# Botnet Detection by Correlation Analysis

André Orvalho

Kongens Lyngby 2012
IMM-M.Sc.-2012-53

Technical University of Denmark
Informatics and Mathematical Modelling
Building 321, DK-2800 Kongens Lyngby, Denmark
Phone +45 45253351, Fax +45 45882673
reception@imm.dtu.dk
www.imm.dtu.dk IMM-M.Sc.-2012-53

# Summary (English)

When a bot master uses a control and commander (C&C) mechanism to assemble a large number of bots, infecting them by using well known vulnerabilities, it forms a botnet. Botnets can vary in C&C architecture (Centralized C&C or P2P are the most common), communication protocols used (IRC, HTTP or others like P2P) and observable botnet activities. They are nowadays one of the largest threats on cyber security and it is very important to specify the different characteristics of botnets in order to detect them, the same way a hunter needs to know its prey before preparing methods to catch it. There are 2 important places to look for botnet activity: The network and the infected host.

This project intends to present a study that correlates the behavior on the network with the behavior on the host in order to help detection, studies like [SLWL07] (based on network behavior) and [SM07] (based on host behavior) are two good start points to help on the research. The choice of the architecture was done by looking at the botnet characteristics especially the capacity of changing and evolving which makes methods for detection by misuse obsolete. The system is designed to first look at 4 features of system calls on the host side: First which system call it is, second the name of the application using the system call, third the time between this system call and the last system call and for last the sequence of the past three system calls. A technique of unsupervised learning (the K-means algorithm) will be used to calculate the values for the threshold using an unclassified training set. when on the real world the collection is used to calculate the values to compare with the threshold. If it passes the threshold than the necessary information is passed to the network evaluation block. On the network side and before receiving any data from the host side, it will calculate the threshold for the flows given on the training set. When using the data from the host to narrow down the number of flows to look at, it verifies if their values pass the threshold. The feature used to calculate the threshold is the time between flows. If the network finds flows that pass the threshold for the network evaluation block than it will emit reports and alarms to the user.

The small experiences done show some promising signs for use on the real world even though a lot more further testing is needed especially on the network bit. The prototype shows some limitations that can be overcome by further testing and using other techniques to evolve the prototype.

# Preface

This thesis was prepared at the department of Informatics and Mathematical Modeling at the Technical University of Denmark, under supervision of Professor Robin Sharp and Sead Muftic, professor at the School of information and communication technology from the Royal Institute of Technology in Sweden. It is included on the Erasmus Mundus program in fulfillment of the requirements for acquiring a M.Sc. in security and mobile computing. It presents the efforts of a six month study on the subject.

It deals with the problem of botnet detection co-relating information both from the host and from the network. Even though it is a very present and relatively big problem, researchers haven't been able to find an acclaimed solution for its detection and prevention. Research on the area is undergoing in many different universities in many different countries, trying to advance with a solution that can give a minimal guarantee of success.

The choice of the subject was done based on the intereste in security and the challenging nature of the problem. The security breach provoked by such a threat is hard to quantify but it is well known to be very present and troublesome. It is a problem with a lot of variants to look at and the available solutions do not look to be sufficient. The ability of contributing to the efforts to solve this problem gives a sentiment of served duty to the author. Hopefully this will help further work in the right direction, might the solution provided not be the most suitable one.

The thesis consists of the idea to a prototype to help detection of botnets based on the hypothesis that good behavior and bad behavior have different patterns either on the host and the network and that those patterns are co-related. This was done by studying the behavior at the host and then using that information to correlate with what's happening on the network, trying to find evidence of malicious behavior on the system to alert the system administrator allowing it to take the necessary measures.

Lyngby, 30-June-2012

André Orvalho

# Acknowledgements

I would like to thank both my supervisors for the support and counseling during the produced work. I would also like to thank all my colleagues at both schools for the good group work and challenging opportunities to work with so many different people and so many different brilliant minds. To finalize I would like to thank my parents for all the support given my whole life until now and the good preparation for my next step.

# Contents

Chapter 1

# Introduction

A bot is a host which is controlled from another location. A bot master is a host which controls others from afar. When a bot master uses a control and commander (C&C) mechanism to assemble a large number of bots, infecting them by using well known vulnerabilities, it forms a botnet[SLWL07]. Normally the attack is performed by first infecting the host. For example using an email with an image, that when it's rendered, installs the botnet code on the host. Once the host is running, the botnet code will induce the host to connect to a Rendezvous point, in order to communicate with the bot master. Then and through the Rendezvous point, the bot master will give orders to the host. These orders can include the request of all the information collected by the zombie or to send more SPAM to its network in order to infect other hosts. This process can differ considering the botnet architecture and its purpose. Figure 1 represents the typical process of infection and contact to perform attacks.

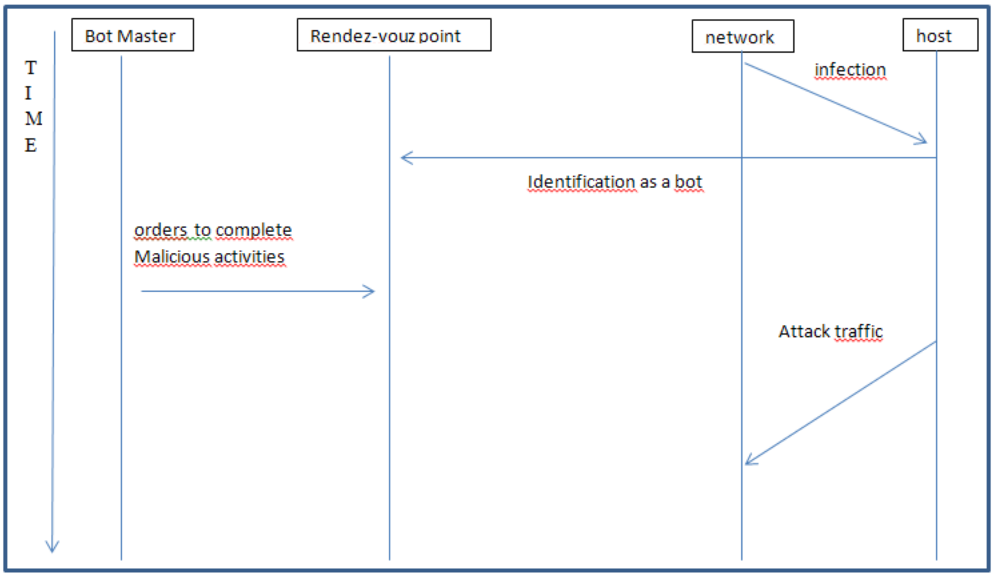


Figure 1 - infection and attack – example of one possible communication protocol

Botnets can vary in C&C architecture (Centralized C&C or P2P are the most common), communication protocols used (IRC, HTTP or others like P2P) and observable botnet activities. Classifications on which attacks they perform or the evasion techniques used by botnets can also be done, but they are out of scope of the problem[Mic06].Those characteristics have been target of studies and are often used to implement techniques helpful to the detection of botnets.

This is one of the largest threats on cyber security nowadays since these mechanisms are one of the biggest forces used for distributed DOS (Denial-o-Service) attacks, malware/spam distribution and phishing/identify theft[Mic06]. Those threats figure on the top of attacks found online. The biggest problem to detect and prevent these kinds of attacks has to do with the mechanism used. The attacker is separated from its victim by a layer of other hosts, creating a level of indirection, consequentially becoming very hard to detect where the attack comes from[SLWL07]. The amount of power of the botnet is directly related to its size, as more hosts, bigger bandwidth will be at its disposal and of course increasing the reachability of the attacker.

Normally the detection of the C&C structure would solve the problem on a centralized architecture. But first the level of indirection makes it very hard to identify the machine or machines which the orders are being sent through. And the growing usage of new architectures like P2P makes it even harder to find it. In these architectures if one C&C mechanism is destroyed does not mean that the whole bot has been deactivated. Also if a bot uses a protocol like HTTP to communicate, it turns the task of detecting its activity much harder once the biggest percentage of traffic on the network belongs to that protocol. Time is one of the keys for detection. In Almost every bot the execution of its activities is timed correlated, which means, every X minutes/hours the same behavior can be observed. The problem is most of the times the botnet code is not provided, which makes it harder to know how far to look back, turning this choice into something like looking for a needle on a haystack.

There is 2 important places to look for botnet activity: The network and the infected host. On the network one can look at the content or at the flow characteristics. Looking at the contents it is a long shot since most of them will be encrypted. But looking at the flow characteristics is not an easy task either, since they are too many and a decision must be made to find which ones show more relevant evidence of botnet activity. On the host itself, the search must be made looking at the processes behavior registering features like which programs can be used by the infected host to perform the botnet code(examples: IRC client or browser) and which kind of system calls are done by these programs that can be suspicious and be under control of botnet code.

The goal is to present a study that correlates the behavior on the network with the behavior on the host in order to help detection, but until now the author could not find any work that tried to do this. One research could be found that could use information from any of those two types of behavior [OKA10], but either only network or either only host based. Here the author presents a tool that studies communities through a given model. This model is defined using a few chosen parameters which are used to train the algorithm to monitor the activity. These parameters can either be host or network based. The study uses the average of those parameters values on a community (thus avoiding noisy individual applications) to verify if passes a certain threshold which is provided on the model as what is acceptable. Once it is detected hosts values pass the given threshold they are classified as possible to show botnet evidence and measures are taken.

On studies about detection using network based behavior, the most significant would be [SLWL07], where the author uses as an hypothesis if it's possible to find botnet evidence only using the traffic flows and not its content. First it applies a sequence of filters and after tries to classify them using machine learning algorithms, concluding it does not give any advantage, thus bypassing the classification stage. Next he will correlate the flows in pairs, finding the most correlated ones and using it to the final stage to apply some social engineering in order to detect how the botnet is acting. There is also other relevant works such as [KRH07], which tries to find a methodology to track botnets on tier-1 ISP's. Using what the author calls a scalable non-intrusive algorithm, follows some steps to achieve the goal: first it will detect hosts with suspicious behavior like scanning ports, emailing spam or DDoS traffic generation. Second uses details from the packets header to correlate them in the same control conversations (equal IP source and destination, ports, etc...). After it calculates which ones have a stronger correlation in order to release alarms and/or reports. The article [GPZL07] is based on the assumption that bots belonging to the same botnet will likely have the same kind of behavior and show a spatial-temporal correlation. Here the flows are separated by who is talking to whom and who is doing what. Aggregating flows with the same packet header parameters and using an ad-hoc algorithm to correlate the flows in the end. Using the same assumption, [GZL08] will use the information on statistical algorithms to detect botnets with theoretically bounds for false positives (FP) and false negatives (FN) rates.

About detection concerning the usage of the host based behavior information fewer examples can be found. On [SM07], the author uses as bottom line that processes which use parameters that come from the network to its system calls, are likely to show remote control. Without relying on any botnet structure or C&C protocol it uses techniques to show that by looking at those system calls receiving tainted arguments it can achieve acceptable results with low FP and FN rates. The author in [HFS98], uses temporal models to represent normal user behavior. These models are built by monitoring special processes and then using a specification language to write the special privileges for each process. These temporal models can then be used to distinguish normal behavior from the malicious one.

There is other methods for detection of botnets and there is some work done. For example, monitoring the IRC traffic at the standard port and analyzing the payload for known botnet commands is what is done at [CJD05]. For the work done on [RFD06], the author bases himself on assumption that the bot masters will look up on the DNS black lists trying to find its bots. Using heuristics based on possible techniques used by the bot masters to do the lookups it will try to distinguish legit lookups from the ones done by bots.

The present work is structured the following way: On chapter 2, taxonomy of botnets is provided. Chapter 3 Describes the Design of the solution for the given problem. The work developed on the host side behavior is described on chapter 4 and chapter 5 will describe what is done on the network side. Finally chapter 6 will give a description how these behaviors are correlated and chapter 7 will present the conclusions, discussion and further work.

Botnets are serious and complex threats, hard to detect and very resilient. Achieving a solution that can detect and help solving this problem might also require complex and elaborated tools. The goal of this project is to find a way to correlate behaviors in order to help on detection even though no published work could be found using the same technique.

Chapter 2

# Taxonomy of botnets

Botnets are new-adopting, trendy and fast-evolving targets. Their features vary from the way they propagate, passing through the C&C architecture and communication mechanism to the way they perform attacks and which activities we can observe from it[BCJX09]. The knowledge base about these threats is still being populated and there is not a perfect understanding of the whole problem due to the reasons stated above.[Mic06]

It's very important to specify the different characteristics of botnets in order to detect them, the same way a hunter needs to know its prey before preparing methods to catch it. First because botnets are complex structures: They can spread as easy as a worm, prevent detection like a virus and include different attacks as it was a toolkit.[CJD05] Second a strategy is needed to know where to look, accounting botnets are so complex, there is many different ways and it's not feasible to try every option.

Given the scope of the problem, the chapter presents specifications of the Botnets variations according to C&C architecture, communication protocols used and observable botnet activities.

## 2.1 C&C Architecture

The C&C architecture relates to the way the botnet structures the communication between the bot master and its zombies, basically defining the topological model the botnet is using.

This characteristic is as important to understand as C&C's are to botnets. The C&C is believed to be the bottleneck of the botnet[Mic06]. Most of the times and depending on the architecture used, If an IDS(Intrusion detection system) is able to find the server or servers used by the bot master to make is C&C structure run and break it, it takes down the communication link between the bot master and a large number of bots. Without that communication link the botnet is not able to launch attacks in a bigger scale, since its power is directly connected to its size(number of bots available). This makes the C&C structure essential to the botnet.

Moreover the C&C mechanism is very unique and unlikely to change on every different bot and its variants, yet it's important to denote, that new architectures can emerge which have never been seen or that some of these architectures can be implemented with certain tweaks to complicate detection.

Three different C&C architectures have been identified: Centralized, P2P and random model.[Mic06]

### 2.1.1 Centralized Architecture

As the name indicates, the bot master choses a single central host to be the point, where every zombie connects to, when infected. This host is important to have high bandwidth since all communications pass through him. It can be the bot master itself but it would lose the advantage of indirection on the communications. Usually should be a compromised host[Mic06], even though there are open IRC servers, for example, that can be used.

After the infection and once it communicated with the central host, the zombie will be redirected to the rendezvous point. There the master will give the orders to the bots, to launch attacks, infect other hosts or simply return logs of information retained from that host. A representation of this architecture is given on Figure 2.

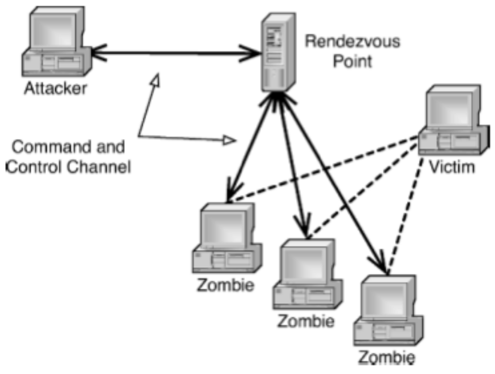


Figure 2 - Representation of a centralized architecture on an IRC-based botnet[SLWL07]

This architecture still has high survivability on the real world since there is not great countermeasures against botnets. Even though sys admins are more aware of this problem and little mechanisms like captcha's and black lists are being used to prevent bots from attacking certain systems, not all of them use it and most are still vulnerable. The existence of many options on tools to implement and customize such a bot make it very easy to setup and because of these reasons, this architecture is still the most prevailing on the internet until today.

It has other advantages like the small messaging latency which makes it much easier to coordinate and launch the attacks. The major drawback is that here there is only one C&C structure being the only source of connection between the bot master and the zombies. So once found and deactivated the bot master has no way of contacting the botnet and that makes the botnet useless. Very Known bots use this architecture such as AgoBot and SDBot.

### 2.1.2 P2P architecture

On this architecture the bot master organizes the bots and the C&C's as if it was P2P network. It can either be structured or unstructured. [DGL07] This technology is much more resilient to detection and deactivation than the centralized model, accounting that it is not dependent on a few single servers so once one is deactivated that has almost no effect on the deactivation of the whole botnet. This characteristic is the main reason why this architecture is a growing trend between the attacker community. Figure 3 shows a possible organization for such a structure.

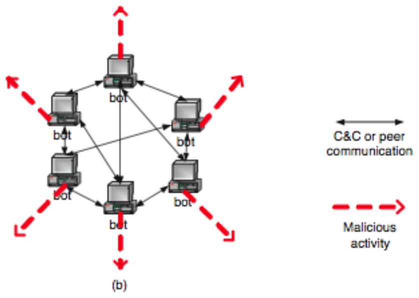


Figure 3 - representation of a P2P architecture [GPZL07]

Obviously this architecture comes with a price. Its major drawback is the scalability problem. It only supports conversations between small groups which makes it harder to launch big attacks. Also there is no guarantee on message delivery and has a higher latency associated with message propagation when comparing to the centralized model. This brings serious problems to the coordination on how the bots deploy to attack, but brings a much more robust model to the table.

#### 2.1.2.1 Structured

On a structured model the bot master use well defined criteria and algorithms to structure the way bots and resources are organized. This is held to make sure any bot can find its way to another, even when not directly connected. They are very similar to other C&C architecture, the random model, where every node has almost the same degree of importance for the network.[DGL07]

This defined criteria and algorithms are very important to robustness of the net but it can be the bottleneck of this structure at the same time so it's very important they are chosen carefully and accounting the botnet requirements.

#### 2.1.2.2 Unstructured

Here no requirements are imposed to the way things are organized. Peers relate in ad-hoc fashion and in the ideal world this kind of model would have no centralized command, but in reality it can be found in 3 different ways: pure, Hybrid and centralized. The pure one is where every node has the same degree and there is only one routing layer with no preferred nodes. The Hybrid is mix between pure and centralized where there are some nodes called super nodes which have a higher degree of command. The centralized is where a single server is used to tag the functionalities and to setup the whole system, but the connections between the peers are in no way controlled by any algorithm.

The major drawback of this implementation is that sometimes when trying to find information(which is done by flooding) from some node or contact it to attack, it might be hard to find it, if it's not well connected around the net or the net is very congested, which can happened because of the flooding.

### 2.1.3 Random

This description comes up on [CJD05] as a possible model, but was not found yet on the real world. Here the zombies do not use the C&C to contact the bot master, but instead, they wait for the bot master to recruit them for an activity. After infection the bot does not look for its master, instead it does nothing waiting to be summoned by the master. When wanting to launch an attack, the bot master scans the network looking for infected machines and sending them the necessary information to launch it. It's very important that the bot master knows which networks to scan, keeping the information of which networks were infected in a database. It is possible to see the way bot masters probe for zombies on networks on figure 4

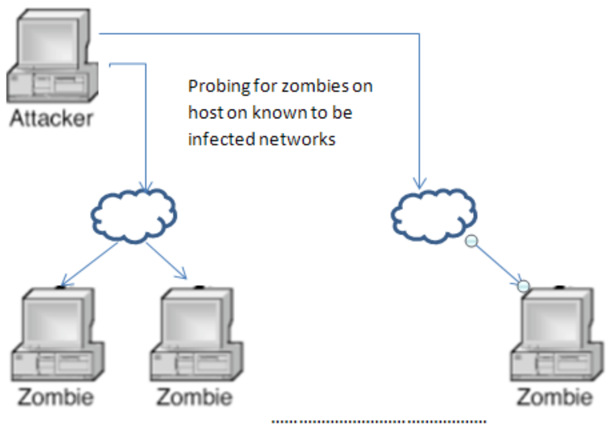

Attacker
Probing for zombies on host on known to be infected networks
Zombie
Zombie
Zombie

Figure 4 - How the random model does the probing operation botnet

This would have all sort of advantages. First it has very high survivability because there is not so much botnet activity correlated behavior with time that can be used to detect it. Second it is very easy to implement, since the protocol is pretty simple. And finally it is hard to discover and destroy because of lack of information and lack of connection between the bot master and its bots.

The biggest problems associated with this structure are the way in which detection of bots is done, the probing technique is easy to detect by a well programmed firewall and the message propagation latency, which can be huge depending on how far bots are from each other.

## 2.2 Communication protocols

The communication protocols used by botnets to exchange information are of an enormous use for the researchers and system administrators. The understanding and knowledge of how they communicate can answer a lot of questions, like where to look for evidence used in detection and what kind of traffic to look at. First because if we know the protocol being used we can monitor where normally that traffic is sent through and filter the inbound as well for that kind of traffic. And second we can define a normal pattern of what a packet or flow for that protocol is like and filter what looks suspicious. Besides that, this information provides understanding on what the botnet is based, where it came from and what possible software tools are being used. The task is not easy and has more constrains and obstacles than as described above but still this information is of high value.

### 2.2.1 IRC protocol

Still the most used protocol by bot masters to contact their bots as it was the place where the first bots appeared. The IRC protocol is designed mainly for communications between large groups but it allows private communications between single entities, which gives the master the right tool to communicate in an easy and flexible way.[Mic06] Typically the bots will all connect to the same IRC channel on the C&C designed server or on a free IRC server. Here the bot master will issue a command and the bots will use their parser to understand those commands and follow the order, either for attacking or other malicious activity. One could think by examining the content of IRC traffic, that we could detect botnet commands, but first the IRC channels allow users to have a password on the channel encrypting the messages and second every bot has a different syntax and since there is a lot of families and hundreds of variations, it's impossible to try every possibility.

On most corporate networks the use of IRC clients or servers is normally blocked, which makes it easier to know if a bot is acting since there is no IRC traffic and if there is it means someone has been infected. But, nowadays and to pass this obstacle, bots been able to tunnel IRC traffic under HTTP, which makes it harder for detection methods to find it. There are already some IDS(Intrusion detections system) which can detect this kind of traffic.[Mic06]

Even though at corporate locations some measures been taken to detect this kind of activity, most of the homes and small companies are not well protected against this and that makes this protocol a very capable tool for attackers to use since there is a huge number of different software mechanisms that make it very easy to setup and use.

### 2.2.2 HTTP Protocol

The popularity of using this protocol has been related to the attention that has been paid to the usage of IRC. The main advantage is basic to see, most of the traffic on the internet is HTTP so it's much harder to find malicious activity since the target to examine is much bigger.

It works simply using the url to post the necessary codes. After being infected, the bot visits an url with its id, which is on the server controlled by the bot master, the server answers with a new url which is going to be parsed by the bot to understand the orders for malicious activity.

Normally the firewalls block inbound traffic of IRC but they cannot do the same for HTTP(or they would block every legit request), then they would have to apply the right filters for abnormal headers or payload if not encrypted.

The practice of malicious activities using this protocol tends to grow and maybe one day might be the most used, the advantage of bypassing firewalls in an easy way and the presentation of a much higher percentage of traffic to look at, makes it a powerful tool. Maybe the major drawback is the latency inherent to contacting a big number of bots and joining forces to launch a big attack.

## 2.3 Other Observable activities

This is where real work can be done. The information given above about other bot characteristics is very important to understand how they work. Also it influences which other botnet activities to look at. But the detection of the C&C structure or the bot master is complicated by the level of indirection present on the communications between bots and the bot master. And the detection of the botnet activity by simply looking at the protocol used is very hard given that most of the messages are encrypted and, for the example of HTTP, the amount of traffic is huge and takes a long time to look at.

All these reasons make it important to account other activities which can potentially help to discover the botnet activity and deactivate it. When communications between the zombies and the command happens, certain patterns can be extracted and used to detect malicious behavior. Examples like abnormal DNS queries when hosts are found to query for improper domain names or if the IP address associated to a particular domain keeps changing. As well as looking at the network traffic flow and its characteristics or the system calls that happen from a certain software tool which are uncharacteristic.

The detection of malicious activity can be done using 2 different ways: Misuse or Anomaly detection. The misuse detection is done by having a database of known patterns of bad behavior(signatures) and detecting similar activities. The anomaly detection is based on the assumption that the attacker is unknown and creates a database with all patterns of normal behavior of the system, detecting everything that falls out of those as abnormal.[FHSL96]

Here is presented 3 types of activities that can be observed: Network based, Host based or Global correlated behaviors.

### 2.3.1 Network-Based behavior

On a network level when the communications between the zombies and the bot master is taking place some observable patterns can be detected. For example, apart of differences between bots and the way they practice their attacks, they always have to connect using the C&C server on someway(unless using the random architecture, which was never found on the wild yet) or bot masters have to pass their orders to the bots somehow and that can produce observable behaviors.

There is 2 main ways of doing this. One is by looking at the packets headers and contents. For example on an IRC based bot, one could try to intercept the packets on the standard port for IRC traffic and examine the content for botnet commands. But there are a big number of problems with this approach. First the standard port is only a good practice advice, there is no rule that obligates the attacker to send the traffic through that port. On other hand the traffic can easily be encrypted which makes it impossible to examine for bot commands. Bots are hundreds and every single one can have a different syntax making it very hard to keep an updated database of which commands can be used. Finally there is also the question of privacy when looking at the content of other people's conversations.

The other approach is to look at the flows and their characteristics. The main problem is which flow characteristics to look at. Basically there are a lot and correlating the results of groups of them is a big field of research. It can be a statistical problem, creating filters to select the flows that can have malicious activity which comes with a percentage of FP(False positives) or FN(False negatives) associated. One can also use machine learning algorithms to try to classify the flows in different groups[SLWL07]. One simple example on how to filter flows is looking at host activity on the network that can be helpful. Information on which hosts are trying to scan ports, emailing spam and generating DDoS traffic can indicate which flows to analyze[SLWL07]. On the classification of flows, there is a set of characteristics like bandwidth or packet timing that can evidence bursty activity which can evidence automated answers showing outside control on the host network activity.

One drawback of using this type of behaviors is that most of the times its observation is limited by the peering edge, not involving large number of flows happening on the backbone or on the whole enterprise network traffic [BCJX09].

### 2.3.2 Host Based behavior

The execution of botnet code inside the host will output sequences of system calls that can be observed and studied to build patterns. For example, correlated when the host is infected and must contact the bot master with its id or when it receives orders to perform malicious activities this triggers a sequence of system calls that are of interest. This sequence can be characterized by as many features as needed from the host activity or the system calls itself.

The problem is to know how far to look back on the sequence of system calls and what other features use to characterize the system calls that can be helpful. The size of the sequence will dictate, how many similar patterns can be found and how they can be correlated. Because of that, this choice must be done carefully. The process can either be done by looking at normal activities from a botnet like writing on a log file the entrances from the keyboard and verifying how many system calls are needed for that and use that number, or by empirical experience until finding the right pace for the right correlated sequences.

There is many ways in which these sequences can be looked at. Examples are the temporal orderings, the type or parameters passed, the instructions between the system calls or the interactions with other processes[HFS98]. When looking at the temporal ordering, the creation of the database or pattern is done by looking at how they relate in time, in another words, which system call comes after the other. The types of parameters passed can be used to see, for example, what kind of system calls happen after one process which received a parameter from a suspicious source like the internet or another process already flagged with malicious activity. Instructions between system calls are important and can be used to narrow down which sequences we should be looking at. This is when instructions that are out of the normal pattern are used by that process, it means the next sequence might be interesting. And the interactions with other processes can also be a clue to where to look at. For example when processes use others to copy important information and transmit it, then those sequences are important to be flagged and put in quarantine looking for repetitions of those instructions.

Other kinds of behavior can be used for detection. The act of Monitoring special files like /etc/host.equiv which contains the processes allowed executing system calls as root without having to introduce a password[HFS98]. The observation of the number of introduced failed passwords or number of times protection violations are triggered can be monitored in order to detect host based behavior that is not correct.[Den87]

### 2.3.3 Global Correlated behavior

Basically is a question of union is strength. One must correlate behavior from the network that might influence the host behavior, with that same host behavior and vice-versa. The problem here is how to correlate them since there is so many choices and obviously without hugely increasing the inherent overhead. One example would be to correlate in time flows with bursty characteristics with the number of failed input of passwords or the system calls from a certain process that received parameters from the internet or untrusted entities.

There is many ways of correlating these behaviors. Using the time to correlate would tell which behavior preceded the other and could be used to create the signatures of patterns to detect the equal bots or to register good behavior patterns to detect abnormal activities. Another way to correlate them would be application based, simply looking at the flows and system calls related to that application. Finally, one could also correlate by other means like the parameters passed from network based behavior process to a host based one or when one process on the network interacts with another process on the host. An explicit and consistent definition of the ways of correlating behaviors from both worlds has not been given anywhere yet and needs to be extended to be better understood and used.

This can give from the point of view of efficiency one great weapon to be used since it would give a more global snapshot of what is happening, but, it is key to find the right behaviors which can lead to good results. The major drawbacks would be the infinity world of choices and how hard would it be to setup on the real world. First the choices must be made, then tried on a safe environment and after should be tried on the real world. This is a process which is most empirical and takes a long amount of time. Also the problem of setting up a safe environment is that is hard to get the real experience to train a database on the case of anomaly detection. Since the real world is more complex than a safe environment and accounting the risk of false positives is very high it may take a few tries and a long time on training further in the real world to create a usable tool. For the case of misuse, having a database of signatures might never be enough and need daily update since in the real world the threats are always evolving and different variants of bots appear all time.

CHAPTER 3

# Design

## 3.1 Motivation

The usage of techniques for detection by misuse is obsolete. The evolving nature of botnets and the number of families and variations on the wild make it almost impossible to create a credible database to be used on a tool like a scanning anti-virus to detect botnet evidence. The time inherent to such a task makes it a powerless weapon against a fast-attacking technology like a botnet. The necessity nowadays is towards a generic scanner that has no demand of updating the information to be able to recognize something is wrong. That's the example of techniques for detection by anomaly where patterns of good behavior are drawn and everything else that is detected is considered to be abnormal [WDF$^{+}$03].

On chapter two were presented many reasons why techniques like looking at the traffic for packet content or at a standard port are not suitable for the given problem. Moreover the best results found on related work were all either related with host based behavior or when looking at the network and trying to correlate flows through its characteristics[SLWL07] [SM07] [OKA10]. Even though those works show some good results, they still look like missing something or show some limitation which cannot be overcome just by the presented work. For example on [SLWL07] it was only developed to be useful for IRC centralized botnets which is obviously a limitation accounting to the growing trend of other

C&C structures. It alerts also to the importance of choosing carefully the methods to solve the statistical problem of classifying traffic in order to avoid high rates of FP's and FN's. Moreover, it explains how co-relation is fundamental on the detection since evaluation on isolated traffic flows might not be enough. The system described on [SM07] has limitations when the parameters passed to the system calls (the main element to classify the system call as tainted or not and possible evaluation) suffers an out-of-band encryption or the act of passing a parameter can be spoofed as being from a legit user. To finalize [OKA10] is based on the results by community and if only a few machines on that community are infected the results might not detect it. Scalability is also an issue which must be solved, even though the author gives guidelines on how to do it.

After reading all the available information on detection of botnets one must think if there is not a possibility to gather all those techniques and trying to use the best of each building a strategy that will give more certainty on detection. After all, union is strength. When using a lot of techniques on one, more information is needed. This necessity will certainly increase the overhead but if properly used, more information means also more accuracy and more accuracy will give smaller rates of FP's and FN's. Thus is very important to find a balance between overhead and accuracy when choosing the amount of information necessary.

Accounting these reasons, the choice of the architecture was based on trying to be as generic as possible not basing it on any known bot architecture or protocol, but looking more on how a machine would behave on both sides (host and network) giving just enough information to create a faster scanner.

## 3.2 Architecture

Figure 5 represents the architecture of the purposed prototype for detection. It's a simplistic view of what is happening on the prototype. First the information about what is happening on the host is passed to the host evaluation block. Here the data is processed to be used on the K-means algorithm (see chapter 4) and if there is reason for alarm (if the given threshold is passed - see chapter 6 for the definition of the thresholds), the necessary information is passed to the network evaluation block. The network evaluation block receives the information from what happened on the host, especially the times of the possible to be malicious activity and the information from what happened on the network (flow characteristics). This information is going to be correlated in order to give a stronger evidence of malicious activity and if once again the threshold is passed the prototype will output alarms to the user and a report explaining what kind of applications are performing these malicious activities and at what times such activities are happening.

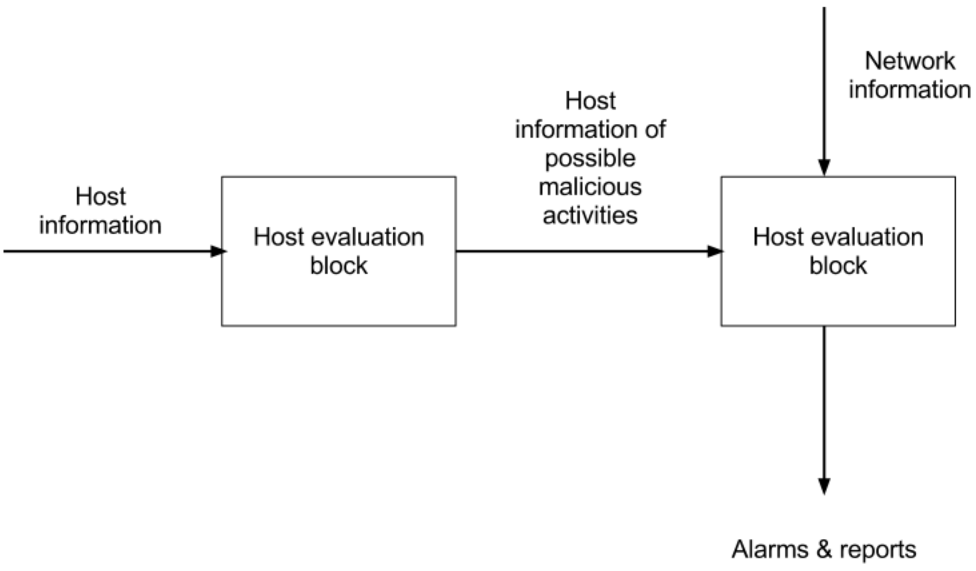


Figure 5 - Prototype architecture

On the host side the inputs will be vectors containing information of system calls on four points: The name of the system call, the application/process running it, the time between the present system call and the last one and the registration of the past three system calls. On the network side the input will be first a report from the host side with the times from the suspicious activities and from the network the time that flows were registered and the time intervals between them. The output will be the name of the application, the times that malicious activity was registered by it and the reasons why it was registered as malicious, meaning, the values that passed the thresholds.

The choice of looking at host based behavior first and then use it to narrow down the information on the network behavior was done based on the idea that the network will have much more operations running since it treats the data of all the hosts. Obviously the number of flows would be much higher making it harder to search for the important information. On the presented architecture, we collect both kinds of data at the host, which means we get the system calls form the host and the flows that only relate to that host. So the network flows will only be analyzed for the flows timely correlated with suspicious activity on the host, helping narrowing down the data to look at.

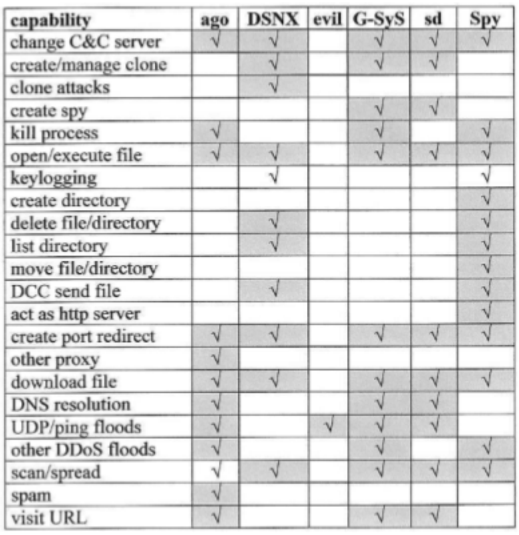
The design was chosen attending the botnet characteristics and the tendencies of their evolution. This prototype is host based and presents a simplistic architecture in order to help to turn detection into a softer operation, retaining information from both sides and giving a tool capable of detecting any different kind of malicious activity.

Chapter 4

# Host based behavior

On this chapter a description is given of what kind of host based behavior is important to collect and how that information will be treated in order to help detection.

System calls are used as information in order to create profiles of good behavior of the system. Obviously it is impossible to register every system call and the information that concerns it, so a set of features must be chosen to be used to create the profile. The choice is for a simple set in order to be efficient and give the information that is strictly necessary but not insufficient. The first thing is to decide which system calls to look at. For example, when looking at the main botnet activities presented on Figure 6 all of them perform read or writes like when changing of C&C server or at the creation of a port redirect which between the presented activities for the given botnet species on the paper are among of the most common. Hopefully these activities will present different patterns from the normal activity and maybe will happen more often or timely correlated (changing the C&C server for example is performed quite regularly and sending data to the bot master is performed normally with the same time interval). The chosen system calls are the read and writes plus the information about which application has done it(giving the information if it was done from or to the network) and the relative time when it was done(giving us the time difference between system calls) as well as the three system calls to happen before the registered one.



| capability | ago | DSNX | evil | G-SyS | sd | Spy |
|---|---|---|---|---|---|---|
| change C&C server | √ | √ |  | √ | √ | √ |
| create/manage clone |  | √ |  | √ | √ |  |
| clone attacks |  | √ |  |  |  |  |
| create spy |  |  |  | √ | √ |  |
| kill process | √ |  |  | √ |  | √ |
| open/execute file | √ | √ |  | √ | √ | √ |
| keylogging |  | √ |  |  |  | √ |
| create directory |  |  |  |  |  | √ |
| delete file/directory |  | √ |  |  |  | √ |
| list directory |  | √ |  |  |  | √ |
| move file/directory |  |  |  |  |  | √ |
| DCC send file |  | √ |  |  |  | √ |
| act as http server |  |  |  |  |  | √ |
| create port redirect | √ | √ |  | √ | √ | √ |
| other proxy | √ |  |  |  |  |  |
| download file | √ | √ |  | √ | √ | √ |
| DNS resolution | √ |  |  | √ | √ |  |
| UDP/ping floods | √ |  | √ | √ | √ |  |
| other DDoS floods | √ |  |  | √ |  | √ |
| scan/spread | √ | √ |  | √ | √ | √ |
| spam | √ |  |  |  |  |  |
| visit URL | √ |  |  | √ | √ |  |

Figure 6 - Shows the capabilities for certain families of botnet studied in [SM07]

## 4.1 Classification

Classification is the process of mapping items onto categories[LS98]. This process is important to distinguish different types of traffic and allow to filter or to find what's necessary. Many techniques from statistical analysis to data mining can be used for classification. It can be a very slow process and it must deal with a certain rate of false positives and false negatives so the choice of the method must be adequate to the problem otherwise these drawbacks will come to haunt you.

Statistical methods are very powerful but normally depend on assumptions about the distributions that might not correspond to reality leading to a high false alarm rate [LTG$^+$92], a good example of this technique is the Naive Bayesian algorithm. The study in [MZ05] uses some variants of this algorithm to classify network flows. Data mining techniques are algorithms that allow researchers to build models from which they can infer patterns useful for classification of certain items, being artificial neural networks(ANN) one of the best examples on this case. In this paper [NSB10] there is an example of a work that uses an ANN to classify traffic and distinguish if it looks like a pattern of good or bad behavior. Many other investigations used ad hoc classification techniques like for example on [RY08], where the traffic is aggregated by traffic packets characteristics and host similarities.

On the present case, sequences of system calls must be mapped into good behavior and bad behavior. To perform the classification on the experiment, the K-means algorithm an unsupervised learning technique will use system calls and its characteristics as input vector.

### 4.1.1 Artificial Neural Networks

Artificial neural networks are machine learning techniques that can be used for pattern recognition. They are formed by a set of neurons based on the animal neuron, having a layer on the input and output, and depending on the topology, they can have an intermediate one as well. They can be seen as a non-linear function between the input and the output, which the transition is done normally by an activation function. Each Neuron is a linear automata, realizing a weighted sum from several inputs [DMS92]. On the beginning they have no knowledge base and have to be trained using a training set, mapping input vectors to output ones, adjusting the weights for better results. The training can either be supervised, unsupervised(also known as competitive) or by reinforcement. One of their best characteristics is the capacity to as soon as correlation is found it is taken in account as well as if an input is irrelevant it will be able to neutralize the weight of that neuron, this if well trained to do so.

They figure as good solutions to unstructured problems containing daedal relationships between variables, i. e., when the rules to classify the target input are hard to identify. But normally don't give good results when human reasoning is necessary. One big advantage comparing to statistical models is their capability to find non-linear relationships on confusing data [FB07] as well as the capacity to adapt easily to other systems(unless re-training is necessary which can be costly in terms of time). Another advantage is the scalability provided for large sets of data.

The major drawbacks are the high computation intensity required and the long training cycles for the learning to be satisfactory. There is one problem that one must address and is very hard to calculate before testing: the size of the neural network. Normally the entrance should have as many neurons as the input vector size. And the output depending on the goal, can have one or more. But on topologies with an intermediate layer, tests must be done to realize which number of neurons better suits the solution of the problem.

### 4.1.2 Supervised Learning

On supervised learning the training set is given in the form of input and the expected output. To train a neural network using supervised learning, the input is given and the expected output is compared to what came out on the neural network. If they do not correspond then the network uses a learning algorithm which will correct the weights in order to correct the output.

A typical example would be the multilayer perceptron with the back-propagation error learning algorithm. This kind of neural network would be from the feed-forward type. It starts by pre-defining a N sized window sample from the training set (in a sequential form) and it learns how to predict the next value on the series. [DMS92]. The training is finished after the algorithm reaches its point of convergence and during this process the weights at the neurons go from random on the start to the desired function in the end.

The usage of such a network has many drawbacks: to begin with N (the sample size) is initially fixed and cannot be changed during the network life or a change would require a complete retraining. This same number can act as bottleneck for the network: if its value is too high the process can be perturbed with unnecessary information or if it is too low the accuracy of the system can be affect with the lack of relevant information. These kinds of network don't have a high rate of adaptability since a partial retraining can lead to forget everything it has learned before. On the present case this problems can be solved by testing many different N's and choosing the one with better results.

Recurrent networks are other type of supervised learning networks where part of the output is sent back into the network, creating memory inside of the it. The input is the most recent element and the output is feed backed to keep trace of past events on internal memory. The long term memory is coded on the connections and the short time one is coded on the neurons. This memory is limited by the number of neurons and connections. There are several drawbacks associated to this topology as well: First the decision of the size of the network is a problem hard to address. Second and on experiences done by [DMS92] there is values the network never learns and there are states where the configurations are unstable and it is hard to understand why.

### 4.1.3 Unsupervised Learning

On this case no solutions are provided on the training set. The distributions are unknown but the number of neurons on the input of the neural network is the same as the K classes of input. So in this case the neurons "compete" between themselves to see which one is fired first, for that to happened they must be in a full mesh in order to know each one's state. The learning algorithms are based on structure presented on figure 7.

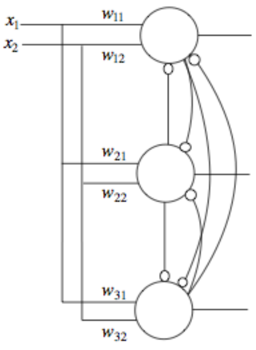


Figure 7 - How competitive neural networks work [Roj96]

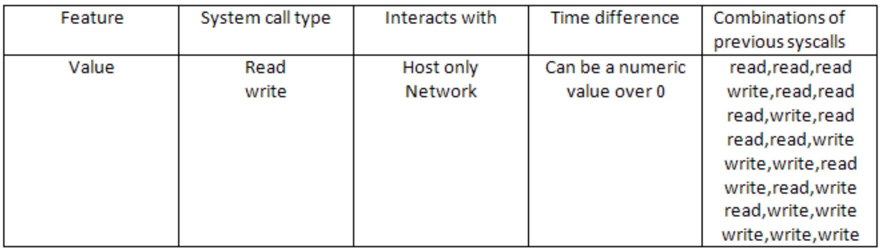
Other techniques can be used for unsupervised learning besides a neural network. Algorithms for clustering techniques like K-means are widely used in help for classification. The K-means algorithm is one of the most used clustering algorithms and has a simple idea of identifying the k-centroids of the clusters of the data in order to minimize the mean square distance from each point to its nearest center[KMN$^+$02]. The K number of clusters to form must be given in advance. It starts by choosing randomly k-centroids(some variants use an heuristic to do this) from the data set and then goes from there measuring the distance between points and their centers. This randomly chosen centers can be updated at the end of the calculations for each vector or a few iterations. This algorithm can be slow to converge depending on the way the centroids are chosen but it is proven to always terminate, thus without a promise of finding the optimal solution. One way to reduce this problem is to run the algorithm more than once [FB07].

## 4.2 Experiment

The Experiment consists on collecting data (system calls information) from a machine and using that to find different patterns. On the present case the collected data is not classified so a technique for unsupervised learning is needed.

Since K-means is a very common solution, one of the more studied [KMN$^+$02] for the clustering problem and is proven to always terminate, it was the chosen approach. This gives also the possibility of exploring more approaches(different k's or heuristics for choices of the centroids). The latency to converge is not a problem in the present training set because is not that big and the patterns are very regular, but finding an heuristic to calculate the initial centroids in order to find an optimal solution should be explored in further work.

The choice of the features was done carefully thinking in what, patterns of botnets, can be built on. First the choice of the system calls to look at was done based on one of the most used by regular processes and by botnet activity, read and writes. Those happen most of the times right after commands have been given to a botnet or infection as well as in legit processes they will be very regular on their patterns. The second feature is the type of process using that system call to perform its activity, it's either a process that can receive parameters from the network or not. This process requires human intervention, where one should define which processes can be receiving external parameters from the network and the ones that do not. This pre-classification is kept on a database and once one finds a system call to classify, looks at the process invoking it and compares if it is on the database, if not, classifies it as possible to receive information over the network(default behavior). One more feature to look at is the time spent between system calls, giving an average of how long it takes for a new one to happen. This is done by subtracting the time of the given system call with the time of the last system call. Besides giving information if something unusual is happening if the time patterns are very different, it gives us the time slot to also know which network flows to look at(for correlation). The last feature and one of the more important is the pattern of system calls. For every system call it will record the past 3 ones giving a window of 4. This should help K-means evaluating better the clusters for different activities on the same process, or same activities for different processes depending on which regularities and centroids K-means will find. On figure 8 a description of the possible values for each feature is given.



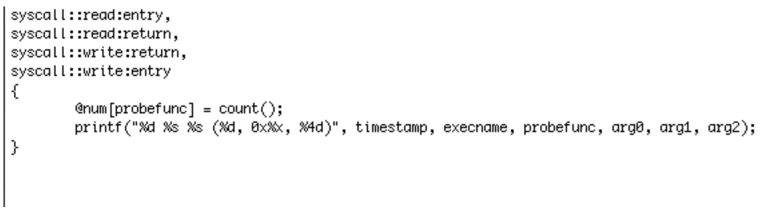
| Feature | System call type | Interacts with | Time difference | Combinations of previous syscalls |
|---|---|---|---|---|
| Value | Read<br>write | Host only<br>Network | Can be a numeric value over 0 | read,read,read<br>write,read,read<br>read,write,read<br>read,read,write<br>write,write,read<br>write,read,write<br>read,write,write<br>write,write,write |

Figure 8 - Possible values for each feature

### 4.2.1 setup

The collection of data was done on the network of the department of Informatics and Mathematical Modeling at the Technical University of Denmark, during 10 minutes only, giving over 250 megabytes of information and almost 3 million system call occurrences. Most of the information is supposed to represent good behavior since the network is well isolated from external intrusions but since there is no way to be 100% sure of such a thing the choice of using unsupervised learning methods seemed more appropriated. The choice for such a small amount of time was because the machine was collecting too much information since tcpdump(see chapter 6 for more information) was running at the same time.Also the given information was of high regularity, not needing such a big sample. An experience of 30 minutes was giving over four gigabytes of information which was being impossible to process on the machine. Still this 10 minutes are very representative of what was happening, since the 30 minute file was just repeating itself.

The machine to perform the task was a Dell optiplex 990 using an AMD64 as processor and FreeBSD 9.0 installed as operative system. The tool to collect the information was Dtrace, which is the standard tool that comes with this FreeBSD distribution for debugging and collecting information about the machine. It uses what it calls probes to collect the information. The script to choose the probes to find the necessary information is the following:



```
syscall::read:entry,
syscall::read:return,
syscall::write:return,
syscall::write:entry
{
        @num[probefunc] = count();
        printf("%d %s %s (%d, 0x%x, %4d)", timestamp, execname, probefunc, arg0, arg1, arg2);
}
```

Figure 9 - The script used to collect system calls from the hosts

The software chosen for execution of K-means was WEKA a very well-known tool for applying machine learning techniques to large sets of data. A script in java was developed to represent the data in an acceptable form for WEKA(arrf). K-means was run with three different values for the number of clusters, 2, 3 and 4.

### 4.2.2 results

For the K-means instance with k = 2 the obtained results are presented on figures 10 and 11 in Appendix A.

For the K-means instance with the k = 3 the obtained results are presented on figures 12 and 13 in Appendix A.

For the K-means instance with the k = 4 the obtained results are presented on figures 14 and 15 in Appendix A.

Chapter 5

# Network based behavior

The choice of what network based behavior to look at is dependent on the information collected on the host. This choice was done based on the idea that the network will have much more operations running since it treats the data of all the hosts. The presented architecture is set to first look at the host and only after to its network activity (in case of suspicion). Here a description of what are the reasons for the choices done on the network side and how those choices can help detection is provided along with an explanation of how the data was collected and treated.

As explained on chapter two, there is two different ways of collecting information on the network that can be useful for detection. Either by looking at packet headers/contents or flow characteristics. Looking at packet contents is an option that has several disadvantages (listed in chapter 2), but when looking at the flow characteristics or packet headers one can use the same techniques specified on chapter four for classification. For example using temporal models to detect bursty activities or detect flows on the same conversation can be achieved implementing classification. Classification (using other parameters) was used in [SLWL07] but the author decided to bypass that step since none of the algorithms used gave results good enough that provided an advantage. Attending that the overhead inherent to such an activity is large and it is already being performed on the host evaluating block. The decision for using other techniques seems more logic. This decision must be done between what flow characteristics or packet headers information can be used and how they should be treated in order to provide a good hint of botnet activity

The first choice is what features can help to identify botnet evidence after what was collected on the host. The choice was only one feature: packet timing's. Normally if the machine is doing activities which are controlled from afar that will show some automated code running and the response times will be faster than normal because if it was a human responding to a request it would do it slower than an automated machine. That's what packet timing gives us; if the time between flows has smaller values than usual then it might indicate we are in the presence of botnet activity. The main reason for choosing this feature is that it is independent from any protocol that the bot can be using and does not need to look at any packet contents, just by looking at the flows and obtaining the arriving times one can use it to calculate and compare it to the normal behavior. Also it does not use any information on the packet that can be spoofed (like the sending address, etc...) since it is the time that it arrived to our host or sent by it.

The technique to treat the information is a very simple ad hoc algorithm which uses a simple statistic technique. The times between flows will be registered in order to calculate the mean and the standard deviation of the sample taken from the network. Then these values will be used to calculate the threshold. Normally creating a pattern of how the system should behave. Once the prototype is active on the system and it receives the information of the times where suspicious system calls happened on the host. It will verify if the average times between flows happening near the given times are smaller than the defined threshold (more information on these values on the next chapter).

The techniques that exist to treat information at the network side are as vast as on the host side. The choice was done by a more simple approach comparing to the host side since equilibrium between accuracy and overhead was necessary to achieve and the host evaluating block is already providing a big dose of overhead. The presented algorithm and flow characteristic were used on an experiment for the values on the provided training set as explained on the next section.

## 5.1 Experiment

The experiment consisted in using a tool to collect flow information at the same time the system calls were being collected, thus having the times that each one occurred at and be able to co-relate them. Than use this information to calculate the threshold and use the information provided from the experience on the host side to verify if the flow timings were correct. Since this flows were collected on the same conditions as the host based information, the probability of finding any flows to be suspicious is very low and also this are the values that are being used to calculate the threshold so that possibility is even more tiny.

### 5.1.1 setup

The setup is the same as the experience for the host based information since it was done at the same time (necessary for the co-relation). But in this case tcpdump was the tool used to collect the information about the flows. Tcpdump is the standard packet analyzer that is installed on the FreeBSD existing on the machine used for the experience. It works by simply listening on the given port connected to the network and registering the times the flows came in and got out. The 10 minute experience gave us a file with over 11 thousand flows and 1,4 megabytes. To calculate the average and the standard deviation, the data was passed to an arrf file with the same script used before to treat the data from the host system calls and passed to WEKA.

### 5.1.2 results

The results after collecting and treating all data for the flows, are represented on figure 16 at the Appendix B.

Chapter 6

# Global correlated behavior

The use of correlated behaviors is the goal of the whole project. The prototype is based on the idea that the correlation of behaviors from the host activity with the ones from the network activity will help finding some botnet evidence. On this chapter a description on how this can be achieved and the base for an experience on the real world with this activities will be given in detail.

## 6.1 Algorithm

To correlate the information from both sides we must define the numbers that will give us the thresholds to mark system calls and traffic as suspicious. One must define the amount of data that must be evaluated and how that can be used to calculate the limits. The system will collect data during an interval of time, verify if the data passes the threshold on the host side, and if so pass the information to the network side which will verify if it passes its threshold and if so, it will output the alarms and/or reports.

The first one to define must be at the host based side. Here several experiences were done grabbing the system calls and since they represent the biggest amount of data to collect, they will define the space of time where the system will collect data until the evaluation. Here we collected information during 30 min and 10 min. The 30 min experience grabbed a total of 20 million system calls which represented an amount of four gigabytes of memory. This was a way too large set that the computer could not treat. The 10 min set grabbed a 250 Megabyte file with almost 3 million system calls on it, which represented a large amount, sufficient to give a description for the behavior patterns. The problem is that imagining this on a real time system, it would be impossible to be doing this verification every 10 min. A solution in the middle would be the best giving three verifications every hour: a 20 min interval. For the purposes of the experience it was used a 10 min interval, but it is strongly believed that a 20 min would be better on a real world to reduce the overhead.

Once K-means was run for the training set, its results will be used as benchmarks to define the threshold on the host side. After a sample of 20 minutes is collect, The values will be classified (looking at which centroid is more near). If the percentage of system calls for the smallest cluster is bigger than the one found on the training set than the times for all the system calls from that cluster should be evaluated on the network side. The host block should have the flexibility for the user to choose using a training set for a K-means using just 2, 3 or 4 clusters, depending on which one gives better results. On the case that the user decides to choose 4 clusters, the comparison of percentages, should be done on the two smallest clusters. For example if in one system the training set is used on a K-means evaluation of 3 and 4 clusters. On the 4 clusters evaluation it gives percentages like 23%, 24%, 26% and 27% and on the 3 clusters: 42%, 38%, and 20%. The best choice would be for the 3 cluster solution since it has a bigger difference between the smallest and the others.

On the network side, training will gives us the mean and the standard deviation values for the time between flows. The difference between the mean and the standard deviation is the defined threshold to be used to compare the values on the real system. First and using the 20 min sample it will calculate the values and keep it on the database. Once it receives the information from the host block with the times of the suspicious activities and the flows times, it will calculate for each system call timestamp the average value of times between the 10 previous and the 10 after flows. Then it will compare the average time between this 20 flows with the calculated threshold. If the value on the new sample is smaller than the one on the training set, it might mean we are in the presence of automated responses and this system calls/flows information should be reported as suspicious to the user. The choice of the number of flows to look at must be adjusted on further tests with values from a real botnet activity, evaluating which one will give a smaller rate of false positives.

The definition of the thresholds is very important for the whole function of the prototype and their definition should always be adjustable to the system and the environment they are in. The provided algorithm is the proposition done for the provided training set. The interval of time to collect information should be adjustable due to performance reasons depending on which machine is being used and after the running of K-means adjusting the values to right K and the calculation of the best number of flows to look at, then the algorithm should be prepared to find evidence of malicious activities.

Chapter 7

# Conclusion

## 7.1 Discussion & limitations

One of the most important things the performed experiences wanted to show was that the prototype should be flexible in choosing the number of clusters on the host based information. As shown on figures 10 and 11 for a k equals to two, the information is divided uniformly, giving both clusters almost around 50% each of the registered system calls. Figure 11, shows the distribution of registers through the clusters concerning each feature in which we can see there is no particular patterns for any of the clusters. All of them include samples of all different values of the features, giving no advantage on the process of classification for which behaviors are more common and normal. On figure 14 and 15, the results for four clusters are presented. This results show two very big clusters and two small clusters. On this case the 2 small clusters could be used to detect the abnormal behavior, but figure 15 shows that those two clusters have the same problem has in the example of $k = 2$: There are samples of all different values of the features for almost every feature. It gives a better classification than k=2 but does not look sufficient to distinguish between good behavior and bad behavior.

The results for $k = 3$, seem to fit. First There are 2 clusters with a very big percentage of the registers and a smaller one that represents behavior less regular (Figure 12). Then this cluster on Figure 13, shows evidence of only of one value of some of the available features, especially on the type of application (the graphic on the left up side) and time interval (the graphic on the right down side). Since it is expected that the malicious behavior comes from the network, this cluster helps finding the less normal behavior that comes from it. The values on the time interval for this cluster will be for the biggest periods without a system call which can normally happen on bursty activity by a botnet, with periods of intense activity and periods of silence.

Unfortunately the times registered by the Dtrace tool were not correct and couldn't be corresponded to the ones on the network side not allowing to test the values on the network side. But since this is the same values that were used by the training set to calculate the mean the standard deviation is probable that the values would not pass the threshold and not giving evidence of any malicious activity.

One limitation to our work is the pre-classification done on the type of applications at the host side. This classification must be done by human intervention, which gives room to flaws and mistakes. It is very important to be done carefully. As well if it is not done makes the feature useless to the algorithm making it weaker. it is a sensible point and should be paid attention while the setting up of the purposed prototype.

Another limitation is the case where botnets use defenses to fake the response times. If a botnet is using an automated algorithm that delays the answers to the received commands it will bypass our defenses, this phenomena is known as stealth. This can be overcome by looking if the times are too near the average. Which means we need to calculate the mean and the standard deviation for the given sample and verify if the standard deviation was too small comparing to the value from the training set. If so this kind of traffic could be flagged as automated as well.

The use of these features to classify and detect botnet evidence show promising signs on the host side, where it seems to be possible with those features to find profiles of good behavior. On the network side it is more complicated since without testing the number of flows to look at on the real world, thus finding the best number for a small rate of False positives, it is hard to say if it would work or not. The limitations seem to be not big obstacles if tweaks are done on further work.

## 7.2 Further work

Some Aspects of the prototype need to be further tested in order to be sure they work on the real world. Also there are alternatives to the choices made in algorithms, parameters and thresholds that can be overviewed for a better solution to the problem.

The tests were only done on data that was not classified but is almost a 100% sure to not be tainted with any botnet activity. That said it should be important to obtain results on the K-means algorithm on a data said known to contain botnet code and if possible to know which system calls were done by it. If so one could verify if the biggest percentage of those system calls would group together on a single cluster and chose the K number of clusters that would better characterize it. Using that cluster as the one to verify if there were any system calls that belong to it. This classification could also be used (if the results were good) to train a neural network giving a more reliable and already automated tool to calculate if a system call belongs or not to botnet activity. The percentage could still be the threshold used to trigger the information to the network side.

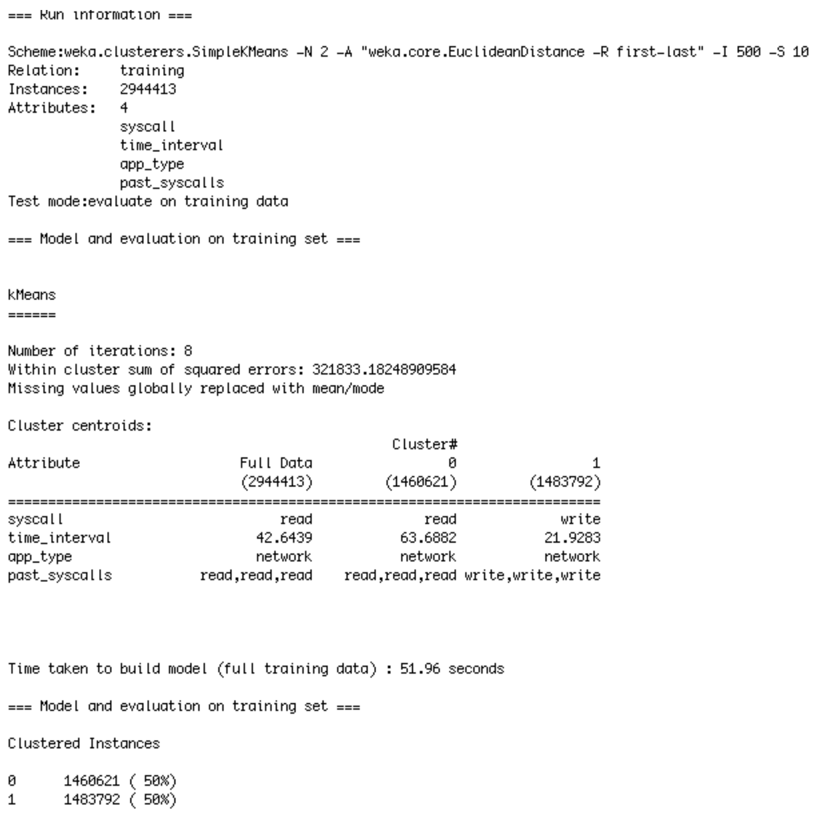
The number of system calls used on the last feature should also be tested for different values since it would be interesting to see which one would give a smaller rate of FN and FP. The best results obtained on previous works were always with small windows, but numbers like the past 5, 6 or 7 system calls could be used to test and find the balance between accuracy and overhead.

On the network side, other features like the application related to that flow or the addresses that communications are happening with could be useful to classify the flows and help co-related them with the system calls. These ways should be explored to better understand the feature that gives best results.

There is a lot of solutions that should be tested in order to compare which one is more efficient and since this is based on information from the system itself, it should be implemented with the possibility of using different solutions for different systems.

Appendix A

# Host Based results



```
=== Run information ===

Scheme:weka.clusterers.SimpleKMeans -N 2 -A "weka.core.EuclideanDistance -R first-last" -I 500 -S 10
Relation:     training
Instances:    2944413
Attributes:   4
              syscall
              time_interval
              app_type
              past_syscalls
Test mode:evaluate on training data

=== Model and evaluation on training set ===


kMeans
======

Number of iterations: 8
Within cluster sum of squared errors: 321833.18248909584
Missing values globally replaced with mean/mode

Cluster centroids:
                                            Cluster#
Attribute                   Full Data              0                1
                            (2944413)      (1460621)        (1483792)
=====================================================================
syscall                          read           read            write
time_interval                 42.6439        63.6882          21.9283
app_type                      network        network          network
past_syscalls          read,read,read read,read,read write,write,write




Time taken to build model (full training data) : 51.96 seconds

=== Model and evaluation on training set ===

Clustered Instances

0      1460621 ( 50%)
1      1483792 ( 50%)
```

Figure 10 - results obtained by running K-means for 2 clusters: centroids and percentages

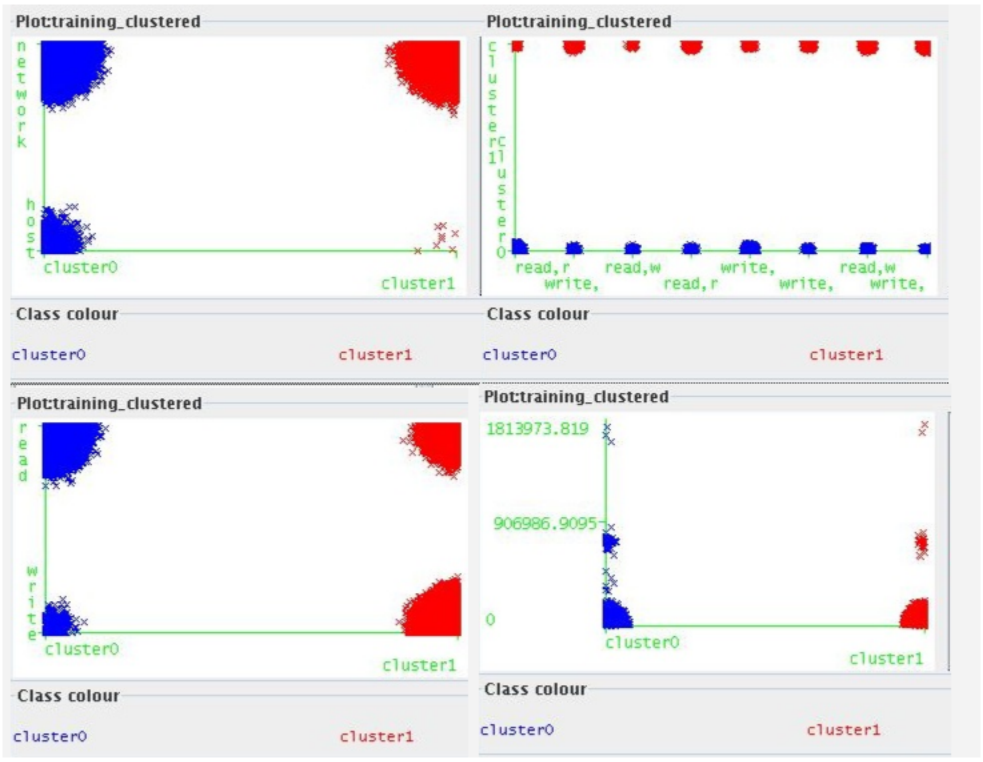
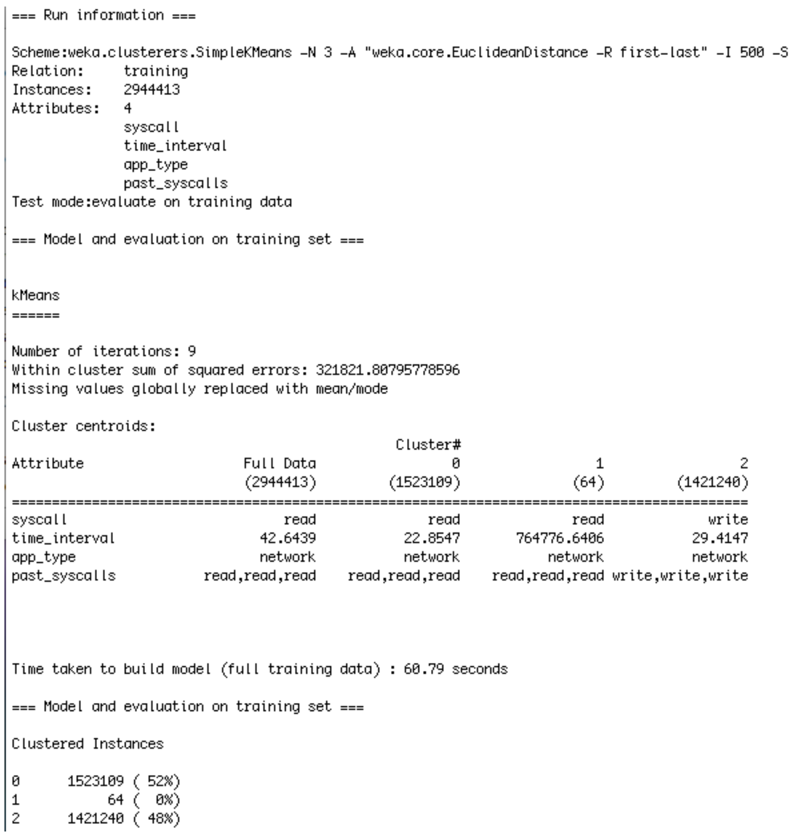


Figure 11 - results obtained by running K-means for 2 clusters: point disposal for each feature, per cluster.

Finished k=2



```
=== Run information ===

Scheme:weka.clusterers.SimpleKMeans -N 3 -A "weka.core.EuclideanDistance -R first-last" -I 500 -S
Relation:     training
Instances:    2944413
Attributes:   4
              syscall
              time_interval
              app_type
              past_syscalls
Test mode:evaluate on training data

=== Model and evaluation on training set ===


kMeans
======

Number of iterations: 9
Within cluster sum of squared errors: 321821.80795778596
Missing values globally replaced with mean/mode

Cluster centroids:
                                         Cluster#
Attribute                  Full Data            0               1                2
                           (2944413)    (1523109)            (64)        (1421240)
=====================================================================================
syscall                         read         read            read            write
time_interval                42.6439      22.8547     764776.6406          29.4147
app_type                     network      network         network          network
past_syscalls         read,read,read read,read,read  read,read,read write,write,write




Time taken to build model (full training data) : 60.79 seconds

=== Model and evaluation on training set ===

Clustered Instances

0      1523109 ( 52%)
1           64 (  0%)
2      1421240 ( 48%)
```

Figure 12 - results obtained by running K-means for 3 clusters: centroids and percentages

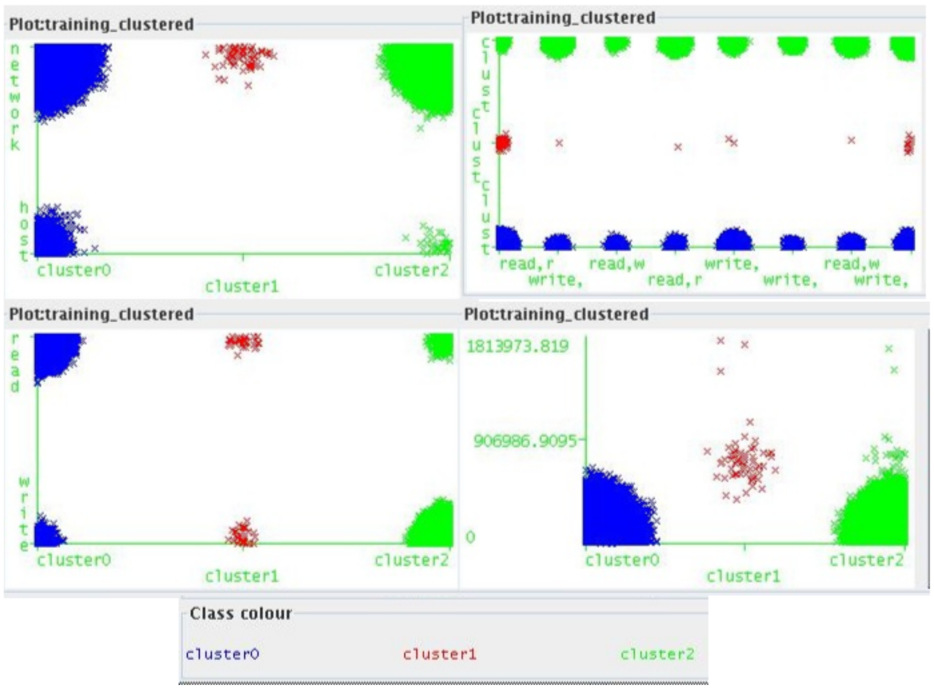
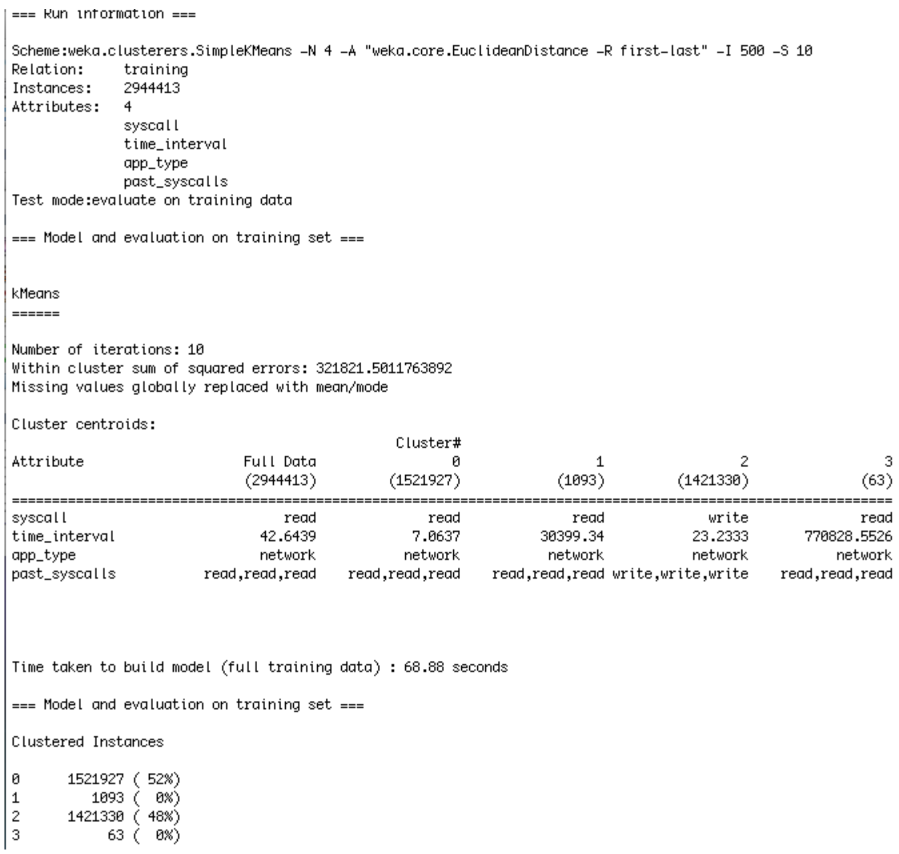


Figure 13 - results obtained by running K-means for 3 clusters: point disposal for each feature, per cluster.

Finished k=3



```
=== Run information ===

Scheme:weka.clusterers.SimpleKMeans -N 4 -A "weka.core.EuclideanDistance -R first-last" -I 500 -S 10
Relation:     training
Instances:    2944413
Attributes:   4
              syscall
              time_interval
              app_type
              past_syscalls
Test mode:evaluate on training data

=== Model and evaluation on training set ===


kMeans
======

Number of iterations: 10
Within cluster sum of squared errors: 321821.5011763892
Missing values globally replaced with mean/mode

Cluster centroids:
                                              Cluster#
Attribute                    Full Data               0                 1                 2                 3
                             (2944413)       (1521927)            (1093)         (1421330)              (63)
=============================================================================================================
syscall                           read            read              read             write              read
time_interval                  42.6439          7.0637          30399.34           23.2333       770828.5526
app_type                       network         network           network           network           network
past_syscalls           read,read,read  read,read,read    read,read,read write,write,write    read,read,read




Time taken to build model (full training data) : 68.88 seconds

=== Model and evaluation on training set ===

Clustered Instances

0      1521927 ( 52%)
1         1093 (  0%)
2      1421330 ( 48%)
3           63 (  0%)
```

Figure 14 - results obtained by running K-means for 4 clusters: centroids and percentages

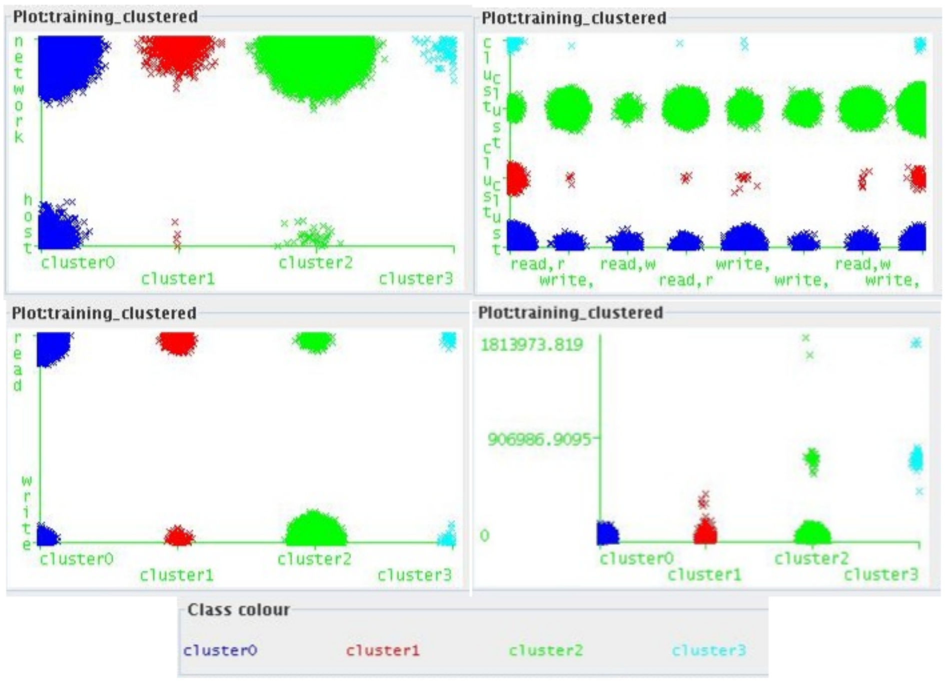
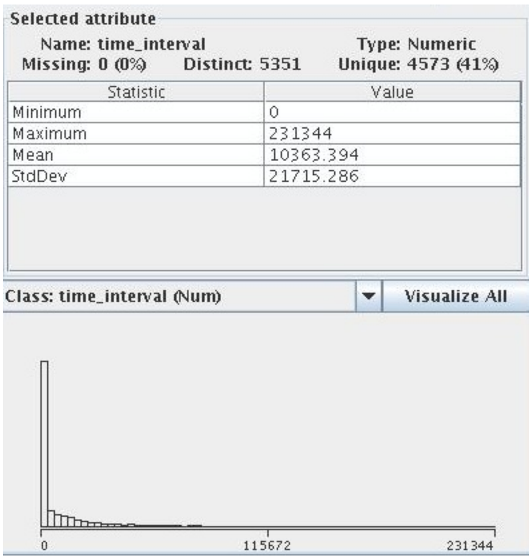


Figure 15 - results obtained by running K-means for 4 clusters: point disposal for each feature, per cluster.

Finished k=4

Appendix B

# Network Based results


Selected attribute

Name: time_interval Type: Numeric
Missing: 0 (0%) Distinct: 5351 Unique: 4573 (41%)

| Statistic | Value |
|---|---|
| Minimum | 0 |
| Maximum | 231344 |
| Mean | 10363.394 |
| StdDev | 21715.286 |

Class: time_interval (Num)

Figure 16 - results obtained by calculating the mean and standard deviation for the collected flows

# Bibliography

[BCJX09] M. Bailey, E. Cooke, F. Jahanian, and Y. Xu. A survey of botnet technology and defenses. 2009.

[CJD05] E. Cooke, F. Jahanian, and McPherson D. The zombie roundup: Understanding, detecting, and disrupting botnets. In *SRUTI 05:Steps to Reducing Unwanted Traffic on the Internet Workshop*, 2005.

[Den87] D. E. Denning. An intrusion-detection model. *ieee transactions on software engineering*, 13 no 2:222–232, 1987.

[DGL07] D. Dagon, G. Gu, and C. P. Lee. *A taxonomy of botnets structures*, chapter 8, page 22. 2007.

[DMS92] H. Debar, Becker M., and D. Siboni. A neural network component for an intrusion detection system. 1992.

[FB07] K. M. Faraoun and A. Boukelif. Neural networks learning improvement using the k-means clustering algorithm to detect network intrusions. 2007.

[FHSL96] S. Forrest, S. A. Hofmeyr, A. Somayaji, and T. A. Longstaff. A sense of self for unix processes. 1996.

[GPZL07] G. Gu, R. Perdisci, J. Zhang, and W. Lee. Botminer: Clustering analysis of network traff ic for protocol- and structure-independent botnet detection. 2007.

[GZL08] G. Gu, J. Zhang, and W. Lee. Botsniffer: Detecting botnet command and control channels in network traffic. 2008.

[HFS98] S. A. Hofmeyr, S. Forrest, and A. Somayaji. Intrusion detection using sequences of system calls. 1998.

[KMN$^+$02] T. Kanungo, D. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, and A. Y. Wu. An efficient k-means clustering algorithm: Analysis and implementation. 2002.

[KRH07] A. Karasaridis, B. Rexroad, and D. Hoeflin. Wide-scale botnet detection and characterization. 2007.

[LS98] W. Lee and S. J. Stolfo. Data mining approaches for intrusion detection. 1998.

[LTG$^+$92] T. F. Lunt, A. Tamaru, F. Gilham, R. Jagannathan, C. Jalali, P. G. Neumann, H. S. Javitz, A. Valdes, and T. D. Garvey. A real-time intrusion-detection expert system(ides). Technical report, 1992.

[Mic06] Trend Micro. Taxonomy of botnet threats. page 15, 2006.

[MZ05] A. W. Moore and D. Zuev. Internet traffic classification using bayesian analysis techniques. 2005.

[NSB10] A. Nogueira, P. Salvador, and F. Blessa. A botnet detection system based on neural networks. 2010.

[OKA10] A. J. Oliner, A. V. Kulkarni, and A. Aiken. Community epidemic detection using time-correlated anomalies. 2010.

[RFD06] A. Ramachandran, N. Feamster, and D. Dagon. Revealing botnet membership using dnsbl counter-intelligence. 2006.

[Roj96] R. Rojas. *Neural Networks*. 1996.

[RY08] M. Reiter and T. F. Yen. Traffic aggregation for malware detections. 2008.

[SLWL07] W. T. Strayer, D. Lapsely, R. Walsh, and C. Livadas. *Botnet Detection Based on Network Behavior*. 2007.

[SM07] E. Stinson and J. C. Mitchell. Characterizing bots remote control behavior. In *Botnet Detection Countering the Largest Security Threat*, 2007.

[WDF$^+$03] J. Wang, P. S. Deng, Y. Fan, L. Jaw, and Y. Liu. Virus detection using data mining techniques. 2003.