Botnet Tracking Tools

GIAC (GSEC) Gold Certification

Author: Pierce M Gibbs, pierce.m.gibbs@gmail.com
Advisor: Richard Carbone

Accepted: August 8th 2014

Abstract

Botnets are a serious threat to internet security. Botnets consist of networked collections of compromised machines called robots or 'bots' for short. Bots are also called 'zombies,' and botnets are also called 'zombie armies.' Bots are controlled by nodes called 'botmasters' or 'botherders.' Bots are infected with malicious code that performs work on behalf of the botmaster or botherder.

Botnets are used to conduct cyber warfare, perform massive identity theft, store and disseminate malware and pornography, execute massive spam campaigns, and implement distributed Denial of Service attacks.

Botnets today can provide the processing power of a supercomputer and perform a sustained Denial of Service attack powerful enough to take a country off line.

This paper will discuss botnet detection tools and techniques, organization and architectures, protocols, and lifecycle.

1. Introduction

Botnets are a serious threat to internet security. Botnets consist of networked collections of compromised machines called robots, or ‘bots’ for short. Bots are also called ‘zombies,’ and botnets are also called ‘zombie armies.’ Bots are controlled by nodes called ‘botmasters’ or ‘botherders.’ Bots are infected with malicious code that performs work on behalf of the botmaster or botherder. Typically, bots contact the botmaster for instructions, software updates, and to deliver status and exfiltrated data.

The bots are controlled remotely by a botmaster, botherder, or command and control (C&C) computer. Generally, the individual bots are not physically owned by the botmaster and are dispersed geographically across the world and across the entire IP address space (Feily and Shahrestani, 2009).

1.1. Size and Power of a Botnet

The power of a botnet is directly proportional to the size or number of infected hosts that are on line and running botnet code. However, the size of a botnet is a dynamic characteristic and can be difficult to determine. Bot machines that are powered off, suffering from a hardware or software failure, or otherwise unavailable for botnet duties might not be counted in the botnet size. Additionally, new machines can be infected, and infected machines can be cleaned further adding to the inexactness of calculating botnet size.

Empirical evidence indicates that botnets with millions of bots are not uncommon. The Conficker worm has been estimated to have infected between 9 and 15 million machines (Q. Wang, Z. Chen, C. Chen, & N. Pissinou, 2010). Chamelon, TLD4, Kelihos, and BredoLab botnets had roughly 120,000; 4.5 million; 300,000; and 30 million bots, respectively (Dev, J, 2013).

The power of a botnet is also often difficult to determine. With the increase of web sites developed by nontechnical personnel using vulnerable tools such as Word Press, intelligent attackers are going after higher end web servers rather than home PCs.

Smart attackers are also looking for vulnerable machines on higher-end corporate networks rather than the home computer.

The theoretical bandwidth provided by cable modem technology is 30Mbs, although the upstream bandwidth realized by a home user would likely be closer to 2Mbs. The processing power provided by an i5 processor core is approximately 100 GFLOPS (Product Export Compliance Matrix, 2012). A modest botnet of 200,000 home PCs can command 200 TFLOPs of processing power and 400 Gbps of bandwidth.

Even a small botnet consisting of only a thousand low-end home computers with a typical home broadband connection poses a great threat. Average upstream rates of 128Kbit/s can yield the botnet more than 100Mbits/s of bandwidth which is greater than the bandwidth possessed by most corporate internet connections (Paul Bacher, et al., 2008).

A larger botnet or a botnet composed of higher-end servers can provide the processing power of a supercomputer and perform a sustained denial of service attack powerful enough to take a country off line (Storm botnet, retrieved 2014).

1.2. Infection Techniques

The techniques botnets use to infect other machines and recruit new bots include worm-like replication, transmission via infected media, and a whole host of social engineering tactics.

Replication is where the bot scans potential target machines, looking for exploitable vulnerabilities. As is the case with worms, no human intervention is required. To evade intrusion detection systems, the more sophisticated botnets will scan a subnet using bots from a wide range of IP addresses.

Another infection technique is via infected media. Thumb drives and CD/DVDs are two common types of media used to distribute botcode. The botcode can be contained within a legitimate program and installed when the user installs the program. The botcode can also be installed, transparent to the user, via the *autorun* and *autoplay* features on machines running Windows.

Drive-by downloads are when a user visits a website and malware is unknowingly downloaded to the user machine by exploiting web browser vulnerabilities. The increasing number of plug-ins, add-ons and browsers has increased the web browser attack surface which contributes to the effectiveness of drive-by downloads.

A watering hole is a term used to describe a website frequented by a group. Often, there is a set of waterholes that the group frequents. Infecting one or more of the sites increases the likelihood that members of the group are infected. Waterholes can be an effective infection technique for group members that are resistant to phishing.

Social engineering techniques are often used to effect infection. Phishing and spam are two social engineering methods used to facilitate infection. Clever bot infection attempts tied spam to global events, enticing users to click on links in emails or on compromised websites. Offers of free music downloads and YouTube videos have also been used as enticement tactics.

1.3. Goals and Usages of a Botnet

The goals of an attacker in terms of bot recruitment can be categorized into two broad categories. In the first case, the target of the attack is the computer that is being recruited. In the second case, the recruited bot is used to attack another target. Computers in a botnet are either the target of the attack, the perpetrator of the attack, or both. When bots are the perpetrator of the attack, the effectiveness of the attack is usually dependent upon the collective power of the botnet.

Stated differently, the attacker is either after the valuable information on the computer or the storage, processing, and communication capabilities of the computer, or both.

The value of information on a personal computer is often overlooked. There are many applications for home use, like tax preparation software, that store volumes of key personal and financial information. Browsers can cache website and account information. Email clients store contact information. Many home computers have sensitive work information on them, exposing a company's intellectual property to potential disclosure. These are just a few examples of the type of information that can be exfiltrated and abused by an attacker or sold to cyber criminals.

A common botnet usage is the sending of spam emails. The spam emails can be used to carry malicious payloads in an attempt to infect the recipients of the spam emails. Spam can also be used to influence or manipulate user behavior such as the purchasing of advertised products, visiting of infected websites, or downloading of music or videos with malicious content.

Most cryptographic controls just buy the user time, in hopes that by the time the control is no longer good, the protected asset no longer has value. That is the theory behind password expirations. Password lifetime is set to be less than the time it takes to discover the password by brute force or other methods. The processing capabilities of a bot can be used for cryptanalysis purposes. Password cracking, brute force key discovery, and rainbow table creation are but a few examples. The collective power of a botnet greatly reduces the time a control is effective.

Data storage is another bot resource an attacker can use without permission. Anonymity is important to the attacker, so storing ill-gotten gain across the botnet keeps incriminating evidence away from the attackers' machines. Additionally, there are efficiency, redundancy, and availability benefits from having a distributed data store. Ill-gotten gains can include personal information seized, pornography, intellectual property, and malware.

Botnet rental has become a lucrative business. Botnets are rented and sold, usually for malicious purposes. Decentralized botnet architectures allow massive botnets to be partitioned into groups of smaller children botnets that can be parceled out for use and then reintegrated into the parent botnet after usage.

This paper will discuss botnet detection tools and techniques. This paper gives a brief introduction, a brief background on botnet characteristics, a summary of detection techniques, an overview of the BotMiner tool, reviews of two case studies where botnets were used to characterize behavior, and a conclusion.

2. Background

This section provides a discussion of botnet characteristics. Botnets can be categorized by three characteristics – botnet organization and architecture, communication protocol, and lifecycle.

2.1. Botnet Organization and Architecture

Early botnets were organized in a centralized or structured manner with each bot communicating directly, or via covert channel, with the command and control node. Botnets receive commands from the command and control node and send results back to the command and control node. Botnets that are more sophisticated create a hierarchy of command and control servers to make it harder to track down the main command and control server. Subordinate command and control servers, often called bot controllers, are often themselves compromised machines. The multi-tier C&C architecture of botnets provides anonymity for the botmaster (Feily and Shahrestani, 2009). Proxying and indirection are other approaches used to a hide botmaster's network identity. In a centralized architecture, the command and control server is a single point of failure. An entire botnet can be shut down by blacklisting the command and control (IRC) servers. A depiction of a centralized organization is shown below.

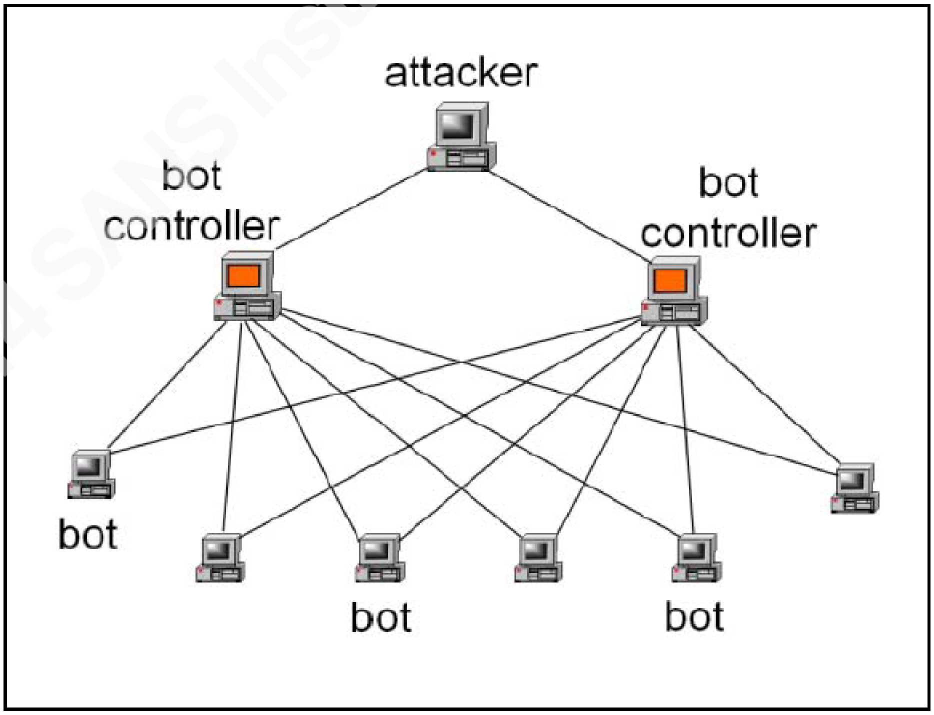


Figure 1. Botnet- centralized organization (Raghava, N., Sahgal, D., & Chandna, 2012).

More recently, decentralized botnets organizations have emerged. Decentralized botnet architectures are generalized as peer-to-peer (P2P) architectures.

There is no centralized command and control server in a P2P botnet. Bots are initialized with a list of hundreds of trusted peers that a bot will communicate with. The botmaster will communicate with a single bot peer (Raghava, N., Sahgal, D., & Chandna, 2012).

P2P botnets do not have a single point of failure. Control can be passed from bot to bot. Because bots do not have a centralized C&C server (or set of C&C servers) that all bots must communicate with, the P2P botnet is much harder to detect and destroy. Shutting down a server or small set of servers does not destroy the entire botnet (Wei Lu et al., 2009).

A depiction of a decentralized organization is shown below.

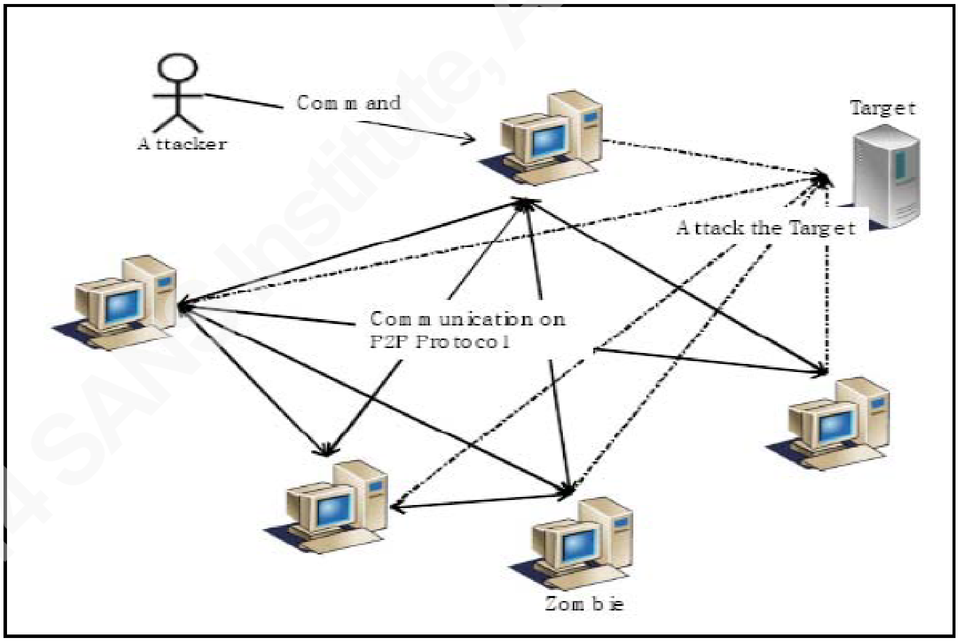


Figure 2. Botnet – decentralized organization (Raghava, N., Sahgal, D., & Chandna, 2012).

2.2. Botnet Communication Protocols

Early botnets utilized IRC chat protocol for communication in a centralized botnet. More recently, HTTP has been utilized, particularly because virtually every node on the internet is expected to access the internet over HTTP. Additionally, there are wide varieties of tools that support HTTP that can be leveraged in a botnet environment. Communicating over a common protocol makes detection of botnet activity more difficult.

2.3. Botnet Lifecycle

Nabil et al suggest a lifecycle consisting of 3 phases - Injection and spreading phase, command and control phase, and botnet application phase (Nabil. Hachem, et al., 2011).

In the spreading and infection phase, botnets spread using traditional malware propagation techniques – distribution of malicious emails, exploitation of software vulnerabilities, instant messaging, and via P2P file sharing network.

During the command and control phase, the botmaster commands the botnet and the botnets exfiltrate information using an appropriate architecture such as centralized or decentralized, and an appropriate communication protocol including IRC, HTTP, or P2P.

Destructive botnet behavior during the botnet application phase can be characterized as DDoS attacks, spamming and spreading malware and advertisements, espionage, and hosting malicious applications and activities.

According to Feilly et al., "a typical botnet can be created and maintained in five phases including: initial infection, secondary injection, connection, malicious command and control, update and maintenance" (Feily, M., & Shahrestani, A. 2009). Feilly's life cycle is shown below.

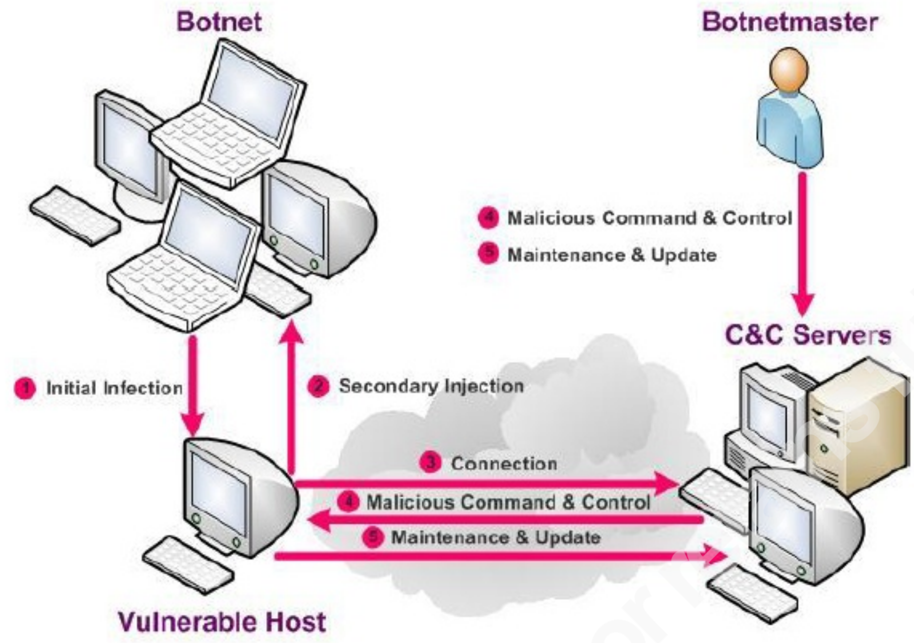


Figure 3. A Typical Botnet Life-cycle (Feily, M., & Shahrestani, A. 2009).

2.3.1. Initial Infection

Infection is a two-stage process. In the primary injection phase, machines are compromised in the traditional ways. Enticing a user to click on a link to a malicious website is one such way of effecting an initial infection. Another method is sending an email with a malicious payload. The payload can contain links to websites or infected PDF, Excel, or other types of files. If the computer used to perform the scan is on the same subnet, or the subnet is not properly defended, tools such as Nessus and Nmap can be used to perform port scans and vulnerability assessments, looking for open ports and vulnerable services. Once vulnerable machines are identified, a small piece of code such as shell code is copied to the victim machine.

2.3.2. Secondary Infection

Once the shell code is resident on the victim machine, phase 2 infection begins. The shell code retrieves the botcode from a specific location via any of a variety of protocols such as TFTP, FTP, HTTP, IRC, or P2P. The location can be an IP address that has been hard-coded into the shell program, hostname used in DNS lookup, or any other scheme that allows the shell to locate the repository and retrieve botcode.

In phase 2 infection, the malicious botcode is downloaded from a specified location and executed, making the infected computer a bot. The botcode runs, usually in the background, whenever the bot is running.

2.3.3. Connection

In the connection phase, the bot establishes a connection back to a command and control server in a centralized architecture or to a peer in a decentralized architecture. The connection can be based on one or more of many protocols. Usually a well-established protocol is used for ease of use. HTTP, IRC and P2P are generally well supported making them easy to use. Lesser known protocols or custom communication channels may be used for stealth purposes. Encryption is also used to disguise the channel.

2.3.4. Malicious Command and Control

During the malicious command and control phase, the bots receive commands from the botmaster and carry out their nefarious activities. “The attacker can ask the infected computers called ‘Agents’ or ‘Zombies’ to perform all sorts of tasks for him, such as sending spam, performing DDoS attacks, phishing campaigns, delivering malware, or leasing or selling their botnets to other fraudsters anywhere” (Nabil, et al., 2011).

2.3.5. Maintenance

The last phase in the botnet lifecycle is the botnet maintenance phase. In the botnet maintenance phase, binaries are updated for a variety of reasons. A binary may be updated to fix bugs in the prior release. A botnet binary may be replaced with a new binary that has updated capability and new exploit features. A botnet binary may be updated and replaced so that it provides a different signature so it can evade signature-based detection. A binary may be replaced to change the manner in which the bot finds its command and control server or peer. In some botnets, server name and IP address of the command and control server are short-lived and change frequently. To impede detection, some botnets use dynamic DNS (DDNS) to resolve server location and the resolved location is updated in the bot.

2.4. Botnet Detection Approaches

A simple taxonomy for classifying Botnet Detection Approaches is to categorize the technique as static or behavioral. However, the following sections classify botnet detection approaches into signature-based, anomaly-based, DNS-based and mining-based detection.

2.4.1. Signature-based detection approaches

Signature-based detection approaches are approaches that identify malware by its characteristics or signature. These signatures can be categorized as host-based or network-based. Host-based signature detection can be described as examining internally visible behavior whereas network-based detection can be described as examining externally visible behavior. The strength in signature detection techniques is that they generally have low rates of false positives. However, each machine is required to securely maintain a list of signatures that has to be updated as new threats are discovered. Clever malware will attempt to tamper with detection tools the signature lists, thus making the detection technique another attack vector. These signature lists have to be updated for this approach to be effective. Unfortunately, signature-based detection is not effective against unknown threats.

2.4.1.1. Host-based signature detection approaches

A host-based signature detection approach attempts to detect malware on the host by examining and monitoring the host's internal state, host-resident resources and behavior. In host-based signature detection, the distinguishing characteristic could be instruction patterns or series of instructions in the binary. Discriminating traits could be the collection of system calls a binary performs. Differentiating characteristics could be the IP address that the malware attempts to connect to or the hostname of the command and control server embedded in the binary. Characteristics peculiar to a particular attack could be the profile of when the malware is most active such as time of day or when particular conditions exist such periods of low or high processor utilization. A simple and often overlooked signature is the hash of the binaries. The hash of the binary can confirm that the binary is indeed a particular piece of malware. A hash of system files can be used to ensure the integrity of the system. Another characteristic is the set of files accessed or modified by the binary. Attempts to modify the registry, elevate privileges, or attached to various system resources can be signatures. The number of processes and threads created and the pattern of interaction between processes can be a signature. The length of time the processes live can also be a signature.

2.4.1.2. Network-based signature detection approaches

In network-based signature detection approaches, the machines behavior on the network is characterized as a signature. Aspects of network behavior that are examined include the IP addresses the machine communicates with, the ports used in the communication, the services communicated with, and the protocols used such as TCP, UDP, IRC, HTTP, and others.

The pattern of the traffic between machines can also be used to characterize a signature. Characteristics such as sequence, frequency, length, and content of messages constitute the pattern. Communications between the botmaster and bot will have different signatures than the communication between bot and targets.

2.4.2. Anomaly-based detection approaches

Anomaly-based detection techniques attempt to detect botnets based on several network traffic anomalies including high network latency, high volumes of traffic, traffic on unusual ports, and unusual system behavior that could indicate presence of malicious bots in the network (Saha and Gairola, 2005). Traditionally, anomaly-based detection approaches focus on the detection abnormal behavior. Statistical models are developed that characterize normal usage patterns and behavior that does not fall into the norm are categorized anomalous. The anomaly can be host-based or network-based. High network latency and high volumes, as mentioned by Feilly might actually be completely benign, normal, and acceptable behavior. Additionally, Feilly seems to suggest that anomalous behavior is anomalous network behavior (Feily, M., & Shahrestani, A. 2009). However, unexpectedly high CPU, memory, or disk usage should also be detected in an anomaly-based approach.

The strength of anomaly-based detection is that it can be effective against new and emerging threats as any statistically abnormal behavior will be detected. The weaknesses of anomaly-based detection approaches are two-fold. Firstly, a considerable amount of time and effort may be required to capture data and determine what constitutes normal behavior. Secondly, anomaly-based detection techniques can be susceptible to high rates of false positives. Any safe, secure, and normal activity that falls outside of the statistical limits will be detected as anomalous.

2.4.3. DNS-based detection approaches

DNS-based detection approaches exploit the fact that bots have a need to locate resources. In a centralized organization, bots need to locate their command and control server. This is probably the most common feature to exploit. Additionally, command and control servers have to relocate to another host. The DNS entry for the command and control server would after to be updated to reflect the new location. to avoid detection. The other host can be owned by the attacker or a botnet controlled by the attacker. DDNS allows rapid and frequent updates of the IP address (host name pairing) which allows a command and control server to move from IP address to IP address and for the bot to quickly find the new IP address of the host. The essence of DNS-based detection is to monitor DNS traffic as well as to monitor the state of the DNS database in the DNS server. The weakness of this approach is that not all botnets use DNS and this approach does not work on non-DNS based botnets.

2.4.4. Mining-based detection approaches

Mining-based detection techniques are perhaps the most comprehensive of all the detection techniques. Host-based and network-based activity is monitored and logged. Mining-based techniques examine all of the available log files, correlating events and trends to make assertions about the existence, or lack thereof, of malicious activity. Moreover, it is a technique that can incorporate some or all of the preceding techniques discussed in this paper.

2.4.5. Summary of published detection approaches

"One of the well-known signature-based Botnet detection techniques is Rishi that matches known nick-name patterns of IRC bots. Rishi is primarily based on passive traffic monitoring for suspicious IRC nicknames, IRC servers, and uncommon server ports. They use n-gram analysis and a scoring system to detect bots that use uncommon communication channels, which are commonly not detected by classical intrusion detection systems" (Zeidanloo, H.R., et al., 2010) & (Goebel and Holz, 2007).

The *nepenthes* platform is "a framework for large-scale collection of information on self-replicating malware in the wild. The basic principle of nepenthes is to emulate only the vulnerable parts of a service. This leads to an efficient and effective solution that offers many advantages compared to other honeypot-based solutions. Furthermore, nepenthes offer a flexible deployment solution, leading to even better scalability. Using the nepenthes platform we and several other organizations were able to greatly broaden the empirical basis of data available about self-replicating malware and provide thousands of samples of previously unknown malware to vendors of host-based IDS/anti-virus systems" (Baecher et al, 2006).

"Binkley and Singh, (2006), proposed an effective algorithm that combines TCP-based anomaly detection with IRC tokenization and IRC message statistics to create a system that can clearly detect client Botnets. This algorithm can also reveal bot servers (Binkley and Singh, 2006). However, Binkley's approach could be easily crushed by simply using a minor cipher to encode the IRC commands" (Zeidanloo, H.R., et al., 2010).

Lee, et al. introduced a tracking method of botnets by analyzing the relationship of domain names in DNS traffic generated from botnets. They showed they could find a set of unknown malicious domain names and their relationship by examining the DNS queries from the clients that accessed the known malicious domain names (Lee, et al., 2010).

Freiling, Holz, and Wicherski presented an approach to (distributed) DoS attack prevention that is based on the observation that coordinated automated activity by many hosts needs a mechanism to remotely control them. To prevent such attacks, it is therefore possible to identify, infiltrate, and analyze this remote control mechanism and to stop it in an automated fashion. Freiling, Holz, and Wicherski showed that this method can be realized on the Internet by describing how they infiltrated and tracked IRC-based botnets which are the main DoS technology used by attackers today (Freiling, Holz, and Wicherski, 2005).

AsSadhan and Moura observed that C2 traffic exhibits a repeated pattern of behavior. This is due to the nature of the pre-programmed behavior of bots. They explored this behavior and looked for periodic components in C2 traffic. They used periodograms to study the periodic behavior, and applied Walker's large sample test to detect whether the traffic has a significant periodic component or not, and, if it does, then it is bot traffic. This test is independent of the structure and communication protocol used by the botnet, and does not require any a priori knowledge of a botnet's behavior. Since they only looked at the aggregated traffic behavior, it is also more scalable than other techniques that examine individual packets or track the communication flows of different hosts" (AsSadhan and Moura, 2009).

"Another option is the use of artificial neural networks." (Sharafat, Rasti, and Yazdian, 2003) The advantage of using neural networks in anomaly detection is that features of 'normal' and 'abnormal' behavior can easily be learned by the neural network, as opposed to applying mathematics to describe the features of the data to the anomaly detector" (Vural and Venter, 2010).

Masud et al., (2008), proposed robust and effective flow-based botnet traffic detection by mining multiple log files. They introduced multiple log correlation for C&C traffic detection and they classified an entire flow to identify botnet C&C traffic. This method does not impose any restriction on the botnet communication protocol and is therefore applicable to non-IRC botnets. Furthermore, this method does not require access to payload content. Hence, it is effective even if the C&C payload is encrypted or is not available (Feily and Shahrestani, 2009) & (Masud et al., 2008).

Zhang and Lee proposed an approach that uses network-based anomaly detection to identify botnet C&C channels in a local area network without any prior knowledge of signatures or C&C server addresses. This detection approach can identify both the C&C servers and infected hosts in the network. The approach is based on the observation that, because of the pre-programmed activities related to C&C, bots within the same botnet would likely demonstrate spatial-temporal correlation and similarity. For example, they engage in coordinated communication, propagation, and attack and fraudulent activities. Zhang and Lee's prototype system, BotSniffer, can capture this spatial-temporal correlation in network traffic and utilize statistical algorithms to detect botnets with theoretical bounds on the false positive and false negative rates (Zhang, and Lee, 2008).

Botminer exploits the fact that bots behave similarly in regards to communication patterns with C & C servers or peers and bots behave similarly in regards to the malicious behavior that they perform.

"BotMiner uses cluster analysis to detect machines that exhibit certain similarities with respect to various flow attributes, such as packet byte rates, packets per flow, flows per address, and temporal flow volumes. In addition, BotMiner analyzes common port pattern usage and targets, which might further help detect macrolevel patterns commonly exhibited in activities such as spam, scanning the Internet for infection targets, or participation in denial-of-service (DoS) attacks" (Gu, Perdisci, Zhang, and Lee, 2008; Porras, 2008).

Bots need to find resources and often use DNS which, making DNS-based detection approaches attractive. However, it is difficult to discern between legitimate and malicious use of DNS updating. "In 2005, Dagon (Dagon, D., 2005), proposed a mechanism to identify botnet C&C servers by detecting domain names with abnormally high or temporally concentrated DDNS query rates. This technique is similar to the approach proposed by Kristoff (Kristoff, J., 2004). However, both techniques have the same weakness and could easily be evaded by using faked DNS queries. Furthermore, according to the evaluation in Raghava (Raghava, N., Sahgal, D., & Chandna, S., 2012), this technique generates many false positives due to misclassification of legitimate and popular domains that use DNS with short time-to-live (TTL)" (Feily and Shahrestani, 2009).

2.5. BotMiner

BotMiner correlates activities in the form of outgoing traffic patterns and communication relationships of communicating to nodes compute a score that relates to the probability of a node's membership in a botnet. The Bot-Miner architecture can be summarized into three functional areas – traffic monitoring, clustering, and correlation. Shown below is a depiction of tits architecture.

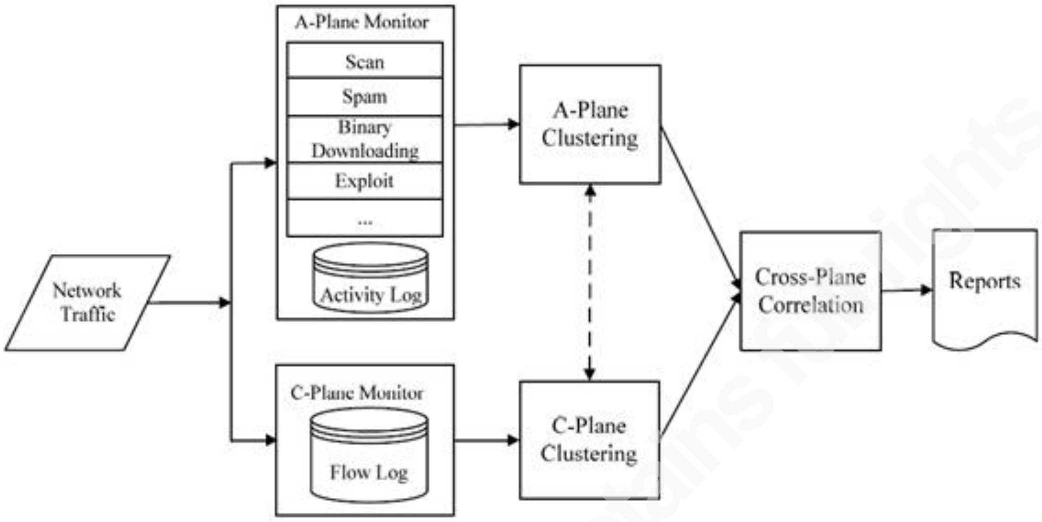


Figure 5: BotMiner architecture (Gu,G., Perdisci,R., Zhang, J., & Lee, W., 2008).

2.5.1. Traffic Monitoring

Traffic monitors are used to characterize the activity and communication patterns and flows. There are two types of monitors.

The A-plane monitors activities of the bot such as scanning subnets, spamming, downloading binaries, performing an exploit etc. In many cases, this is monitoring the outgoing traffic. The A-plane monitor logs information on “who is doing what” (Gu, Perdisci, Zhang, and Lee, 2008).

The C-plane monitoring is concerned with determining who talks to whom, when and how. Information collected include source and destination IP addresses, source and destination ports, protocols, services used, length of time a connection is established, number of packets sent, who initiated the connection, who terminated the connection.

2.5.2. Clustering

During the clustering activities, A-plane events are clustered according to the type of event such as spamming or scanning. During C-plane clustering, machines are grouped according to similar communication patterns.

2.5.3. Cross-plane Correlation

Cross-plane correlation involves cross checking clusters in both planes that show similar activities between nodes that share communication patterns. A score is computed indicating the likelihood of botnet membership.

2.6. Honeynet

A honeynet is a collection of honeypots, designed to lure attackers while capturing information about threats and to distract and divert attackers away from production machines. Generally, honeypots are real machines with vulnerabilities. Vulnerabilities range from unpatched operating systems to out of date applications to poorly configured firewalls. To attract attackers, the honeypots on the honeynet must look like targets of value.

Honeypots are not machines that are part of an organization's day-to-day business activities. Any interaction with a honeynet implies malicious or unauthorized activity. "Any connections initiated inbound to your honeynet are most likely a probe, scan, or attack. Any unauthorized outbound connections from your honeynet imply someone has compromised a system and has initiated outbound activity. This makes analyzing activity within your honeynet very simple" (Know Your Enemy: Honeynets, http://old.honeynet.org/papers/honeynet, 2006).

This section describes the Architecture, Data Control, Data Capture and Alerting capabilities of Generation II Honeynets.

2.6.1. Architecture

The Generation II honeynet is "an architecture, a highly controlled network used to contain and analyze attackers in the wild" (Know You Enenmy: GenII Honeynets, 2005). The main element of the honeynet architecture is the gateway. The gateway is also called the honeywall. The honeywall separates the victims from, and protects, the rest of the world. All traffic entering or leaving the honeynet traverses the honeywall. The honeywall contains three interfaces – eth0, eth1, and eth2. Eth0 is the external interface and separates the honeynet from the production network thus protecting the production network, which is important. Eth1 is the internal interface and is connected to the honeypots. Eth0 and eth1 are bridged at layer 2 to make it harder to detect the honeywall. Packets crossing eth0 to eth1 and etht1 to eth0 do not have their time to live (TTL) values decremented, which is detectable by attackers. Eth2 is a remote administration interface. Eth2 is used for honeywall and honeypot configuration, as well as log and captured data retrieval. The gateway is a secured Linux installation IPTables and firewalling. The honeynet architecture is depicted below.

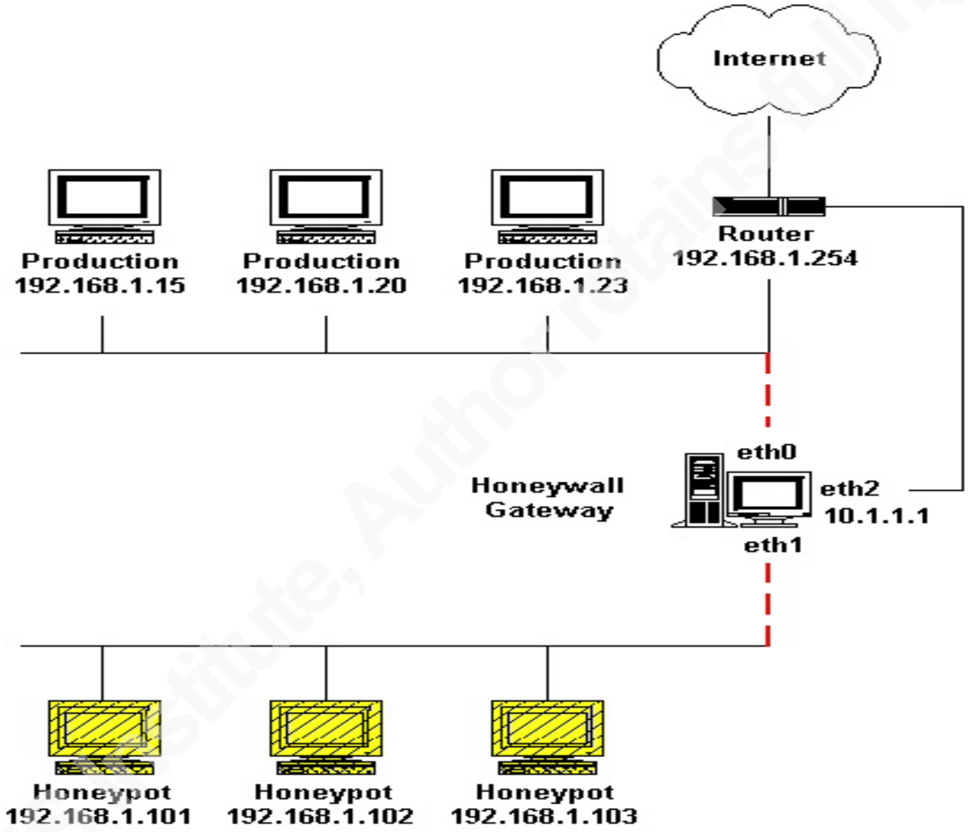


Figure 6. HoneyNet Architecture (Know You Enemy: GenII Honeynets 2005)

2.6.2. Data Control

Data control is designed to minimize the attacker's risk to other, non-honeynet systems while learning as much as possible about the techniques. The attacker has to be contained without the attacker's knowledge. The data control function of the honeynet employs two technologies – connection counting and network intrusion protection (NIPS). The connection counting is used to limit the number of outbound connections a honeypot can initiate. Network intrusion detection devices inspect traffic looking for known signatures of exploits and alerts when an intrusion is detected. NIPS are NIDs that, in addition to alerting, will block, drop, or prevent the connection. IPTables can be configured to provide connection limit capability in the honeywall. NIPS is used to recognize and block known exploits through packet inspection. Snort_inline (www.snort.org) is the tool of choice for NIDS and packet blocking.

2.6.3. Data Capture

"The key to Data Capture is collecting information at as many layers as possible. No single layer tells us everything." (Know You Enenmy: GenII Honeynets, 2005). Attacker's activities, system activity, network activity are all important pieces of information that should be collected and analyzed.

"The Honeynet Project has identified three critical layers of Data Capture; firewall logs, network traffic, and system activity." (Know You Enenmy: GenII Honeynets , 2005). Firewall activity is logged by the IPTables. All inbound and outgoing connections should be captured. A second snort process is used as not to overload the snort_inline process. The snort process logs all network traffic. The open source kernel module, Sebek (https://projects.honeynet.org/Sebek), is used to monitor system activity and capture the attacker's keystrokes.

2.6.4. Alerting

Honeynets need to be monitored at all times by dedicated personnel or by system services. A Simple Watcher utility called Swatch is used to monitor log files for patterns and alerting personnel via phone, email, alarms etc. An area for improvement is to provide log file correlation and analysis to make better determinations about the logged activity and the presence of bot code.

3. Case Studies

This section presents and summarizes two case studies illustrating the use of honey pots in the analysis of botnets and malicious code. In the first case study, the investigators characterize traffic on a college campus network using a honeywall. In the second case study, the investigators examine the communications patterns of a peer-to-peer botnet during the secondary injection phase.

3.1. Virtual Distributed Honeynet at KFUPM

Sqalli, AlShaikh, and Ahmed (Sqalli, M., AlShaikh, R., & Ahmed, E., 2010) performed a case study to evaluate the effectiveness of the Honeywall CDROM in exploring attacks on the King Fahd University of Petroleum and Minerals (KFUPM) network (Sqalli, M., AlShaikh, R., & Ahmed, E., 2010). One area of interest to the investigators is the source of the attack and the destination or resource being attacked. Another point on interest is the type of attacks occurring on the network. The types include denial of service attacks and port scanning. Lastly, the case study sought to explore the tools used to perform the attack such as ssh, and rsh,

Honeywall CDROM and KeyFocus' KFSensor were the two products evaluated for use in the case study. KFSensor is a windows-based IDS. KFSensor simulates vulnerable system services. The Honeywall CDROM solution was chosen primarily because of acquisition cost. The Honeywall CDROM is open source and free to use.

The analysis is largely a network traffic analysis. Malware detection and classifications, from a host-internal perspective, are not considered.

3.1.1. Experimental Setup

A segment of the university's network was simulated on single physical host. The physical host was an Intel-based Core 2 Duo with 80GB of disk space and 2GB of RAM. The host was virtualized with VMWare. The virtualized components contained on the host are summarized below:

3 VLANs – 2 campus VLANs and a private VLAN for the logging service

4 servers – 2 linux and 2 windows servers for deployment on the campus VLANs

1 honeywall – controls and captures network and contains logging server

Virtual Machines (VMs) were created to host the 4 honeypot servers. Each server was configured with 256MB ram and a 6GB HD and loaded with Fedora Linux or Windows XP. The Honeywall VM was configured with 1GB ram and a 10GB HD. The Honeywall implementation is shown in figure 7 below.

Sebek and Snort were introduced in 2.6.3. Sebek is an opensource product that monitors host activity, primarily by intercepting system calls. Sebek is available as a linux kernel module or as a windows driver. Sebek allows for a wide variety of data capture to from keystroke capture to recovery of files copied over the network. Socket, open, and fork are among the system calls that Sebek can intercept. Sebek has client and server components. Clients run on the honey nets and report captured data to the server. The server is typically executing on the honeywall. Sebek clients and servers communicate over a covert channel using Sebek's protocol. Attackers often encrypt their communications. Sebek captures data and instructions at the kernel level, after the code and data has been decrypted.

Snort is an open source, network intrusion detection application. Snort provides network protocol analysis, traffic analysis, content searching and packet logging. Snort is a rule-based product and based on rulesets, Snort is able to detect a wide range of attacks.

Snort and Sebek server were installed on the honeywall. Sebek clients were installed on the 4 honeypot servers. Sebek and Snort are supported by the Honeywall CDROM. Installation, configuration, and log analysis and display are supported by the Honeywall CDROM GUIs.

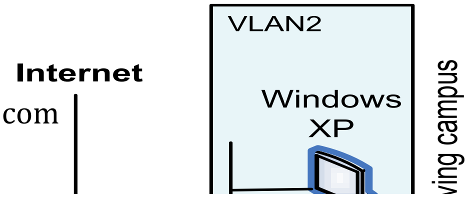
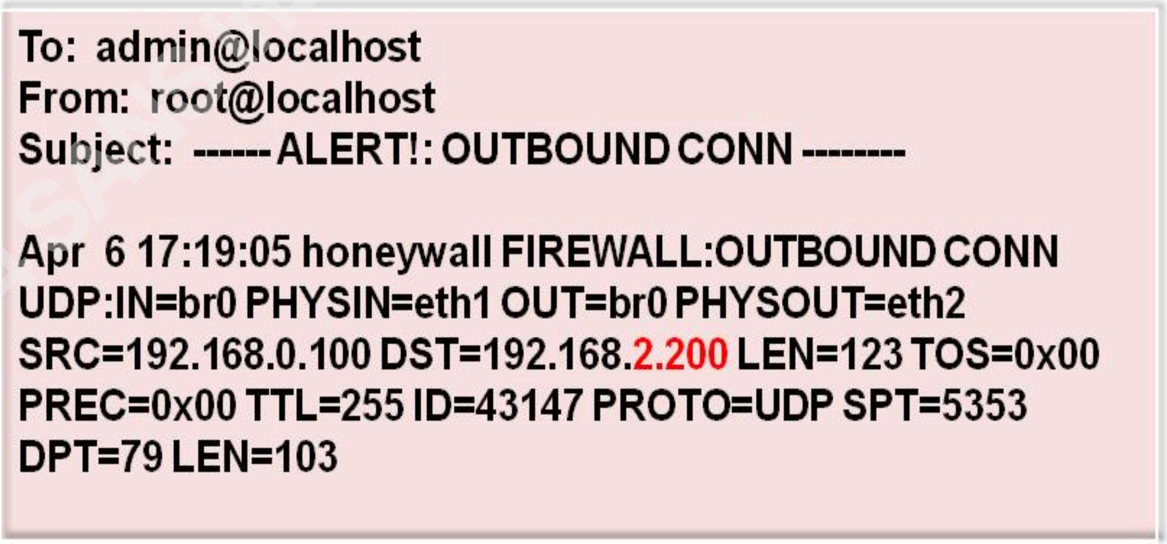


Figure 7. Virtual Distributed Honeynet at KFUPM (Sqalli, M., AlShaikh, R., & Ahmed, E., 2010).

Rather than use Swatch for alerting, as suggested by the Honeynet ptoject, a simple tool was written to monitor log files and inform the system administrator, via email, of any successful instrusion event. A sample email is illustrated below:



```
To:  admin@localhost
From:  root@localhost
Subject:  ------ ALERT!: OUTBOUND CONN --------

Apr  6 17:19:05 honeywall FIREWALL:OUTBOUND CONN
UDP:IN=br0 PHYSIN=eth1 OUT=br0 PHYSOUT=eth2
SRC=192.168.0.100 DST=192.168.2.200 LEN=123 TOS=0x00
PREC=0x00 TTL=255 ID=43147 PROTO=UDP SPT=5353
DPT=79 LEN=103
```

Figure 8. Alert Notification (Sqalli, M., AlShaikh, R., & Ahmed, E., 2010).

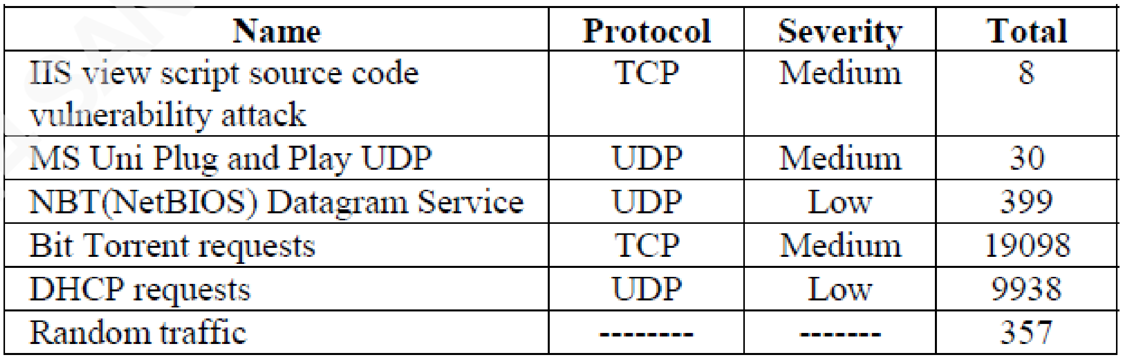
3.1.2. Data Collection

Due to the risk that honeypots pose in regards to being used to attack other system resources, the honeywall was not deployed on the real network. Instead, network traffic was recorded and then replayed on the honeywall's simulated network segment. The data was recorded using Wireshark and replayed using Tcpreplay. Wireshark is network traffic analyzer that provides packet capture and logging capability. Tcpreplay is a suite of tools that allows the editing and replaying of previously captured network traffic.

Twenty different collection events were performed. Each event recorded 90 minutes of traffic. In total, 30 hours of traffic was recorded.

3.1.3. Results and Conclusion

Over a 30 hour period, more than 30,000 activities were recorded. 35% of the traffic was considered low risk and 65% was considered medium risk. Although the table below lists the IIS vulnerability attack as medium, it should perhaps, be considered high. Bit torrent is a peer to peer protocol for efficient uploading and downloading of large files. Bit torrent traffic, which comprises roughly half of all internet traffic would be expected to account for, perhaps, more than half of traffic on a college campus. However, more discussion as to how the risk was determined to be medium is warranted.



Name	Protocol	Severity	Total
IIS view script source code vulnerability attack	TCP	Medium	8
MS Uni Plug and Play UDP	UDP	Medium	30
NBT(NetBIOS) Datagram Service	UDP	Low	399
Bit Torrent requests	TCP	Medium	19098
DHCP requests	UDP	Low	9938
Random traffic	--------	-------	357

Figure 9. Traffic Summary (Sqalli, M., AlShaikh, R., & Ahmed, E., 2010).

Honeywall CDROM useful design and implementation solution for gathering and analyzing traffic on distributed honeypots. Although Sebek was installed, it was not apparent what Sebek was used for. Sebek is primarily a host-based data gathering tool and this study is mostly a network-based study. Additionally, traffic was replayed through the honeywall which particularly since the honeypots were not used to

3.2. Peer-to-Peer Botnets

Although commands from an attacker in a peer-to-peer architecture may experience greater latency than commands issued in a command and control architecture, peer-to-peer architectures are increasingly popular because of their resiliency. The command and control server is a static, single point of failure in a centralized architecture. In a peer-to-peer architecture, the command and control is distributed, and dynamic. "A peer-to-peer network is a network in which any node in the network can act as both a client and a server" (Grizzard, J., Sharma, V., Nunnery, C., & Dagon, D, 2007).

There are a variety of primary infection methods. The goal of the primary infection is to maintain persistence on the host and connect to the bot network. The focus of this use case is to examine the secondary injection methods of the Trojan.Peacomm botnet.

The Trojan.Peacomm botnet uses the Overnet network protocol. The Overnet network protocol is based on the Kademlia algorithm.

A Kademlia network is basically a distributed hash table that provides for efficient storage and retrieval of key – value pairs. Keys are identifiers and are typically hashes. The keys can be identifiers of data or nodes.

Each node in the network has a unique ID that is sent with every message a node transmits. Each Kademlia node stores contact information about other nodes that is used to build internal routing tables.

Foundational to the Kademlia algorithm is an XOR-based notion of closeness. Node identifiers are XOR'd with the result being an indication of proximity.

There are four primitive operations defined in Kademlia – store, find_value , find_node, and ping.

The find-node primitive takes a node ID as a parameter. The recipient of the find_node command returns the k closest nodes to the searched for node. k is a system-wide parameter. Recursively, the initiator resends the find_node command to the newly discovered nodes. These primitive invocations are asynchronous and in parallel with one another. The lookups terminate when the node has been located or the initiator has received responses from all k closest nodes. Nodes that do not respond are removed from the initiator's routing table. (Maymounkov, P. & Mazieres, D., 2002). Store and find_value primitives behave similarly in terms of finding closest neighbors and sending commands to newly discovered neighbors until exhaustion of neighbor information or successful storing or locating of values.

3.2.1. Experimental Setup

The execution environment was a machine virtualized with VMWare GSX 3.2. The honeypot VM was a Windows XP VM. A honeywall controlled the honeypot's access to the internet. Pcap logs on the honeypots network traffic were captured. Snapshots of the honeypot's state were captured using the malware analysis tool PerilEyez. A Trojan.Peacomm binary was observed running in the honeypot for two weeks.

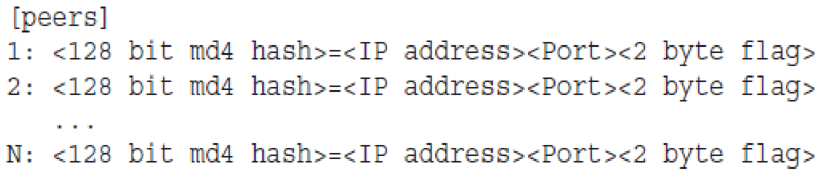
3.2.2. Initial Bot Infection

Snapshots of the system were taken before and after the initial infection. The snapshots were compared using PerilEyez. The snapshots captured the state of running services, open ports, and the file system.

Trojan.Peacomm creates a system driver, wincomm32.sys, and injects it into the services.exe Windows process, establishing persistence and a communication channel to the botnet.

Trojan.Peacomm also disable s the Windows firewall and opens TCP ports 139 and 12474, and opens UDP ports 123, 137, 138, 1034, 1035, 7871, 8705, 19013, 40519.

The Trojan.Peacomm binary has a hardcoded list of 146 peers and that peer list is copied into wincomm.ini. The bot joins the botnet by announcing itself to its peers. The peer list contains an 128-bit hash identifier, ip address, port, and flags. A section of the ini file is shown in the figure below.



```
[peers]
1: <128 bit md4 hash>=<IP address><Port><2 byte flag>
2: <128 bit md4 hash>=<IP address><Port><2 byte flag>
   ...
N: <128 bit md4 hash>=<IP address><Port><2 byte flag>
```

Figure 10: format of wincomm32.ini (Grizzard, J., Sharma, V., Nunnery, C., & Dagon, D, 2007).

3.2.3. Secondary Injections

The binary contains a hardcoded key that is used to retrieve the URL that contains the secondary injection location. The value returned from a search on that key is encrypted. The binary also has a hardcoded key that is used to decrypt the encrypted URL.

Using the hardcoded keys, the bot searches for the location of the secondary injection executable, decrypts the location, downloads and executes the secondary injection code. Secondary injections are downloaded using HTTP.

3.2.4. Network Trace Analysis

The honeybot was on a network segment that was NAT'd, hence approximately 10 machines shared the same IP address as the honeybot. The network traffic is for the IP address traffic. The infection occurs shortly after time 0 and the number of unique IP addresses increases dramatically around 800 seconds. The dramatic increase corresponds to initial infection and secondary injection activity, during which time the bot establishes communications with the botnet and searches for and receives the secondary injection. Around 2000 seconds, the botnet reaches somewhat of a steady state as the bot discovery is mostly complete.

Analysis of the Overnet packets revealed 5 hashes that were repeatedly searched for. One of the hashes was the botnets own hash which is to be expected since botnets publish their own hashes periodically per the Overnet algorithm. Two hashes are never found. The other two hashes are found by multiple nodes and correspond to the secondary injection URL.

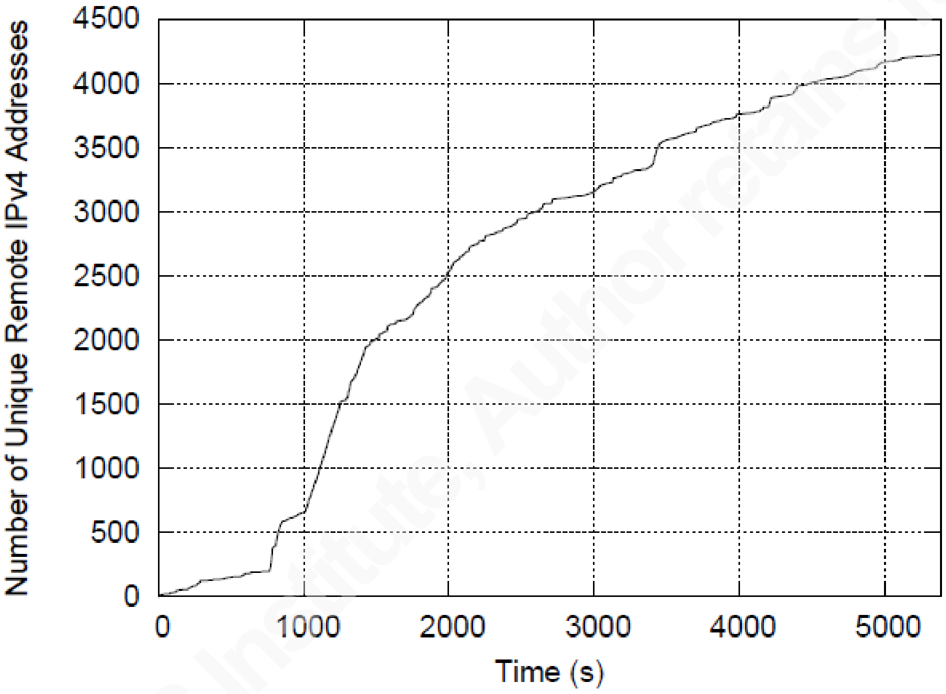


Figure 11 Addresses Contacted over Time (Grizzard, J., Sharma, V., Nunnery, C., & Dagon, D, 2007).

4. Conclusion

Botnets are powerful collections of cooperating computers, easily having as much processing power as a supercomputer and controlling enough network bandwidth to knock some countries off of the internet. Background information was presented on botnet infection techniques and the goals of a botnet.

An overview of botnet organization, architecture, and protocols were discussed. A model to describe a typical botnet was also discussed. A survey of botnet detection approaches was presented.

Botnet detection tools BotMiner and Honeynets were discussed and two case studies were presented.

Centralized Command and Control architectures typically have lower latency in terms of bots receiving issued commands however peer-to-peer architectures are more resilient and are harder to disrupt by taking a Command and Control server off-line. The Kademlia protocol, due to the parallel nature of the commands, provides peer-to-peer architectures with latency levels almost as low as in a centralized scheme.

Critical to detecting either a decentralized or centralized botnet is the ability to analyze data on the host and to analyze network traffic. Both are required. The BotMiner tool is based on have both sets of analysis available and both case studies utilized both approaches.

5. References

AsSadhan, B., José, M., & Moura, F., (2009), Detecting Botnets using Command and Control Traffic, Eighth IEEE International Symposium on Network Computing and Applications.

Bächer, P., Holz, T., Kötter, M., &Wicherski, G., Know your enemy: Tracking Botnets, 08/10/2008, www.honeynet.org/papers/bots , retrieved 6/4/2014.

Baecher, P., Koetter, M., Holz, T., 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 (RAID.

Binkley, R., & Singh, S., (July 2006), An algorithm for anomaly-based Botnet detection. In Proceedings of USENIX SRUTI'06, pages 43-48.

Dagon, D., (2005), "Botnet Detection and Response, The Network is the Infection," in OARC Workshop.

Dev, J.,(2013) Usage of botnets for high speed MD5 hash cracking, Innovative Computing Technology (INTECH), Third International Conference on, IEEE

Feily, M., & Shahrestani,A., (2009) A Survey of Botnet and Botnet Detection, Third International Conference on Emerging Security Information, Systems and Technologies.

Freiling,F. C., Holz, T., & Wicherski, G., (2005), Botnet Tracking: Exploring a Root-Cause Methodology to Prevent Distributed Denial-of-Service Attacks, , Proceedings of 10th European Symposium on Research in Computer Security, ESORICS.

Goebel, J., & Holz, T., (2007), Rishi: Identify Bot Contaminated Hosts by IRC Nickname Evaluation. In Proceedings of USE NIX HotBots'07.

Grizzard, J., Sharma, V., Nunnery, C., & Dagon, D, (2007) Peer-to_Peer Botnets: Overview and Case Study, In USENIX Workshop on Hot Topics in Understanding Botnets (HotBots'07).

Gu, G., Zhang,J., & Lee, W., (2008), "Botsniffer: Detecting botnet command and control channels in network traffic," in Proc. 15th Annual Network and distributed System Security Symposium (NDSS'08).

Gu,G., Perdisci,R., Zhang, J., & Lee, W., (2008), "BotMiner: Clustering Analysis of Network Traffic for Protocol- and Structure-Independent Botnet Detection," Proc. 17th Usenix Security Symp., Usenix Assoc.

Hachem, N., Ben Mustapha, Y., Granadillo, G., & Debar, H., (2011), Botnets: Lifecycle and Taxonomy, Conference on Network and Information Systems Security (SAR-SSI).

Know You Enemy: GenII Honeynets, old.honeynet.org/papers/gen2, 12 May 2005, retrieved November 14, 2012.

Know Your Enemy: Honeynets, http://old.honeynet.org/papers/honeynet , 31 May 2006, retrieved 6/4/2014.

Kristoff, J., (2004), "Botnets," in 32nd Meeting of the North American Network Operators Group.

Lee, J., Kwon, J., Shin, H., & Lee, H., (2010), Tracking Multiple C&C Botnets by Analyzing DNS Traffic, Secure Network Protocols (NPSec), 2010 6th IEEE Workshop on.

Lu, W., Tavallaee, M., Ghorbani, A., (2009), Automatic Discovery of Botnet Communities on Large-Scale Communication Networks, ACM Symposium on Information, Computer and Communications Security.

Maymounkov, P. & Mazieres, D. (2002), Kademlia: A Peer-to-peer Information System Based on the XOR Metric, In Proceedings of IPTPS02, Cambridge, USA.

Masud, M.M., Al-khateeb,T., Khan, L., Thuraisingham,B., & Hamlen, K.W., (2008), Flow-based identification of botnet traffic by mining multiple log file, in Proc. International Conference on Distributed Frameworks & Applications (DFMA), Penang, Malaysia.

Porras, P., (2009), Directions in Network-Based Security Monitoring, , Security & Privacy, IEEE Volume: 7 , Issue: 1.

Product Export Compliance Matrix. (2012). Retrieved July 4, 2014, from http://download.intel.com/support/processors/corei5/sb/core_i5-3500_d.pdf.

Raghava, N., Sahgal, D., & Chandna, S., (2012), Classification of Botnet Detection Based on Botnet Architechture, International Conference on Communication Systems and Network Technologies.

Saha, B., & Gairola, A., (2005), "Botnet: An overview," CERT-In White PaperCIWP-2005-05.

Sharafat,R., Rasti, M., & Yazdian, A, (2003) Neural network based anomaly detection in computer networks: a novel training paradigm, , in ISCA 16th International Conference: Computer Applications in Industry and Engineering, Las Vegas, NV.

Sqalli, M., AlShaikh, R., & Ahmed, E., (2010) Towards Simulating a Virtual Distributed Honeynet at KFUPM: A Case Study, in The IEEE UKSim 4th European

Modelling Symposium on Mathematical Modelling and Computer Simulation (EMS), Pisa, Italy.

Vural, I., & Venter, H.S., (2010), Using network forensics and artificial intelligence techniques to detect Bot-nets on an organizational network, Seventh International Conference on Information Technology.

Wang, Q., Chen, Z., Chen, C., Pissinou, N., (2010), On the Robustness of the Botnet Topology Formed by Worm Infection, Global Telecommunications Conference (GLOBECOM 2010), IEEE.

Wikipedia, Storm botnet, http://en.wikipedia.org/wiki/Storm_botnet, retrieved 6/28/2014.

Zeidanloo, H., Shooshtari, M., Amoli, P., Safari, M., & Zamani, M., (2010), A Taxonomy of Botnet Detection Techniques, Conference on Computer Science and Information Technology (ICCSIT), 2010 3rd IEEE International.