



The Australian National University  
Faculty of Engineering and Information Technology  
Department of Computer Science

# **Detecting Potential Peer-to-Peer botnets using the payload of network packets**

**Karun Dambiec**

**Supervisor: Dr Warren (Huidong) Jin,  
NICTA/CSIRO**

This is the final report for the COMP8760 Computer  
Science Project (Semester 2, 2008)

October 2008

© Karun Dambiec

Typeset in Palatino by T<sub>E</sub>X and L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub>.

Except where otherwise indicated, this thesis is my own original work.

Karun Dambiec

29 October 2008



---

# Acknowledgements

---

Thankyou to NICTA for allowing me to visit in 2008. Thankyou to Dr Huidong (Warren) Jin from NICTA and CSIRO for supervising. I would like to acknowledge Warren's support and encouragement throughout the completion of this project, especially given the short timeframe I had for the project and peer-to-peer bots being a relatively new research area. I have learned a lot within the short timeframe regarding threats to computer networks, and gained an understanding of how these threats could evolve, and the potential damage as a result of them.



---

# Abstract

---

Peer-to-Peer botnets are becoming an increasing threat to the security of computer systems and networks. The first peer-to-peer botnets only began to be created recently, with the Storm botnet being introduced in 2007. They comprise of a network of computer systems performing some malicious purpose for a botmaster and using peer-to-peer protocols for communication. The first common use of peer-to-peer protocols was Napster in 1999. The SGNet honeypot project is used to collect data for malicious traffic, and provides the dataset we use. As peer to peer botnets use a decentralised point of control and some peer to peer botnets encrypt their communication, we found that it is not a simple process to determine if network traffic is caused by a peer-to-peer botnet.

The main difficulty with peer-to-peer botnets is the detection of potential peer-to-peer botnet traffic. This difficulty occurs as a result of peer-to-peer botnets commonly using encrypted network traffic for its communication, and their decentralised control structure. We assume that there is some peer to peer botnet which send an encrypted packet to a system to determine if it is already infected, and a potential peer with the peer-to-peer bot before attempting to inject the system with it or attack the system. We assume that if encrypted data occurs before the plain text data, it is encrypted.

We propose a simple method for detecting encrypted network traffic, and detecting potential peer-to-peer botnets. We carried out an experiment to evaluate its performance using encrypted files, and found that it was 97% accurate. This method that we proposed allows us to determine which sessions have encrypted data before plaintext data. We then used the algorithm to find a total of 697 sessions from the period of 8th to 14th August 2008 where encrypted data occurred before plaintext data, and a total of 320 sessions from the period of 15th to 21st August 2008. The method we use for detecting encryption can be improved through being modified to be able to detect the difference between binary data being executable files and encrypted data. Although subject to the usual problems associated with not being able to differentiate between encrypted and binary data, no obvious systematic effects on the results were observed. We found using this method that there were ICMP packets at the beginning of some sections where the contents were encrypted.

We assume Peer to Peer bots are likely to start using ICMP more often, and that some peer-to-peer botnets can be detected through the examination of ICMP packets which they send. We contributed two methods to detect potential peer to peer botnet traffic. We propose a second method which extends the method we proposed previously to detect potential peer-to-peer botnet traffic, by using the contents of ICMP packets, and

includes a simple, yet effective method of detecting encrypted ICMP traffic.

In this work, we have made progress towards the goal of being able to detect new peer-to-peer botnets as they come into existence. Future work includes further automating and improving the method used to determine whether network packets are encrypted or plaintext.

---

# Contents

---

<b>Acknowledgements</b>	<b>v</b>
<b>Abstract</b>	<b>vii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Peer-to-Peer Botnets . . . . .	2
1.2 Honeypots . . . . .	2
1.3 SGNet Honeypot Project . . . . .	2
1.4 ICMP . . . . .	3
1.5 Project Objectives . . . . .	3
1.6 Project Schedule . . . . .	3
1.6.1 Proposed Timetable . . . . .	3
1.6.2 Actual Project Timetable . . . . .	5
1.7 Thesis Organisation . . . . .	5
<b>2 Peer-to-Peer Botnets</b>	<b>7</b>
2.1 Introduction to Peer-to-Peer Botnets . . . . .	7
2.1.1 Peer-to-Peer Botnet Topology Diagram . . . . .	8
2.1.2 IRC Botnet Topology Diagram . . . . .	9
2.1.3 Comparison of Peer-to-Peer and IRC Botnets . . . . .	9
2.2 Peer to Peer Protocols . . . . .	9
<b>3 Honeypots</b>	<b>11</b>
3.1 Introduction to Honeypots . . . . .	11
3.1.1 Why is a honeypot important? . . . . .	11
3.2 Honeypot Categories . . . . .	11
3.2.1 Low Interaction . . . . .	12
3.2.2 High Interaction . . . . .	12
3.2.3 Research Honeypots . . . . .	12
3.2.4 Production Honeypots . . . . .	12
3.3 SGNet Honeypot Project . . . . .	13
3.3.1 Introduction to SGNet Honeypot Project . . . . .	13
3.3.1.1 Epsilon - Pi - Gamma - Mu model . . . . .	13
3.3.1.2 Argos . . . . .	13
3.3.1.3 Nepenthes . . . . .	14
3.3.2 Data Description . . . . .	14
3.4 SGnet Database . . . . .	14

---

3.4.1	ScriptGenSession Table	14
3.4.2	TinySession	15
3.4.3	Packet	15
3.5	Horasis Python Library	15
3.5.1	LargeSession	16
3.5.2	TinySession	16
3.5.3	SGSession	16
3.5.4	Packet	16
3.5.5	Source	16
3.5.6	Environment	16
3.5.7	Injection	17
3.5.8	Activity	17
3.5.9	Malware	17
3.5.10	Behaviour	17
3.5.11	PE Info	17
3.5.12	Definition of Terms	17
<b>4</b>	<b>Internet Control Message Protocol</b>	<b>19</b>
4.1	Introduction to ICMP	19
4.2	Levenshtein Distance	20
<b>5</b>	<b>Hypotheses</b>	<b>21</b>
5.1	Hypothesis about Peer to Peer bots which check for peers before injection	22
5.2	Hypothesis about Peer to Peer bots which use ICMP for communication	22
<b>6</b>	<b>Experiments and Results</b>	<b>25</b>
6.1	Method for detecting encrypted network traffic	25
6.1.1	Performance Measure	25
6.1.2	Experiment Setup	26
6.1.3	Encryption Detection Effectiveness	26
6.2	Method for detecting peer-to-peer bots where peers are checked for before injection	28
6.2.1	SGNet data	28
6.2.2	Period of 8th - 14th August 2008	29
6.2.3	Period of 15th - 21st August 2008	33
6.3	Method for detecting peer-to-peer bots where they communicate using ICMP packets	40
6.3.1	SGNet Data	41
6.3.2	Standard ICMP Packet contents	41
6.3.3	Period of 8th - 21st August 2008	43
6.4	Discussion	45
<b>7</b>	<b>Conclusions</b>	<b>49</b>
7.1	Future Work	50
7.2	What I Learnt	50

---

**References**



---

# List of Figures

---

2.1	Diagram of Peer-to-Peer Botnet Topology . . . . .	8
2.2	Diagram of IRC Botnet Topology . . . . .	9
19		
5.1	Flow of Packets . . . . .	22
6.1	Histogram of Percentage of Alphabetical Characters for Packets . . . . .	27
6.2	Unencrypted file number 27 . . . . .	27
6.3	Encrypted file number 27 . . . . .	27
6.4	Histogram of Percentage of Alphabetical Characters for Packets for Pe- riod of 8th-14th August 2008 . . . . .	29
6.5	Percentage of Plain text characters for the first packet of each sessions during the period 8th - 14th August 2008 . . . . .	32
6.6	Sessions 726715, 726716, 726717 . . . . .	33
6.7	Histogram of Percentage of Alphabetical Characters for Packets for Pe- riod of 15th-21st August 2008 . . . . .	34
6.8	Percentage of Plain text characters for the first packet of each sessions during the period 15th - 21st August 2008 . . . . .	37
6.9	Contents of Payload of Packets for Session 737250 . . . . .	38
6.10	Sessions 737250 . . . . .	39
6.11	Combined Graph of Sessions . . . . .	40
6.12	Histogram of Levenshtein Distance against Windows XP ICMP Imple- mentation . . . . .	43
6.13	Histogram of Levenshtein Distance against Mac OS X ICMP Implemen- tation . . . . .	44
6.14	Histogram of Levenshtein Distance against Linux ICMP Implementation	45
6.15	Diagram of Peer-to-Peer Botnet Topology . . . . .	46
6.16	Diagram of IRC Botnet Topology . . . . .	46



---

# List of Tables

---

1.1	Proposed Timetable . . . . .	4
1.2	Actual Timetable . . . . .	5
3.1	Classes of Sessions in SGNet . . . . .	14
6.1	Confusion Matrix for Measuring Performance of Encryption Detection Method . . . . .	26
6.2	Confusion Matrix of Evaluation of Encryption Detection Method . . . . .	28
6.3	Packets Examined Around First Peak in Histogram for Period of 8th-14th August 2008 . . . . .	30
6.4	Packets Examined Around Second Peak in Histogram for Period of 8th-14th August 2008 . . . . .	30
6.5	Packets Examined Around Second Peak in Histogram for Period of 8th-14th August 2008 . . . . .	31
6.6	Packets Examined Around First Peak in Histogram for Period of 15th-21st August 2008 . . . . .	35
6.7	Packets Examined Around Second Peak in Histogram for Period of 15th-21st August 2008 . . . . .	35
6.8	Packets Examined Around Third Peak in Histogram for Period of 15th-21st August 2008 . . . . .	36



# Introduction

---

When most people think of security, they do not think of computer security and generally think of security guards denying or allowing access to buildings, the police or intelligence agencies preventing crime or a terrorist attack. It is not often most people associate security with computer systems or networks. Computer security has become more important in the protection of information than physical security, and has to be considered to prevent people from accessing computer systems or networks, or information stored on those computer systems or using computer networks for malicious purposes.

There are an increasing number of threats to the security of computer systems. One of these threats is peer-to-peer botnets. As a result of the increasing number of bots that make up a peer-to-peer botnet, and the difficulty of detecting them before an attack occurs, a distributed attack by a peer-to-peer botnet on an organisations computer network could severely affect an organisations ability to function and impact it financially.

Since these attacks are increasing and becoming more sophisticated in nature, there is a need for further research into peer-to-peer botnets, and for proactive security mechanisms to be developed rather than defensive mechanisms that merely protect a network from attacks.

In 2003, total internet attacks increased by 67.5% compared to the previous year[1]. As the number of internet users is continuously increasing, the total number of potential hosts for a peer-to-peer botnet increases and the number of internet attacks occurring will increase.

Until recently there has been very little research being done on peer-to-peer botnets, as peer-to-peer protocols only came into popular use initially for sharing of music with the advent of Napster in 1999. The Storm peer-to-peer botnet was introduced in January 2007, and peer-to-peer botnets only became a topic of discussion in the media around the time of its introduction.

Most research relating to botnets, has focused on IRC botnets which have a centralised point of control being an IRC Server. Peer-to-peer botnets are an increasing threat

to computer networks and systems, due to their increasing difficulty to detect as a result of their decentralised peer-to-peer structure, the size and potential bandwidth utilised, and increasing use of encryption.

In order to understand this work it must be understood what a peer-to-peer botnet is, a honeypot, the SGNet Honeypot project, ICMP and Frequent Sequential Pattern Mining. Once these fundamental concepts are understood, the contributions of our work can then be examined.

## 1.1 Peer-to-Peer Botnets

Peer-to-Peer botnets comprise of bots which run on a collection of computer systems, to perform some malicious purpose for a botmaster. They communicate with each other using peer-to-peer protocols. Some of these protocols used for communication within a botnet, encrypt the data that is sent between the peers in the botnet. The communication that forms part of an attack on a computer system or network, by a peer-to-peer botnet is not encrypted as this would require the peer-to-peer botnet to share encryption keys with its potential target.

## 1.2 Honeypots

A honeypot is a system which collects any data that has been sent to it over a network, and it allows us to record information about any unauthorised traffic or attempts to attack a network[2]. There are three main categories of honeypots being low-interaction, medium-interaction and high-interaction. Each category differs in the amount of interaction that is possible with an attacker. Dependent on the implementation and data collection method used by a honeypot, they can collect detailed information about unauthorised network traffic. Any traffic sent to a honeypot is unauthorised, as it is not used by normal users of the network on which it is on and not used for production purposes in an organisation.

## 1.3 SGNet Honeypot Project

SGNet is a worldwide honeypot project, consisting of a distributed network of 25 honeypots with multiple participants. It uses a combination of technologies being ScriptGen, Argos and Nepenthes to allow a high level of interaction with its attackers, so it can collect more detailed information than previous honeypot projects such as Leurrecom[3].

## 1.4 ICMP

The Internet Control Message Protocol or ICMP is a protocol for notification of events relating to the availability of a computer network and hosts on it[4]. Its purpose is to provide notifications about problems in the network environment, and its common uses are to notify a person whether or not a computer system or network is reachable[4].

## 1.5 Project Objectives

The objectives for this project were to investigate peer-to-peer botnets using the data that is provided as a part of the SGNet Honeypot Project, to gain an understanding of peer-to-peer botnets. We proposed two methods that could be used to detect potential Peer-to-Peer botnet traffic.

## 1.6 Project Schedule

### 1.6.1 Proposed Timetable

We created a timetable for the project, to allow it to be managed effectively as follows:

---

Week	Dates	Activities
1	21/7-27/7	Initial Meeting Define Research to be undertake Begin Review of Background Literature Begin Initial Presentation
2-3	28/7-3/8	Literature Review Data Exploration Finalisation of Study Contracts
4	11/8-17/8	Initial Project Presentation Continued Data Exploration Continuation of Literature Review
5-7	18/8-7/9	Research
8-10	8/9-28/9	Research
Mid Semester Break	29/9-12/10	Research
		Implementation
11	13/10-19/10	Evaluation of Results
12-13	20/10-2/11	Final Report Writeup Final Presentation Submission of Final Report

---

**Table 1.1:** Proposed Timetable

### 1.6.2 Actual Project Timetable

Week	Dates	Activities
1	21/7-27/7	Initial Meeting Define Research to be undertake Begin Review of Background Literature Begin Initial Presentation
2-3	28/7-3/8	Literature Review Data Exploration Finalisation of Study Contracts
4	11/8-17/8	Initial Project Presentation Continued Data Exploration Continuation of Literature Review
5-7	18/8-7/9	Research
8-10	8/9-28/9	Research
Mid Semester Break	29/9-12/10	Research Implementation
11	13/10-19/10	Evaluation of Results
12-14	20/10-29/10	Final Report Writeup Final Presentation Submission of Final Report

**Table 1.2:** Actual Timetable

The timetable changed significantly to the end, as a result of changes to the submission dates for the project submission items. The project was completed on schedule and the proposed timetable was achieved.

## 1.7 Thesis Organisation

We first introduce peer-to-peer botnets in Chapter 2, and compare them briefly to IRC botnets. Chapter 3 introduced honeypots, describes the SGNet Honeypot project our data is from and the data available from it. We introduce the ICMP protocol, and Levenshtein distance in Chapter 4, which we use the payload of ICMP packets in one of our experiments. In Chapter 5, we discuss our hypotheses and the basis for them, and in Chapter 6 we provide details of our experiments and the results. We discuss the significance of our results in Chapter ??, and we provide our conclusions and discuss future work in Chapter 7.



# Peer-to-Peer Botnets

---

## 2.1 Introduction to Peer-to-Peer Botnets

To be able to understand peer-to-peer botnets, we first need to know the origins of malicious bots. Malicious bots originated from simple bots whose purpose was to automate the repetitive tasks a human operator had to perform. They were initially mainly used to operate IRC channels when a human operator was unavailable, and ensure that when they returned the human operator, would still have control over the IRC channel and remain its operator. [5]

These malicious bots that were developed initially used IRC channels to form a botnet where the centralised control point was the IRC channel and a botmaster who instructed the bots to perform actions through this channel. Lately they have been evolving to become peer-to-peer bots in where their is no centralised control point, and no dependency on a central server.

A peer-to-peer botnet uses protocols that were developed originally for peer-to-peer systems. Peer-to-peer systems are systems which were originally designed to enable to sharing of primarily music files illegally. As a result of this they generally include features to ensure nodes remain anonymous making it more difficult to detect these nodes, ensure they are not affected by other nodes failing, and that it is not possible to prevent a peer-to-peer network from functioning as a result of nodes becoming malicious and hence attacking other nodes. [6] Not all peer-to-peer systems are completely without a centralised point of control, and in the case of a peer-to-peer botnet, it is likely to be a hybrid approach with the bots being a part of a peer-to-peer network where this network does not require a centralised point to exist, however a botmaster still remains a centralised point of control in terms that it instructs the botnet to perform certain actions [6].

A peer-to-peer botnet consists of a collection of computer systems with an agent known as a bot who communicate with its peer bots directly rather than via a central point of control such as an IRC channel. A bot performs some malicious action for the owner of the botnet known as a botmaster.

A sufficiently large peer-to-peer botnet or any form of botnet can have a severe impact on an individual country or company by preventing access services vital to the functioning of that countries economy or a company. In Estonia, in April 2007, an attack using a denial of service attack using a botnet occured that denied access to services which citizens depending on such as internet banking[7]. This resulted in Estonian businesses and the country not being able to operate on a normal basis both within Estonia and internationally until the attacks ceased after several days.

Peer-to-peer botnets are likely to become more advanced as they evolve in order to prevent their detection and monitoring of, and to circumvent defences against attacks by peer-to-peer botnets. Research needs to be done into both methods of detection, and defence against attacks by peer-to-peer botnets, as well as the development of future botnets. [8] propose a hybrid peer-to-peer botnet which will be a lot harder to detect, create network defences for and to prevent it from operating. They propose possible defence mechanisms against the hybrid peer-to-peer botnet. Even though the peer-to-peer botnet they propose could start being used by the blackhat community, defences against it were suggested in the same publication proposing it.

Botmasters may encrypt their communication between peers to avoid it been detected as easily as if it was plain text traffic[8]. We make the assumption, that there is a peer-to-peer botnet that checks if hosts are already infected with the botnet, before attempting to inject itself into a host. In this case we assume this check is encrypted, and the injection into the host is done using plaintext.

A peer-to-peer botnet is a collection of bots which unlike an IRC botnet do not have a centralised point of control, and act aboth as a client and server in a similiar manner to that of peer-to-peer filesharing applications[5].

### 2.1.1 Peer-to-Peer Botnet Topology Diagram

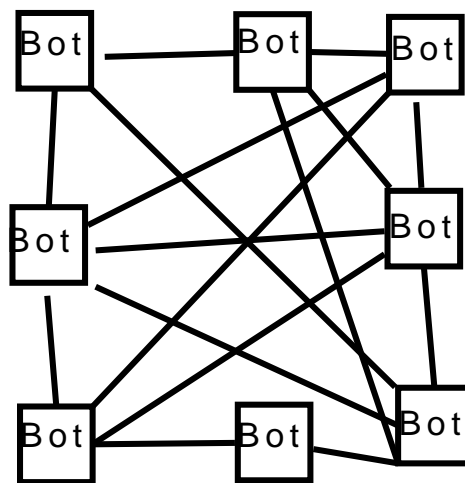


Figure 2.1: Diagram of Peer-to-Peer Botnet Topology

### 2.1.2 IRC Botnet Topology Diagram

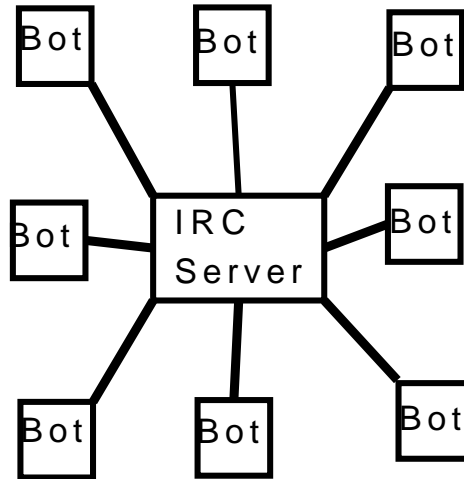


Figure 2.2: Diagram of IRC Botnet Topology

### 2.1.3 Comparison of Peer-to-Peer and IRC Botnets

We provide a diagram of the topology of a Peer-to-Peer botnet and IRC botnet above. The main difference between them is that IRC botnets require a centralised point of control being an IRC server, whereas peer-to-peer botnets have a decentralised point of control. An IRC Botnet connects to the server and performs actions automatically as requested by the botmaster who sends messages to the IRC channel the bots are a member of. A Peer-to-Peer botnet connects to its peers, and hence is not reliant on an IRC channel or server.

## 2.2 Peer to Peer Protocols

We briefly introduce the background to peer-to-peer protocols which are used by peer-to-peer bot nets. Peer-to-peer protocols became in common use initially for sharing music files with the advent of Napster in 1999. Napster used a partially decentralised model for their peer-to-peer network, where music files were transferred without requiring a centralised server and, the list of music available, the peers which provided the music and search mechanisms were provided by a centralised server. With Napster being shutdown as a result of a court order, alternative services and protocols including Gnutella and Kazaa started.

As the original purpose of peer-to-peer systems and protocols was to share files, possibly illegally, they include features to enable anonymity, ensure they are not affected by the failure of a single node and to ensure it is not possible to prevent the functioning of a peer-to-peer network as a result of nodes in the system attacking other

nodes in the system[9]. This results in bots that use peer-to-peer protocols being more difficult to detect when compared to IRC bots.

# Honeypots

---

## 3.1 Introduction to Honeypots

A honeypot is a system which collects any data that has been sent to it over a network, and it allows us to record information about any unauthorised traffic or attempts to attack a network[2]. There are three main categories of honeypots being low-interaction, medium-interaction and high-interaction. Each category differs in the amount of interaction that is possible with an attacker.

### 3.1.1 Why is a honeypot important?

We can analyse the data collected by the honeypot at a later date to determine trends in the behaviour of peer-to-peer botnets, and to find methods of detecting attempted intrusions in order to prevent them in the future. If a honeypot is behind a firewall, it can be used to determine that no attacks are occurring from the inside, and to indicate that network defences are working well[10].

## 3.2 Honeypot Categories

Spitzner[11] proposed three classes of honeypots being low interaction, medium interaction, and high interaction. Each class has a different level of interaction with an attacker. Also Mokube and Adams[2] proposed two different types of honeypots being research honeypots and production honeypots. The purpose of each of these types is different and would operate differently. As part of this work it is necessary to understand the basic concepts relating to honeypots, and the difference between low and high interaction honeypots.

### **3.2.1 Low Interaction**

A low interaction honeypot is one in which a very limited amount of interaction occurs with an attacker. This offers advantages being that an attacker could not potentially take over the machine and use it for their malicious purposes, and the attacker has limited access to the operating system allowing the honeypot to record data with a low risk of being compromised[10]. This limited interaction has disadvantages as the attacker is not able to complete their attack, as it would not allow them to inject a malicious piece of code. Also as it does not allow us to have much interaction with the attacker, the honeypot may only capture the initial connection request or a portion of the potential data that could have been captured if the attack was completed. These can be used as part of an intrusion detection system as the assumption that all traffic to a low interaction honeypot is unauthorised can be made, and all sources of the network traffic received by the honeypot can then be blocked.

### **3.2.2 High Interaction**

A high interaction honeypot is one in which an attacker is able to attack a honeypot that behaves in a very similar manner to a real system. The advantages of this compared to a low interaction honeypot, is that it is able to gain more information about new attacks, and vulnerabilities[10]. Whereas the low interaction honeypot, may only be able to respond to previous forms of attacks. It allows us to gather more detailed information regarding the behaviour of an attacker, and allow for the injection of malicious code. There are several potential disadvantages to these in that the attacker could potentially take over the system which runs the honeypot and cause it to perform malicious actions as a part of a botnet.

### **3.2.3 Research Honeypots**

A research honeypot is one that aims to collect detailed information about new attacks, changing trends in attacks, and to collect as much useful data as possible[2]. It differs from a production honeypot in that it has different objectives.

### **3.2.4 Production Honeypots**

Unlike a research honeypot, the objective of a production honeypot is not to collect as much data about attacks as possible[2]. Rather its objective is to collect information that could be used to prevent potential attacks. These honeypots could be a part of a network intrusion detection system, or the information be given to a firewall or administrator, to carry out actions to defend against the attack.

---

## 3.3 SGNet Honeypot Project

### 3.3.1 Introduction to SGNet Honeypot Project

SGNet is a worldwide honeypot project, consisting of a distributed network of approximately 25 honeypots with multiple participants. It uses a combination of technologies being ScriptGen, Argos and Nepenthes to allow a high level of interaction with its attackers, so it can collect more detailed information than low interaction honeypot projects[3].

ScriptGen is a tool that analyses traffic between a real machine and an attacker, generates a finite state machine of the interactions that occurred for a protocol, and then creates a script of these interactions which is then used by honeyd to emulate a protocol. As it generates an emulation of a protocol it will not be completely accurate, however it will be sufficient for most protocols to convince an attacker, a honeypot based on its scripts is a real system. [12]

Further refinements to ScriptGen based honeypots have been made by [13] to allow 0-day attacks to be handled by the honeypot and new scripts for the honeypot to be generated online without requiring a real system to be used initially and the interactions recorded. We can now gather more detailed data regarding new attack forms as they occur as these enhancements are used in the SGNet project.

#### 3.3.1.1 Epsilon - Pi - Gamma - Mu model

SGNet uses a model which they call the Epsilon - Gamma - Pi - Mu model. This model is used for attacks where an injection of some malware is the goal of the attack. We describe the main components of it:

- **Epsilon**  $\epsilon$  is the network traffic or exploit that is required to be sent to the host before the attacker is able to gain control over it in order to carry out the next stage of its attack.
- **Gamma**  $\gamma$  is the network traffic that is sent by the attacker to gain control over a part of a system, and redirect the control to a remote location.
- **Pi**  $\pi$  is the payload which is sent to a system as a part an attack on it.
- **Mu**  $\mu$  is the malware which has been downloaded by the SGNet honeypot project, if it has been able to download malware successfully.

#### 3.3.1.2 Argos

Argo is a system based on an emulator which accepts network traffic, allowing an attack to perform an code injection while replacing the injections's shellcode with its

own forensic shellcode which is specific to the operating system of the attack, and then generates a signature of the injection[14]. The signature of the attack generated by Argos, could then possibly be used by an intrusion detection system. ScriptGen uses Argos to trace the beginning of the data that would be executed by a system, in order to allow the finite state machine that it uses to learn the exploit that forms a part of the attack. This results in it being able to learn the exploit up to the payload  $\pi$  stage where the code injection occurs, and hence Argos does not need to be used again by SNet for attacks of the same type. [3].

### 3.3.1.3 Nepenthes

Nepenthes is used to identify the behaviour of an attack's payload. Its objective is to download malware through the emulation of vulnerabilities that the malware uses, so that the malware can be analysed [3]. Its use is relevant to downloading malware from peer-to-peer botnets that use injection based methods to propagate.

## 3.3.2 Data Description

We give a description of the dataset we use, so that it can be understood how the data is relevant to the method we develop. The SNet deployment stores data in a Oracle Database which we accessed remotely using a combination of SQL Queries to the database and the Horasis python library provided by the SNet Honeypot Project.

## 3.4 SGnet Database

We describe the components of the database for the SNet project which we use.

### 3.4.1 ScriptGenSession Table

As of the 18th of September 2008, the SNet database contained 1 962 928 sessions of the following classes:

Session Type	Number
NP	22 254
HY	26 020
AG	625 711
SG	1 285 958

**Table 3.1:** Classes of Sessions in SNet

---

It contains four classes NP, HY, AG and SG which occur when one of the following happens:

- SG occurs when the session is able to be completely handled by the finite state machine that is using by the SGNet deployment [3].
- HY occurs when the session is not able to be completely handled by the finite state machine, and is sent to other components of the SGNet honeypot project [3].
- NP occurs when a SGNet honeypot passes the session to Nepenthes, and it is able to recognise the shellcode that forms a part of the session [3].
- AG occurs when a SGNet honeypot passes the session to Argos, and the session's payload is successfully recognised by Argos [3].

### 3.4.2 TinySession

The honeypot database contains a table of TinySessions. A TinySession is all attacks by a system to a particular honeypot IP address. Each tiny session refers to the packets which were sent during that session.

### 3.4.3 Packet

The honeypot database contains a collection of packets. For each packet, it contains information about the source ip and port, destination ip and port, payload and timestamp.

## 3.5 Horasis Python Library

Horasis is a python library which provides us with a collection of objects, that allows us to retrieve data by creating new objects. Most information can be accessed directly via SQL without the library. The individual packets provided in the database, we could only retrieve through the use of Horasis, as the format in which they were encoded was dependent on the Python Impacket library to convert it back into binary data.

We discuss only objects which relate specifically to the SGNet Honeypot Project. The main object types it provides are Source, Environment, Packet, Injection, Activity, PE\_Info, Behavior, Malware, SGSession, TinySession and LargeSession.

### 3.5.1 LargeSession

A **LargeSession** is an object which refers to a collection of **TinySession** objects that all come from the same source IP address, that have been received by different hosts on the same sensor in the SGNet project. Each sensor in the project operates a number of virtual machines, each of which we refer to as a host.

### 3.5.2 TinySession

A **TinySession** is an object which refers to a collection of packets that have been received and sent from/to a source IP address from/to a host, and the associated ScriptGen sessions. These **TinySessions** consist all packets sent to multiple ports on the host from a source.

### 3.5.3 SGSession

A **SGSession** or ScriptGen Session object is an instance of each session which goes through ScriptGen. It consists of data being received from individual source ips with individual ports going to a specific destination IP address with a destination port.

### 3.5.4 Packet

A **Packet** is the finest granularity of object available using Horasis. It consists of a Unique Packet Identifier, **TinySession** identifier, Timestamp, and the packet header, and payload. It can either be a TCP, UDP or ICMP packet.

### 3.5.5 Source

A **Source** is any host that attacks a system within a twenty-five hour period. It provides information regarding the IP address of the host, and a timestamp indicating when the twenty five hour period started. Twenty-five hours is the longest a host can be a source from when the timestamp started, to allow the SGNet deployment to be able to handle dynamic IP addresses.

### 3.5.6 Environment

An **Environment** is an object representing a sensor in the SGNet Honeypot project. It is a deployment of SGNet consisting of several hosts on a system. The information provided by this object is an identifier, name and the country in which it is in.

### 3.5.7 Injection

An **injection** object contains the information relating to an attempt to inject some executable code into a honeypot. The information is contained using the Epsilon - Gamma - Pi - Mu model we discussed earlier.

### 3.5.8 Activity

An **activity** is a point at which data that forms a part of the Epsilon - Gamma - Pi - Mu model can be identified (I need to look more at this and possibly clarify/correct).

### 3.5.9 Malware

**Malware** is an object that represents an instance of malware being downloaded by SGNNet. It contains all information relating to the Malware that has been discovered when it was downloaded, and it consists of a link to a Behaviour object.

### 3.5.10 Behaviour

**Behaviour** is an object that contains all information that has been gathered by Anubis regarding its behaviour. Anubis(<http://anubis.iseclab.org/>) is a tool that analyses malware and gathers data regarding the malware's behaviour.

### 3.5.11 PE Info

**PE Info** provides data about the environment, operating system, characteristics about a specific piece of malware that was downloaded.

### 3.5.12 Definition of Terms

An **Attack** is any attempt by a system to use a honeypot for malicious purposes.



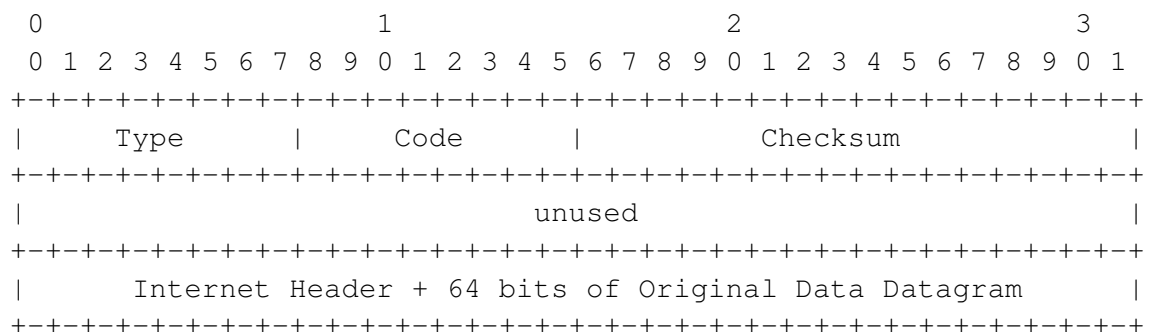
---

# Internet Control Message Protocol

---

## 4.1 Introduction to ICMP

The Internet Control Message Protocol or ICMP is a protocol for notification of events relating to the availability of a computer network and hosts on it[4]. Its purpose is to provide notifications about problems in the network environment, and its common uses are to notify a person whether or not a computer system or network is reachable[4]. The most common applications of ICMP that are known are Ping and Traceroute. Ping is an application which sends a host a ECHO message, and the host ECHOs it back to the sender. Traceroute is where a host uses ICMP packets to trace the path packets travel to reach a destination.



**Figure 4.1:** Diagram of ICMP from RFC 792[4]

ICMP has started being used for applications other than its original purpose. The BHO trojan which is a keylogger uses ICMP packets to send the keys it has logged to a destination. It allows data to be included as part of the ICMP packet. The standard procedure for a reply to an ICMP ECHO packet, is to send an ECHO reply with the same packet contents.

The data contained in ICMP packets may be encrypted.

## 4.2 Levenshtein Distance

We use the Levenshtein distance to determine the distance between the standard implementations providing by operating systems for ICMP packets, and the data provided by the SGNNet honeypot project. We introduce the basic concept of Levenshtein distance.

Levenshtein distance allows us to determine the number of operations that is required to transform one sequence into another sequence[15]. For the transformation of the word pear to beers, we calculate the Levenshtein distance as follows:

1. pear  $\rightarrow$  bear (We substitute b for p)
2. bear  $\rightarrow$  beer (We substitute e for a)
3. beer  $\rightarrow$  beers (We insert an s at the end)

As it was necessary to make 3 transformations to transform pear into beers we have a levenshtein distance of three.

By using this distance, we can find that the smaller the distance between ICMP payloads, the more similar two ICMP packets are.

## Hypotheses

---

Peer-to-peer botnets have now started to use encryption for communication between peers to be able to avoid detection by intrusion detection systems and firewalls. As communications are encrypted each time a peer-to-peer bot, communicates with a peer in its botnet the signature for the communication would be different. As a result signature based methods where the signature is generated based on the contents of a network packet, would not be able to be used due to the constantly changing signature of the communication.

Not all peer-to-peer botnets use encryption for communications between peers even if they implement a peer-to-peer protocol that includes encryption as part of the protocol specification. Phatbot is one such bot. It is based on a protocol known as WASTE which includes encryption as a part of its specification, however the authors chose to disable it due to its key sharing requirements[16]. As it uses plaintext for communications, it is simpler to detect using signature based detection techniques. It could however be possible for a botmaster to modify the bot to use encryption which is included in the WASTE protocol, and hence make it much more difficult to detect.

Sinit is a peer-to-peer bot which sends out packets to random hosts to determine if they are a potential peer. It does not rely on a seed list of peers, and only requires all code sent to be executed by the botnet to be signed with a private encryption key. It was easily detected due to its choice of using the same port as the DNS protocol and its packets contents not corresponding to that of a normal DNS request. [17]. It does not however encrypt the packets that are sent to random hosts to determine if it is a peer. Nor does it attempt to inject hosts with the bot, rather it relies on them going to a website.

Nugache is one peer-to-peer bot which encrypts all of its communications, and therefore any communications between an infected system and its peers is not easy to detect. Nugache aims to distribute itself through clients. [18; 8]

Possible methods for bots to distribute themselves other than via email or through websites, include buffer overruns and exploits in software. A peer-to-peer botnet can choose to use one of these methods as there could be a large community of systems

which do not have the latest patches to prevent any discovered exploits. To maximise a bots efficiency in distributing itself, it could check to see if a system it was intending to inject itself into was already infected. This would result in the bot not having to attempt to inject itself into a system, therefore decreasing the amount of network traffic caused by the bot, allowing it to maximise the number of systems it can infect within a time frame.

In Wang et al(To be Published), they propose that a botnet in the real world, would not reinfect existing bots that are a part of a peer-to-peer botnet as this could improve detection of the botnet. We make the assumption that in order to minimise the number of reinfections occurring, there there would need to be some method of determining if the bot is already infected. We assume this would occur from a packet which is encrypted sent to the potential bot, and if no response is received the system is then injected with the bot. We assume this injection would occur using packets sent in plaintext.

## 5.1 Hypothesis about Peer to Peer bots which check for peers before injection

We assume there are some peer-to-peer botnets which send an encrypted packet to a system to determine if it is already infected, and a potential peer with the peer-to-peer bot before attempting to inject the system with it or attack the system. It was hypothesised that their is some peer-to-peer bot which performs the following sequence while injecting systems with the bot:



Figure 5.1: Flow of Packets

## 5.2 Hypothesis about Peer to Peer bots which use ICMP for communication

ICMP is a protocol which is commonly used for providing feedback to hosts on a network regarding the state of a host or computer network. As a part of the protocol it provides the ability to send data as part of an ICMP packet. The BHO Trojan which is a key logger, uses ICMP packets to send the keys a user has entered. It uses this as it is not common for network security devices, being firewalls and intrusion detection systems to check the contents of packets for this protocol.

We assumed that there are ICMP packets which do not have the standard payload that is sent by the standard operating system utilities for generating ICMP packets. We can calculate the Levenshtein distance between the standard operating system implementation of ICMP and the payload of the ICMP packets that we have received. By finding packets which are not similar to the standard operating system implementation of ICMP, we assume we can detect a potential peer-to-peer bot before it manages to attack other systems in the network, by detecting its communication with peers and potential peers as early as possible.



---

# Experiments and Results

---

Detection of potential peer-to-peer bots is the first step in being able to respond to this threat proactively. In this chapter, we will discuss the methods we proposed to detect potential peer-to-peer bots, and the experiments we carried out in relation to these proposed methods.

We first propose a method of detecting encrypted network traffic, and carry out an experiment to determine if it is accurate. As peer-to-peer bots vary in their communication methods and behaviour, we proposed two methods of detecting potential peer-to-peer botnets. The first method proposed involves detection of an encrypted check of a potential peer, before a peer-to-peer bot attempts to inject a system with itself. Not all peer-to-peer bots use injection methods to propagate, and some propagate via emailing themselves to users or having users download and run them from websites. The second method we proposed involved finding the levenshtein distance between the payload from operating system implementations of ICMP and the payload of ICMP packets collected by the SGNet Honeypot Project which were selected from sessions where encrypted packets occurred before non encrypted packets.

## 6.1 Method for detecting encrypted network traffic

We proposed a method for detecting encrypted network traffic, where the encrypted network traffic appears at the beginning of a session. We carried out experiments to determine if it was reliable at detecting encrypted data.

### 6.1.1 Performance Measure

We defined the performance measure for our method of detecting if data is encrypted or not to be the percentage of true positives in a confusion matrix. A true positive occurred when the data was classified into the encrypted class correctly. A true negative occurred when the data was classified into the plain text class correctly.

---

	Encrypted	Plaintext
Encrypted	True Positives	False Positives
Plaintext	False Negatives	True Negatives

**Table 6.1:** Confusion Matrix for Measuring Performance of Encryption Detection Method

### 6.1.2 Experiment Setup

We tested the method we proposed for detecting encrypted data using files that were encrypted using the PGP Desktop application, and the plain text versions of these. These files simulated encrypted payloads of a network packet, and plaintext payloads of network packets.

We have tested the experiment with data from the SNet HoneyPot Project. As the data provided by the project was multiple gigabytes in size and would require a sizeable time to retrieve, we only retrieved data for the period of 8th August 2008 to 14th August 2008, and 15th August 2008 to 21st August 2008.

### 6.1.3 Encryption Detection Effectiveness

We created the variable percentage of plain text characters for each packet. The percentage of plain text characters consisted of the percentage of alphabetical characters in lower case and upper case, and numerical characters that were contained in the packet. We excluded spaces to remove the effect, they had on plain text and encrypted data. A plain text document could contain a lot of spaces, which would have affected the percentage of plain text characters count.

We used the following algorithm to calculate the percentage of plain text characters:

**Data:** Packet

**Result:** Percentage of Plain Text Characters

**for each character  $c$  in packet do**

**if  $c$  is a letter between A to Z, a to z or a number 0 to 9. then**

        | Increment number of plaintext characters

**end**

**else if  $c$  is not a space then**

        | Increment number of non plaintext characters

**end**

**end**

**Return:** number of plaintext characters divided by number of non plaintext characters

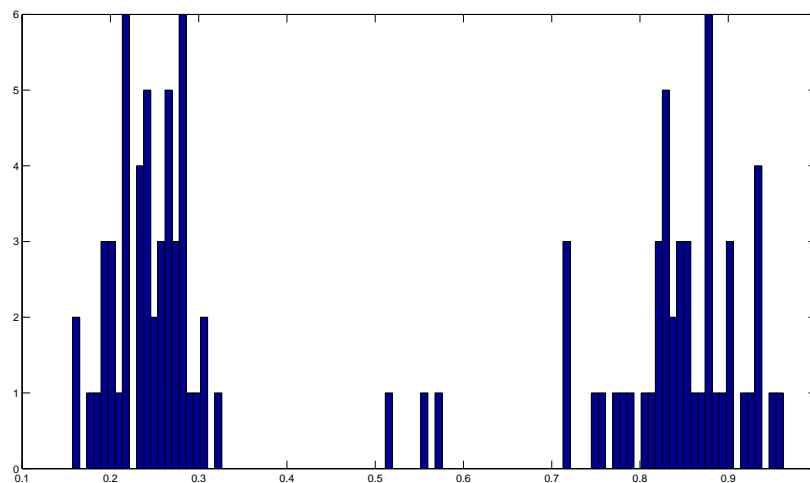
**Algorithm 1:** An algorithm for determining the percentage of plain text characters in the payload of a network packet

We calculated the percentage we receive using the algorithm for the example data below, where \* represents any non plain text character.

AbCd123\*

We have a percentage of 87.5% for the above data, as there are 7 plain text characters and one non plain text character.

We tested the method for detecting encrypted packets against the encrypted files that were created with PGP, and the plain text versions. We created a histogram showing the percentage of plain text characters. We examined the peaks of the histogram to determine the percentage which we could be confident of was encrypted packets, and plaintext packets.



**Figure 6.1:** Histogram of Percentage of Alphabetical Characters for Packets

The following file when ran against our algorithm returning a percentage of approximately 82%

"fkd9ik23s"

**Figure 6.2:** Unencrypted file number 27

The encrypted version of the file returned a percentage of approximately 23.6%

PGP  
 \gD\_yeOrł\&}=etuV57,-Bx|[ŁHb Ln}

**Figure 6.3:** Encrypted file number 27

We then selected the percentage of plain text characters that are in a file to be plain text as 70%, as it is the start of an area around the peak. We then created a confusion matrix of the results:

	Encrypted	Plaintext
Encrypted	47	0
Plaintext	3	50

**Table 6.2:** Confusion Matrix of Evaluation of Encryption Detection Method

We found that given the test data, the method for determining if data is encrypted is 97% accurate at classifying the data as encrypted or plaintext.

## 6.2 Method for detecting peer-to-peer bots where peers are checked for before injection

### 6.2.1 SNet data

We retrieved two weeks of data separately for the week of 8th to 14th August 2008, and the week of 15th to 21st August 2008.

For the first week, we retrieved all tiny sessions from the SNet Database for the period of 8th to 14th of August 2008 using the following SQL Statement:

```
SELECT * FROM TINY_SESSION WHERE TINY_SESSION.BEGIN_AT
> to_date('8-AUG-8') AND tiny_session.begin_at < to_date('14-AUG-8');
```

For the second week, we retrieved all tiny sessions from the SNet Database for the period of 15th to 21st of August 2008 using the following SQL Statement:

```
SELECT * FROM TINY_SESSION WHERE TINY_SESSION.BEGIN_AT
> to_date('15-AUG-8') AND tiny_session.begin_at < to_date('21-AUG-8');
```

The initial information we retrieved about the sessions for each week from the Tiny\_Session table consisted of the following attributes:

- Tiny Session ID
- Source ID
- Begin At
- End At
- Sequence Ports ID

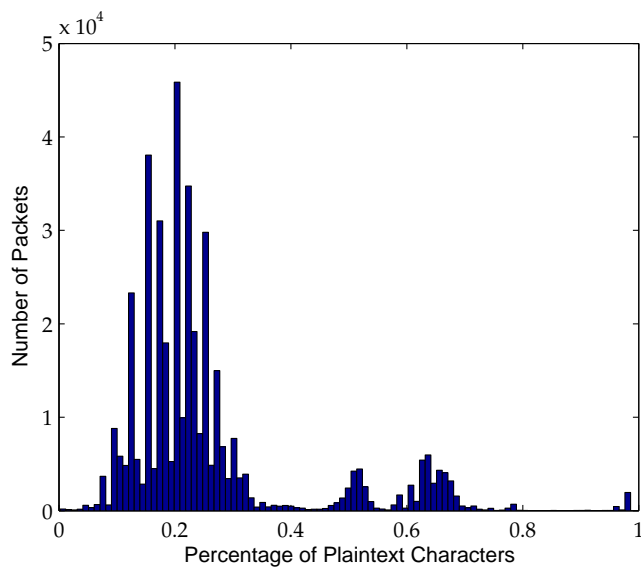
Next we retrieved the packets for each session using the python horasis library and stored them in a file identified by their aggregated session id, and packet id.

Each packet we retrieved from the database consisted of the following:

- Tiny Session ID
- Packet ID
- Timestamp
- Payload

### 6.2.2 Period of 8th - 14th August 2008

We used our encryption detection method to create a histogram displaying the percentage of plain text characters for all packets.



**Figure 6.4:** Histogram of Percentage of Alphabetical Characters for Packets for Period of 8th-14th August 2008

We then examined the packets that had a percentage around the peaks of the histogram to determine the percentage which we could be confident of was encrypted packets, and plaintext packets.

We examined manually the contents of ten packets around the first peak in the histogram:

---

Session ID	Packet ID	Percentage
720013	33765204	15.79%
725519	34007568	13.64%
725519	34007573	23.42%
725519	34007572	39.37%
728321	34161751	26.96%
728321	34161788	20.51%
749437	34918626	28.44%
721856	33805633	33.33%
716194	33503581	22.09%

**Table 6.3:** Packets Examined Around First Peak in Histogram for Period of 8th-14th August 2008

Our examination of these ten packets found they were not plain text packets, and therefore were encrypted packets.

We then manually examined the contents of ten packets around the second peak in the histogram:

Session ID	Packet ID	Percentage
713621	33265594	50.25%
713621	33265672	50.75%
727806	34062121	51.22%
715568	33476685	51.23%
718405	33674536	50.25%
721862	33804596	48.52%
721862	33804608	49.70%
725257	33978314	51.24%
740526	34572799	41.74%
727804	34061180	48.10%

**Table 6.4:** Packets Examined Around Second Peak in Histogram for Period of 8th-14th August 2008

Our examination of these ten packets found that they were network packets being sent in plain text by protocols which also incorporate some non plaintext characters.

We then manually examined the contents of ten packets around the third peak in the histogram:

---

Session ID	Packet ID	Percentage
727806	34062123	61.80%
727810	34061212	63.33%
727817	34061843	62.30%
727819	34062653	65.57%
727831	34062089	65.57%
727848	34061314	66.67%
713613	33266507	68.85%
713616	33265969	68.85%
713616	33265970	67.21%
713632	33266557	67.21%

**Table 6.5:** Packets Examined Around Second Peak in Histogram for Period of 8th-14th August 2008

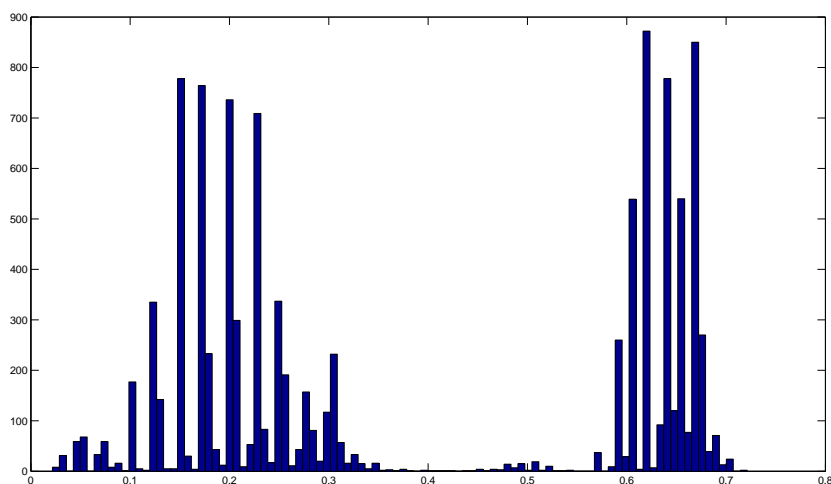
Our examination of these ten packets found that they were network packets which contained mainly plain text with a decreasing number of non plain text characters compared to the previous peak.

We considered packets that use a protocol which is mainly plain text with some non plain text characters to be a plain text packet. As we have examining the peaks in the histogram, we assume any packet above 43% of plain text characters is a plain text packet, and anything below 43% is an encrypted packet.

Hence we now have:

- Plain text packets  $> 43\%$
- Encrypted/non plaintext Packets  $< 43\%$

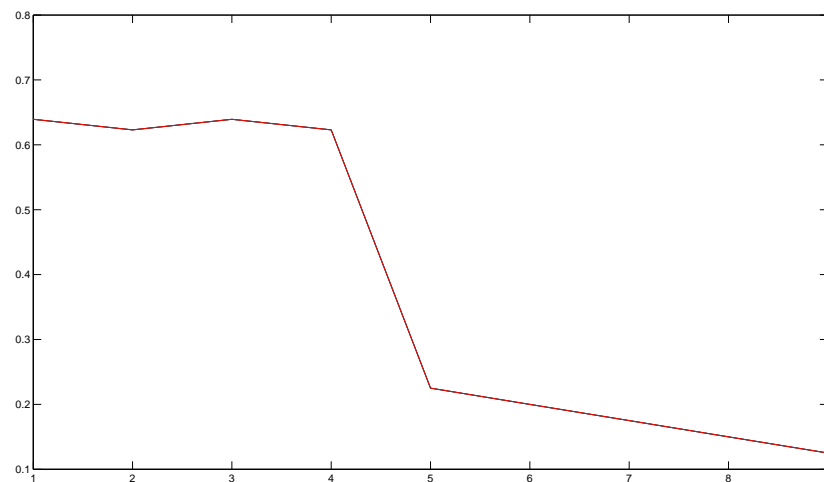
We created a list of all the sessions with the percentage of each packet in the sequence listed in order for each session. Next we determined which sessions had plaintext packets after encrypted packets, by checking if there were any packets with a percentage of over 43% after initial packets with a percentage of list than 43%, We found that there were 697 sessions where plaintext packets occurred after packets which were encrypted. Then we created a histogram of the percentage of plain text characters for the first packet in each session:



**Figure 6.5:** Percentage of Plain text characters for the first packet of each sessions during the period 8th - 14th August 2008

We examined sessions where the percentage of plain text characters was high at the beginning of the session, and close to the maximum percentage given in Figure 6.5. we found three sessions to be of interest as a result of the packet contents. The sessions were number 726715, 726716 and 726717. These sessions were interesting as they all contained the text "hello ???" in the payload of an initial ICMP packet that was the first stage of the session. A normal payload generally contains the letters of the alphabet rather than this text[19]. There is a trojan BHO which uses ICMP packets for communication[20]. The contents of the packets and the regular occurrence of packets which do not match the standard operating system utilities, is indicative that this is caused by an emerging or modified peer-to-peer botnet which sends these ICMP packets containing this string to determine if a system is a potential peer, and is expecting a different response from that of returning the packet payload to it. As after it sends these packets, the sessions then attempt to connect to the system using the SMB protocol.

We found that for all three sessions, they have the same pattern occurring, where the percentage of plain text characters remains above 60% for the first 4 packets, and then declines rapidly as shown in the following graph:



**Figure 6.6:** Sessions 726715, 726716, 726717

We provide a copy of the ICMP stage of one of the sessions below, with some identifying information removed.

```
CID: 34041834 TS: 1218537340.000000
IP 195.5.253.81 -> #####
ICMP type: ECHO code: UNKNOWN
```

```
6865 6c6c 6f20 3f3f 3f          hello ???
```

```
CID: 34041835 TS: 1218537340.000000
IP ##### -> 195.5.253.81
ICMP type: ECHOREPLY code: UNKNOWN
```

```
6865 6c6c 6f20 3f3f 3f          hello ???
```

Further work needs to be done to verify that this is caused by a peer-to-peer bot. If this payload contents is not part of any operating systems ICMP implementation, it could be used to identify a possible attack through connecting using the SMB protocol as occurred in this session before it occurs.

### 6.2.3 Period of 15th - 21st August 2008

We carried out a similar procedure for the second period of 15th - 21st August 2008, as we did for the first period of 8th-14th August 2008.

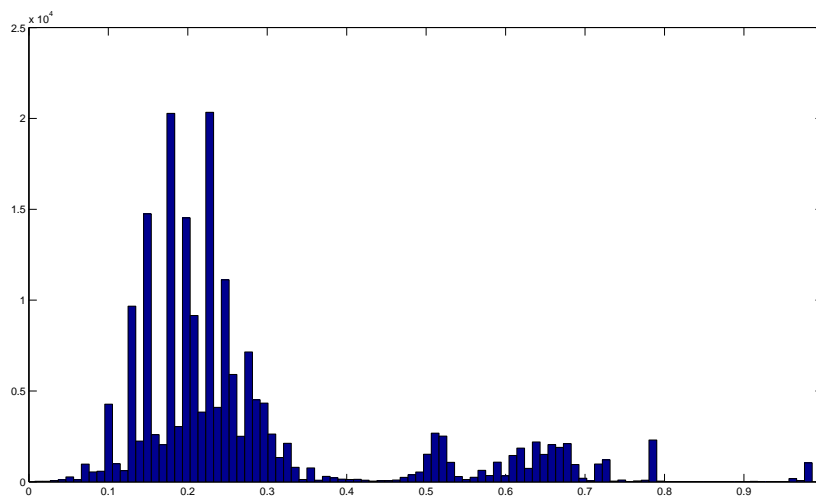
We then used our encryption detection method to create a histogram displaying the

---

percentage of plain text characters for all packets. We examined the peaks of the histogram to determine the percentage which we could be confident of was encrypted packets, and plaintext packets.

We then selected the percentage of plain text characters that are in a packet to be plain text as 50%. Next we go through each aggregated session, and determine if there are encrypted packets before the plain text packets, or not.

We then determined which sessions had plaintext packets after encrypted packets, by checking if there were any packets with a percentage of over 60% after initial packets with a percentage of list than 50



**Figure 6.7:** Histogram of Percentage of Alphabetical Characters for Packets for Period of 15th-21st August 2008

We examined manually the contents of ten packets around the first peak in the histogram:

Session ID	Packet ID	Percentage
732013	34280644	23.39%
732013	34280645	23.39%
735284	34326903	22.22%
735284	34326964	24.73%
735285	34326964	20.51%
735286	34326999	18.42%
735290	34321835	22.50%
737185	34409869	22.50%
737185	34409867	20.25%
739259	34509923	21.43%

**Table 6.6:** Packets Examined Around First Peak in Histogram for Period of 15th-21st August 2008

Our examination of these ten packets found they were not plain text packets, and therefore were encrypted packets.

We then manually examined the contents of ten packets around the second peak in the histogram:

Session ID	Packet ID	Percentage
739259	34509937	48.41%
739259	34509973	51.24%
739259	34510003	52.24%
739521	34556060	50.74%
739521	34556086	50.74%
747174	34776173	52.73%
747174	34781174	53.14%
733282	34305418	50.25%
733282	34305431	50.25%
733283	34034559	51.2%

**Table 6.7:** Packets Examined Around Second Peak in Histogram for Period of 15th - 21st August 2008

Our examination of these ten packets found that they were network packets which were either potential non plain text data or plain text data.

We then manually examined the contents of ten packets around the third peak in the histogram: We examined these to determine if the payload contents is encrypted or plain text.

---

Session ID	Packet ID	Percentage
731996	34279904	63.93%
731996	34279906	63.93%
732003	34279959	65.57%
732011	34280574	63.93%
732013	34280640	68.42%
732015	34280003	65.57%
732016	34279997	67.21%
732029	34280564	66.10%
732030	34280576	63.93%
735284	34326878	65.57%

**Table 6.8:** Packets Examined Around Third Peak in Histogram for Period of 15th-21st August 2008

Our examination of these ten packets found that they were network packets which were the standard operating system implementation of ICMP packets, and were mainly plain text.

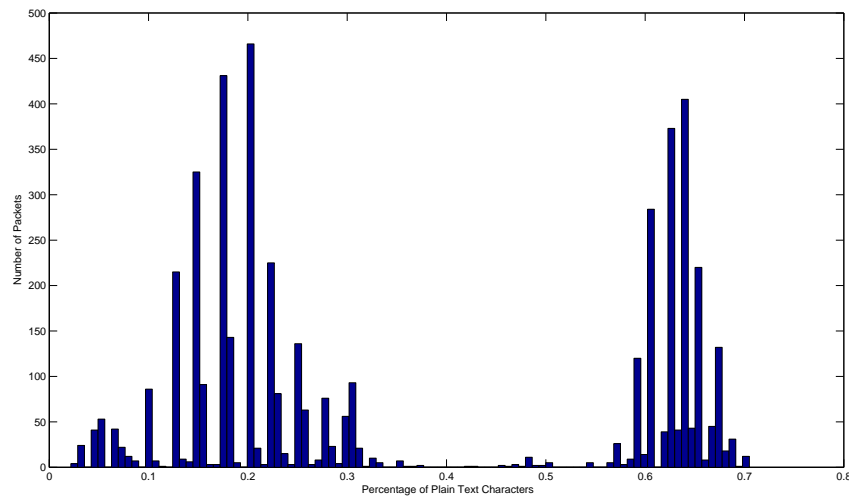
We considered packets that use a protocol which is mainly plain text with some non plain text characters to be a plain text packet. As we have examining the peaks in the histogram, we will assume any packet above 60% of plain text characters is a plain text packet, and anything below 50% is an encrypted packet. Anything between 50% and 60% could either be a plain text or non plain text packet.

Hence we now have:

- Plain text packets  $> 60\%$
- Possible Plain Text or encrypted packets  $> 50\%$  and  $< 60\%$
- Non Plain Text Packets  $< 50\%$

We created a list of all the sessions with the percentage of each packet in the sequence listed in order for each session.

We found that there were 320 sessions where plaintext packets occurred after packets which were encrypted. We created a histogram of the percentage of plain text characters for the first packet in each session:



**Figure 6.8:** Percentage of Plain text characters for the first packet of each sessions during the period 15th - 21st August 2008

Most of the first packets at the beginning of each session are encrypted as shown in the histogram above. However the majority of these sessions do not have plain text data occurring after the encrypted data.

Although subject to the usual problems associated with not being able to differentiate between encrypted and binary data, no obvious systematic effects on the results were observed.

We examined the sessions where the plaintext packets occurred after encrypted packets, and found that some of the sessions, had packets which indicated that the system could be checking for potential peers. As it is not possible for us to discuss the contents of all these sessions we select one of them to examine.

We examine the extract of the beginning of session 737250, and create a plot of the percentage of plaintext for the packets that make up the session:

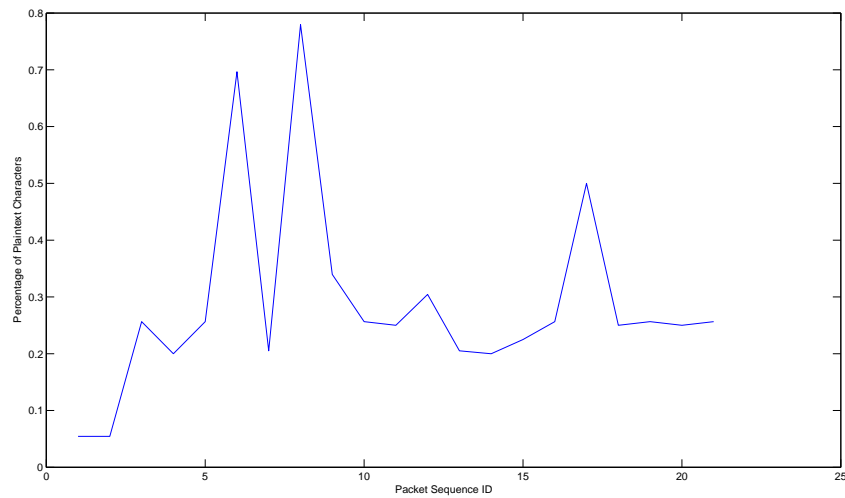
---

```

E < 2Sp P$f P
E < @UpS P8$gPtO
E 4 2Sp P$g9P<
E G 2Sp P$g9PT HEAD / HTTP/1.0
E G 2Sp P$g9PT HEAD / HTTP/1.0
E < @UpS P8$gPtO
E G 2Sp P$g9PT HEAD / HTTP/1.0
E @ 2Sp P$z9P)
E G 2Sp P$g9PT HEAD / HTTP/1.0
E < @UpS P8$gPtO
E @ 2Sp P$z9P)
E 4 2Sp P$z9P(
E G 2Sp P$g9PT HEAD / HTTP/1.0
E < @UpS P8$gPtO
E @ 2Sp P${9P(
E 4 @xpS P9$zP5
E 4 2Sp P$z9P(
E 4l -pS P9$zPD]F
E 4 2Sp P${9P(
E 4m ,pS P9$zPD]F
E 4n +pS P9$zPD]F
E 4o *pS P9$zPD]F
E 4p )pS P9${PD]F
E )q 3pS P9${PD]! HTTP/1.1 200 OK
Server: Microsoft-IIS/5.0
Date: Wed, 30 Jan 2008 16:19:43 GMT
Connection: Keep-Alive
Content-Length: 1270
Content-Type: text/html
Set-Cookie: ASPSESSIONIDQGGQGLRU=FJPPMBFAMHAMEFNHFJJADIAA; path=/
Cache-control: private

```

**Figure 6.9:** Contents of Payload of Packets for Session 737250



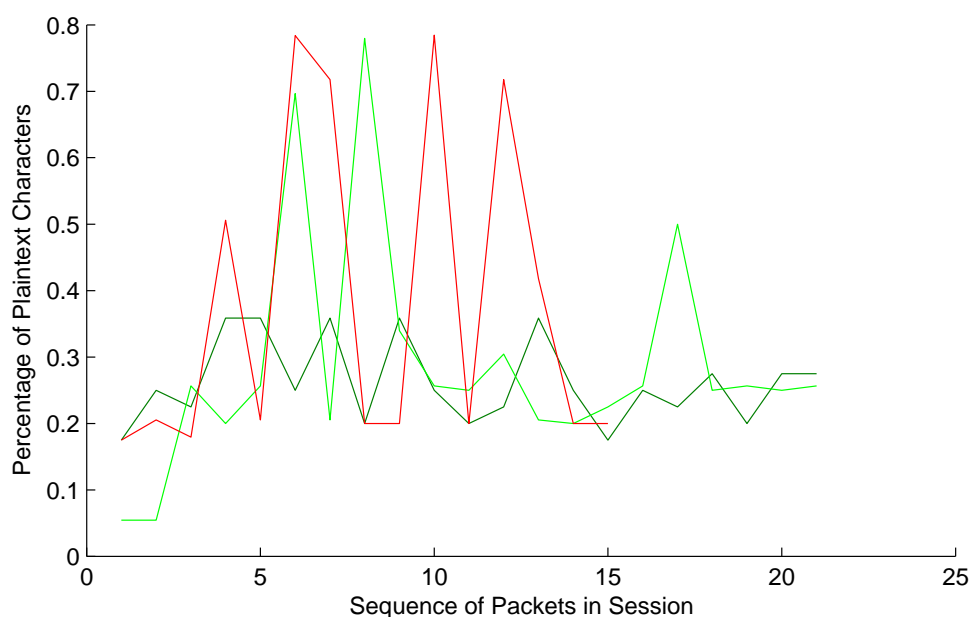
**Figure 6.10:** Sessions 737250

The has in the initial packets shown at the beginning of the above, several ICMP packets with encrypted binary data, and several requests to ports on the system. After no response is sent to these requests, the attacker then connects to the HTTP server which is running on the system.

We found a total of 10 sessions in which similar behaviour to Session 737250 occurs being sessions:

- 737250
- 737824
- 735097
- 732049
- 732413
- 733044
- 734066
- 735088
- 736220
- 737199

We created a graph of three of the sessions which we found to have similar behaviour:



**Figure 6.11:** Combined Graph of Sessions

All sessions shown in the graph, start initially with encrypted packets of a similar type, and then proceed on to a combination of plaintext and encrypted packets with the percentage of plaintext characters varying between sessions. We observed a similar pattern where an encrypted packet is sent before a plain text packet, and another encrypted packet is sent.

This behaviour is indicative of a peer to peer botnet attempting to check for peers before attempting to inject the computer with the peer to peer bot. As a result of current data connection mechanisms used by the SGNNet honeypot project, we are currently unable to confirm if it was a peer to peer bot, as it was not downloaded successfully by the honeypot project.

### 6.3 Method for detecting peer-to-peer bots where they communicate using ICMP packets

There is nothing to prevent a bot running on a system intercepting an ICMP packet, and replying with different data. The ping application does not check to see if the data in the reply is the same other than that it is an ECHO REPLY packet. This reply to a packet could be modified by a malicious bot, and used for communication similar to what the BHO Trojan used.

We propose a method of detecting peer-to-peer botnets which communicate using

ICMP packets, by using Levenshtein distance to find the distance between the standard operating system implementation of ping and other ICMP packets.

### 6.3.1 SGNet Data

We used a random sample of fifty sessions from SGNet for the period of 8th - 21st August 2008 where encrypted packets occurred before plaintext packets at the beginning of the session, and used only ICMP packets from those sessions.

The initial information we retrieved about the sessions from the Tiny\_Session table consisted of the following attributes:

- Tiny Session ID
- Source ID
- Begin At
- End At
- Sequence Ports ID

Next we retrieved the packets for each session using the python horasis library and stored them in a file identified by their aggregated session id, and packet id.

Each packet we retrieved from the database consisted of the following:

- Tiny Session ID
- Packet ID
- Timestamp
- Payload

ICMP packets are commonly used for detecting if a host with a particular IP address is available. This is done through sending an ECHO ICMP packet, and if it is available the host replies using an ECHO REPLY ICMP packet. Using the previous method we found that ICMP packets which match the standard operating system implementation of ping are around the peak of 65% for the period of 8th - 21st August 2008, and that ICMP packets which contain data which does not match the standard operating system implementation of ping are around 20%.

### 6.3.2 Standard ICMP Packet contents

To be able to determine if the ICMP data is sent by the standard operating system utilities, we captured the payload of standard ICMP packets by running the ping utility on the Windows XP, Mac OS X and Linux operating systems, and using tcpdump and Wireshark.

We ran the ping utility on Windows XP, and captured the following payload of an ECHO packet:

```
..k F... ..5...E.
.<%T.... ..{...
G....\.. A.abcdef
ghijklmn opqrstuv
wabcdefg hi
```

We captured the following payload of the ECHO REPLY packet in response to the ECHO packet we had sent.

```
....5... k F...E.
.<'...-. .#..G...
{....\.. A.abcdef
ghijklmn opqrstuv
wabcdefg hi
```

We then ran ping on Mac OS X and captured the following payload ECHO packet:

```
..k F... ..$.E.
.T....@. .*...{..C
..... ..B\..HJ.
.....
..... .. !"#$$%
&' () *+, - ./012345
67
```

We captured the following payload of an ECHO REPLY packet.

```
....$. k F...E.
.T@2... e..C....
{..... ..B\..HJ.
.....
..... .. !"#$$%
&' () *+, - ./012345
67
```

We then ran the ping utility on Linux and captured the following payload of an ECHO packet:

```
E..T..@.@.<.....D.....b.H.....
!"#$$%&' () *+, - ./012345
```

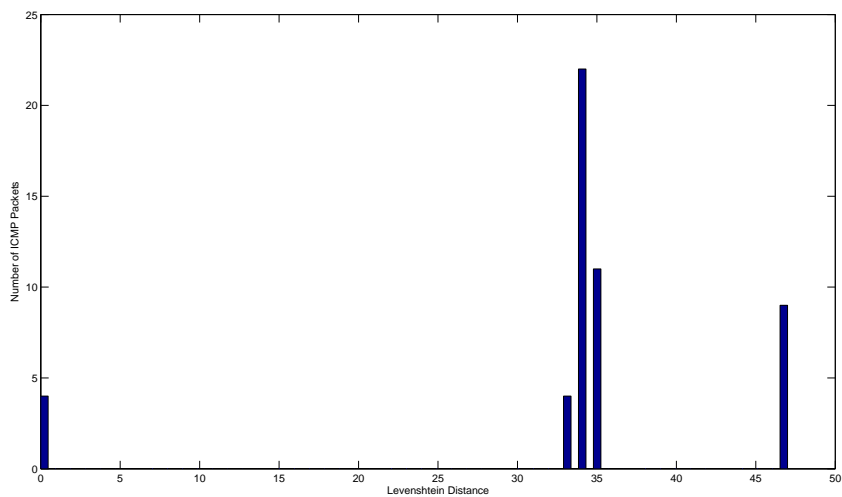
We received the following payload for the ECHO REPLY Packet

```
E..T) ...@.S.....L.....b.H.....
!"#$$%&' () *+, - ./012345
```

### 6.3.3 Period of 8th - 21st August 2008

We used levenshtein distance to calculate the similarity measure between the ICMP packet, and the standard operating system implementation. For our levenshtein distance we use the standard ascii character set.

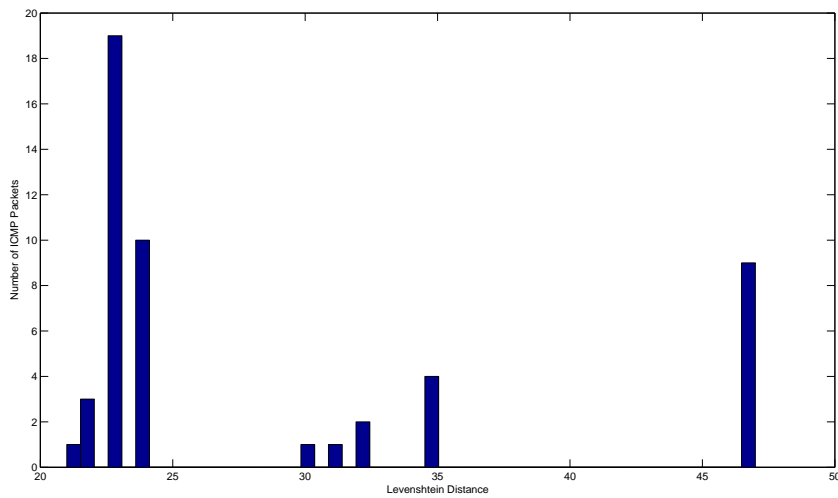
First we calculated the Levenshtein distance between the ICMP packets sent by the Windows XP implementation of Ping which sends ICMP Echo packets, and the ICMP ECHO packets collected by the SGNet Honeypot project. We created the following histogram showing the number of occurrences of packets with the same levenshtein distance:



**Figure 6.12:** Histogram of Levenshtein Distance against Windows XP ICMP Implementation

By examining the histogram for the windows XP ICMP implementation, we found that four ICMP packets used the standard operating system implementation, and forty-six ICMP packets did not use the standard operating system implementation. Using the method we developed previously in this work, we found that the contents of the ICMP packets with the non standard operating system implementation for windows xp are encrypted.

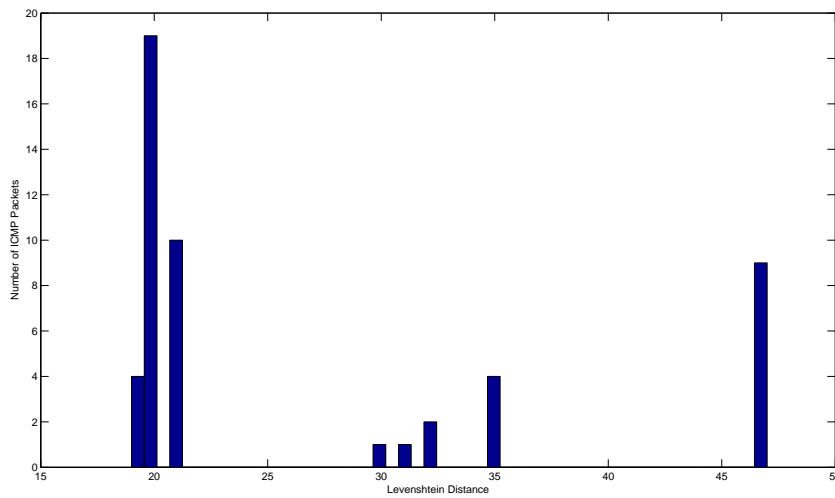
Then we calculated the Levenshtein distance between the ICMP packets sent by the Mac OS X implementation of Ping which sends ICMP Echo packets, and the ICMP ECHO packets collected by the SGNet Honeypot project. We created the following histogram showing the number of occurrences of packets with the same levenshtein distance:



**Figure 6.13:** Histogram of Levenshtein Distance against Mac OS X ICMP Implementation

By examining the histogram above, we found that fifty ICMP packets did not use the standard Mac OS X operating system implementation. Using the method we developed previously in this work, we found that the contents of the ICMP packets with the non standard operating system implementation are encrypted, except for the packets that matched the windows XP Operating System Implementation.

Then we calculated the Levenshtein distance between the ICMP packets sent by the Linux implementation of Ping which sends ICMP Echo packets, and the ICMP ECHO packets collected by the SGNet Honeypot project. We created the following histogram showing the number of occurrences of packets with the same levenshtein distance:



**Figure 6.14:** Histogram of Levenshtein Distance against Linux ICMP Implementation

By examining the histogram above, we found that fifty ICMP packets did not use the standard Linux operating system implementation. Using the method we developed previously in this work, we found that the contents of the ICMP packets with the non standard operating system implementation are encrypted, except for the packets that matched the windows XP Operating System Implementation.

As a result of the analysis of the levenshtein distance, and the previous method proposed, we found these sessions could use ICMP with an encrypted payload to determine if a system is a peer. Our analysis of the ability for peer-to-peer botnets to communicate with ICMP indicate these packets are sent by peer to peer botnets, however as a result of there currently being limited research around peer-to-peer botnets, it is not possible for us to determine if these packets are being sent by peer to peer botnets.

## 6.4 Discussion

The main difficulty with peer-to-peer botnets is the detection of potential peer-to-peer botnet traffic. This difficulty occurs as a result of peer-to-peer botnets commonly using encrypted network traffic for its communication, and their decentralised control structure.

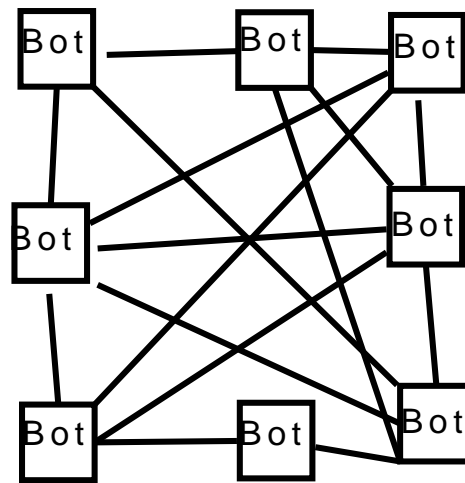


Figure 6.15: Diagram of Peer-to-Peer Botnet Topology

Whereas with a IRC botnet there is a centralised point of control being an IRC server or IRC channel. With IRC botnets, if you find the centralised point of control, you can then quickly locate all IRC bots which are connecting to the IRC server. This is not possible with peer to peer botnets, as they lack a central server.

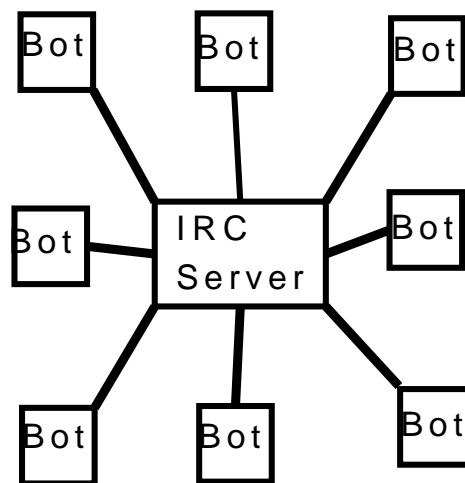


Figure 6.16: Diagram of IRC Botnet Topology

In this work, we have examined two methods that we proposed which can be used to detect potential peer-to-peer botnets that behave in a certain method. As a result of peer-to-peer botnets being a new research area, and that there are no current methods to determine if a peer-to-peer botnet is sending network traffic, we have not been able to determine which sessions our method found were as a result of a peer-to-peer bot. The methods we proposed, can detect traffic which could potentially be from a peer to peer botnet, the traffic can then be investigated further.

---

We have made progress through our experiments, and gaining further understand of peer-to-peer botnets towards our goal of being able to detect network traffic from new peer to peer botnets. Future work involving simulation of peer to peer botnets, or capturing traffic between systems we know to be infected with peer to peer botnets will allow for further progress to be made. The goal of being able to detect peer-to-peer botnets is longterm, and a long process of research is required to achieve this goal.



## Conclusions

---

There is currently a long way to go in research in peer to peer bots and peer to peer botnets, as they are relatively new with the storm peer-to-peer botnet being introduced in 2007. This work has contributed two methods that can be used to detect Peer to Peer bots which comprise a part of a peer to peer botnet.

We contributed a method of detecting communication between potential peer to peer bots, before they attempt to infect a system, when the peer to peer bot sends encrypted network packets to check if the system is already infected. The method cannot detect all peer to peer botnets, as a result of their varying behaviours. It can detect peer to peer botnets which behave in the way we proposed.

Also we contributed a second method of detecting potential peer to peer bots using levenshtein distance, where they use code for sending ICMP ECHO packets which is a not a standard operating system implementation, and where they use ICMP packets for the purpose of communication.

The work done in this thesis, is only the beginning of research into the detection of peer to peer botnets, and the future development of proactive mechanisms to defend against this threat to computer and information security. There is currently no single method which can be used to detect all peer-to-peer botnets, due to the way they communicate being different. In this work, we have made progress towards the goal of being able to detect new peer-to-peer botnets as they come into existence.

We developed an understanding of what needs to be done for future experiments to improve the quality of data available regarding peer-to-peer botnets. Most new peer-to-peer bots use websites, or email to propagate rather than injecting a computer system with the bot, so a user having to run the initial installation package. As a result of this, honeypots need to make changes to how they capture data, in order to be able to capture better quality data relating to peer-to-peer botnets.

## 7.1 Future Work

Until recently, there has been little research done relating to peer-to-peer botnets, even though they are becoming an increasing threat to the security of computer networks and systems. This is mainly as a result of development of malicious bots that use peer to peer protocols beginning recently. Most research we found relating to botnets, focused on IRC botnets where there was a centralised point of control. Through our indications examining the honeypot data we had, we found that it was not able to collect sufficient data regarding peer-to-peer botnets.

There is a wide range of future work that needs to be done on peer-to-peer botnets. An area worth carrying out future work on is improving methods of detecting potential peer-to-peer botnets, and methods to collect data about peer-to-peer botnets and their future behaviour. This work when done, will allow us to be able to have a greater understanding of what data needs to be collected, and will allow network security mechanisms to be able to respond proactively to this new threat.

Additional future work includes automating the selection of the peaks which are used to determine whether network packets are encrypted or plain text, and to improve the method so that it can differentiate between binary data which is not encrypted such as executable files, and binary data.

Research is just beginning into peer to peer botnets, and it is an interesting and useful area in which to be conducting research.

## 7.2 What I Learnt

Throughout the project, I learnt a range of skills and found it to be a valuable learning experience. I gained skills in conducting research, and as a result learnt research can be interesting, yet time consuming, and that it is not easy to come up with new ideas that are successful. My project management and time management skills were further improved. Overall I found completion of this project to be an extremely valuable process, despite any difficulties encountered as a result of conducting research in a relatively new area, and given the short time frame for the project.

---

# References

---

- [1] T. Peng, C. Leckie, and K. Ramamohanarao, "Survey of network-based defense mechanisms countering the dos and ddos problems," *ACM Comput. Surv.*, vol. 39, no. 1, p. 3, 2007.
- [2] I. Mokube and M. Adams, "Honeypots: concepts, approaches, and challenges," in *ACM-SE 45: Proceedings of the 45th annual southeast regional conference*, (New York, NY, USA), pp. 321–326, ACM, 2007.
- [3] C. Leita and M. Dacier, "Sgnet: A worldwide deployable framework to support the analysis of malware threat models," *edcc-7*, vol. 0, pp. 99–109, 2008.
- [4] J. Postel, "Rfc 792 - internet control message protocol," 1981. from <http://tools.ietf.org/html/rfc792>.
- [5] J. B. Grizzard, V. Sharma, C. Nunnery, B. B. Kang, and D. Dagon, "Peer-to-peer botnets: overview and case study," in *HotBots'07: Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets*, (Berkeley, CA, USA), pp. 1–1, USENIX Association, 2007.
- [6] D. Wallach, "A survey of peer-to-peer security issues," *Software Security Theories and Systems*, pp. 253–258, 2003.
- [7] M. Lesk, "The new front line: Estonia under cyberassault," *Security & Privacy, IEEE*, vol. 5, pp. 76–79, July-Aug. 2007.
- [8] P. Wang, S. Sparks, and C. C. Zou, "An advanced hybrid peer-to-peer botnet," *IEEE Transactions on Dependable and Secure Computing*, 2008. to be published.
- [9] D. Wallach, "A survey of peer-to-peer security issues," *Software Security Theories and Systems*, pp. 253–258, 2003.
- [10] R. McGrew, "Experiences with honeypot systems: Development, deployment, and analysis," in *System Sciences, 2006. HICSS '06. Proceedings of the 39th Annual Hawaii International Conference on*, vol. 9, pp. 220a–220a, Jan. 2006.
- [11] L. Spitzner, *Honeypots: Tracking Hackers*. Addison-Welsey, Boston, 2002.
- [12] C. Leita, K. Mermoud, and M. Dacier, "Scriptgen: an automated script generation tool for honeyd," in *ACSAC '05: Proceedings of the 21st Annual Computer Security Applications Conference*, (Washington, DC, USA), pp. 203–214, IEEE Computer Society, 2005.
- [13] C. Leita, M. Dacier, and F. Massicotte, "Automatic handling of protocol dependencies and reaction to 0-day attacks with scriptgen based honeypots," *Recent Advances in Intrusion Detection*, pp. 185–205, 2006.
- [14] G. Portokalidis, A. Slowinska, and H. Bos, "Argos: an emulator for fingerprinting zero-day attacks for advertised honeypots with automatic signature generation," in *EuroSys '06: Proceedings of the 1st ACM SIGOPS/EuroSys European Conference on*

*Computer Systems 2006*, (New York, NY, USA), pp. 15–27, ACM, 2006.

- [15] Wikipedia, "Levenshtein distance," September 2008. from [http://en.wikipedia.org/wiki/Levenshtein<sub>d</sub>istance](http://en.wikipedia.org/wiki/Levenshtein_distance).
- [16] J. Stewart, "Phatbot trojan analysis," 2004. from <http://www.secureworks.com/research/threats/phatbot>.
- [17] J. Stewart, "Sinit p2p trojan analysis," 2003. from <http://www.secureworks.com/research/threats/sinit>.
- [18] R. Lemos, "Bot software looks to improve peerage," May 2006. from <http://www.securityfocus.com/news/11390>.
- [19] Wikipedia, "Ping," October 2008. from <http://en.wikipedia.org/wiki/Pings>.
- [20] "Data stolen via icmp - security labs alert," July 2006. from <http://securitylabs.websense.com/content/Alerts/1178.aspx>.