking stack, then the virtual machine the attacker gets handed off to would be Linux with Apache installed.

3.5.1 DNS Tunnels

Examples: iodine[18], dns2tcp[19], NSTX[20], DNScat[21], OzzymanDNS[22]

DNS tunnels offer an exceptionally robust, easy to deploy and very difficult to mitigate method of establishing a covert channel between two or more endpoints on the internet. Many security vendors make use of this fact in their security appliances and will often embed requests for signature verification into DNS queries and responses. This is in order to ensure that these requests are not blocked by firewalls at the client's location.

The important components of a DNS tunnel infrastructure are:

Domain: The most onerous requirement of a DNS tunnel is that a subdomian of a publicly resolvable domain must be delegated to the IP address of the DNS tunnel's server endpoint. It is the delegation of this subdomain that allows the rest of the scheme to operate.

Server: The server in a DNS tunnel is a DNS server that is designed to handle DNS requests in a non-RFC-compliant fashion. When this server receives requests for *.tun.example.com, the requests are treated as tunneled network traffic and not as DNS requests.

Client: The client in a DNS tunnel is similar to the server in that it encodes raw network traffic into the query string of a DNS packet in the form of a query for "*.tun.example.com".

The server-side of the tunnel typically decodes the query strings from the DNS packets and puts together the raw network traffic. When sending information back to a client, the server crafts responses to the original query and encodes the reply network traffic into answer portions of the packets.

The reason that DNS tunnels are so difficult to detect and block is that all packets that cross the internet conform to the RFC for DNS, and as such will not be directly distinguishable from normal DNS queries. Because these abnormal queries look, from a protocol standpoint, identical to normal queries, it is necessary to apply some statistical measures in order to be able to determine when a DNS tunnel is present.

In order to detect a DNS tunnel, it is possible to exploit the fact that all queries and responses must be for subdomains of some common domain. By identifying the top-most domain[23] for each DNS query and response, it is possible to group all queries and responses by domain and then compute statistics on each domain. By computing the entropy of the queries for each domain, it is possible to clearly identify when a domain is being used to support a DNS tunnel due to what will be an exceptionally high entropy[24].

Once a DNS tunnel has been detected, it is then possible for a standard IDS/IPS appliance to block the tunnel by blocking the domain that it is using. Because the tunnel must have a subdomain delegated to the tunnel server, it is possible to block the DNS tunnel without disrupting any other internet traffic. That is, it is possible to block a DNS tunnel with zero collateral damage.

[18] http://code.kryo.se/iodine/
[19] http://www.hsc.fr/ressources/outils/dns2tcp/index.html.en
[20] http://savannah.nongnu.org/projects/nstx/
[21] http://tadek.pietraszek.org/projects/DNScat/
[22] http://www.dnstunnel.de/
[23] It is important to identify the topmost domain that is not a public suffix. There is a community maintained list of public suffixes available at http://publicsuffix.org/ that can be used for this purpose.
[24] Intuitively, a DNS tunnel will rarely repeat the same query more than once, and will have very many queries over a window of time. This will be very different from normal domain behaviour where only a few subdomains will be represented, and they will occur with great frequency.

3.5.2 ICMP Tunnels

Examples: PTunnel[25]

ICMP tunnels offer an alternative to DNS tunnels that make use of ICMP packets that are almost as ubiquitously allowed through firewalls as DNS. They offer the distinct advantage over DNS tunnels that they do not need to have a publicly resolvable DNS subdomain delegate to the server. Instead, they rely on the fact that ICMP messages such as echo pings are allowed through firewalls as part of being a "good netizen". Like DNS tunnels they do require specialized client and server software in order to translate the network traffic into the payload of the carrier stream.

ICMP tunnels typically embed their payloads into the data content of large ping packets (as is done by PTunnel), and so provides an avenue for blocking ICMP tunnels. Since ICMP pings are typically used diagnostically to test layer 3 connectivity, there is no need for ping packets to be any larger than the minimum allowable packet size. Since ICMP sits on top of a layer 3 protocol such as IPv4 or IPv6, then the minimum necessary packet size is 8 bytes for the ICMP header plus the size of the layer 2 and 3 headers[26]. For IPv4 with no options set, this results in a minimum packet size of 40 bytes.

This information on packet sizes can be used to protect against ICMP tunnels. By modifying the packets as they cross the internet gateway so that ICMP pings are never longer than their minimum acceptable length (that is 8 bytes plus the length of the layer 2 and 3 headers), it is possible to make it impossible for ICMP tunnels to successfully operate. By employing this method of mitigation, it still allows devices to verify layer 3 connectivity by using ICMP pings which is the primary purpose of ICMP pings.

The collateral damage incurred by this method is minimal, and will only affect applications which rely on the data field of ICMP pings. One such situation is using large ping packets to verify that jumbo frames[27] are in fact enabled on hosts, or a portion of the network. Using large ICMP pings will no longer suffice for this task if the packets need to cross this packet-shaping point, since the packets will always be reduced in size by the packet shaper[28], it is necessary to find a different method of verifying jumbo frames.

3.6 Black-listing vs White-listing

Black-listing and white-listing are two diametrically opposite approaches to blocking and allowing content. While black-listing involves deciding which content to block, and allowing all else, white-listing is the process of deciding which content to *allow* and blocking all else. In a situation where the overwhelming majority of content is unwanted, and only a select set of content is allowable, white-listing is the clear choice.

In a situation where internet content can be partitioned[29] into three categories - allowable, unallowable, and unknown - the situation becomes more complicated. It is no longer an option to simply apply a white-listing technique, since there may still be a great deal of allowable content in

[25] http://www.cs.uit.no/~daniels/PingTunnel/

[26] See http://en.wikipedia.org/wiki/Ping for a full description of ICMP echo requests.

[27] Jumbo frames are a gigabit ethernet feature that allows for the use of an MTU larger than 1500. This larger MTU is typically in the range of 7000-9000 bytes.

[28] This is an acceptable situation, since it is very rare that someone will be attempting to verify jumbo frames across the internet. Typically, jumbo frames are only used on LANs between two hosts that are performing bulk data transfer.

[29] Partitioning a collection of items involves splitting them into sets, where all of the sets together don't leave any item out, and no item appears in more than one set. That is, each item appears in exactly one set.

the unknown class of traffic. Black-listing, however, can result in large and difficult to maintain lists of content or providers that must be blocked. White-listing presents some serious problems of its own though, especially in situations where a considerable portion of the traffic is of unknown reputation.

In situations such as these, a hybrid approach is worth considering. By white-listing what can be white-listed, and only applying black-listing techniques to those portions that need it, it may be possible to reduce the total number of rules that need to be maintained. Fewer rules will likely result in simpler maintenance and administration of the rule sets.

3.7 Detecting Periodic Events

It is possible to apply some non-trivial mathematics to be able to identify strongly periodic[30] events in network traffic. Based on what is known about network infrastructure, very few things are strongly periodic. Routing table updates and DHCP renewals are two network management applications that produce strongly periodic events that would be detectable.

Because of how network switches operate, certain very fine-grained[31] periodic information is lost in the multiplexing process. As a switch multiplexes 10 ports into one trunk port, the order of the packets is not guaranteed and so the very fine-grained timing information is inherently lost. This is acceptable, and in fact inconsequential, because any periodic information that was that fast is of little interest. The periodicity information that survives the networking infrastructure intact is of significant interest since this behaviour is necessarily caused by the application.

Some examples of strongly periodic applications are:

- Routing protocol updates
- DHCP renewals
- Scripted events, run with cron or other schedulers, that produce network traffic.
- Email clients polling servers for new email
- Keyloggers or other applications *calling home* at a regular interval.
- Websites that use periodic refreshes to check for updates.

These application events would be detectable if it was possible to break up traffic into the frequency components of it. In theory, if employees were regular enough, it would be possible to observe a spike when the office administrator comes in every day at 8 AM to start her computer.

Extending this further, it would be possible to identify what portions of the network traffic are largely time-of-day dependent. Once these portions are determined, it is possible to 'subtract' them from the data and observe what remains.

The mathematics behind this approach are not well formulated, and this is currently an unsolved problem due to several implementation-specific challenges.

[30] Strongly periodic events are events that occur, with very little deviation, at specific intervals. An event that occurs every five minutes, give or take a couple seconds, is strongly periodic. Similarly, an event that occurs every week at the same time, give or take a couple minutes, is strongly periodic.

[31] On the order of milliseconds.

4 Types of Botnet Command and Control Channels

This section lists some common botnet command and control channel methods. This is not meant to be exhaustive, but rather instructive in order to get investigators thinking in the right mind set:

- Custom protocol
- IRC using an IRC chat channel
- Chat client protocols such as Jabber, MSN, AIM, ICQ, etc...
- Well-known web protocols such as HTTP or FTP.
- Web-based applications such as twitter updates, Facebook status updates or Linked In profile fields.[32][33]
- JPEG or other image metadata. These images can be hosted on any free image hosting website.

The last few items on the above list are exceptionally difficult to detect unless algorithms are specifically looking for that type of behaviour.

Any communications can, and should be assumed to be, encrypted. In the cases where there is a carrier stream, such as the JPEG meta-info, it should be assumed that the meta-info in the JPEG is encrypted, even if the JPEG is not.

4.1 Reliance on DNS

Botnets can operate in one of two fashions: either they use DNS, or they do not. Botnets that do not use DNS are subject to making use of static lists of IP addresses, and so suffer from vastly reduced flexibility. Botnets that make use of DNS are subject to requiring the DNS subsystem in order to function, which introduces an important choke point for the botnets that can allows us to stop them before any network data is sent.

To start with an example, the Konficker worm that plagued Windows for many generations made use of DNS in a very noticable way. Whenever it was time for the botnet to check in, each bot would generate up to 500,000 DNS names, all using the same random algorithm, and the same seed, and test them all. If they resolved, then the bots would attempt to fingerprint the command and control server that the DNS name resolved to. This would have caused major spikes of strange DNS behaviour at regular intervals. If this DNS traffic was blocked, then the botnet would be unable to check in at all, as well as rendering it unable to obtain updates or commands. In other words, stopping the DNS answer from returning would neuter the botnet.

This is still the case, but botnets are significantly less obvious. The reliance on DNS, however, still exists and should still be considered an important avenue for mitigating and detecting botnets.

[32] These social networking websites often have robust APIs that allow the controller to programmatically update the fields.

[33] It is very important to note that the content of these Twitter updates, or LinkedIn fields need not be encrypted or unintelligible. There may a type of 'language', where if a status update says *Hey, go check out www.example.com!* is interpreted by the bots as a command to execute a DoS on www.example.com. http://revision3.com/hak5/snubs-sugar contains a very informative interview with a botnet C&C author that discusses many of these points.