We are IntechOpen,
the world's leading publisher of
Open Access books
Built by scientists, for scientists

4,800
Open access books available

122,000
International authors and editors

135M
Downloads

Our authors are among the

154
Countries delivered to

TOP 1%
most cited scientists

12.2%
Contributors from top 500 universities

CLARIVATE ANALYTICS
BOOK CITATION INDEX
INDEXED

WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index
in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?
Contact book.department@intechopen.com

Numbers displayed above are based on latest data collected.
For more information visit www.intechopen.com

Advanced Methods for Botnet Intrusion Detection Systems

Son T. Vuong and Mohammed S. Alam
University of British Columbia
Canada

1. Introduction

Today, our dependence on the internet has grown manifold. So has the need to protect our vast personal information accessible via web interfaces such as online passwords, corporate secrets, online banking accounts, and social networking accounts like Facebook. The appearance of botnets in the internet scene over the last decade, and their ever changing behavior has caused real challenges that cannot be easily remedied.

According to literature, a *botnet* is defined to be a set of infected hosts (also called bots or zombies) that run autonomously and automatically, controlled by a botmaster (bot herder) who can co-ordinate his/her malicious intentions using the infected bots. Some of the prominent malicious tasks that can be credited to botnets include DDoS (Distributed denial-of-service), spam, phishing, ransomwares and identity theft.

In a botnet DDoS attack, the botmaster can command all its bots to attack a particular server (example: update.microsoft.com) at a particular date, time and for a duration via a malicious or anonymous proxy used as a stepping-stone to hide the actual commanding node. In a spam campaign, the nodes that form the bot network are responsible for sending spam by behaving as spam relay points, delivering spam mails to a list of intended victim email addresses selected by the botmaster. For example: a node which is part of a spam botnet could be sent a list of email addresses to spam for the day with a payload of the spam that is to be mailed. These spam messages could advertise pharmaceutical products and may also deliver further infection executables via email links or attachments to recruit more bots, as done by botnets such as *Storm* and *Waledac*. In a phishing scam, botnets are responsible for acting as web proxies or web servers to deliver hoax site content to benign users to gather their e-banking or credit card credentials. For example, the sites could host content which looks like a banking site requesting for login details credentials which when entered by the user, can be used by the botmaster to access legitimate banking sites. Eventually the funds are transferred to accounts that leave no trails (Nazario & Holz, 2008).

Botnets such as *Storm* have been known to infect over 2 million hosts while *Conficker* has infected over 9 million hosts according to some estimates. As can be seen, the far reaching effects of malicious intentions of botnets and their masters are a real threat.

This chapter will cover a concise survey of botnet detection systems as well as provide a novel mobile-agent based method that has been adapted from mobile-agent based intrusion detection systems, for handling botnets. We provide the necessary background needed to understand botnets such as the offensive techniques utilized by botnets; the defensive

techniques developed by researchers; and also focus on a mobile agent based technique to detect infected hosts.

2. Botnet offense

In order to better understand the challenges that the security community faces in order to dismantle botnets, we first need to understand how botnets function, and the many tools and techniques employed by them.

2.1 Setting up a command and control server

The first step in creating a botnet is to setup the Command and Control (C&C) server. This is the location where the infected hosts report to the botmaster, letting it know that a host has been infected successfully. This is also the location where the infected hosts retrieve the full list of commands that the infected bot should run. Section 2.3 covers some of the communication features of a C&C server.

2.2 Bot lifecycle

Unlike the initial advanced botnets such as *Agobot* which carried a list of exploits to perform on a vulnerable host and its entire command set at the time of initial infection, every advanced bot today uses multiple stages in order to form a botnet (Schiller et al., 2007; Gu et al., 2007). This was mainly done first, to avoid signature detection by network intrusion detection systems such as snort (Roesch, 1999) and second, to reduce the initial infection size of the bot binary to make it less traceable while using *drive-by-download* attacks.

Stage 1 of a bot's lifecycle is the initial infection/exploit of a host. In this step the bot binary has to first infect the host by attempting to exploit one or more security vulnerabilities that might pre-exist on a system. Section 2.4 provides further details on the associated techniques that botmasters could use in this step. Once infected, *stage 2* is the process by which the bot reports back to the botmaster using the command and control (C&C) channel to inform him that the host has been successfully compromised. Information related to the host such as opened backdoors, host operation system settings and network capabilities are just some of the details that are reported back during this phase. In *stage 3* the bot downloads new executables. This process is also referred to as *egg downloading*. This could be the component that detects and disables antivirus software, or could provide potential updates to the bot malware with its full command list to make it more functional. In *stage 4* the downloaded malware is executed on the bot. The bot at this stage has become fully functional. In *stage 5*, the bot starts listening to the command-and-control channel to retrieve payload information from peers or servers and could execute the commands that are passed on using the payload. It is not necessary that the channel used in stage 3 is the same channel used in stage 5. In *stage 6*, the bot could optionally report the results of executing the commands to the server. This feature is used by many botnets to track the functionality of the bot so that the botnet could be load-balanced.

2.3 Botnet communication structure

The most important component of a botnet that decides if it can be easily dismantled is its communication structure which is used to command and control the infected hosts of a botnet. The type of communication used between a bot client and its command-and-control

server or between any two bot clients can be differentiated into two types: *Push-based* commanding or *pull-based* commanding. Each method has its own advantages and disadvantages.

In a push-based communication, the bot master "pushes" the command that the bots are to run. The advantage of push-based communication lies in the fact that botmasters can instantaneously ask bots to perform a certain task. This allows for tighter control. However, the inherent disadvantage of such a method is the amount of traffic that can be observed leaving the server, or the tight timing correlation between various monitored nodes that are part of the same botnet, leading to easier detection of infected hosts. This weakness has been utilized by most botnet detection techniques such as *Botsniffer* (Gu et al., 2008). An example of push-based communication is the use of IRC servers for command-and-control.

In a pull-based communication, each bot is allowed to periodically retrieve the next command to run from a server. This helps not only to avoid flash-crowds at a command-and-control server, but the injection of random delays in command retrieval from bot to server makes it more difficult to trace a command-and-control server. This allows the server to hide behind traditional web traffic. Most existing botnets used for spamming (5 of top 9) use http protocol, a pull-based communication, to masquerade communication as legitimate users (Steward, 2009). In addition to the primary channel of communications, bots also have a secondary communication usually in the form of backdoors created by Trojans/bot software installed in each infected host. This channel is only used by the botmaster if the primary communication channel has been compromised.

We now elaborate a little on some of the more common communication structures used by botnets.

2.3.1 IRC (Internet Relay Chat)

In the beginning, most botnets used a centralized approach for managing botnets (Bacher et al., 2005). This was done using the IRC (internet relay chat) protocol or modified versions of the protocol using freely available sources such as *UnrealIRCd* (unrealircd, 2010). As per (Schiller et al., 2007), the main reasons for using IRC were its interactive nature for two way communication between server-client; readily available source code for easy modifications; ability to control multiple botnets using nicknames for bots and password protected channels; and redundancy achieved by linking several servers together.

Most IRC servers are modified from the original IRC protocol so that not all clients are visible to each channel member, or the server only listens to commands entered by the botmaster. Most bots parse the channel subject to be the command issued by the botmaster (Bacher et al., 2005). However, since these servers become the single point of failure and are easily detected, botnets have moved to other decentralized methods of control such as P2P; use of other less detectable protocols (http web servers); or use of IRC in combination to DNS fast-flux techniques, as explained in section 2.4.1. This was mainly due to the increased ability of the research community to reverse engineer the bot binary using tools such as *IDA pro* (Hex-rays, 2010) and mimic the behavior of a bot by joining and monitoring a botnet (Bacher et al., 2005; Rajab et al., 2006).

2.3.2 Web based botnet

The most prominent communication structure for botnets after IRC is the used of web servers. This is mainly done since most firewalls cannot distinguish between web-based bot

traffic, and legitimate web traffic. The botmaster could be informed via an http request of the backdoor port to be used for communication along with a password to connect to the bot in case a secondary channel is required for communication.

2.3.3 P2P (peer-to-peer)

Probably the most complex botnet that had been studied to use a P2P scheme was the *Storm/Zhelatin/Pecomm/Nuwar* botnet and its variants. This botnet used a P2P approach to communicate commands between its bot members (Holz et al., 2008) based on the edonkey (*Overnet* protocol based on the Kademlia P2P algorithm) protocol followed by its custom *stormnet* (XOR encrypted communications) protocol to communicate. Using an off-the-shelf protocol that relied on unauthenticated publish-subscribe system allowed researchers to infiltrate the botnet. The number of botnets that use the P2P approach is less mainly due to the complicated nature of the C&C structure and due to the fact that once defenders have control of one infected host, it is easier for them to detect other infected peers connecting to it. *Nugache* is another P2P based botnet that uses encrypted peer communications. A noteworthy feature of *Storm* is the additional feature of automatically attacking anyone attempting to track it i.e. any storm infected node that was not behaving appropriately would be DDoSed by the system. This made it increasingly difficult for researchers to understand how the botnet functioned (Holz et al., 2008).

2.3.4 Other communication protocols and proposed botnet features

Botnets have also been detected to use one of many other uncommon protocols such as instant messaging for C&C. Using instant messaging for C&C has the drawback of being easily tracked and taken down by the instant messaging provider. ftp *dropzones* for banking Trojans have also been observed by (Holz et al., 2009). As per the authors, botnets used for stealing banking credentials submit keylogged data from phishing attacks into dropzones. The authors discovered a few authentication free *dropzones* during their investigations.

Some researchers have also proposed advanced techniques that could be used by botnets in the future. (Singh et al., 2008) discusses the use of email as C&C by using a combination of encryption and steganography in email content. The email content could be send to the user's inbox or spam folder at the direction of the botmaster by picking the right keywords. (Bambenek & Klus, 2008) proposed the possibility of using RSS feeds or XML for communication via websites maintained by botmasters, or public bulletin boards not controlled by the botmaster. (Hund et al., 2008) proposed a new bot design called *Rambot* that uses peer-to-peer technology in addition to using strong cryptography (2048 bit RSA keys) where the public key of botmaster would be hardcoded into the bot binary. Use of Diffie-Hellman symmetric key between bot-bot communications was also proposed by the authors in addition to the possibility of using a credit-point system to build trust among bots. The authors also discuss about peers only sharing partial peer lists with other bots to avoid detection of all peers in the botnet. In order to avoid allowing defenders to simulate multiple nodes on a single host, the authors also discuss about presenting a challenge(5-15 minute task) to any node before it communicates. This however has the drawback of the bot being detected by regular users. (Wang et al., 2007) proposes concepts similar to (Hund et al., 2008) in the use of asymmetric keys for bot-botmaster communication, symmetric keys for bot-bot communication and the use of peer-list exchange where only a partial list of peers are exchanged only during reinfection attempts. (Vogt et al., 2007) proposes creating

infections where thousands of smaller botnets are created with each having its own C&C server. Also all commands require a decryption key based on both the public key of the botmaster and another key that is partitioned such that each botnet has a partial key K_i. Defenders would need to infiltrate each separate botnet to gather the entire decryption key.

2.4 Infecting the user

The primary step that creates the botnet is the initial infection of the user which converts a clean host into a bot. Users can be infected in one of three ways.

Drive-by-downloads

As noted in numerous papers (Provos et al., 2007; Wang et al., 2006; Ikinci et al., 2008; Siefert et al., 2007; Provos et al., 2008) drive-by-downloads have become an emergent threat that exploit weaknesses often seen in web browsers and browser plugins.

In this process, the user is infected by a malicious link embedded in the site that based on the *user-agent* (browser) starts off a series of attacks (attack vector) to download malware into the user's machine without any acceptance by the user other than to have visited the site. This malicious site could be hosted by a malicious entity; could have 3rd party advertisement links which load malicious content; or be a legitimate site that has been infected earlier. 3rd Party advertisements could include the action of syndication (Daswani & Stoppelman, 2007) by which space on a site is sold for advertisement links to 3rd party sites that serve the ad content. Legitimate sites could be infected by a SQL injection attack which would then contain malicious iframe pointers to malicious servers. Unlike network scanning for vulnerabilities, which are blocked by firewalls and NATs, drive-by-download uses a pull-based technique that bypasses most filters. (Provos et al., 2008) notes that many of the infected hosts show connections to IRC servers or HTTP requests soon after infection confirming the fact that drive-by-downloads lead to creation of botnets. The malware or malicious iframe pointers are usually obfuscated within the html source. Instead of reinventing the exploits, these malicious links use ready-made exploit packs such as *Mpack*, *IcePack* or *EL Fiesta* that contain customized infection vectors. Though (Provos et al., 2008) mentions scanning well known URLs to check for maliciousness, a URL may seem benign in the beginning during initial scan, but might start behaving maliciously at a later time. The authors reported the presence of over 3 million malicious URLs detected over a 10 month period, and 1.3% of search results returning malicious URLs in Google searches.

Malicious attachments (social engineering)

A few botnets such as *Storm* & *Waledac* use social engineering as the attack vector. In this process, users are emailed a web link, hosted by a node in the bot network that a benign user would be enticed to visit. Once the URL is visited, the botmaster uses social engineering such as the need for missing flash plug-in or video codec to entice the user into downloading an executable and thus infecting the user. The use of custom packers and added encryption makes it almost impossible for antivirus software to detect maliciousness of the downloaded binary.

Vulnerable hosts

Most botnet attack vectors still target hosts that have not been fully patched. For example, some of the initial botnets such as *SDBot, Rbot, Agobot, Spybot* and *Mytob* were formed due to various windows vulnerabilities. Similarly the recent worm having a botnet commanding

structure (*Downadup/Conficker/Kido*) that exploits MS08-067 spreads primarily due to inadequate patching. As pointed out by (Brumley et al., 2007; Brumley et al., 2008), attack vectors for a vulnerability can be created within hours of a patch being made available by a vendor. The difference between a patched and an unpatched version of the software allows malware authors to detect the underlying vulnerability that unpatched systems are vulnerable to.

2.5 Advanced botnet features
2.5.1 Obfuscation
The primary reason for using obfuscation is to make it difficult for botnet defenders to detect and tear down the inner functioning of a bot malware by simple signature matching. This is accomplished in many ways. For example, web-based malware (used for drive-by-downloads) uses JavaScript obfuscation to hide the attack vector. Web based malware is easier to obfuscate than memory corruption vulnerabilities and cannot be caught by state of the art polymorphic worm detectors such as Hamsa (Li et al., 2006) and Polygraph (Newsome et al., 2005). Another method includes the use of *packing* followed by *encryption* of bot binaries that causes bot binaries to go undetected by signature detectors. For example, storm is packed every minute by a custom packer built into its code whereby the size and thus MD5 hash no longer match to previous bot binary samples of the same malware.

2.5.2 Fast-flux
Most advanced botnets (*Storm, Waledac*) used primarily for phishing and spam use fast-flux techniques to hide the actual servers responsible for updated copies of the malware. In a fast-flux technique, the DNS to IP mapping of the download location of the malware constantly changes such that blocking an IP address does not really help, or correlating information about a particular infection based on just the IP is no longer useful enough. Some botnets use double fast flux using multihoming to change both the A record and NS record of DNS (Holz et al., 2009; Nazario & Holz, 2008; Siefert et al., 2007).

2.5.3 Virtual-machine detection
Quite a few malicious bot applications have inbuilt functionality to check if the host that has been infected is running in a virtual machine. Some characteristics include registry entries caused as artifacts of running various virtual machine software; list of running processes and services; or attempt remote timing attacks (Franklin et al., 2008) where the bot code runs a set of instructions in a loop leading to difference in results compared to a real system. It has also been noted by researchers in the virtual machine field that virtual machines will continue to be detected regardless of hardware support for virtualization (Garfinkel et al., 2007) mainly due to the difference in goals of the virtualization community.

2.5.4 Rootkits
Most bot code packages such as *Rustock, Storm* and *rxbot* uses rootkits to hide its presence to antivirus and malware detection tools. In these cases the bot binary package contains an executable which causes inline-function-hooking of important windows kernel dll functions such as kernel32.dll to hide the actual bot binary files from detection. An example rootkit used by hackers include *Hacker Defender*.

2.5.5 Random generation of domain names

Some newer botnets such as *Conficker* and *Kraken/Bobax* use random generation of domain names in search of the Command and Control servers. While *Kraken* walked through its generated list in serial order, *Conficker* generates a new list every day that has not been registered yet. Once the first C&C server is connected to, *Conficker* could activate its botnet structure (Pierce, 2008). This feature of trying to connect to non-existent servers could act as a give-away in detecting bot infections.

3. Botnet defense

In trying to keep pace with botnets, defenders have constantly tried to mitigate the harmful intentions of botnets by coming up with novel solutions, targeted at the core architectural footprint of botnets. Some of the solutions use static analysis techniques via reverse engineering the bot binaries using programs such as *IDA pro* or *peryleyez* (Holz et al., 2008; Grizzard et al., 2007; Chiang & Lloyd, 2007). Other approaches have used a dynamic analysis approach using tools such as *cwsandbox* (Sunbelt, 2010) or *norman* sandbox by performing windows API hooking; or performing system wide dynamic taint tracking (Tucek et al., 2007; Yin et al., 2007).

Botnet emulation approaches testbeds such as EMUlab/ DETER/ WAIL (Barford & Blodgett, 2007) have also been used to emulate an entire botnet by setting up command-and-control servers, infected clients and local DNS resolvers.

Work related to the area of drive-by-downloads has been done by (Provos et al., 2007; Provos et al., 2008) using honeynets to monitor URLs that might be malicious. These honeynets browse the URL using internet explorer via client-honeypots and track the number of new processes created, number of registries modified, and number of files creates due to visiting a site. The use of honey-clients to monitor changes while visiting URLs such as the Strider Honey monkey project (Wang et al., 2006), the Monkey spider project (Ikinci et al., 2008) or the use of behavior analysis tools such as Capture-BAT (Seifert, 2008) fall under the category of detecting drive-by-downloads. DNS monitor approaches have been used for lookup behaviors commonly used by bots using active methods such as DNS hijacking (Dagon et al., 2006) or passive methods such as DNS Black listing (Ramachandran et al, 2006). We now discuss some of the broader approaches that have been taken for botnet detection.

3.1 Botnet detection using honeypots

The main research methodology to detect and infiltrate botnets in the past few years has been via the use of honeypots. A honeypot can be loosely defined to be a machine that is closely monitored to watch for potential infiltration. The honeypot machine could be a real vulnerable machine but is usually a machine running in a virtual environment. The use of honeypots lies in the fact that any traffic that tries to penetrate or contact a honeypot can be considered as inherently malicious since by default, honeypots do not by themselves contact other hosts unless instructed to do so and hence should not exhibit any network traffic. The use of more than one honeypot in a network is called a *honeynet*. The purpose of a honeypot defines its type. Some of them include (Riden & Seifert, 2008):

Client honeypots: A Client honeypot is a machine that looks for malicious servers, behaving as a client. Some of the prominent projects in this area includes *Strider Honeymonkeys* (Wang et al., 2006), *Monkey Spider* (Ikinci et al., 2008), *Capture HPC* (Seifert, 2008), *Shelia*

(Rocaspana, 2007) and the *MITRE Honeyclient* (Mitre, 2007). Most of these projects use links (URL) gathered from spam traps as seed values, and then actively visit the sites using a virtual machine that contains different levels of patching. This allows the system to detect the vulnerability attacked, and the configuration of the vulnerable machine. *High Interaction Honeypot:* 3rd generation honeywall ('Roo") (Roo, 2005) is a high interaction honeypot that allows the attacker to interact at all levels. The honeywall is placed between a honeynet and the outside world, collecting data from network. Roo uses *snort-inline* (snort-inline, 2005) to block all outgoing attack traffic from the honeynet. *Low Interaction Honeypot:* A low interaction honeypot emulates vulnerabilities rather than hosting an actual vulnerable system. Thus these types of honeypots can be easily detected if an attacker interacts with this node. These are mainly useful for automated worm like bots that spread. Some known examples include:

- *Nepenthes* (Baecher, 2006): Emulates multiple windows vulnerabilities on various windows ports via stateful finite state machine. It has the ability to emulate 16,000 IP addresses on a single machine. Meant to collect self replicating malware automatically. Contains 21 different vulnerability modules. Has a module for parsing shell codes that are XOR encoded and a module for retrieving binary from remote server obtained by parsing shell code.

- *Honeyd* (Provos, 2007b): Implements a small daemon which creates virtual hosts on a network. Allows one to create a simulated network of over 60,000 hosts on a single host allowing real hosts to co-exist among virtual hosts, thus making it extremely difficult for attackers to track down the real hosts in the network. Each host feature can be configured separately. (Li et al., 2008) used one year worth of honeynet data captured using half darknet sensors and half honeyd sensors to reach the conclusion that most botnet nodes scan randomly rather than scanning just a specific local IP range in most cases.

3.2 Spamming botnet detection

Given that the primary utility of botnets is in sending spam, many researchers have looked into analyzing botnets that are used exclusively for sending spam such as the *Storm, Srizbi* and *Rustock* botnets. Though the size of spamming botnets has reduced significantly due to internet service providers blocking C&C servers as well as the domain providers for these botnets, spamming botnets remain an active threat (Steward, 2009). (Ramachandran et al., 2008) used a DNS blacklisting technique (DNSBL) where it creates a graph of nodes that are in any way linked to the known *srizbi* botnet. If a bot belonging to *srizbi* queries a large DNSBL of an internet service provider, correlation of the querying node or the one being queried with the *srizbi* list gives a list of new peers who are infected by *srizbi*. This process could be repeated multiple rounds to find out all associated bots which send spam. Spamming botnets have also been detected based on using hotmail spam mail as seed data and detecting source of the mail using domain-agnostic signature detection (Xie et al., 2008; Brodsky & Brodsky, 2007).

3.3 Network-based botnet detection

Some botnet detection systems have relied on detecting bot traffic using network level data. This is mainly done using network sniffing intrusion detection tools such as snort in addition to other network flow monitors.

Bothunter (Gu et al., 2007) uses a vertical correlation algorithm which tries to capture the different steps of a bot life-cycle. The 5 stages of a bot used in Bothunter are *Inbound scanning* where network monitoring is done to see if an internal host is scanned for from external host; *Exploit usage* where an exploit packet is sent from external to internal host; *egg downloading* where a binary of the malware is retrieved by the infected host from the outside network; Outbound *bot coordination* dialog where Command and Control traffic is observed; Outbound *attack propagation* where the internal host attempts to attack an external host. The system throws a warning if atleast 2 outbound bot attempt stages are seen or evidence of localhost infection followed by a single outward communication from infected host is seen. The authors use a combination of snort and anomaly detection tools called SCADE and SLADE for detection.

BotSniffer (Gu et al., 2008a) uses network-based anomaly detection approach to detect C&C channel for IRC in a local area network by implementing modules as snort preprocessors. Their algorithm is based on the fact that IRC has a tight spatial-temporal correlation on the size and duration of packet lengths observed during an n-gram (2-gram) analysis for homogeneity check of communication packets.

BotMiner (Gu et al., 2008b) uses a horizontal correlation algorithm to detect bot traffic which detects both centralized command-and-control structures and peer-to-peer command-and-control structures. The authors partition every network flow into an Activity-plane (A-plane) and a Communication plane (C-plane) based on the type of traffic. A-plane is monitored by snort and modules from the BotHunter program. C-plane uses binning technique to read four network quantities such as flows per hour, packets per flow, average number of bytes per packet and average number of bytes per second. Once flows that have the same C-plane state are clustered, a cross-correlation plane is calculated to figure out which nodes are part of the same botnets based on a scoring function.

(Strayer et al., 2008) uses network monitoring to try to correlate traffic in a local area network to detect bots based on the tight correlation in the timing of IRC-based bot traffic to the server. The authors used a modified version of the *Kaiten* bot to connect to their own internal IRC server (*UnrealIRCd*) to collect data via *TCPdump*.

Some IRC-based botnet detection work has also been done by (Karasaridis et al., 2007) which looks at traffic flows obtained by a Tier-1 ISP and correlates the data to locate the commanding server and hosts.

Some botnet defense techniques rely on cooperation from every Autonomous System (AS) which is currently not feasible due to privacy issues. (Liu et al., 2008) proposes the use of *Stop-It* servers that are supposed to stop internal nodes from performing denial of service attacks if reported by another autonomous system. Similarly (Simon et al., 2007) also relies on setting an evil-bit for traffic arriving from an autonomous system that cannot be trusted. Overall the system behaves similar to the system proposed by (Liu et al., 2008).

(Stinson & Mitchell, 2008) discusses the evasion techniques that can be used to defeat the various network based botnet detection approaches used. They come to the conclusion that network-based botnet detection systems that rely on packet timings and size of packets can be easily defeated by random modifications to the measured variables associated to a network packet. Similarly (Cooke et al., 2005) reports that there is no simple connection based invariant useful for network detection. Their conclusion was based on data collected from the Internet Motion Sensor project. They measured that the length of the outgoing connection from bot to botmaster varied from 9 hours to less than a second. Some botnets

had traffic that was encrypted along with random noise insertions to thwart signature detection.

3.4 Behavior analysis based botnet detection

More recently, researchers have attempted to detect botnets by tracking their network and host behavior. (Bayer et al., 2009) recently proposed the correlation of behavior analysis of malware via clustering of behavior of host system calls via their ANUBIS dynamic analysis tool and the use of Locality Sensitive Hashing (LSH) clustering algorithm. Their tool works by performing an offline analysis of a malware sample similar to CWSandBox. The authors mention that capturing behavior at a system call level causes data explosion and increased false positives and negatives if an adversary has the knowledge that a system is tracked at a system call level.

(Bailey et al., 2007) uses hierarchical clustering based on measuring normalized compression distance where distances are measured by computing the zlib compressing of features, stored in random order. Each feature is represented by registry modifications made, processes created, file modifications made.

(Rieck et al., 2008) uses support vector machines to calculate which malware group a malicious executable represents, based on supervised learning using 10-fold cross validation of certain bot families. They compute feature vectors computed from CWSandbox reports.

(Lee & Mody, 2006) perform k-medoid clustering of events generated by running malicious executables. Each event is represented by file modifications or registry changes. They use edit distance of events among executables to cluster. They showed that edit distance measurements for distance do not work when the number of events goes higher than 500. Using k-medoid also has the drawback that the actual number of clusters has to be predetermined. Having a *k* which is less than the actual number of clusters cause outliers to be included, thus significantly impacting the cluster features.

(Gao et al., 2005) had proposed the use of applying DNA behavior distance of sequence of system call subsets by calculating distance between *system call phrases* of a given process and its replica. Their approach works by computing the edit distance between any two system call phrases, where a phrase is a sequence of system calls. However their work has limitations as the distance between system calls can be artificially increased by malicious adversaries by making unnecessary system calls.

4. Agent technology

Though various methodologies have existed for botnet detection, the use of agent technology has been mostly overlooked. Given the distributed nature of botnets, and their modular structure allowing for constant updates, it is more intuitive to use a similar technology that is inherently distributed and allows similar kind of code updates for defensive purposes. The need for a clear understanding of agents is necessitated due to the fact that the system that we have developed and extended, is layered on top of an agent platform, *Grasshopper* (Bäumer & Magedanz, 1999), based on the first mobile agent standard MASIF (Mobile Agent System Interoperability Facility), an interoperability standard that allows agents from different mobile agent platforms to interact with each other. Researchers could use other mobile-agent based platforms such as *Aglets* that allow for similar functionality. The term agent or software agent is usually deciphered well in the artificial

intelligence community, where it stands for a program that can behave autonomously to perform a multitude of dynamic tasks based on the logistics that have been programmed into it by a user.

4.1 Agent classification
Based on the mobility of agents, they can be classified into three main types:

Static Agents: The first is the concept of static agents. Static agents are fragments of code that do not move to different locations, and stay at a constant position throughout its life cycle.

Semi-Mobile Agents: Semi-mobile agents, as the name suggests, have some mobility. They are in fact an inherent type of mobile agents, which are created at one logical or physical location, but are moved to another location for its functional life cycle.

Mobile Agents: Mobile agents are a fragment of code, which can move around, from host to host during its life cycle depending on the runtime task allocated to it. Mobile agents are based on a terminology, well known in literature as *mobile code* (Fuggetta et al., 1998). The term mobile code can be defined as the capability to change the binding between the pieces of code, and the location where they are executed.

The scope of the advantages or disadvantages of using any of the above mentioned agent types can vary based on the functionality of the agent based system that is being deployed. If latency is a big issue in the system, one should opt for static and/or semi-mobile agents. This is because the greater the mobility of an agent, the higher the latency introduced into the system caused by the time required to create it at a new location and to transfer the runtime state of the agent. If the host where the agent runs is very fragile or more prone to destruction or tampering, it would be best to use a mobile agent rather than a static agent, as it is easier for mobile agents to find a new location to run at than static agents.

4.2 Advantages and disadvantages of agents
The use of mobile agents offers wide advantages especially in distributed systems that cannot be overlooked. Some of the advantages offered by agents have been clearly listed in (Bieszczad et al., 1998). The major categories of these are summarized as follows:

Reduction in Network Traffic: In case of mobile agents, the agents themselves move to data. i.e. we move the agent code to the data rather than moving the data to the agent code. This allows for a dramatic reduction in the amount of bandwidth consumed in the log correlation process (explained in later sections) as data is almost always larger than the few kilobyte size of agents in general.

Asynchronous autonomous interaction: This is vital in a network where network connections are volatile, such as wireless networks. In such cases, the agent could migrate to a mobile device to gather data. Even if the connection breaks, the agent could continue processing data on the mobile device and report back whenever the connection is reestablished. This adds to the agent's capability to work in a fault tolerant mode.

Software Upgrades: Usually in order to update software on multiple hosts, an administrator has to first stop the server functionality, then uninstall the old version of the software, and then reinstall the new version. The entire software system has to be stopped for upgrades. The advantage of mobile agents or agents in general in this situation is that if each component of the upgraded software is managed by an agent, then it is as easy as disabling the old agent and deploying a new agent which has the required functionality. In this way one could avoid bringing down the entire system and instead stop just a single agent-based component.

Functionality in heterogeneous environments: Most agents today can work in heterogeneous environments. This is due to the fact that these agents are usually written in a language which is portable to multiple platforms, such as java or perl. Since agents sit on top of an agent framework, they can easily function regardless of if the host runs a version of Linux or Windows operating system. The significant reduction in costs of placing agent frameworks in hosts over the past few years have added to the benefits of running agents.

Just like there are advantages to using agents, there are also drawbacks to using agents. The applicability of advantages or disadvantages to using agents is based immensely on the specific user needs or goals that have been put forward. The shortcomings of using a mobile agent-based system have been clearly summarized by (Vigna, 2004).

Agent Security: The one and only reason that has hindered the wide usage of mobile agents in the real world has been its security constraints. One of the key problems associated with mobile agent security is the *malicious host* problem i.e. how much trust can be placed on a host where the agent travels to, given that the agent may have valuable data. Other security concerns that have been mentioned in literature include the concept of *malicious agents* (Vigna et al., 2002) where given the availability of an agent platform in a host, how much trust can be placed on the agent that travels to the host to gather information? This problem has been solved in the agent-security field by allowing a host to run only certain digitally signed agents. Last but not the least, agents can be *tampered* with which means, a legitimate agent could be *brainwashed* while traveling from host to host. (Vigna et al., 2002) has provided a means for auditing an agents trail to detect attacks that modify agents legitimate access permissions and authorization mechanisms for the aglets mobile agent platform.

Lack of Shared Language: Even though many tasks have been overtaken by FIPA (The Foundation for Intelligent Physical Agents) to create a standard ACL (Agent communication language), most agent platforms do not adhere to this language. Hence it is hard for agents to communicate with each other when they are based on different platforms.

Required Agent Platform: Any piece of agent code available today needs to run on an agent platform that contributes to the control and deployment of agents. For example, our system has to use the Grasshopper agent platform to execute its tasks currently. Similarly, to run java applets, the system has to have a java runtime environment available. The dependence of mobile agents on an agent platform is an extra requirement that has to be made, without which they cannot function. The problem is further compounded by the fact that not all agent platforms follow a given set of rules and procedures thus hindering interoperability issues even with the existence of standards such as MASIF.

4.3 Intrusion detection system data correlation

Most detection systems today use the process of *log correlation*, which is a process that takes the alerts generated by multiple intrusion detection systems and produce a brief report on the network being protected (Valeur et al., 2004). The advantage of this method is that if there are multiple intrusion detector sensors deployed in the network, on the occurrence of an intrusion attack, each of these sensors would generate a report on the intrusion type. Allowing log correlation of the information generated by all these sensors would provide a system administrator with a compact but detailed report on the attack allowing him or her to pinpoint the vulnerability easily.

In the conventional log correlation model, distributed sensors, after gathering the data, send all the alerts to a centralized location for correlation purposes. But the major disadvantage of this model is that if the amount of logs generated is large, it would clog the network system

in a low-bandwidth network. Also a centralized approach would overload a node that receives too many correlation tasks at a given time, causing system overload and hence delay in producing the analyzed results.

4.4 Agent based security systems

The earliest relevant work in this area was started by Purdue University's CERIAS (The Center for Education and Research in Information Assurance and Security) group in 1995 when they put forward a proposal for building an autonomous agent based security model by using genetic programming (Crosbie & Spafford, 1995). This was followed up by their work in implementing the earlier proposal (Balasubramaniyan et al., 1998).This system was called AAFID (Autonomous Agents for Intrusion Detection) written earlier in Perl, Tcl/Tk and C, and later revised and written in the perl language to make it more portable. (Helmer et al., 1998) used an anomaly detection technique by using the *Ripper* algorithm on *sendmail* system calls. The architecture mimicked a portion of the Java Agents for Meta-Learning (JAM) project (Stolfo et al., 1997). A distributed hierarchical IDS was proposed by (Mell & McLarnon, 1999) that tries to randomize the location of agents and decentralizing directory services. The system also resurrects agents killed by an intruder as there always exists multiple copies that track the original agent and vice versa. The *Micael* IDS was proposed by (Queiroz et al., 1999). They proposed an additional feature of periodically checking if all agents are active in the system. Another prominent work that detects intrusions using mobile agents is the IDA system (Asaka et al., 1999). This system tries to backtrack intrusion attempts by looking into MLSI (Mark Left by Suspected Intruders) left at each host. They also emphasize tracking the steps that an attacker takes.

The *Sparta* system by (Krügel et al., 2001; Krügel & Toth, 2002) is the most extensive work done till date on using mobile agents and intrusion detection. Sparta, which stands for Security Policy Adaptation Reinforced Through Agents, is an architecture that is capable of monitoring a network to detect intrusions and security policy violations by providing a query like functionality to reconstruct patterns of events across multiple hosts. This is a network-based IDS that *correlates* data from multiple sensors located throughout the network. The authors have created an EQL (Event Query Language) with syntax similar to SQL (Sequence Query Language) used in databases.

Other mobile agent based IDS's include a P2P based IDS (Ramachandran & Hart, 2004) that works in a *neighborhood watch* manner where each agent looks after other agents in its vicinity by using a voting procedure to take action against a compromised agent; the MA-IDS system (Li et al., 2004) which uses encrypted communication between the mobile agents in the system, and use a threshold mechanism to detect the probability for each intrusion depending on the quantity of each intrusion type obtained allowing it to learn in a one dimensional method. Some other mobile agent based IDS's include a position paper (Aslam et al., 2001) that claims to work on D'Agents environment; and work by (Foukia et al., 2001; Foukia et al., 2003) which uses a social insect metaphor and immune systems to model an intrusion detection system.

5. Agent-based botnet detection

Based on our previous experience on mobile agent based intrusion detection systems (Alam et al., 2005; Alam & Vuong, 2007), and an in-depth understanding of the behavior of botnets, we believe the appropriate approach to defend against botnets is to adapt a *mobile-agent*

based paradigm in combination with current host monitoring techniques, to detect bot infected hosts based on bot *behavior analysis*.

Our proposed approach predominantly would work for a local or remote environment with a single administrative entity with access to network level data of monitored hosts and optionally, access to the host machine via mobile-agent middleware if host-based bot behavior features are need. We use network and host behavior monitoring of hosts to detect bot infections based on calculating feature vectors stored as a bot DNA.

As mentioned in previous sections, each botnet exhibits certain traits/features. For example, the *storm* botnet used technique of fast-flux; used a peer-to-peer modeling based on edonkey (kademlia) protocol; used rootkits; certain variants exhibited detection of virtual machines; and infected bots either are used to DDoS, send phishing emails, or advertisements. We believe that each of these attributes could be considered a feature of the botnet. Some of the features describe the communication methods of a botnet while others describe their activity (Gu et al., 2008). Some of these can be detected using network monitoring while others by monitoring host changes. Each variant of a bot over time modifies some of its functionality (features), but the change is subtle from variant to variant in most cases. There are certain exceptions, such as *Conficker.C* which retains only 15 percent of its code intact from *Conficker.B* (Porras et al., 2009) and extensively modifies both its network and host behavior.

With our primary goals to:

1.   detect hosts that exhibit botnet traits with a certain confidence level, and
2.   detect which bot an infected host behaves like,

we believe that our first goal can be captured by calculating an *infection score* based on *weighted botnet features* exhibited by a host. The weights could be calculated based on using machine learning methods such as support vector machines (SVM) on predominant bot families. Our second goal can be accomplished by first learning the behavior of botnet infected hosts by capturing host and network behavior of known bot infections. This is followed by converting the behavior to a set of features represented as a vector stored in synthetic DNA format, allowing application of clustering or hashing algorithms as discussed in section 3.4.

5.1 Mapping bot feature vector

In order to apply the botnet detection problem to DNA, we map labeled buckets representing the range of values exhibited per bot feature/attribute. The purpose of using marked bins is to emphasize the more bot-like feature a host exhibits.

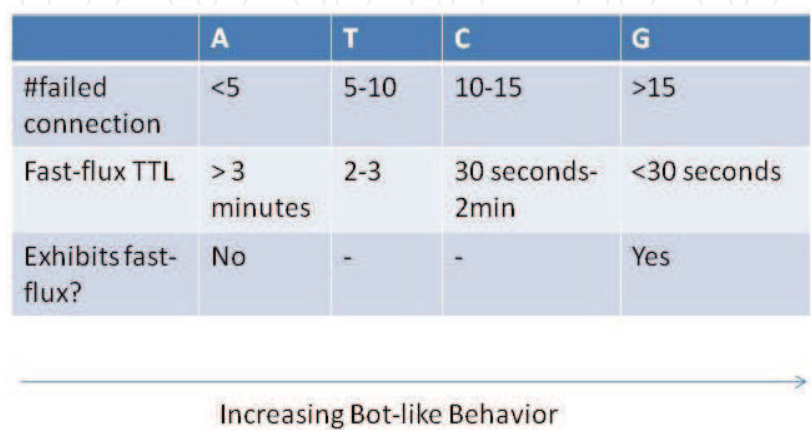|	A	T	C	G
#failed connection	<5	5-10	10-15	>15
Fast-flux TTL	> 3 minutes	2-3	30 seconds-2min	<30 seconds
Exhibits fast-flux?	No	-	-	Yes

Increasing Bot-like Behavior

Fig. 1. Assigning labels to botnet attributes

For example, if an infected node exhibits fast-flux, we only need two possible attribute states: *yes* or *no*. But measuring features such as the rate of outgoing connection, or average packet sizes cannot be captured by a *yes/no* solution. This can be solved to an extent by using a bucket/binning technique by partitioning such numbers into multiple marked bins. Whereas there exists a small number (4) of allowed variations in DNA nucleotides, it reduces the ability to measure accuracy in case of botnet detection. An example labeling is shown in Fig. 1. One also has to assign appropriate bin ranges that distinguish benign traffic from bot-like traffic. These are some of the challenges we are trying to solve based on measuring current bot features.

5.2 Sequencing of hosts

A system administrator would keep multiple DNA sequences of each host in its network: a set of sequences representing network-based DNA and a set representing host-based DNA. We partition the space into two since there might be cases when only one set of sequences are available. For example cellular devices using local wifi access cannot contribute to a host-based sequencing due to the absence of host-monitoring applications on the device in some cases. Fig. 2 shows an example DNA of a host as maintained by the administrator.

Two hosts that exhibit similar DNA sequences behave similarly. Thus if a host shows DNA sequencing similar to a bot DNA sequence with subtle mutations, we know the type of infection and can mark the infected host. Similarly if a host exhibits DNA sequences similar to innocuous DNA, we know it is clean.
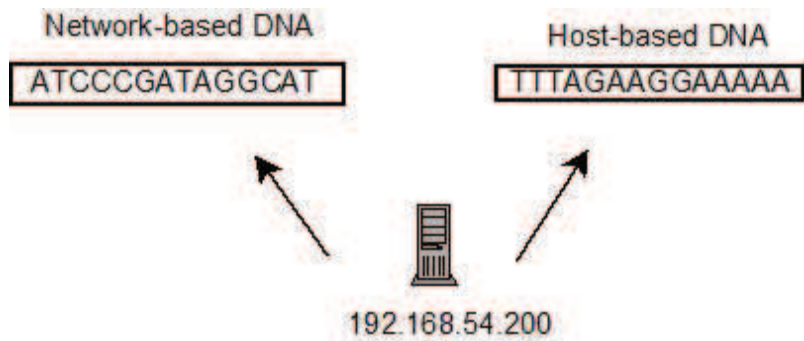

Fig. 2. An example of the DNA structure maintained per host basis

5.3 Capturing the attributes

The effectiveness of our solution is based on selecting the appropriate network or host attributes exhibited by botnets. These lists of attributes are based on the behavior depicted by various botnets over time as discussed in sections earlier. One or more of the attributes requires maintaining some sort of state information. Table 1 provides some of the higher layer network features that are currently tracked.

Fast-flux TTL, Rate of failed connections to ip, Rate of failed connection to dns, IRC, (Incoming http request, Outgoing http response, Outgoing http request, (Failed http response, Successful http response)), Incoming network scanning, (Outgoing network scanning, (Outgoing network scanning, Rate of Scanning)), (Retrieve Binary, (Binary MD5 match, Size of Binary)), (P2P traffic, Active connection rates, P2p active connections), Source IP spoofing, Outgoing SMTP traffic

Table 1. Network features tracked

5.4 Depth of attributes

As shown in Fig. 3, the depth at which a collected attribute resides, decides the length of the feature vector, and associated runtime and memory costs. The depth of the feature also decides if a bot can be appropriately distinguished or categorized under a known botnet. This is a trade-off that has to be kept into consideration. If the features vector comprises a sequence of system call API, this would cause a feature vector explosion (Bayer et al., 2009). In order to tackle this issue, some have abstracted system call objects (Bayer et al., 2009) , or created feature vector generated from system registry changes, file read/write and processes created for host-based attributes (Bailey et al., 2007; Rieck et al., 2008; Lee & Mody, 2006).

The attribute collection strategy regarding host-based behavior is based on the decisions of the researcher designing the agents. We envision having a multi-tiered strategy for host-based attributes where some attributes are collected at a higher layer than others depending on required time sensitivity.
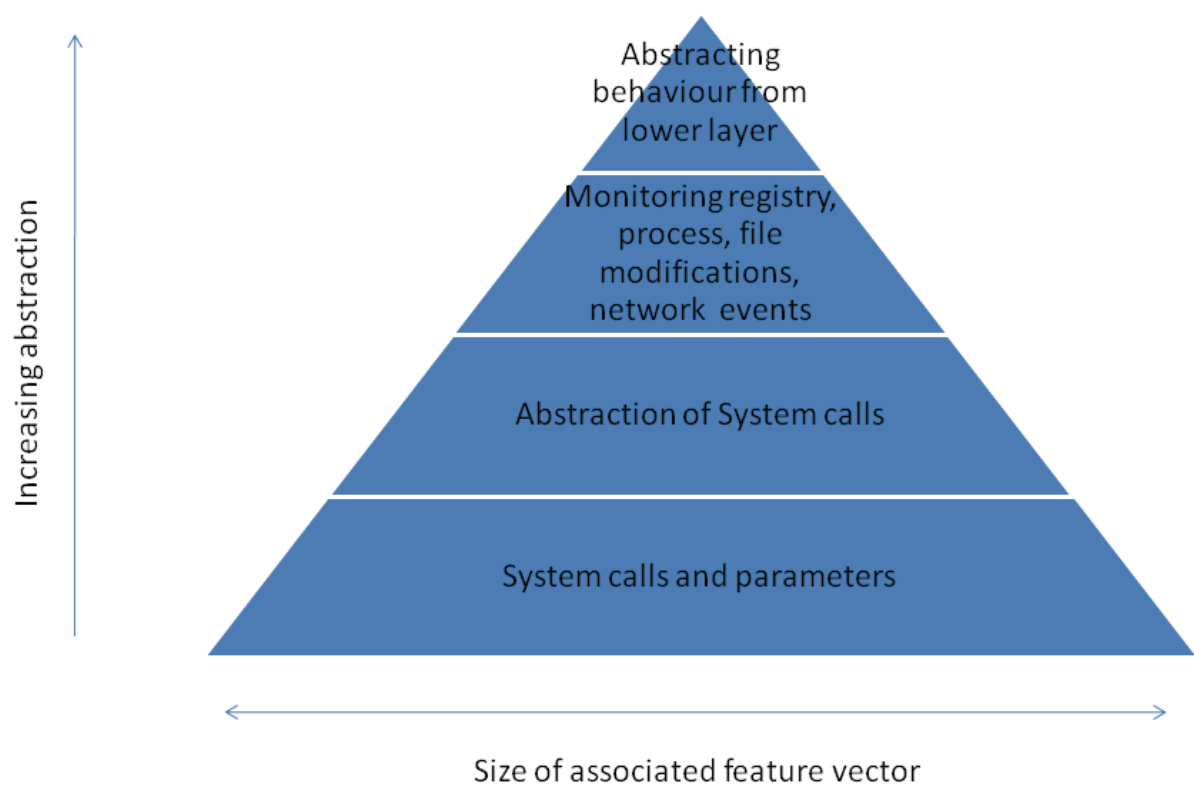

Fig. 3. Feature vector size vs. level of abstraction

5.5 Network attributes

A network-based DNA could be computed by capturing network packets by network-based packet filters such as *snort-inline* by monitoring all network connections between internal hosts and the external network. The primary advantage of *snort-inline* is that it allows active dropping of network packets if needed based on snort rules triggered.

We envision that one would need two sets of network attributes. Those that can be computed based on just packet header traces, and those that can be computed via deep packet inspection. The difficulty in capturing the later is the amount of encrypted channels used by botnets today. Due to the extensive obfuscation technology and encryption used, we have to take into account that some of the attributes that requires deep packet inspection will not be able to detect some bot traffic, and thus should be weighed differently.

5.6 Host attributes

Host-based attributes need to be captured using multiple methods. We could run host monitoring tools such as *Sebek* (Sebek, 2010) in a host that provide some of the host-based DNA, for example: list of dll and system files created, registry entries modifies, their modification dates, running processes, etc. This information and the analysis code that computes the host DNA could reside on the host. But there is an inherent problem in capturing host infections.

No data obtained using analysis tools already present on a host once it has been infected can be trusted. Moreover, the infected host could modify results sent by the infected host. For example most bots such as *storm, rxbot* and *rustock* use rootkits to modify results obtained using windows system API to hide monitoring of processes, network connections, file visibility and file sizes. Similarly *Conficker* (Porras et al., 2009) modifies in memory versions of windows system API leaving the actual dll file on disk untouched. This leads us to the case for using mobile agents.

5.7 The case for mobile agents

The main reason why using a mobile agent based approach is viable in host-based behavior detection is the fact that if our evidence gathering code is already available on an infected node, we cannot trust its result. Thus in order to analyze a host, our evidence gathering code has to travel to the host being analyzed. This could be the code which computes an MD5 hash of some important system files, or retrieve analysis data stored in a pseudo random file stored on the host in an encrypted format to hide from the infection code, or the code that detects the presence of rootkitted files similar to *Rootkit Revealer* or *Rootkit Unhooker*. Similarly, if more than one host exhibits similar malicious activities, or if multiple network sensors are deployed, mobile agents would allow processing of multiple hosts in a parallel manner, minimizing the time to detect infections. Mobile agents would allow us to replace outdated monitoring agents, with new agent code that has updated tracking abilities.

5.8 Protecting the agent and the infected host

An approach that can be taken to protect the infected host from malicious agent is the use of strong asymmetric cryptography. Some mobile agent platforms such as *Grasshopper* and *Aglets* allow only agent code signed and verified to run on a given host with access control policies. Using strong asymmetric key to sign the agent and its verification by the infected host environment would protect the host.

Similarly, once the agent performs its analysis task, the mobile agent would travel back to the analyzing host where it would be marked as tainted. The analyzing host could perform a check such as performing an MD5 hash on the agent to see that the agent code has not been modified before its results are processed.

Any agent travelling to an infected host also has to verify that the agent middleware has not been compromised in any way before starting its processing. The absence of an agent middleware that is supposed to exist could act as a sign of maliciousness. Using Aglets agent middleware has the added advantage of us being able to add functionality to the agent middleware as required since it is open-sourced.

5.9 Feature extraction (infection score)

Though we rely on measuring the various network and host-based attributes, not all attributes have equal weight in detecting botnet communication or activity. Moreover certain botnet families exhibit higher frequency of a certain attribute versus others. For example, certain attributes such as the use of fast-flux for communication or a user machine exhibiting SMTP traffic are symtoms of botnet behavior in case of botnets such as *Storm* and *Waledac*, but these features may not be utilized by IRC based bots such as *Agobot*. Thus, we see that not all attributes should be weighed equally for all botnets.

One approach would be to partition the features into multiple sets each assigned to a weight category, or each feature assigned an individual appropriate weight. This would constitute a part of feature extraction, where certain features are brought into focus while other probably noise given less emphasis. Whereas taking the first approach is easier, it is also prone to more inaccuracy. The second approach is more accurate but harder to compute.
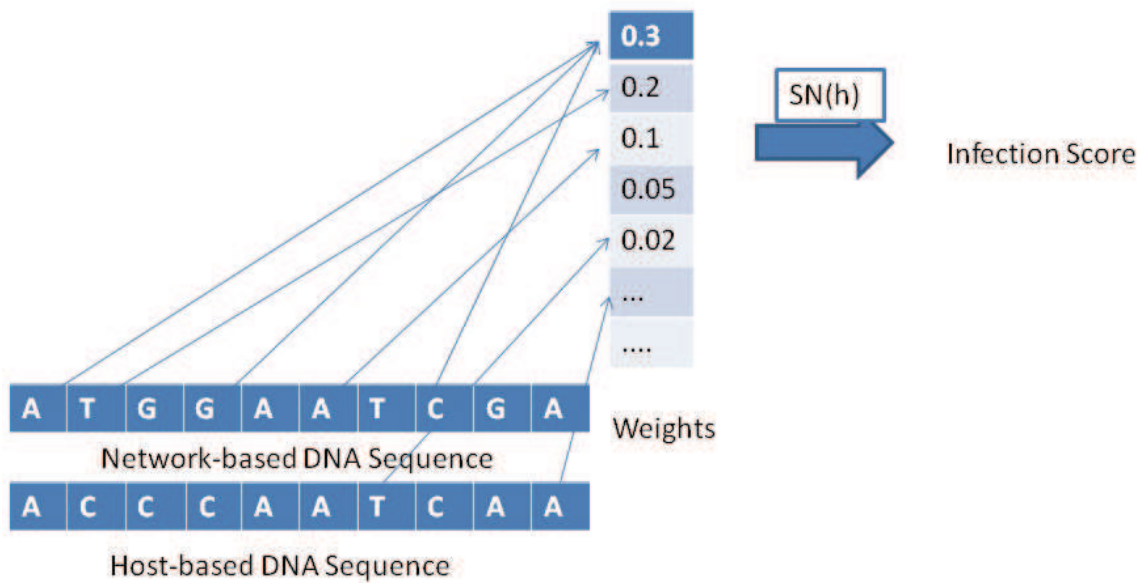

Fig. 4. Computing an infection score for a botnet infected host

The second approach could be accomplished by using Support Vector Machine (SVM), a supervised machine learning approach which is less prone to noise in the data sample. Creating an SVM for each bot family, and comparing the host DNA to each bot family SVM would allow us to measure which bot a host behaves like. The purpose of an SVM is to compute an optimal hyperplane that separates one class of n-dimensional points from another class (Rieck et al., 2008). Thus the main reason for using SVM in our case would be to compute actual weight assignments. This allows us to compute the infection score as shown in figure 4 where SN(h) is the computed infection score.

A host that exhibits a higher infection score, and if the infection score exceeds a threshold score set by an administrator, would automatically trigger a correlation requirement of host and network features exhibited and a further analysis.

Similarly, if the host or network DNA exhibits similar patterns to a known infection (based on distance measurement) after clustering bot behavior, it would also trigger a DNA-based correlation.

5.10 Scenario of use

In this section we describe the scenario of use of our approach.

A system administrator will receive continuous updating of the DNA sequencing of a given host and its probability of infection. The network-based DNA of a host will be updated based on the network traffic seen by the snort-inline processor. The host-based DNA will be reported periodically by individual hosts within the local network.

If the probability of the host being infected crosses a certain threshold based on the infection score, or a host approaches a DNA match close to a botnet, a bot correlation trigger flag will be raised.

Based on the infection model seen, a mobile agent would be created with the required host-detection functionality.

The agent would be deployed to the infected host, where it would perform analysis tasks as described in earlier sections. If the infected host denies a real agent to run, this could be a sign of maliciousness.

The agent could return with advanced-detailed results such as an encrypted list of rootkitted processes/files, or just the host-based DNA results. The agent is placed in the tainted bin, to verify the integrity of the agent, since it had travelled to a probable infected host. If the agent has retained its integrity its results are measured to be valid.

Based on both the host and network-based results, the node could block all incoming/outgoing network traffic by automatically modifying snort-inline for the given host. It could also provide details such as if the infection matches a known botnet, or is a new botnet pattern. It would also allow us to correlate hosts that have exhibited similar bot behavior pattern.

6. Conclusion

In this chapter, we have primarily focused on the various mechanisms utilized by botnet owners to maintain and protect their botnets; and the defensive mechanisms designed by researchers to study, detect and dismantle the malicious botnets. As can be understood, botnets utilize multiple advanced technologies that are constantly updated. Hence we have proposed the use of mobile agents which too can be constantly updated to defend against the ever changing behavior of bot binaries.

7. References

Alam, M.; Gupta, A.; Wires, J. & Vuong, S. (2005). APHIDS++: Evolution of a Programmable Hybrid Intrusion Detection System, *Proceedings of Mobility Aware Technologies and Applications*, pp. 22-31, Volume 3744/2005, Montreal, January, 2007, SpringerLink.

Alam, M. & Vuong, S. (2007). APHIDS++ : A Mobile Agent Based Intrusion Detection System, *Proceedings of 2nd International conference on Communication Systems Software and Middleware*, pp. 1-6, ISBN 1-4244-0613-7, Bangalore, January, 2007.

Asaka, M.; Taquchi, A. & Goto, S. (1999). The Implementation of IDA: An Intrusion Detection Agent System. *Proceedings of the 11th conference on Forum of Incident Response and Security Teams*. Brisbane Australia, July, 2007.

Aslam, J.; Cremonini, M.; Kotz, D. & Rus, D. (2001). Using Mobile Agents for Analyzing Intrusion in Computer Networks. *Proceedings of the Workshop on Mobile Object Systems at ECOOP 2001*. Budapest, Hungary, June, 2001.

Bacher, P. ; Holz, T. ; Kotter, M. & Wicherski, G. (2005). Know your Enemy : Tracking Botnets using Honeynets to learn more about bots. Honeynet Project, [online] Available at: http://www.honeynet.org/papers/bots, [Accessed 10 September 2010].

Baecher, P; Koetter, M;Dornseif, M. & Freiling, F. (2006). The nepenthes platform: An efficient approach to collect malware. *Proceedings of the 9th International Symposium on Recent Advances in Intrusion Detection,* pp. 165-184, Springer Link, Hamburg, Germany, September, 2006.

Bailey, M.; Oberheide, J.; Andersen, J.; Mao, Z. M.; Jahanian, F. & Nazario, J. (2007). Automated classification and analysis of internet malware. *Proceedings of the 10th international Conference on Recent Advances in intrusion Detection* (Gold Goast, Australia, September 05 - 07, 2007, pp 178-197 C. Kruegel, R. Lippmann, and A. Clark, Eds. Lecture Notes In Computer Science. Springer-Verlag, Berlin, Heidelberg.

Balasubramaniyan, J. S., Garcia-Fernandez, J. O., Isacoff, D., Spafford, E. & Zamboni, D. (1998). An Architecture for Intrusion Detection Using Autonomous Agents, *Proceedings of the 14th Annual Computer Security Applications Conference* (ACSAC), ISBN 8186-8789-4, IEEE Computer Society, Washington, DC, December, 1998.

Bambenek, J. & Klus, A. (2008). Botnets and proactive system defense. Botnet Detection: Countering the Largest Security Threat, edited by Wenke Lee, Cliff Wang, and David Dagon, ISBN 978-0-387-68766-7, Springer-Verlag, 2008.

Barford, P. & Blodgett, M. (2007). Toward botnet mesocosms. *Proceedings of the First Conference on First Workshop on Hot Topics in Understanding Botnets,* Cambridge, MA, April, 2007, USENIX Association, Berkeley, CA.

Bäumer, C. & Magedanz, T. (1999). The Grasshopper Mobile Agent Platform Enabling Shortterm Active Broadband Intelligent Network Implementation. *Proceedings of the First international Working Conference on Active Networks* S. Covaci, pp 109-116, Ed. Lecture Notes In Computer Science, vol. 1653. Springer-Verlag, London.

Bayer, U; Comparetti, P; Hlauschek, C; Kruegel, C & Kirda, E. (2009). Scalable Behavior-based Malware Custering. *Proceedings of the 16th Annual Network and Distributed System Security Symposium,* ISOC, SanDiego, CA, February, 2009.

Bieszczad, A; White, T & Pagurek, B. (1998). Mobile Agents for Network Management, *Proceedings of IEEE Communications Surveys*, September 1998.

Brodsky, A. & Brodsky, D. (2007). A distributed content independent method for spam detection. *Proceedings of the First Conference on First Workshop on Hot Topics in Understanding Botnets,* Cambridge, MA, April, 2007, USENIX Association, Berkeley, CA.

Brumley, D.; Caballero, J.; Liang, Z.; Newsome, J. & Song, D. (2007). Towards automatic discovery of deviations in binary implementations with applications to error detection and fingerprint generation, *Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium*, pp. 1-16, Boston, MA, N. Provos, Ed. USENIX Association, Berkeley, CA.

Brumley, D.; Poosankam, P.; Song, D. & Zheng, J. (2008). Automatic Patch-Based Exploit Generation is Possible: Techniques and Implications. *Proceedings of the 2008 IEEE Symposium on Security and Privacy*, pp. 143-157, SP. IEEE Computer Society, Washington, DC, May, 2008.

Chiang, K. & Lloyd, L. (2007). A case study of the rustock rootkit and spam bot. *Proceedings of the First Conference on First Workshop on Hot Topics in Understanding Botnets*, Cambridge, MA, April, 2007, USENIX Association, Berkeley, CA

Cooke, E.; Jahanian, F., & McPherson, D. (2005). The Zombie roundup: understanding, detecting, and disrupting botnets. *Proceedings of the Steps To Reducing Unwanted Traffic on the internet on Steps To Reducing Unwanted Traffic on the internet Workshop*, Cambridge, MA, July, 2005, USENIX Association, Berkeley, CA.

Crosbie, M. & Sapfford, G. (1995). Defending a Computer System using Autonomous Agents. *Proceedings of the 8th National Information Systems Security Conference*.

Dagon, D ;Zou, C & Lee, W. (2006). Modelling Botnet Propagation using time zones. *Proceedings of The 13th Annual Network and Distributed System Security Symposium (NDSS 2006)*, San Diego, CA, February 2006, ISOC.

Daswani, N. & Stoppelman, M. (2007). The anatomy of Clickbot.A. *Proceedings of the First Conference on First Workshop on Hot Topics in Understanding Botnets*, Cambridge, MA, April 2007, USENIX Association, Berkeley, CA.

Foukia, N.; Billard, D. & Harms, J. (2001). Computer System Immunity using Mobile Agents. *Proceedings of HP Openview University Association 8th Annual Workshop*. 2001.

Foukia, N.; Hassan, S.; Fenet, S. & Albequerque, P. (2003). Combining Immune System and Social Insect Metaphors: A Paradimg for Intrusion detection and response system. *Proceedings of the 5th International Workshop for Mobile Agents for Telecommunication Applications*, Marakech, Morocco, October, 2003, Lecture Notes in Computer Science, Springer.

Franklin, J.; Luk, M.; McCune, J.M.; Seshadri, A.; Perrig, A., & van Doorn, L. (2008). Towards Sound Detection of Virtual Machines, *Botnet Detection: Countering the Largest Security Threat*, edited by by Wenke Lee, Cliff Wang, and David Dagon, ISBN 978-0-387-68766-7, Springer-Verlag, 2008.

Fuggetta, A.; Picco, G.& Vigna, G. (1998). Understanding Code Mobility, *Proceedings of IEEE Transactions on Software Engineering*, pp. 342-361, May, 1998

Gao, D.; Reiter, M. & Song, D. (2005). Behavioral Distance for Intrusion Detection. *Proceedings of the 8th International Symposium on Recent Advances in Intrusion Detection*, pp. 63-81, Seattle, Washington, September 2005, Springer.

Garfinkel, T.; Adams, K.; Warfield, A.; & Franklin, J. (2007). Compatibility is not transparency: VMM detection myths and realities. *Proceedings of the 11th USENIX Workshop on Hot Topics in Operating Systems*, pp. 1-6, San Diego, CA, May, 2007, G. Hunt, Ed. USENIX Association, Berkeley, CA.

Grizzard, J. B.; Sharma, V.; Nunnery, C.; Kang, B. B. & Dagon, D. (2007). Peer-to-peer botnets: overview and case study. *Proceedings of the First Conference on First*

Workshop on Hot Topics in Understanding Botnets, Cambridge, MA, USENIX Association, Berkeley, CA.

Gu, G.; Porras, P.; Yegneswaran, V.; Fong, M. & Lee, W. (2007). BotHunter: detecting malware infection through IDS-driven dialog correlation. *Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium*, pp. 1-16, Boston, MA, August 2007, N. Provos, Ed. USENIX Association, Berkeley, CA.

Gu, G.; Zhang, J.; Lee, W. (2008). BotSniffer: Detecting Botnet Command and Control Channels in Network Traffic. *Proceedings of The 15th Annual Network and Distributed System Security Symposium (NDSS 2008)*, San Diego, CA, February 2008, ISOC.

Gu, G.; Perdisci, R.; Zhang, J. & Lee, W. (2008). BotMiner: clustering analysis of network traffic for protocol- and structure-independent botnet detection. *Proceedings of the 17th Conference on Security Symposium* , pp. 139-154, San Jose, CA, July 28 - August 01, 2008, USENIX Association, Berkeley, CA.

Helmer, G.; Wong, J.S.K.; Honavar, V. & Miller, L. (1998). Intelligent agents for intrusion detection. *Proceedings of IEEE Information Technology Conference*, pp. 121-124, Syracuse, NY, USA, September 1998.

Holz, T.; Steiner, M.; Dahl, F.; Biersack, E. & Freiling, F. (2008). Measurements and mitigation of peer-to-peer-based botnets: a case study on storm worm. *Proceedings of the 1st Usenix Workshop on Large-Scale Exploits and Emergent Threats*, pp. 1-9, San Francisco, California, April 2008, F. Monrose, Ed. USENIX Association, Berkeley, CA.

Holz, T.; Engelberth, M. & Freiling, F. (2009). Learning more about the underground economy: a case-study of keyloggers and dropzones. *Proceedings of the 14th European Conference on Research in Computer Security,* pp. 1-18, Saint-Malo, France, September 21 - 23, 2009, M. Backes and P. Ning, Eds. Lecture Notes In Computer Science. Springer-Verlag, Berlin, Heidelberg.

Hund, R.; Hamann, M. & Holz, T. (2008). Towards Next-Generation Botnets. *Proceedings of the 2008 European Conference on Computer Network Defense* , pp. 33-40, Dublin, Ireland, December 11 - 12, 2008, EC2ND, IEEE Computer Society, Washington, DC.

Hex-rays. (2010). IDA Pro. [Online] Available at: http://www.hex-rays.com/idapro/

Ikinci, A.; Holz, T. & Freiling, F. (2008). Monkey-Spider: Detecting Malicious Websites with Low-Interaction Honeyclients. *Proceedings of Sicherheit, Schutz und Zuverlssigkeit*, pp. 407-421, April 2008, Germany.

Jansen, W & Karigiannis, T. (1999). Mobile Agent Security. *NIST Special Publications*, Computer Security Resource Center. [Online] Available at: http://csrc.nist.gov/publications/nistpubs/800-19/sp800-19.pdf [Accessed 12 September 2010]

Karasaridis, A.; Rexroad, B. & Hoeflin, D. (2007). Wide-scale botnet detection and characterization. *Proceedings of the First Conference on First Workshop on Hot Topics in Understanding Botnets* , Cambridge, MA, April 2007, USENIX Association, Berkeley, CA.

Lee, T. & Mody, J. (2006). Behavioral classification. *Proceedings of European Expert Group For IT-Security*, EICAR 2006, Hamburg, Germany, May 2006.

Li, Z.; Goyal, A. & Chen, Y. Honeynet-based Botnet Scan Traffic Analysis. (2008). *Botnet Detection: Countering the Largest Security Threat*, pp. 25-44, ISBN 978-0-387-68766-7, edited by Wenke Lee, Cliff Wang, and David Dagon, Springer-Verlag.

Krügel, C.; Toth, T. & Kirda, E. (2001). SPARTA, a Mobile Agent Based Instrusion Detection System. *Proceedings of the IFIP Tc11 Wg11.4 First Annual Working Conference on Network Security: Advances in Network and Distributed Systems Security*, pp. 187-200, Belgium, November 26 - 27, 2001, B. D. Decker, F. Piessens, J. Smits, and E. V. Herreweghen, Eds. IFIP Conference Proceedings, vol. 206. Kluwer B.V., Deventer, The Netherlands.

Krügel, C.& Toth, T;. (2002).Flexible, Mobile Agent based Intrusion Detection for Dynamic Network. *Proceedings of the European Wireless*. February 2002.

Li, C.; Song, Q.; Zhang, C. (2004). MA-IDS Architecture for Distributed Intrusion Detection using Mobile Agents. *Proceedings of the 2nd International Conference on Information Technology for Application (ICITA 2004)*.

Li, Z.; Sanghi, M.; Chen, Y.; Kao, M. & Chavez, B. (2006). Hamsa: Fast Signature Generation for Zero-day PolymorphicWorms with Provable Attack Resilience. *Proceedings of the 2006 IEEE Symposium on Security and Privacy* , pp. 32-47, Oakland, CA, May 21 - 24, 2006. SP. IEEE Computer Society, Washington, DC.

Liu, X.; Yang, X. & Lu, Y. (2008). To filter or to authorize: network-layer DoS defense against multimillion-node botnets. *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication*, pp. 195-206, Seattle, WA, USA, August 17 - 22, 2008. SIGCOMM '08. ACM, New York, NY.

Mell, P. & McLarnon, M. (1999). Mobile Agent Attack Resistant Distributed Hierarchical Intrusion Detection Systems. *Proceedings of the Second International Workshop on Recent Advances in Intrusion Detection*. Purdue, IN, USA, September 1999, [Online] Available at: http://www.raid-symposium.org/raid99/

MITRE Honey Client Project. (2007). [Online] Available at: http://www.honeyclient.org/trac

Queiroz, J.; Carmo, L. & Pirmez, L. (1999). Micael: An autonomous mobile agent system to protect new generation networked applications. *Proceedings of the Second International Workshop on Recent Advances in Intrusion Detection*. Purdue, IN, USA, September 1999, [Online] Available at: http://www.raid-symposium.org/raid99/

Nazario, J. & Holz T. (2008). As the net churns : fast-flux Botnet Observations. *Proceedings of 3rd International Conference on Malicious and Unwanted Software*, Virginia, October 2008, IEEE.

Newsome, J.; Karp, B. & Song, D. (2005). Polygraph: automatically generating signatures for polymorphic worms, *In Proceedings of IEEE Symposium on Security and Privacy, 2005*, pp. 226- 241, Oakland, CA, 8-11 May 2005. IEEE.

Pierce, C. (2008). Owning Kraken Zombies, A detailed Dissection. [Online] Available at: http://dvlabs.tippingpoint.com/blog/2008/04/28/owning-kraken-zombies . Last accessed: 12th September, 2010.

Porras, P.; Saidi, H. & Yegneswaran, V. (2009). Conficker C Analysis. [Online] Available at: http://mtc.sri.com/Conficker/addendumC/ , Last Updated: 4 April, 2009.

Provos, N.; McNamee, D.; Mavrommatis, P.; Wang, K. & Modadugu, N. (2007). The ghost in the browser analysis of web-based malware. *Proceedings of the First Conference on First Workshop on Hot Topics in Understanding Botnets*, Cambridge, MA, April 2007, USENIX Association, Berkeley, CA.

Provos, N. (2007). HoneyD. [Online] Available at: http://www.honeyd.org. Last accessed: 7th September 2010.

Provos, N.; Mavrommatis, P., Rajab, M. A., & Monrose, F. (2008). All your iFRAMEs point to Us. *Proceedings of the 17th Conference on Security Symposium*, pp. 1-15, San Jose, CA, July 28 - August 01, 2008. USENIX Association, Berkeley, CA.

Rajab, M.; Zarfoss, J.; Monrose, F. & Terzis, A. (2006). A multifaceted approach to understanding the botnet phenomenon. *Proceedings of the 6th ACM SIGCOMM Conference on internet Measurement*, pp. 41-52, Rio de Janeriro, Brazil, October 25 - 27, 2006, ACM, New York, NY.

Ramachandran, A.; Feamster, N. & Dagon, D. (2006). Revealing botnet membership using DNSBL counter-intelligence. *Proceedings of the 2nd Conference on Steps To Reducing Unwanted Traffic on the internet - Volume 2*, pp. 49-54, San Jose, CA, July, 2006, USENIX Association, Berkeley, CA.

Ramachandran, G. & Hart, D. (2004). A P2P intrusion detection system based on mobile agents. *Proceedings of the 42nd Annual Southeast Regional Conference*, pp. 185-190, Huntsville, Alabama, April 02 - 03, 2004. ACM-SE 42. ACM, New York, NY.

Riden, J. & Seifert, C. (2008). A guide to different kinds of honeypots. [Online] Available at: http://www.symantec.com/connect/articles/guide-different-kinds-honeypots, February 2008, Symantec.

Rieck, K.; Holz, T.; Willems, C.; Düssel, P. & Laskov, P. (2008). Learning and Classification of Malware Behavior. *Proceedings of the 5th international Conference on Detection of intrusions and Malware, and Vulnerability Assessment*, pp. 108-125, Paris, France, July 10 - 11, 2008, D. Zamboni, Ed. Lecture Notes In Computer Science, vol. 5137. Springer-Verlag, Berlin, Heidelberg.

Rocaspana, J. (2007). Shelia: a client-side honeypot for attach detection. [Online] Available at: http://www.cs.vu.nl/~herbertb/misc/shelia/. Last accessed: 10 September, 2010.

Roesch, M. (1999). Snort – lightweight intrusion detection system for networks. *Proceedings of USENIX LISA'99*. Seattle, Washington, November 1999.

Roo. (2005). Honeywall. The Honeynet Project. [Online] Available at: https://projects.honeynet.org/honeywall/

Sebek. (2010). Sebek. The Honeynet Project. [Online] Available at: https://projects.honeynet.org/sebek/

Schiller, C.; Binkley J.; Evron G.; Willems, C; Bradley, T.; Harley D.; Cross M. (2007). *Botnets : The Killer Web Application*, ISBN 1597491357, Syngress.

Siefert, C; Steenson, R.; Holz, T.; Davis, M. (2007). Know Your Enemy: Behind the scenes of Malicious WebServers. Honeynet.org. Nov 2007, [Online]: Available at: http://www.honeynet.org/papers/mws. Last Accessed: 12 September, 2010.

Siefert. C. (2008). Capture-HPC Client Honeypot. [Online] Available at: https://projects.honeynet.org/capture-hpc . Last Accessed: 12 September, 2010.

Simon, D. R.; Agarwal, S. & Maltz, D. A. (2007). AS-based accountability as a cost-effective DDoS defense. *Proceedings of the First Conference on First Workshop on Hot Topics in Understanding Botnets*, Cambridge, MA, April 2007, USENIX Association, Berkeley, CA.

Singh, K.; Srivastava, A.; Giffin, J.& Lee, W. (2008). Evaluating Email's Feasibility for Botnet Command and Control. *Proceedings of the 38th IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, Anchorage, Alaska, USA, June, 2008, IEEE.

Snort-inline. (2005). [Online] Available at: http://snort-inline.sourceforge.net/

Steward, J. (2009). Spam Botnets to Watch 2009. [Online] Available at: http://www.secureworks.com/research/threats/botnets2009/. Secureworks, January, 2009.

Stinson, E. & Mitchell, J. C. (2008). Towards systematic evaluation of the evadability of bot/botnet detection methods. *Proceedings of the 2nd Conference on USENIX Workshop on offensive Technologies*, pp. 1-9, San Jose, CA, USENIX Association, Berkeley, CA.

Stolfo, S.; Prodromidis, A.; Tselepis, S.; Lee, W.; Fan, D. & Chan P. (1997). JAM: Java Agents for Meta-Learning over Distributed Databases. Proceedings of the third international conference on Knowledge Discovery and Data mining, pp. 74-81, Newport Beach, CA, USA, August, 1997, ISBN 1-57735-027-8, AAAI Press.

Strayer, W.; Lapsely, D; Walsh, R & Livadas, C. (2008). Botnet Detection Based on Network Behavior. *Botnet Detection: Countering the Largest Security Threat*, pp. 1-24, edited by Wenke Lee, Cliff Wang, and David Dagon, ISBN 978-0-387-68766-7, Springer-Verlag, 2008.

Sunbelt. (2010). CWSandbox. [Online] Available at: http://www.sunbeltsoftware.com/Malware-Research-Analysis-Tools/Sunbelt-CWSandbox.

Tucek, J.; Newsome, J.; Lu, S.; Huang, C.; Xanthos, S.; Brumley, D.; Zhou, Y. & Song, D. (2007). Sweeper: a lightweight end-to-end system for defending against fast worms. *Proceedings of Special Interest Group on Operating Systems Operating Systems Review,* pp. 115-128, Volume 41, No. 3, June 2007, ACM, Newyork, NY, USA.

Unrealircd. (2010). [Online] Available at: http://www.unrealircd.com/. Last accessed: 12 September, 2010.

Valeur, F.; Vigna, G.; Kruegel, C. & Kemmerer, R. A. (2004). A Comprehensive Approach to Intrusion Detection Alert Correlation. *Proceedings of IEEE Transactions on Dependable and Secure Computing,* pp. 146-169, Volume 1, No. 3, July 2004, IEEE.

Vigna, G. (2004). Mobile agents: ten reasons for failure. *Proceedings of the IEEE International Conference on Mobile Data Management (MDM '04),* pp. 298-299, Berkeley, CA. IEEE.

Vigna, G.; Cassell, B. & Fayram, D. (2002). An Intrusion Detection System for Aglets. *Proceedings of the 6th international Conference on Mobile Agents*, pp. 64-77, Barcelona, Spain, October 22 - 25, 2002, N. Suri, Ed. Lecture Notes In Computer Science, vol. 2535, ISBN 3-540-00085-2, Springer-Verlag, London.

Vogt, R.; Aycock, J. & Jacobson Jr, M. (2007). Army of Botnets. *Proceedings of the Network and Distributed System Security Symposium, NDSS 2007*, San Diego, California, USA, 28th February - 2nd March 2007, ISOC.

Wang, P.; Sparks, S. & Zou, C. (2007). An advanced hybrid peer-to-peer botnet. *Proceedings of the First Conference on First Workshop on Hot Topics in Understanding Botnets*, Cambridge, MA, April 2007, USENIX Association, Berkeley, CA.

Wang, Y.; Beck, D; Jiang, X.; Roussev, R.; Verbowski, C.; Chen, X. & King, S. (2006). Automated Web Patrol with Strider HoneyMonkeys: Finding web sites that exploit Browser vulnerabilities. *Proceedings of the Network and Distributed System Security Symposium, NDSS 2006*, San Diego, California, USA, February 2006, ISBN 1-891562-22-3, ISOC

Xie, Y.; Yu, F.; Achan, K.; Panigrahy, R.; Hulten, G., & Osipkov, I. (2008). Spamming botnets: signatures and characteristics. *Proceedings of SIGCOMM Computer*

Communication Review, pp. 171-182, Volume 38,No. 4, October 2008, ACM, Newy York, NY, USA.

Yin, H.; Song, D.; Egele, M.; Kruegel, C. & Kirda, E. (2007). Panorama: capturing system-wide information flow for malware detection and analysis. *Proceedings of the 14th ACM Conference on Computer and Communications Security*, pp. 116-127, Alexandria, Virginia, USA, October 28 - 31, 2007, ACM, New York, NY.
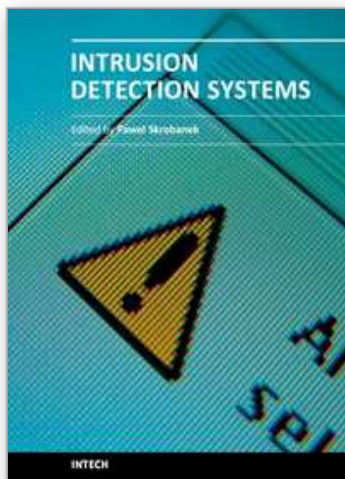
Intrusion Detection Systems

Edited by Dr. Pawel Skrobanek

ISBN 978-953-307-167-1

Hard cover, 324 pages

Publisher InTech

Published online 22, March, 2011

Published in print edition March, 2011

The current structure of the chapters reflects the key aspects discussed in the papers but the papers themselves contain more additional interesting information: examples of a practical application and results obtained for existing networks as well as results of experiments confirming efficacy of a synergistic analysis of anomaly detection and signature detection, and application of interesting solutions, such as an analysis of the anomalies of user behaviors and many others.

How to reference

In order to correctly reference this scholarly work, feel free to copy and paste the following:

Son T. Vuong and Mohammed S. Alam (2011). Advanced Methods for Botnet Intrusion Detection Systems, Intrusion Detection Systems, Dr. Pawel Skrobanek (Ed.), ISBN: 978-953-307-167-1, InTech, Available from: http://www.intechopen.com/books/intrusion-detection-systems/advanced-methods-for-botnet-intrusion-detection-systems

INTECH
open science | open minds