

Detection of Denial of Service attacks using a naïve Bayesian classifier

Ikt 407 - WebMining

Autumn 2006



HØGSKOLEN I AGDER

Agder University College

Faculty of Engineering and Science

Author(s):

Kristen R. Gravelseter

Richard Imenes

Åsmund Myklevoll

Supervisor(s):

Morten Goodwin Olsen

Version:

Status: FINAL

Pages: 20 (including this page)

Modified date: 2006-11-26

Keywords: <list of up to 5 keywords>

Abstract: This project is a proof -of-concept of self -learning intrusion detection systems. The project goal is to prove that it is possible to make this using a Naïve Bayesian classifier. Through the project we have tested a dataset using Orange, and seen what kind of classification results we can get. This project does not contribute with any new knowledge in the field of self-learning intrusion detection systems, but has acted as part of our education and has given us more knowledge about the subject.

The project has its own blog at ikt407.wordpress.com

License:

This work is licensed under the [Creative Commons Attribution-ShareAlike License](http://creativecommons.org/licenses/by-sa/2.5/). To view a copy of this license, visit <http://creativecommons.org/licenses/by-sa/2.5/> or send a letter to Creative Commons, 543 Howard Street, 5th Floor, San Francisco, California, 94105, USA.

Table of contents

Table of contents	2
Figure overview	3
Table overview.....	3
1.0 Executive summary.....	4
2.0 Introduction.....	5
Overview	5
3.0 Background	7
3.1 Denial of Service (DoS).....	7
3.2 Bayesian Classifiers	9
3.2.1 Bayes Theorem	9
3.2.2 Naïve Bayesian classifier	10
3.2.3 Example of a Bayesian classifier	10
3.3 Orange.....	10
4.0 Problem Description	12
5.0 Requirements	13
6.0 Implementation	14
6.1 Orange implementation.....	14
6.2 Dataset.....	14
7.0 Testing and evaluation	16
7.1 Testing.....	16
7.2 Evaluation	17
7.3 Improvements	18
8.0 Discussion	19
8.1 Project outcomes.....	19
8.2 Evaluation of the overall results	19
9.0 Conclusion	21
Appendix.....	22
Glossary & Abbreviations.....	22

Figure overview

Figure 1 - A typical DDoS attack	8
Figure 2 - Orange model with logical drawing.....	14

Table overview

Table 1 – Results of test	16
Table 2 - Original weight.....	16
Table 3 - Modified weight.....	16
Table 4 – Result of test.....	17
Table 5 – Result of test	17
Table 6 – Result of test.....	17

1.0 Executive summary

In the world of computers there are people who dedicate themselves to creating new ways of attacking other computers. In order to keep up with this development the major security companies put a lot of research into self-learning intrusion detection systems (IDS), and one way of making these self-learning IDS's is by using an algorithm for classification of the network traffic. It would be very useful if the IDS could detect malicious traffic without having to be constantly updated with the signatures of new attacks, but rather recognize the pattern of attacks in such a way that if you are being attacked in a way that has never been done before, your system will classify the traffic as malicious, and thus take action against it.

In this report we look to prove this concept of self-learning IDS's by using a Naïve Bayesian classifier to classify a data set which we have made ourselves, and distinguish between normal traffic, and Denial of Service (DoS) traffic.

The fields we captured for use in the data set are packet source ip, packet source port, packet destination ip, packet destination port, packet size, and packet type.

Our test data set is a relatively simple one, and the success rate we get is 99,16%. Because of the simplicity of our dataset we see that the amount of training required for the classifier is not much. We see that if we reduce the amount of training data from 70% to 10% it does not have any real effect on the results we get. This is probably due to the fact that in our data set, all of the DoS traffic we generated originates from one single host, and thus we could almost classify just based on host address.

We reduce the weight on source and destination address to see how this affect the results we have gotten, but it does not have the expected effect. We then remove source and destination host completely from the equation, and we get an expected drop in classification rate to 91,4%.

We then tried to make our dataset more realistic by reducing the amount of DoS traffic to 1% of the data set. We then saw that the success ratio was around 98% even with heavily reduced amount of training data.

In retrospect we see that our dataset is too simple to get a simulation of real life traffic, but it is more than good enough for our purpose of showing that it is possible to make a self-learning classifier.

2.0 Introduction

Why is it so that a program that is designed to stop unpredictable malicious traffic needs a defined and predictable list of signatures to be able to sort out the bad packages from network traffic?

This question might as well been the trigger for writing this project. Attackers will try to outsmart such programs by designing a new attack that is not defined in the predictable list used by filtering programs. Is there a way to protect e.g. an online service from a Denial of Service (DoS) attack?

As we see it the main weakness for most of the programs designed to stop malicious traffic is that the program has a defined set of signatures. These needs to be updated by the provider before any new forms of attack will be filtered. It is not even certain that a filtering program like the one mentioned will be able to protect an online service from a DoS attack, as the program will use plenty of recourses to filter the malicious traffic from the legitimate traffic. That is, if it is able to detect the nature of the attack at all...

The requirements for the method of sorting traffic that we are looking for are therefore quite high: Not only has the method to be efficient and precise, it – preferably – has to be self-learning. Fortunately there is quite a load of research available on that field, so we have already found a classification algorithm that might do the job: Naïve Bayes classification.

We have based some of our research and solution on a master thesis written by Tor Oskar Wilhelmsen, as we found that his work was close related to our problem.

Since time is sparse we will only give a proof of concept to show that it exists a way of making a filter that not only will be able to classify most of the malicious network traffic, but also will be able to learn. We will use a tool by the name Orange to make the job related to classifying easier and the dataset we use is one we have created. We understand that our discoveries do not rock the world in any fashion, but we have provided useful information for researching further into this subject.

2.1 Overview

In chapter 3 we look at the background for this project. We define DoS attacks and see how the attacks usually are handled in part 3.1. Bayesian Classifiers is also a subject that is covered in 3.2 as well as an example showing how classifiers work. The program we use to make the work related to classifying easier is also mentioned and there is a brief explanation of Orange's features in 3.3.

Detection of DoS attacks using a Naïve Bayesian classifier

The fourth chapter covers our problem description. Here we conclude and limit our work on this subject. Requirements are stated in chapter 5. There are two main requirements that we state: Our success rate on the classifying and the use of Orange.

The next chapter – Chapter 6 – covers the implementation. This explains how we use Orange in our work, which dataset we decided on using and a bit about how we prepared the dataset for Orange use.

In chapter 7 we show some test results and try to evaluate these as we go. Here you can find a few tables that show how well the classifier works. Also, we mention a bit about how we could improve the tests of our classifier.

In the latter part, chapter 8, we discuss the findings and review why we chose the learning environment and classifier as we did and how we might improve the results we got.

Last, but not least, we conclude in chapter 9. Here we look at our work and see if it will affect the research on this subject. The very last part includes references and abbreviations.

3.0 Background

3.1 Denial of Service (DoS)

Denial of Service (DoS) attacks is becoming a common problem to service providers on the net. There are three types of attack that are considered potentially dangerous: Smurf/fraggle, SYN Flood and DNS (Domain Name Service) attacks. Smurf attack is when a single computer sends a forged ping to a network distribution address, thereby getting all of the computers on the network to reply. The source address is of course forged, and the victim with this IP address will endure massive amounts ICMP (Internet Control Message Protocol) reply traffic that needs to be handled. [1]

The other method is called SYN Flood. This occurs when an attacker sends a malformed SYN packages that usually is sent to initiate a connection between the server and the client. The server will never get a response from the client, and while waiting for the connection to time out, the server's resources will be depleted. The DNS attacks is not that big a problem anymore, as the new version of BIND (Berkeley Internet Name Domain) tolerate this problem well. [7]

There is also what we call a DDoS (Distributed DoS) attack. The difference is that the DoS attack is executed from the network of compromised computers, rather than from a single computer. Both DoS and DDoS have in common that they try to render a service unavailable for legitimate users. DDoS is more effective than DoS since the distributed attack is dispatched from many clients that more easily will overpower even the best equipped server. This is not only because the huge amount of requests that the server will receive, but also because the request-part of a protocol is usually consumes less resources than responding to the request. On figure [Fig 1.] we see an illustration of a DDoS attack,

and its massive effects. [6]

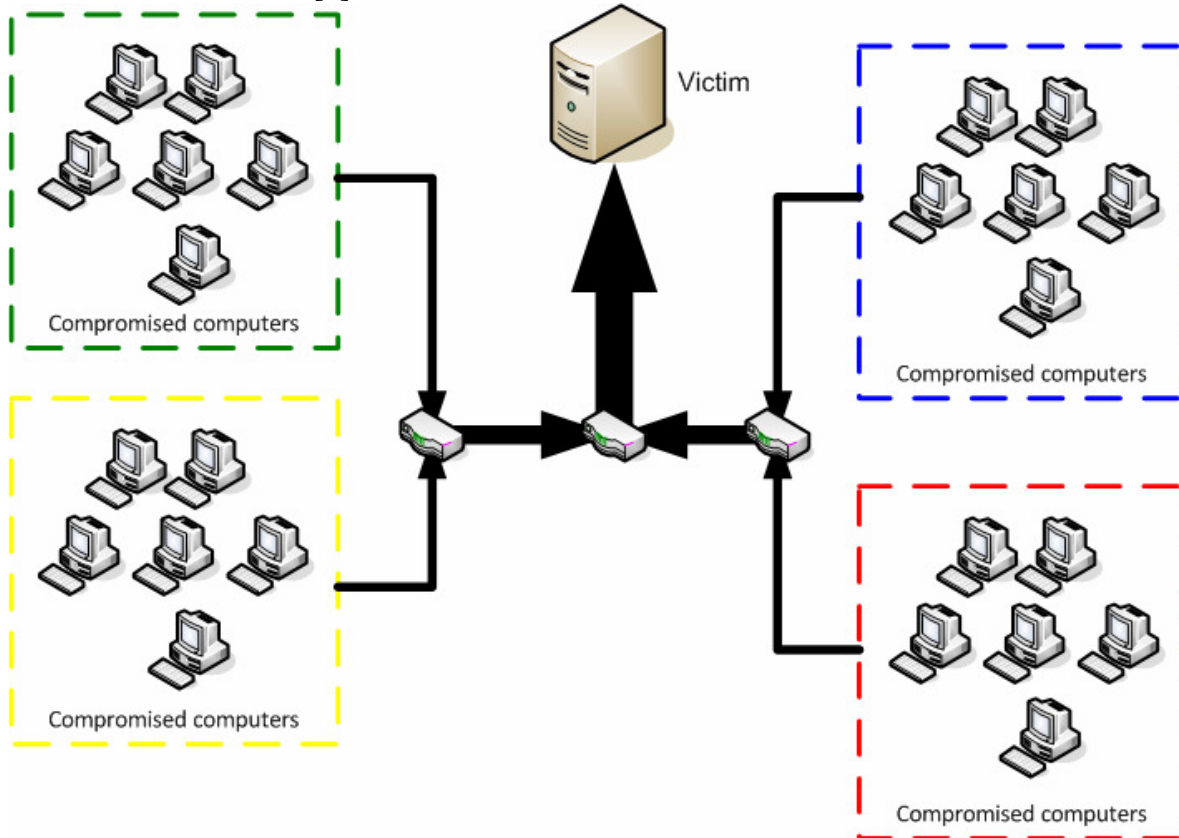


Figure 1 - A typical DDoS attack

You do not have to be a hardcore programmer or a hacker to initiate an attack - anyone who has the time and effort are able to attack. This is a part of the reason why this problem is growing fast. Also, the general public has more bandwidth available now than before, so every infected host will be able to send huge amounts of traffic. Many people take their computer security lightly, thinking that no one would have interest of hijacking their computer - they have nothing to hide. Most people do not regard their lack of computer security as a problem that can affect other services or servers. Logs from 1 600 networks in 2003 shows that intrusion and such is a huge problem - the logs showed as many as 3 million scans per day! [3] The goal for most of this host scanning is to find vulnerable machines to infect.

The cost for such an attack can also be quite big. Some service providers pay for the traffic that storms in at times, and that alone can sum up to be a huge bill. The fact that the service provider is not able to provide the given service to its users can also become a big cost. That's why many service providers would like an easy solution to this increasing problem.

There are quite a few programs that offer some kind of DoS recognition, but most of these programs use "signatures" to compare against. New types of attack will not be detected before the database of the program is updated. If there is a way to make a program recognize the attack without having to compare it to a specific signature, this

would not only be less resource consuming, but also it would mean that one would notice the attack as early as possible.

We see from this that it is no good way to stop a DDoS attack without rendering the service (partial) unavailable for all or many of its users, at least not at the firewall of the service. Even if the malicious traffic is dropped at the firewall the problem with stolen bandwidth would still be an issue. This makes one to realize that the traffic has to be filtered at a point that is closer to a router in close proximity to the source of the attack. The best solution does not only include the service provider (the victim) and its ability to discover DoS traffic, it includes teamwork with the upstream provider as well. It is also very important to have a battle plan ready before the attack occurs – so when the attack is discovered the correct measures can be taken, and the involved people can be contacted as soon as possible. It is ultimately the upstream provider or an ISP (Internet Service Provider) that needs to drop malicious traffic based on simple rules.

Another research project which has been carried through on this faculty (University College in Agder) with connections to our own project is “Selvorganiserende læring av trafik kategorier i Bayesiansk pakkebasert IDS” by Tor Oskar Wilhelmsen. His idea was to look at the relations of the bits in the IP-package and find a way to organize the attributes. Then, using a naïve Bayes classification, he was able to detect 13 of 16 different attacks (including DoS, U2R, R2L and Probe). This kind of classification does not require a lot of computer resources in comparison with “signature”-based discovery of DoS and other malicious traffic. The solution Tor Oskar Wilhelmsen suggests is even at some degree self learning, which is more than we can say about most other solutions. [4]

3.2 Bayesian Classifiers

In order to understand what we are working with we need to have some theoretical background to substantiate our understanding of the subject. What we did was read about Bayes Theorem to understand it, and then about Bayesian classifiers.

3.2.1 Bayes Theorem

Bayes' theorem (also known as Bayes' rule or Bayes' law) is a result in probability theory, which relates the conditional and marginal probability distributions of random variables. [1]

In probability theory there are cases of conditional probability, where the probability of one event depends on the probability of a previous event. Bayes theorem states the relationship between these events.

$$\Pr(A|B) = \frac{\Pr(B|A) \Pr(A)}{\Pr(B)}.$$

The formula tells us the probability of A given B, by multiplying the probability of B given A with the probability of A, and dividing with the probability of B.

3.2.2 Naïve Bayesian classifier

The naïve Bayesian classifier is based on bayes theorem, and is a relatively simple algorithm for machine learning. [2] The Bayesian classifier has proved itself, and according to research it has performances in line with decision tree and neural network classifiers. [3] The Bayesian classifier demands a lot of training data in order to be effective in classification of real data.

3.2.3 Example of a Bayesian classifier

Here is an example of how a naïve Bayesian classifier works. [3]

Suppose your data consist of fruits, described by their color and shape. Bayesian classifiers operate by saying "If you see a fruit that is red and round, which type of fruit is it most likely to be, based on the observed data sample? In future, classify red and round fruit as that type of fruit."

A difficulty arises when you have more than a few variables and classes -- you would require an enormous number of observations (records) to estimate these probabilities.

Naive Bayes classification gets around this problem by not requiring that you have lots of observations for each possible combination of the variables. Rather, the variables are assumed to be independent of one another and, therefore the probability that a fruit that is red, round, firm, 3" in diameter, etc. will be an apple can be calculated from the independent probabilities that a fruit is red, that it is round, that it is firm, that it is 3" in diameter, etc.

In the example we presume that the apple is a metaphor for DoS, which is the type of data we want to classify, and the different attributes associated with the apple is comparable to the different bits we look at.

The tasks where a Bayesian classifier can be implemented include spam filters, document classification and in network traffic classification and also more.

3.3 Orange

Detection of DoS attacks using a Naïve Bayesian classifier

Orange is a component-based data mining application, which includes a lot of techniques for preprocessing, modeling and data exploration. Orange is based on C++ components that are either directly accessed, accessed through Python scripts, or accessed via Orange Widgets.

Orange Widgets is a way for the user to program visually. Instead of writing commands you drag boxes and connect lines between them to make the configuration of what you want Orange to do. This way of setting up the program is very intuitive and easy to get started with. It is also possible to double-click boxes to get more specific tasks for the box, or the function the box represents.

In Orange you can build your own components, or you can use the components that already exist.

Default features of Orange: [4]

- *Data input/output: Orange can read from and write to tab-delimited files and C4.5 files, and supports also some more exotic formats.*
- *Preprocessing: feature subset selection, discretization, and feature utility estimation for predictive tasks.*
- *Predictive modeling: classification trees, naive Bayesian classifier, k-NN, majority classifier, support vector machines, logistic regression, rule-based classifiers (e.g., CN2).*
- *Ensemble methods, including boosting, bagging, and forest trees.*
- *Data description methods: various visualizations (in widgets), self-organizing maps, hierarchical clustering, k-means clustering, multi-dimensional scaling, and other.*
- *Model validation techniques, that include different data sampling and validation techniques (like cross-validation, random sampling, etc.), and various statistics for model validation (classification accuracy, AUC, sensitivity, specificity, ...).*

4.0 Problem Description

In the world of computer communication there are people who thrive on creating new ways to attack other computers. At the rate these new attacks are made, the security companies are having trouble keeping up with the attack-makers. Since this is the case, there is a lot of money being put into research of new ways to detect attacks. And the field where most research time, and money is put into is self-learning classifiers. In this project we aim to prove that it is possible to create self-learning classifiers by classifying DDoS attacks using a naive Bayesian classifier with Orange. We do this to get a deeper understanding of classification, and to learn more about self-learning classifiers. After this project is finished, others can use it as a basis for deeper going projects, like a master thesis or similar.

In this project we will look into the possibility of using a dataset provided by Massachusetts Institute of Technology (MIT) [5], and a dataset created by us, and use Orange [4] to train and classify the packages. We will also use tcpdump for further testing on the data.

This project differs from other similar research projects by just being a proof of concept. We do not intend to revolutionize the world by creating the perfect classifier, but rather just prove to ourselves, and readers of this report that a self-learning classifier is something that is possible to create.

5.0 Requirements

When setting up the requirements for this project we thought through the aspects of the assignment in order to find some values or programs that we absolutely needed to use. What we ended up with is this:

Requirement 1: A success rate of 95%

Why: In order to see that our proof of concept project is really a proof of concept, we wanted the classification success rate as high as possible. When we read Tor Oskar Wilhelmsens report “Selvorganiserende læring av trafikkkategorier i Bayesiansk pakkebasert IDS” [2] we saw that he achieved a success rate around 99%. We therefore set our expectations a little lower, and ended up with 95%. It is also set this high to prove that could be possible to actually implement a classifier in a real system without getting to many faulty classifications.

How to test: In Orange there are statistics for the classification of each dataset you run through, and we will just use the statistics from Orange.

Requirement 2: Use Orange

Why: We set the use of Orange as the data mining tool because of the easy use of visual programming, and because of the statistical outputs we can get from it.

How to test: We don't really need to test Orange, but we will see if it works when we get the classifications, and the statistics that we need to compute our success rate.

6.0 Implementation

6.1 Orange implementation

We have implemented our classification solution in Orange. We have tried several different approaches to our problem, and have found what we believe is the optimal solution. Orange gives us the ability to easily and effectively try out different methods of classification.

Figure 1 is a capture from Orange showing our logical model. It clearly displays how the logical build-up of our classification system works, and how each part is connected to the others. The diagram is to be read from left to right.

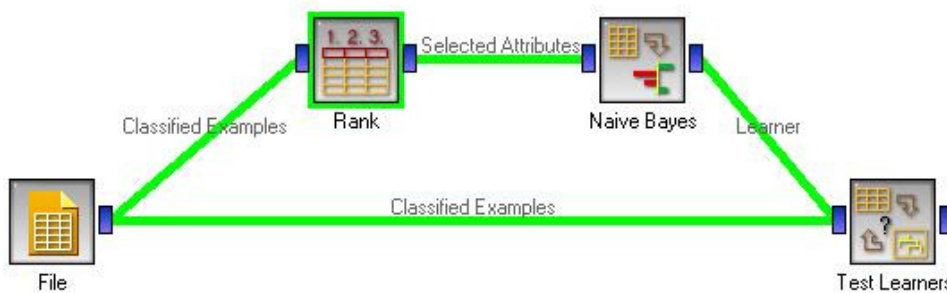


Figure 2 - Orange model with logical drawing

During development and testing several others widgets are in use to control dataflow and get a better overview of the classification process. The above figure is the bare essentials to get the system working.

The file widget is the starting point where the file is selected. The rank object is used to alter the weighting of the different fields to better suit our needs. As we will discuss later, it is of great importance to us to adjust some fields to compensate for the flaws in our dataset. Usually, however, this might not be needed. The Naive Bayes widget is the where the classification takes place, and the Test Learner is where we view the results.

We can also adjust the size of the training set in Test Learners, and further play with different classification methods.

This might look simple, but there is advanced logic under the hood of this model.

6.2 Dataset

Early in the project we where planning on using the MIT DARPA data set. We later discovered that this was difficult, due to the nature of the data dump, and the sheer size of the capture. To be able to use this set properly we would need to import the data dump to an SQL database. It also did not contain important fields directly available to us, such as protocol type and size of the packet. It would be possible to extract this information from the TCPDump files, but we chose to instead pursue another venue.

Detection of DoS attacks using a Naïve Bayesian classifier

We decided instead to generate our own data set with Wireshark [9] (previously known as Ethereal). This path had both advantages and disadvantages, but in the end we were happy with the results.

The advantages are that we have much greater control in which parts of the packet we capture and which parts we are not interested in.

The disadvantages are that we get a somewhat less varied dataset, and the dataset in itself is smaller. We will correct some of the consequences of this, but

For our purposes we captured the following fields:

- Packet source address (IP address)
- Packet source port
- Packet destination address (IP address)
- Packet destination port
- Packet size
- Protocol type

We used a custom written Perl script to parse the Wireshark logs, and prepare the captured data for classification. The non-DoS traffic was generated by capturing regular web surfing, SSH usage, ftp traffic, mail traffic of a home network.

To generate the DoS traffic we attacked the Wireshark sniffer using Nessus [10] with only DoS plug-ins enabled. This provides us with a wide variety of different DoS attacks for our classification. We then merged the two logs and formatted it for Orange.

7.0 Testing and evaluation

7.1 Testing

We achieved excellent results for our first classification with standard settings. In table 1 we are using 70% of the dataset for training, and the other 30% for classification.

Classification Accuracy	Sensitivity	Specificity	Area under ROC curve	Information score	Brier score
0,9916	0,9889	0,9932	0,9992	0,9230	0,0165

Table 1.

When we reduced the training set to 50%, we did not see any reduction in accuracy. In fact the statistics were identical to that of the first test, which indicates that even 50% training set is enough to achieve the maximal results.

If we further reduce the dataset to 10% we still maintain exactly the same results as the two prior tests. Our dataset contains almost 30.000 records, and 3.000 records is obviously enough for the classifier to classify accurately with such an easy-to-classify dataset.

This is most likely due to the nature of our dataset, and its weaknesses. All DoS traffic originates from a single host (our Nessus server). Further almost none of the legitimate traffic originates from that host, so one could almost filter exclusively on source address alone. Also DDoS traffic would come from multiple hosts, so source address, and in some respects destination address would be impossible to use as classification criteria.

To attempt to rectify this error, we use the Rank widget to lessen the weight of source and destination address. Table 2 shows the default weight distribution, and table 3 shows our tweaked weight scheme. We also restore the training rate to the default 70% and we will keep this ratio for the rest of the test period.

Attributes	Weight
Source address	55
Source port	5
Destination address	55
Destination port	5
Size	5
Protocol	29

Table 2 – Original weight

Attributes	Weight
Source address	1
Source port	10
Destination address	1
Destination port	10
Size	30
Protocol	30

Table 3 – Modified weight

Even after our alterations we maintain the same accuracy. We now wish to limit the information available to the classifier. We do this by using a filter that removes both destination and source address completely. We now filter the traffic only based on port,

Detection of DoS attacks using a Naïve Bayesian classifier

size and protocol. This results in an expected drop in classification success that is displayed in table 4. The results are still quite high with such a limited dataset. This is most likely due to the high ratio of DoS to regular traffic.

Classification Accuracy	Sensitivity	Specificity	Area under ROC curve	Information score	Brier score
0,9140	0,8387	0,9579	0,9796	0,7254	0,1020

Table 4.

We now realize that we need to alter the dataset to make it more realistic. We do this by reducing the ratio of DoS traffic to regular traffic, so that we only have 1% DoS traffic left.

We now run the same tests as before to check whether this will change the results of the classifier. The first test with 70% train data and standard settings is displayed in table 5.

Classification Accuracy	Sensitivity	Specificity	Area under ROC curve	Information score	Brier score
0,9893	0,9273	0,9957	0,9993	0,4005	0,0143

Table 5.

This is interesting, because we now see that a reduced set has clearly an impact on the classifiers results. We are still working with an easy-to-classify dataset, but the results still apply to the real world.

We now repeat the test where we reduce the amount of training data available to the classifier to 10%. The results are surprisingly enough the same as the above test, which indicates to us that even 0,1% of the data traffic is enough to classify this data set.

We explore further by once again removing the source and destination from the set. Table 6 shows the numbers from that test.

Classification Accuracy	Sensitivity	Specificity	Area under ROC curve	Information score	Brier score
0,9769	0,8491	0,9899	0,9462	0,3100	0,0436

Table 6.

7.2 Evaluation

What conclusions can we draw from our testing so far? Well, we see clearly that Bayesian classification is well suited for this type of classification.

Obviously a more varied address space in both source and destination for all the traffic would make the classification harder. Real world data would also feature a more singular type of attacks (i.e. when attacks are launched, it is most likely the same attack for all attack nodes). We would also have to factor in far less percentage of DoS attacks to

legitimate traffic. In our dataset the DoS traffic make up 36% of the sample data, while a real-world scenario would probably be far less than 1%.

Our reviewed data set had some drop in accuracy, but nothing in the scale that we where hoping for. This only indicates that our dataset is too easy to classify, even without the source and destination address. This again means that the data Nessus generated for us is very distinctive.

7.3 Improvements

How can we improve our tests? If we increased the dataset with 100.000 more packets of regular, varied traffic and used random generated addresses for source and destination addresses for DoS traffic, we would be closer to real traffic. Then DoS traffic would make up less than 1%, and DoS could not be found by source address alone.

In our review of the data set, we simply reduced the set so that the DoS part made up 1%. Given more time, we could have used a larger data set and more varied traffic. This would have made the DoS traffic less distinctive, and it would also allow for a larger DoS part. A larger DoS part would also give greater variance of DoS data, which in turn would make the DoS traffic harder to classify. This would reduce our accuracy to 95-98%, which should be more realistic.

It is strange for us to seek less accuracy, but we also seek out realism and 99,9% accuracy is not realistic in real world data.

8.0 Discussion

8.1 Project outcomes

We have achieved our initial requirement with flying colours. This proved to be surprisingly easy, all things considered. However the data set which we used for our testing was, as previously mentioned, easy for the classifier. Thus we believe that our results would have been slightly poorer had the data set been closer to reality. However, we are confident that our target of 95% correct classification would have been achieved even then.

Our proof of concept in this project displays clearly what we expected when we started; that Bayesian classifiers are a fast and efficient way to classify large data sets. We attempted several ways to reach our goal. First we attempted to copy Tor Oskar Wilhelm's master thesis of 2005, and his way of classifying the packets. After a while we had to give up on this plan, due to the complexity of his solution.

We then decided to move on to produce our own data set, which was what we probably should have done from the start. As soon as our data set was completed, and our test system in Orange was in place, we achieved staggering results. We had to re-check the results several times to be sure it was accurate.

As mentioned earlier this was mostly due to the simplicity of the data set, but also due to the Bayesian classifiers effectiveness. Later trials clearly prove this point.

If we are to draw conclusions from our experience with Bayesian classifiers, we can clearly see the uses and advantages of this self-learning classifier. This is also reflected in the worldwide use of Bayesian classifiers in everything from spam filters in E-mail clients to DoS detection in network appliances.

8.2 Evaluation of the overall results

So how do our results relate to real world research in traffic classification, and in what way does our work contribute to the community?

When we started up this project we had several choices to make, before we started any work. We had to choose which environment we would do the testing in, which dataset we should use and lastly which classifier we should use.

We began by looking over the different options we had for environment. We evaluated both Orange and YALE [8], and ultimately chose Orange. As our time was limited we needed a stable platform which was quick to develop on. Orange fit that bill perfectly, being more mature and easier to use than YALE (in our opinion).

Detection of DoS attacks using a Naïve Bayesian classifier

Then we had to evaluate which classifier would be best. Our original assignment quoted Bayesian classifiers as the focal point, but we were given a choice to select other classifiers. After careful consideration we confirmed that Bayesian classifiers would be best suited to our needs. We did not run any tests on other classifiers to compare performance, but given more time this would be a natural expansion of the project.

So how real is our tests and can we relate it to real-world scenarios? In principle one could use our method for live or off line evaluation of network traffic. Our tests, however, lack the variety and sheer size of a normal busy network. The consequence of this is that real-world performance for a similar real-time setup would be less accurate.

To make this system perform better, one could combine several classifiers (which would look at different aspects of the packet), and also consider repetitions such as many SYN packets from one or a few hosts without the target confirming them.

9.0 Conclusion

Our project is mainly a proof-of-concept project, and therefore our aim has not been to improve the processes or techniques used in Bayesian classification. We have merely proved the fundamental functionality of the Bayes theorem when applied to network traffic. This has, of course, been done before and in that respect we do not expand on anything that has not been done before.

The manner of our classification, with regards to how we classify traffic is also a rather simple approach. This leads us to conclude that this project does not contribute anything to world, other than that we now have proved that Bayes theorem is sound with regards to network classification.

Appendix

References

- [1] http://en.wikipedia.org/wiki/Bayes'_theorem (26.11.06)
- [2] ”Selvorganiserende læring av trafikkategorier i Bayesiansk pakkebasert IDS” by Tor Oskar Wilhelmsen
- [3] http://www.resample.com/xlminer/help/NaiveBC/classiNB_intro.htm (26.11.06)
- [4] <http://www.ailab.si/orange> (26.11.06)
- [5] <http://web.mit.edu/> (26.11.06)
- [6] http://www.ddosprotection.com/pdf/What_is_DDoS.pdf (19.11.06)
- [7] <http://www.securityfocus.com/infocus/1655> (17.11.06)
- [8] <http://rapid-i.com/> (26.11.06)
- [9] <http://www.wireshark.org/> (26.11.06)
- [10] <http://www.nessus.org/> (26.11.06)

Glossary & Abbreviations

Short definitions of terms used and references to further relevant glossary sources

Term	explanation	URL
DoS	Denial of Service	http://en.wikipedia.org/wiki/Denial_of_Service
DDoS	Distributed Denial of Service	http://en.wikipedia.org/wiki/Denial_of_Service
ICMP	Internet Control Message Protocol	http://www.faqs.org/rfcs/rfc792.html
BIND	Berkeley Internet Name Domain	http://www.isc.org/sw/bind/
IDS	Intrusion Detection System	http://en.wikipedia.org/wiki/Intrusion-detection_system
ISP	Internet Service Provider	http://en.wikipedia.org/wiki/ISP