Combining a Bayesian Classifier with Visualisation: Understanding the IDS

Stefan Axelsson
Department of Computer Science
Chalmers University of Technology
Göteborg, Sweden
sax@cs.chalmers.se

ABSTRACT

Despite several years of intensive study, intrusion detection systems still suffer from two key deficiencies: Low detection rates and a high rate of false alarms.

To counteract these drawbacks an interactive detection system based on simple Bayesian statistics combined with a visualisation component is proposed, in the hope that this lets the operator better understand how exactly the intrusion detection system is operating. The resulting system is applied to the log of a webserver.

The combination proved to be effective. The Bayesian classifier was reasonably effective in learning to differentiate between benign and malicious accesses, and the visualisation component enabled the operator to discern when the intrusion detection system was correct in its output and when it was not, and to take corrective action, re-training the system interactively, until the desired level of performance was reached.

Categories and Subject Descriptors: K.6.5 [Security and Protection]: Unauthorized access

General Terms: Security, Human factors

Keywords: Intrusion detection, Naive Bayesian Classification.

1. INTRODUCTION

A significant problem with intrusion detection systems, is the high number of false alarmsThis is perhaps not surprising when making an analogy with the common burglar alarm. Burglar alarms operate under a very restricted security policy. Any activity whatsoever on the premises is suspicious. Intrusion detection systems on the other hand are active when the computer system is in full operation. There is much benign activity taking place. The analogy with a burglar alarm is apt then, and serve to explain the high number of false alarms. In the shop lifting scenario however, an ordinary burglar alarm would not be appropriate since there would be a multitude of normal, benign activity (the shopkeeper even encouraging this). The shoplifting problem is currently addressed among other things by *surveillance*, i.e. human supervision of the potential shoplifters. The human, using her senses unaided, is too expensive to employ directly, and therefore technology is brought to bear in the form of video cameras, video recorders, etc. Taking this analogy more literally leads to the idea of applying some form of *information visualisation* to the intrusion detection problem and this paper draws on techniques from this field.

Analogously to the equipment utilised to aid the human security operator in the shoplifting scenario, a prototype tool was developed where the state of a Bayesian classifier is visualised to further an understanding by the operator of exactly what the intrusion detection system is 'learning' and how that affects the quality of the output–e.g. in the form of false alarms. To ascertain the effectiveness of the approach an empirical study of the access requests made to a fairly large public webserver is made using it.

The rest of this paper is organised as follows: First intrusion detection is summarised in section 2, and then Bayesian self learning systems is detailed in section 3. The experimental data is discussed in section 4. Section 5 describes the intrusion detection system, and section 6 and section 7 detail the experiment. Related work is presented in section 8. Ideas for future work is presented in section 9 and the paper concluded in section 10.

2. INTRUSION DETECTION

The intrusion detection problem can be viewed from the perspective of traditional detection and estimation theory as consisting of detecting the signal from a malicious process (*hypothesis*) given an input signal that consists of the mixing of the signals of both a malicious process and a benign process, possibly imperfectly observed. However, in contrast with established principles in detection and estimation theory, *signature based* intrusion detection systems (IDS for short) use detection models concentrating on the intrusive signal only with only a rudimentary, undeveloped model of the benign process, and *anomaly based* systems concentrate on the benign model of the signal, ignoring the malicious process. In order to better make a detection one needs explicit models of both the benign and malicious processes under study [5],and our Baysian detector does so.

Anomaly detection systems most often employ some form of automated learning when building their models. As such they often have problems with high false alarm rates, and a potential weakness where the skillful attacker knowing the details of the IDS can slowly change his behaviour over time such that the IDS will learn that the malicious behaviour is in fact 'normal' and never sound the alarm. Signature based systems on the other hand have difficulty with detecting (sometimes trivial) variations of the attacks it knows about. The author conjectures that if the operator of the IDS had some form of insight into the inner workings of the IDS (what it has 'learnt' in a sense) such attacks would have a smaller chance of going undetected.

Automated learning can be roughly divided into two major groups, *directed* and *non directed*. Most IDS fall into the latter category, i.e. they automatically find clusters or other features in the input data and flag outliers as anomalous. Relatively little investigation in *IDS research* has been into the area of directed automated learning systems.

Major problems with all self learning systems are the issues of *over training*, i.e. where the system gains a too specific knowledge of the training set, which prevents it from correctly generalising this knowledge given slightly different stimuli, and *under training* where the system has really seen too few examples on which to base any well founded decision about later stimuli but still classifies as if it had. A goal of our approach is that the visualisation of the inner workings of the IDS will let the operator easily detect instances of over training and under training, and be able to deal with them interactively.

3. NAIVE BAYESIAN DETECTION

We have chosen to implement an IDS based on the principles of Bayesian filtering in the same vein as now popular spam filtering software, popularised by Paul Graham[1].

These simple classifiers operates as follows: First the input is divided into some form of unit which lends itself to being classified as either benign or malicious (in spam classifications typically a piece of email is considered), this unit of division is denoted a *message*. It is the responsibility of the user to mark a sufficient number of messages as malicious/benign beforehand to effect the learning of the system. The system is thus one of *directed* self learning. The message is then further subdivided into tokens—in an email typically the words of the text that makes up the email and various elements of the header. The tokens are scored, such that the score indicates the probability of the token being present in a malicious message, i.e. the higher the relative frequency of the tokens occurrence in malicious messages, relative to its occurrence in benign messages, the more indicative the token is of the message being malicious. The entire message is then scored according to the weighted probability that it is malicious/benign given the scores of the tokens that it consists of.

One can parameterise the scoring of the tokens in a number of ways, we have chosen a simple method that closely follows Paul Grahams presentation:

Let the total number of benign messages seen thus far be denoted by *good*, and the total number of malicious messages be denoted by *bad*. Furthermore let the number of times the token has appeared in benign and malicious messages be denoted by g and b respectively (i.e. if it has appeared twice in the same malicious message that is counted as two occurrences). Then the score of the *token* is calculated as: $score = \frac{b/bad}{b/bad+g/good}$. If both b and g are zero then $score = 0.5$, i.e, if we have not seen the token before, then it is given a neutral score of 0.5, meaning that it is indicative of neither a benign nor a malicious message. The *tokenscore* is furthermore clamped to the range $[10^{-9}, 1-10^{-9}]$, to prevent division by zero when the entire *message* is scored.[2] It should be noted that one doesn't actually mark *tokens* as being benign or malicious, only messages. The score of the tokens is *inferred* from the number of times it occurs in benign and malicious messages. As the dependent probability is never actually calculated (due to efficiency concerns, we would then have to consider the new token given the probability of all preceding tokens, which would lead to a state space explosion) calling this method *Bayesian* is a bit of a misnomer.

The entire message is scored according to the formula (let the scores of the individual tokens the message consists of in order be denoted by p_i respectively, and the total number of tokens in the message by n):

$p_{malicious} = \prod_{i=0}^{n} p_i / (\prod_{i=0}^{n} p_i + \prod_{i=0}^{n} (1-p_i))$.

So the score of the message (i.e. probability the message is bad) is the weighted probability of the probabilities that the tokens the message consists of are indicative of a bad message.

In order to apply this principle to an intrusion detection system, one would typically present it examples of malicious and benign activity and then when the system is trained, present it with unknown input, flagging all messages that scored higher than a set *threshold score* as intrusive, though a more elaborate approach is taken here.

4. THE EXPERIMENTAL DATA

For the experiment, we have chosen to study a webserver access log.

Even though the choice was made to study webserver logs the longer term aim is that the general approach developed here should generalise to other monitored systems. It should be noted that the tool is agnostic in this respect, placing few limitations on the form of the input data.[3]

The author feels compelled to point out that there is a dearth of publicly available corpora suitable for intrusion detection research. To make matters worse the corpora that is most popular (somewhat of a de facto standard) which originated from the Lincoln labs IDS evaluation [6] , is unavailable to us as it is export controlled. The same is true of anomaly based systems with which to compare our results. We feel it would be pointless to compare our approach to a signature based system e.g. Snort ('http://www.snort.org').

The webserver under study serves a university computer science department. At the time of investigation, the server

[1]"A Plan for Spam," 'http://www.paulgraham.com/spam.-html'

[2]A token is thus never considered *perfectly* indicative of a benign nor a malicious message, even though the scores will be referred to as a *perfect* 0.0 or 1.0 for clarity in the remainder of the paper.

[3]That said, lower level, more machine oriented logs may not be the best application of this method. Even when converted to human readable form they require detailed knowledge of e.g. protocol transitions etc. Of course, fundamentally the logs have to make sense to someone somewhere, as any forensic work based on them would otherwise be in vain. Another problem is that of message sequences where the sequence itself is problematic, not one message in itself.

was running Apache version 1.3.26. It was set to log access requests according to the *common* log strategy. The log thus consists of a line based text file with each line representing an single HTTP access request. The *request* field is central. It consists of the request method ('GET', 'HEAD', 'CONNECT', etc), followed by the *path* to the resource the client is requesting, and the method of access ('HTTP 1.0', or 'HTTP 1.1' typically). The *path* in turn can be divided into components separated by certain reserved characters.

The log for the month of November 2002 has previously been studied in detail. The resulting access log contained circa of 1.2 million records. Cutting out the actual request fields and removing duplicates (i.e. identifying the unique requests that were made) circa 220000 unique requests were identified. It is these unique requests that will be studied in the rest of the paper.

The reason the unique *types* of requests are studied instead of the actual request records is that we are more interested in the *types* of attacks that are attempted against us than the particular instance of the attack. This provides a degree of generalisation even in the setup of the experiment as there is no risk of learning any irrelevant features that are then (perhaps) difficult to ignore when trying to detect new instances of the same type of attack later. Note that an entity, i.e. a worm, that lies behind an actual attack often uses several types of attacks in concert. There exists methods for correlating attacks against webservers to find the entity behind them when one already knows of the particular attack requests being made [2]. It should be noted that no detection capability is lost in this way, since knowing the type of attack being performed it is trivial[4] to detect the instances later, should one chose to do so. The choice was made to ignore the result code as many of the attacks were not successful against our system, and the result codes clearly demonstrated this. Ignoring this information actually makes our analysis more conservative (it biases our analysis toward false negatives).

Not all possible attacks against web servers would leave a trace in the access log, e.g. a buffer overrun that could be exploited via a cgi-script accessed through the *POST* request since the posted data wouldn't be seen in the access log. Unfortunately the raw wire data was not available; there is nothing really preventing the use of the IDS on that data, after some post processing. It should be noted however, that few current attacks are of this type, and that there was a multitude of attacks in the access log data with which to test the IDS.

5. VISUALISING A BAYESIAN IDS

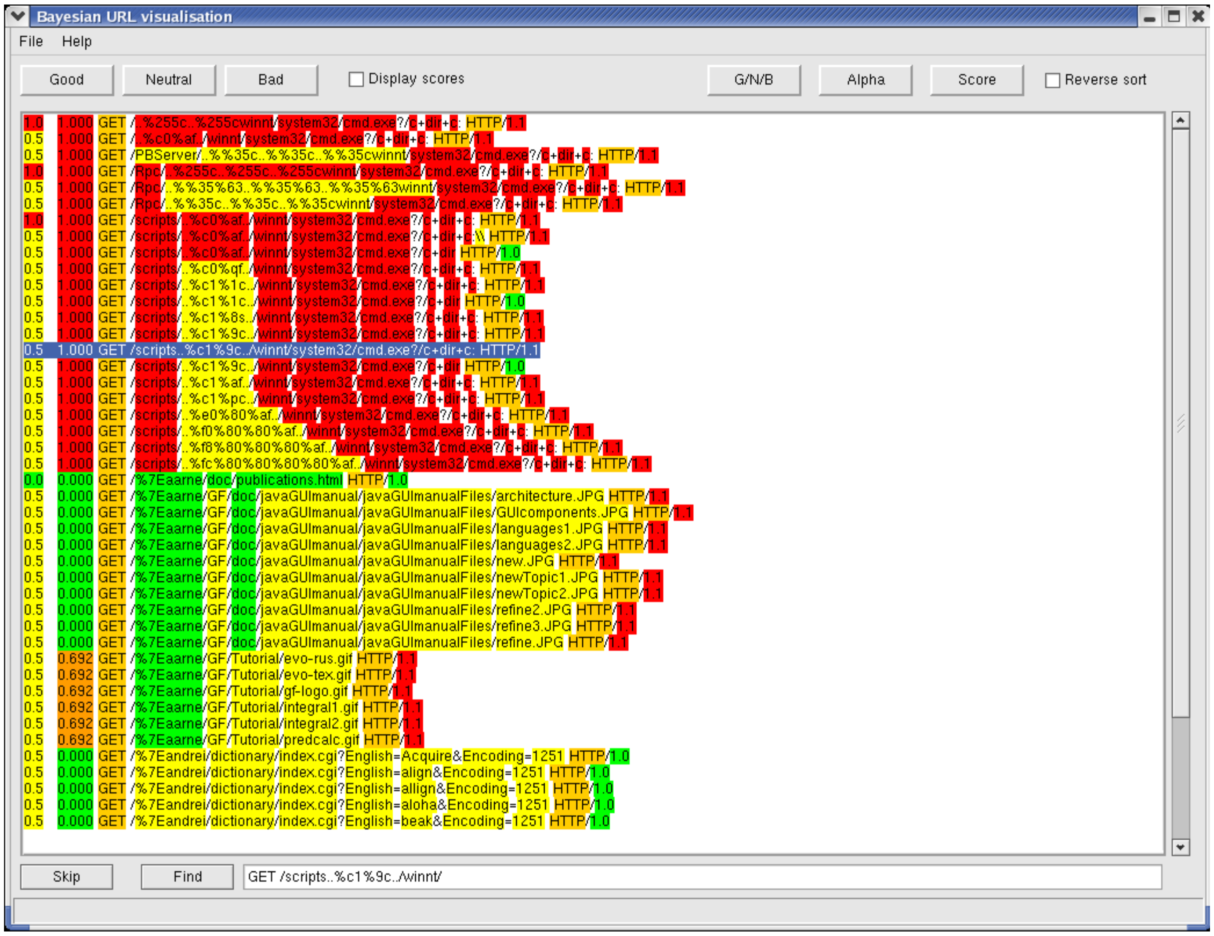
An important problem with self learning systems is that they can be opaque to the user of the system, i.e. it's difficult for the user to ascertain exactly what has been 'learnt' and hence to judge the quality of the output. The problems of not really having the human in the loop when making decisions using decision support systems has been noted in human-machine interaction circles for some time [10, 8]. The operator that doesn't have a relatively correct (or at least consistent) mental picture of the state of the machine he's interacting with will not perform well, probably resorting to ignoring the system he's put to monitor. This problem is complicated in the case of intrusion detection because of the base-rate fallacy [1], i.e. that most alarms will tend not to be a true indication of malicious activity unless the IDS does not have a very low false alarm rate. Hence the correct identification of false alarms is crucial for the operational effectiveness of an IDS. Bayesian self learning systems are not immune to these problems if employed in a naive fashion, i.e. when the system is trained in a 'batch' fashion, where it is first presented with several examples of intrusive behaviour and then several examples of non-intrusive behaviour, to finally be applied to unknown input, delivering only (in the worst case) 'alarm/no alarm' as output to the operator. A natural improvement is to display the score to the operator, but in practise this is only slightly more helpful. As anecdotal evidence we submit the following: When the author first started using the Bayesian spam filter recently added to the *Mozilla* ('http://www.mozilla.org') email client, the filter seemed to learn the difference between spam and non-spam email with surprisingly little training. It wasn't until some rather important email was misclassified as 'spam' that it was realised that what the filter had actually learnt, wasn't the difference between spam and non-spam, but between messages written in English and the author's native tongue. In fairness given a few more benign examples of English messages the system was successfully retrained and was again correctly classifying email, but some rudimentary insight into exactly what the system had learnt would have made us more sceptic of the quality of the classification, even though the classifier seemed to operate perfectly judging by the output.

To that end a (prototype) tool named *Bayesvis* was implemented to apply the principle of interactivity and visualisation to Bayesian intrusion detection. The tool reads messages as text strings and split them up into the substrings that make the tokens. In the first version of the tool URL access requests make up the messages, and they are split according to the URL field separating characters (`;/?:@&=+,$`) but with little modification the tool could accept any input data that lends itself to being split into messages (perhaps marking sessions) and tokens according to its textual representation. Figure 1 is a screen dump of the user interface of the tool.

The 'learning' that a Bayesian system as modelled above does, is encoded in the score of the tokens the IDS use to score the messages. Therefore the scores of the tokens are visualised as their textual representation (black text) on a heatmapped background [9]. A heatmap maps a real number (in our case the probability of the token being indicative of a malicious message, i.e. $p = [0, 1]$) to a colour on the colour wheel, from green via yellow to red that is, the hue of p—in HSV coordinates—is mapped onto the range $[180^o, 0^o]$, fully saturated, and as close to the whitepoint as possible. The total score of the message is visualised in the same manner and also an indicator of whether the user has marked this message as benign or malicious.[5]

[4]The one type of attack that we can think of that would not be detectable is a denial-of-service attack making the same request over and over. Since this would be trivial to detect by other means this is not seen as a significant drawback.

[5]Unfortunately the human eye is much better at discerning between different colours than levels of grey, so a grey scale mapping for the purpose of this presentation is less effective at conveying the nature of our visualisation. Barring the availability of a colour printer, it's suggested that this paper be read in the on-line, colour version.



```
1.0  1.000 GET /..%255c..%255cwinnt/system32/cmd.exe?/c+dir+c: HTTP/1.1
0.5  1.000 GET /..%c0%af../winnt/system32/cmd.exe?/c+dir+c: HTTP/1.1
0.5  1.000 GET /PBServer/..%%35c..%%35c..%%35cwinnt/system32/cmd.exe?/c+dir+c: HTTP/1.1
1.0  1.000 GET /Rpc/..%255c..%255c..%255cwinnt/system32/cmd.exe?/c+dir+c: HTTP/1.1
0.5  1.000 GET /Rpc/..%%35%63..%%35%63..%%35%63winnt/system32/cmd.exe?/c+dir+c: HTTP/1.1
0.5  1.000 GET /Rpc/..%%35c..%%35c..%%35cwinnt/system32/cmd.exe?/c+dir+c: HTTP/1.1
1.0  1.000 GET /scripts/..%c0%af../winnt/system32/cmd.exe?/c+dir+c: HTTP/1.1
0.5  1.000 GET /scripts/..%c0%af../winnt/system32/cmd.exe?/c+dir+c:\\ HTTP/1.1
0.5  1.000 GET /scripts/..%c0%af../winnt/system32/cmd.exe?/c+dir HTTP/1.0
0.5  1.000 GET /scripts/..%c0%qf../winnt/system32/cmd.exe?/c+dir+c: HTTP/1.1
0.5  1.000 GET /scripts/..%c1%1c../winnt/system32/cmd.exe?/c+dir+c: HTTP/1.1
0.5  1.000 GET /scripts/..%c1%1c../winnt/system32/cmd.exe?/c+dir HTTP/1.0
0.5  1.000 GET /scripts/..%c1%8s../winnt/system32/cmd.exe?/c+dir+c: HTTP/1.1
0.5  1.000 GET /scripts/..%c1%9c../winnt/system32/cmd.exe?/c+dir+c: HTTP/1.1
0.5  1.000 GET /scripts..%c1%9c../winnt/system32/cmd.exe?/c+dir+c: HTTP/1.1
0.5  1.000 GET /scripts/..%c1%9c../winnt/system32/cmd.exe?/c+dir HTTP/1.0
0.5  1.000 GET /scripts/..%c1%af../winnt/system32/cmd.exe?/c+dir+c: HTTP/1.1
0.5  1.000 GET /scripts/..%c1%pc../winnt/system32/cmd.exe?/c+dir+c: HTTP/1.1
0.5  1.000 GET /scripts/..%e0%80%af../winnt/system32/cmd.exe?/c+dir+c: HTTP/1.1
0.5  1.000 GET /scripts/..%f0%80%80%af../winnt/system32/cmd.exe?/c+dir+c: HTTP/1.1
0.5  1.000 GET /scripts/..%f8%80%80%80%af../winnt/system32/cmd.exe?/c+dir+c: HTTP/1.1
0.5  1.000 GET /scripts/..%fc%80%80%80%80%af../winnt/system32/cmd.exe?/c+dir+c: HTTP/1.1
0.0  0.000 GET /%7Eaarne/doc/publications.html HTTP/1.0
0.5  0.000 GET /%7Eaarne/GF/doc/javaGUImanual/javaGUImanualFiles/architecture.JPG HTTP/1.1
0.5  0.000 GET /%7Eaarne/GF/doc/javaGUImanual/javaGUImanualFiles/GUIcomponents.JPG HTTP/1.1
0.5  0.000 GET /%7Eaarne/GF/doc/javaGUImanual/javaGUImanualFiles/languages1.JPG HTTP/1.1
0.5  0.000 GET /%7Eaarne/GF/doc/javaGUImanual/javaGUImanualFiles/languages2.JPG HTTP/1.1
0.5  0.000 GET /%7Eaarne/GF/doc/javaGUImanual/javaGUImanualFiles/new.JPG HTTP/1.1
0.5  0.000 GET /%7Eaarne/GF/doc/javaGUImanual/javaGUImanualFiles/newTopic1.JPG HTTP/1.1
0.5  0.000 GET /%7Eaarne/GF/doc/javaGUImanual/javaGUImanualFiles/newTopic2.JPG HTTP/1.1
0.5  0.000 GET /%7Eaarne/GF/doc/javaGUImanual/javaGUImanualFiles/refine2.JPG HTTP/1.1
0.5  0.000 GET /%7Eaarne/GF/doc/javaGUImanual/javaGUImanualFiles/refine3.JPG HTTP/1.1
0.5  0.000 GET /%7Eaarne/GF/doc/javaGUImanual/javaGUImanualFiles/refine.JPG HTTP/1.1
0.5  0.692 GET /%7Eaarne/GF/Tutorial/evo-rus.gif HTTP/1.1
0.5  0.692 GET /%7Eaarne/GF/Tutorial/evo-tex.gif HTTP/1.1
0.5  0.692 GET /%7Eaarne/GF/Tutorial/gf-logo.gif HTTP/1.1
0.5  0.692 GET /%7Eaarne/GF/Tutorial/integral1.gif HTTP/1.1
0.5  0.692 GET /%7Eaarne/GF/Tutorial/integral2.gif HTTP/1.1
0.5  0.692 GET /%7Eaarne/GF/Tutorial/predcalc.gif HTTP/1.1
0.5  0.000 GET /%7Eandrei/dictionary/index.cgi?English=Acquire&Encoding=1251 HTTP/1.0
0.5  0.000 GET /%7Eandrei/dictionary/index.cgi?English=align&Encoding=1251 HTTP/1.0
0.5  0.000 GET /%7Eandrei/dictionary/index.cgi?English=allign&Encoding=1251 HTTP/1.0
0.5  0.000 GET /%7Eandrei/dictionary/index.cgi?English=aloha&Encoding=1251 HTTP/1.0
0.5  0.000 GET /%7Eandrei/dictionary/index.cgi?English=beak&Encoding=1251 HTTP/1.0
```

```
GET /scripts..%c1%9c../winnt/
```

Figure 1: The *Bayesvis* tool.

Once the user has visual access to the internal state of the classifier and hence can start to form an opinion of the learning process it is tempting to let the user interactively guide the learning process, in our case by being able to mark messages on screen as either malicious or benign. (Keyboard shortcuts are of course provided for all controls to speed up the interaction). In order for this to be practical, the experience must be seamless; ideally the user should not experience any delay between her action and the time the result is displayed on the screen. With a few notable exceptions this requirement has been met and the updating of the state of the messages is instantaneous on reasonably current hardware.

In order to present the ideas embodied in the prototype we give a quick presentation of the user interface as *user interaction* is its *raison d'être*. The user interface can be divided into a few major groups roughly corresponding to the controls from top left to right, top to bottom in figure 1.

Saving/loading Via the file menu, the user can save and load a session, but more importantly append new messages (imported as text files) to the end of the current session. The user can also clear the actual messages from the current session, but keeping the tokens and their scores. This enables the user to append new data for classification, without having the display cluttered by the training data.

Marking messages The main window of the display lets the user select messages by left clicking on them,[6] and marking them as either *good*, *bad*, or *neutral*. By default the messages are marked as neutral when first imported. The display is divided into three columns. The first contains a marker that display the state of the message: 0.0, 0.5 and 1.0 depending on the message being marked *good*, *neutral* or *bad* respectively on a heatmapped background. The intended mnemonic is the score the resulting tokens would have, were they part of only one message of the indicated type. The second field is the Bayesian score of the message (also on a heatmapped background), indicating the relative *'badness'* of the message as a whole. The third column fills the rest of the horizontal screen estate and consists of a heatmapped display of the tokenized message. The characters that separate the tokens, and hence are not part of the scoring process (they have no score of their own) are also displayed, but on a white background, which serves to both separate the heatmapped tokens from each other visually, but also provides the original data, without fooling the user into thinking that

[6] A range of messages can also be selected by click dragging or shift clicking, which is useful when we're training the system on a large corpora of already know malicious accesses as we are in this paper.

the separating characters are somehow part of the detection process. The user can also display the actual scores of the tokens in curly braces after the tokens themselves.

Sorting The user can opt to sort the messages alphabetically (optionally in reverse order), but perhaps more interesting is the ability to sort according to message score. Since this tool provides a visual display of the scores in descending order, a cut-off score as in an anomaly based IDS has not been implemented. Instead the user can sort messages according to score and view them in order, deciding herself when she has reached a level that is uninteresting. The last sorting option is the option to sort according to the marking of the messages, with the ordering *good* < *neutral* < *bad*. This is useful during a training or scoring session to quickly find misclassifications (messages with a *good* (green) score that one has marked *bad* (red) will stand out visually among the correctly classified messages). The sorting functions in general and the sorting of scores in particular is the one exception to the instantaneous response that the rest of the tool provide, though it's still reasonable.

Searching At the bottom of the screen is an ordinary sequential search function. More interesting is the *skip* capability used for skipping similar messages, especially when dealing with semi tree like data as is done here.

In figure 1, a few examples of bad and good access requests have been loaded. The user has marked three malicious requests as malicious (which can be seen in the left most column) and one request as benign. As a result all malicious requests have been correctly classified as malicious (they all have a perfect 1.0 score), and most of the benign requests have been marked as benign. A few toward the bottom of the page still have a high score (having a score of 0.692), and the next step would be to mark the first of them as benign and see how that would influence the rest of the misclassified requests.

It is interesting to note that had a batch oriented system been trained with these examples and just the scores been observed we could well have been pleased thus far as all the other benign requests have a perfectly benign score of 0.000. However, when looking at the heatmap of the tokens of the last requests it becomes clear that the reason for the score is the perfect 0.0 score for the token '1.0' that dominates the score of the request as a whole. As it happens, in the requests the system was trained on, the token '1.0' appears once, in a good message, and never in a bad message, and this serves to give it a perfectly *benign* score. As the system has never seen any of the other tokens in the requests they default to a score of 0.5, which is to say that they are not indicative of anything. To the human analyst using the tool, it's abundantly clear that the last requests here are correctly classified more by coincidence than anything else. The system doesn't really have enough input yet to say with any reasonable degree of certainty that these requests are benign, and more training is called for. Contrast this with the malicious requests. As it happens just marking the first request in figure 1 correctly classified all the malicious requests. In this case it is because of the tokens of the 'payload', i.e. the tail of the request that tries to execute a command interpreter on MS Windows operating systems (see section 6 for more details of this type of attack). A few more requests were marked to increase the level of training of the tokens that precede the payload. In this case it is quite apparent to the operator that the detection is of a higher quality given the training set, since the tokens that are marked are quite significant given the type of flaw that is being exploited. In this small example, the strengths and weaknesses of the learning process become visually apparent, and the operator can respond interactively to correct the instances of *under training* seen, in doing so receiving immediate feed back ('click-by-click' literally) and taking into account the new state of the IDS, before performing additional updates. This would not be true of a more traditional IDS working along the same principles. Unfortunately a lack of space and the static nature of a written presentation, prevents of from providing more insight into the interactive nature of this process than possible here.

Bayesvis will be made available under the General Public License.

6. THE TRAINING DATA

The author has previously gone thorough the November 2002 access log by hand, classifying each of the 216292 unique access request for the purpose of intrusion detection research.[7]

It was decided to classify the accesses into two categories, *suspect* and *intrusive*. The reason for using a *suspect* class is that since this is data from the field and we do not know the intentions of the entity submitting the request it is sometimes difficult to decide whether a request is the result of an intrusive process, the result of a flaw in the software that submitted it or a mistake by its user. Also, some accesses are just plain peculiar (for want of a better word), and even though they are probably benign, they serve no obvious purpose. Philosophically, the *suspect* class, consists of accesses that we don't mind if they are brought to the attention of the operator. But on the other hand, as they are not proper indications of intrusions, we accept that they may not be reported as intrusive.

The *intrusive* class was further subdivided into seven different subclasses that correspond to metaclasses (i.e. each of these metaclasses consists of several types of attacks, each of which may be a part of one or several *instances* of attacks) of the attacks that were observed:

Formmail attacks The department was subjected to a spam attack, where spammers tried to exploit a commonly available web mail form to send unsolicited mail via our server. This type of attack stood out, with many different requests found.

Unicode attacks These are attacks against the Microsoft IIE web server, where the attacker tries to gain access to shells and scripts by providing a path argument that steps backward (up) in the file tree and then down into a system directory by escaping the offending '\..\' sequence in various ways.[8] Many variations on this scheme are present in the log data.

[7]An *extremely* tedious task...

[8]See e.g. 'http://builder.com.com/5100-6387_14-1044883-2.html'.

Proxy attacks The attacker tried to access other resources on the Internet, such as web servers or IRC servers *via* our web server, hoping that it would be misconfigured to proxy such requests. We suppose that this is an attempt to either circumvent restrictive firewalls, or more likely, to disguise the origin of the original request, making tracking and identification more difficult.

Pathaccess attacks These are more direct attempts to access command interpreters, cgi scripts or sensitive system files such as password files. A major subclass here is trying to access back doors known to be left behind by other successful system penetrations (by worms for example). Also configuration files of web applications (e.g. web store software) was targeted. Attempts to gain access to the configuration files of the webserver itself were also spotted.

Cgi-bin attacks Attacks against cgi scripts that are commonly available on web sites and may contain security flaws. We believe the availability of several cgi script security testing tools to be the reason for the variety of cgi probes present in our data.

Buffer overrun Only a few types of buffer overruns were found in our log data. All of these are known to be indicative of worms targeting the Microsoft IIS web server.

Misc This class contains seven accesses we are fairly certain are malicious in nature, but which don't fit in any of the previous classes. They are various probes using the *OPTIONS* request method, and a mixture of *GET* and *POST* requests that target the root of the file system. Searching available sources it's been impossible to find any description of exactly which weaknesses these requests target.

In summary, table 1 details the number of different types of access requests in the training data.

Access meta-type	Unique requests
Formmail	285
Unicode	79
Proxy	9
Pathaccess	71
Cgi-bin	219
Buffer overrun	3
Miscellaneous	7
Total attack requests	673
Normal traffic	215504
Suspect	115
Total requests	216292

Table 1: Summary of the types of accesses in the training data.

7. THE EXPERIMENT

The choice was made to train the system on the November log with the identified weaknesses mentioned above[9] and then to evaluate the resulting IDS on the logs for the following months. Examining the logs for the months following November, i.e. December through February they contain on the same order of number of unique requests in themselves, but it turns out that many of these requests are similar to the ones in the November file. However the number of accumulated previously unseen requests for the following months fall off nicely: November 200000, December 87000, January 66000 and February 40000. In addition, studying the requests themselves, many of these only differ in a single component compared to their November counterparts, and hence ought to be easily dispensed with. Thus the number of unique requests that have to be processed decreases nicely as more knowledge about our particular web server is accumulated. As we are only interested in the type of attack, the system will only be tested on the reduced logs where previously seen requests have been filtered out.[10]

Since an interactive tool with feedback is tested, several possible strategies for training presents themselves. A strategy was chosen that is believed to be biased toward *detection*, i.e it will result in as high a detection rate as possible at the cost of more false positives. The strategy is to mark all the previously identified malicious requests as malicious and then mark the false positives as benign until there are no obvious ones left. We name this strategy: *Train until no false positives*. The cut-off score for the URL is set at 0.5 (which is conservative), i.e. a score above 0.5 for a benign access request is considered a false positive. This strategy is in contrast with a strategy that would add more examples of benign activity by actively searching for them and marking them as benign, even though they may not have a score that would make them false positives in our eyes.

Figure 2 shows a detail of a step in the early phases of the training where all the attacks and suspect accesses have been added and marked, but the operator have yet to perform much in the way of correcting false alarms. As seen in the picture, all the accesses are either yellow or red, with tokens such as *GET* being highly indicative of a malicious access. It's not difficult to realise that this would probably not hold for a sufficiently trained IDS, as the majority of all requests are *GET* requests. Hence, the operator starts by marking a few of the benign accesses (one is selected ready for marking in the picture). When a few accesses have been marked the operator re-score, re-sort and repeat the process, until the false positive rate is at an acceptable level, according to the strategy described above. To give an indication; for the November data set it turns out that only 325 accesses need be marked before there are no false positives.[11] This should

[9]It is perhaps unreasonable to assume that every operator of such a tool should do their own security evaluation to acquire training data, but of course nothing prevents training the system on malicious data made available by an external expert, much like intrusion signatures for signature based IDS are typically 'subscribed' to from an external provider.

[10]The reduction itself was performed by judicious use of the *sort*, *uniq*, and *comm* commands.

[11]We feel compelled to point out that the time taken to accomplish this task is of course trivial. If the user is to have any hope of evaluating the output of *any* IDS, then she shouldn't have to spend more than 10 seconds per access request at the very most (probably much less) which means

be contrasted with the total number of malicious/suspect accesses marked ($673 + 115 = 788$) and the total number of benign accesses (215504). Thus, only a small fraction of the benign accesses need be marked as benign for the false positive rate to reach acceptable levels for this data set.[12] Due to the nature of Bayesian classifiers, this does not result in perfect training, three accesses have a score above 0.5 even though they have been marked as *benign* as they are short and contains suspect tokens only. It should be noted that it's fully expected that the false positive training is somewhat fragile, i.e. the system will not give the benefit of the doubt to new access requests, that even though they are benign are sufficiently dissimilar to the ones marked, as the system does not have a great deal of benign data from which to generalise any notion of benign accesses.

The evaluation consists of erasing the training data, saving the tokens with their respective scores and loading the access requests for the next month. Then the accesses are sorted according to URL score and the URLs with a score surpassing our threshold 0.5 is judged for false positives, and the ones with a lower score for false negatives.

Indeed as suspected, latter results (see figure 3) show some false alarms. Here the visualisation of the internal state of the IDS displays its strengths. We see that similar '~andrei' URLs have clearly been marked as benign some time earlier, as much of them are green, but a few instances of the tokens (in this case input to a cgi-script that translates phrases between English and Russian) must have been part of malicious accesses earlier, since they have a perfect score of 1.0, being thought to be highly indicative of malicious accesses. In this case, the majority of the URL consists of benign tokens, and the relatively low score (most proper alarms have a perfect score of 1.0) makes it clear that these are in fact false alarms. As it happens, marking just a handful of these accesses as benign (containing the tokens: *root*, *not* and a few others) suffices to bring the score of these requests well below the threshold. This process is simplified by the instantaneous update of the display. As the first URL is marked (bringing down the score of the *not* token), all other URLs that contain that token are immediately updated, with their corresponding total score. The operator then chooses the next URL that hasn't been affected and mark that one, receiving instantaneous feedback on how that affects the rest of the false alarms. This is a level of interactivity that the author has not seen in any other IDS tool, though it's unfortunately difficult to do justice to in this presentation.

A third example of the detection process is given in figure 4, where the generalisation capabilities of the IDS are demonstrated. Here we see several examples of Unicode type attacks that have not been seen beforehand, which is illustrated by the number of yellow (i.e. not previously seen) tokens. Despite this all the attacks are correctly classified as malicious, since they contain the typical Unicode 'payload' or a variation thereof. The IDS manages to generalise the learnt detection capability for this type of attack (this turns out to be true for the other classes as well) and it's easy for the operator to convince herself that these alarms are genuine, as they contain several highly significant suspicious tokens. To give an indication of the generalisation capability; in the January data alone Bayesvis detected on the order of 200 generalised Unicode attacks.

While a summary of the performance of the Baysian detector itself fails to capture the interactive aspects of the intrusion detection tool, table 2 contains the approximate counts of the instances of true and false alarms and the *suspect* accesses that were classified as benign. As the system was progressively retrained on the false alarms in the files in question the quality of the detection increased. It was always relatively easy to identify false alarms as these typically had relatively many tokens of a benign or neutral nature, with only one or two indicative of maliciousness, hence the number of false alarms as seen in table 2 doesn't say as much about the ease (or difficulty) with which these could be identified. The only exception to this rule were the access requests that consisted of only one or two tokens in total. If these tokens happen to be part of a malicious request, then marking and re-scoring would not tend to change the status of the misclassified request as a whole, since there simply wasn't enough data to work with. If the request consists of only one, biased token, Bayesian classification cannot do much. This is of course a problem for all such classifiers, that becomes readily apparent from the visualisation of the requests. As conjectured the detection rate was impressive, with no missed true attacks, though it should be pointed out that the analysis performed was *not* as thorough as that which lead to the Nov training data. The mis-classifications, i.e false negatives, that did crop up where all in the *suspect* class, and in line with the discussion in section 6 they are not considered missed attacks. In summary we've processed access logs containing close to 5 million access requests[13] (divided onto more than 400000 unique types of accesses) in two to three hours including the one hour to initially train the system (discounting the time taken to find the malicious examples). This time does not differ to a substantial degree from the time the user would have to spend on going through the output of a traditional IDS with similar performance as our Bayesian detector. Unfortunately lack of space precludes us from giving a fuller, more qualitative, treatment that the subject perhaps deserves.

8. RELATED WORK

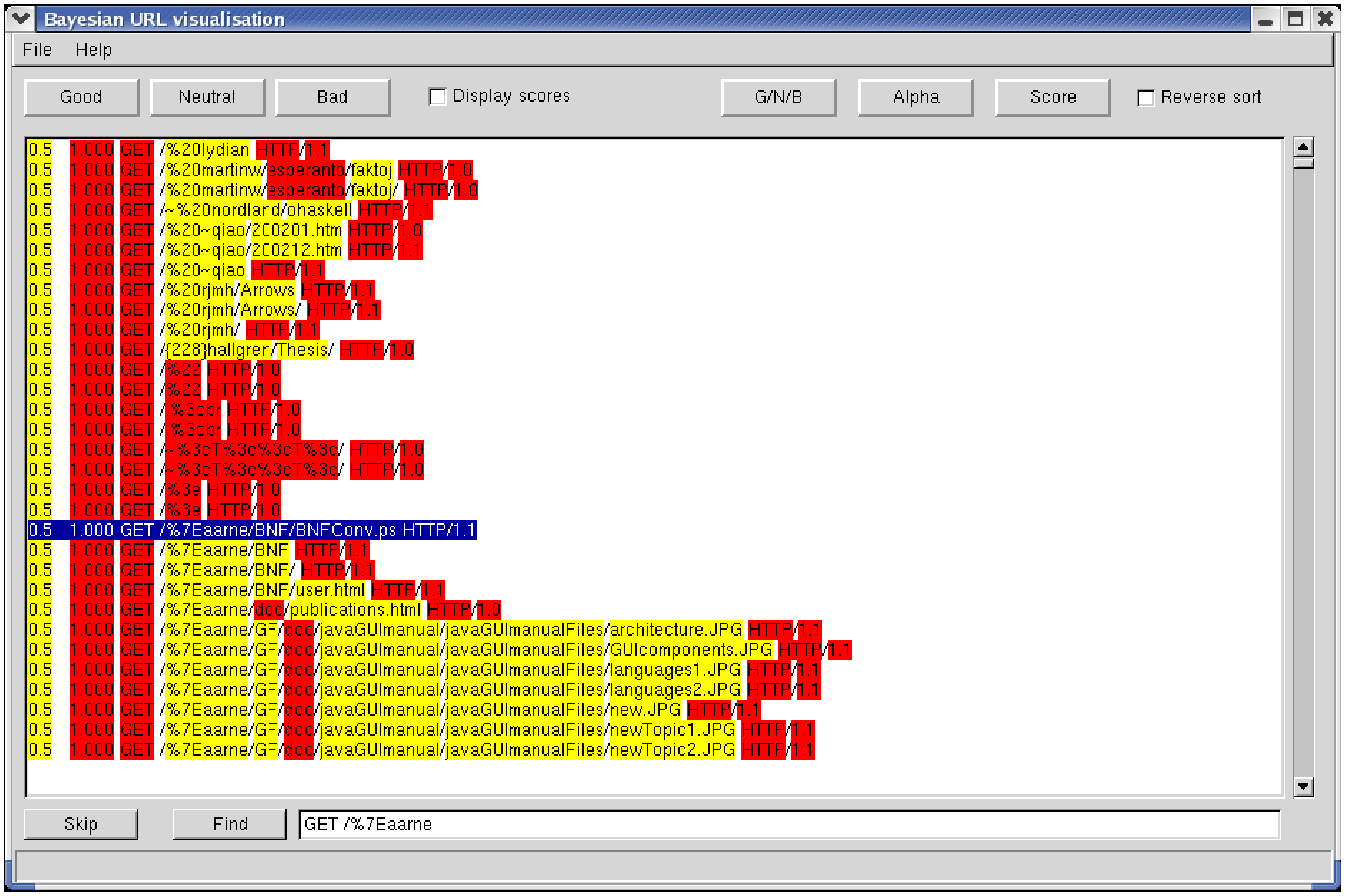
The idea of bringing visualisation to intrusion detection was first published in some years ago and a body of work has started to collect. As such the scope has to be limited to the works that combine some form of self learning detection and visualisation, excluding work in other areas.

A small subfield (e.g. [7]) of anomaly detection and visualisation has arisen through the application of self-organising maps (also called Kohonen maps) [4] to intrusion detection. The question of visualisation arises because the Kohonen map itself is a visual representation of an underlying neural network model. The work cited above shares the characteristic that they all build some neural network model of

that the training would take less than one hour.

[12]Even though it is less likely that we could rely on external security knowledge for the training on benign data than on malicious data, as the benign data is by its nature site specific, this is not as problematic as the site operator *must* have an idea of what data the site provides. One also shouldn't discount the possibility that there is some potential for crossover between benign data for different site due to e.g. directory structures, templates etc. being similar for similar server software in use at different sites, and thus external benign data might still make useful training data.

[13]This is a realistically sized example, though it cannot measure up to the likes of e.g. *Google*.



```
0.5   1.000 GET /%20lydian HTTP/1.1
0.5   1.000 GET /%20martinw/esperanto/faktoj HTTP/1.0
0.5   1.000 GET /%20martinw/esperanto/faktoj/ HTTP/1.0
0.5   1.000 GET /~%20nordland/ohaskell HTTP/1.1
0.5   1.000 GET /%20~qiao/200201.htm HTTP/1.0
0.5   1.000 GET /%20~qiao/200212.htm HTTP/1.1
0.5   1.000 GET /%20~qiao HTTP/1.1
0.5   1.000 GET /%20rjmh/Arrows HTTP/1.1
0.5   1.000 GET /%20rjmh/Arrows/ HTTP/1.1
0.5   1.000 GET /%20rjmh/ HTTP/1.1
0.5   1.000 GET /{228}hallgren/Thesis/ HTTP/1.0
0.5   1.000 GET /%22 HTTP/1.0
0.5   1.000 GET /%22 HTTP/1.0
0.5   1.000 GET /.%3cbr HTTP/1.0
0.5   1.000 GET /.%3cbr HTTP/1.0
0.5   1.000 GET /~%3cT%3c%3cT%3c/ HTTP/1.0
0.5   1.000 GET /~%3cT%3c%3cT%3c/ HTTP/1.0
0.5   1.000 GET /%3e HTTP/1.0
0.5   1.000 GET /%3e HTTP/1.0
0.5   1.000 GET /%7Eaarne/BNF/BNFConv.ps HTTP/1.1
0.5   1.000 GET /%7Eaarne/BNF HTTP/1.1
0.5   1.000 GET /%7Eaarne/BNF/ HTTP/1.1
0.5   1.000 GET /%7Eaarne/BNF/user.html HTTP/1.1
0.5   1.000 GET /%7Eaarne/doc/publications.html HTTP/1.0
0.5   1.000 GET /%7Eaarne/GF/doc/javaGUImanual/javaGUImanualFiles/architecture.JPG HTTP/1.1
0.5   1.000 GET /%7Eaarne/GF/doc/javaGUImanual/javaGUImanualFiles/GUIcomponents.JPG HTTP/1.1
0.5   1.000 GET /%7Eaarne/GF/doc/javaGUImanual/javaGUImanualFiles/languages1.JPG HTTP/1.1
0.5   1.000 GET /%7Eaarne/GF/doc/javaGUImanual/javaGUImanualFiles/languages2.JPG HTTP/1.1
0.5   1.000 GET /%7Eaarne/GF/doc/javaGUImanual/javaGUImanualFiles/new.JPG HTTP/1.1
0.5   1.000 GET /%7Eaarne/GF/doc/javaGUImanual/javaGUImanualFiles/newTopic1.JPG HTTP/1.1
0.5   1.000 GET /%7Eaarne/GF/doc/javaGUImanual/javaGUImanualFiles/newTopic2.JPG HTTP/1.1
```

```
GET /%7Eaarne
```

Figure 2: False positives during the training phase.

Month	Unique requests	True Alarms	False alarms	False negative suspect
December	87000	700	20	15
January	66000	560	40	20
February	40000	240	10	10

Table 2: Summary of the results of the experiment (approximate values).

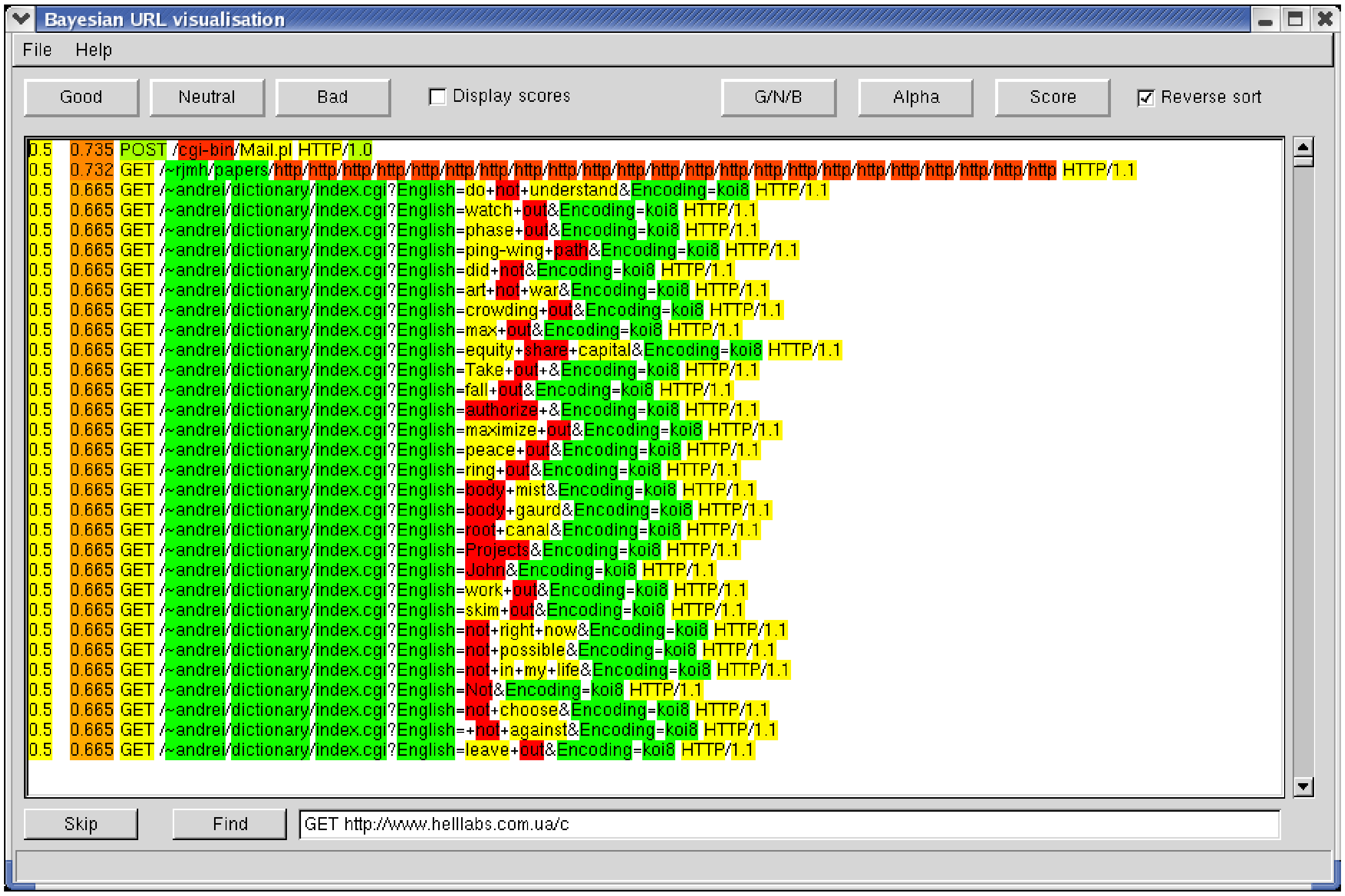
network traffic or host data and then present the resulting two dimensional scatter plot to the user. The scatter plot typically illustrates various clusters within the data. A problem here is that the interpretation of the plot is known to be quite tricky [4].

The question of how the visualisation of a self-organising map furthers the understanding of the security situation, elemination of false alarms and understanding of the nature of alarms is interesting, so there is room for more work on that aspect in this field.

Girardin et. al. [3] also use self-organising maps, but stresses the link to the human operator. Also other visualisation methods is used in addition to the self-organising map, using the self-organising map as an automatic clustering mechanism. They report on successful experiments on data with known intrusions. Their approach differs from our approach in that they use connection statistics etc. from TCP/IP traffic as their input data. While they study parameters of TCP/IP connections, they don't study the data transferred. A higher level protocol (HTTP) is studied here, and the particulars of the connections themselves is abstracted away. They don't illustrate the effects their visualisation have on the understanding of the operator of the security situation, at least not to the degree done here.

These approaches all differ from ours in that neither argue the effect the visual presentation has on the understanding of the security result by the user of the system. Furthermore they all use the raw data as input, not divided into categories (i.e. *accesses* in our case), and in doing so detect instances of attacks, but not necessarily categories of attacks as done here.

A quick survey of the available commercial intrusion detection systems was also made. While many commercial intrusion detection systems claim to support visualisation, the author only managed to find three that uses any degree of visualisation in our sense of the word. We only managed to obtain documentation on one of them: *CA Network Forensics* (previously *Raytheon Silentrunner*, 'http://www.silent-



```
0.5   0.735 POST /cgi-bin/Mail.pl HTTP/1.0
0.5   0.732 GET /~rjmh/papers/http/http/http/http/http/http/http/http/http/http/http/http/http/http/http/http/http/http/http/http/http/http/http HTTP/1.1
0.5   0.665 GET /~andrei/dictionary/index.cgi?English=do+not+understand&Encoding=koi8 HTTP/1.1
0.5   0.665 GET /~andrei/dictionary/index.cgi?English=watch+out&Encoding=koi8 HTTP/1.1
0.5   0.665 GET /~andrei/dictionary/index.cgi?English=phase+out&Encoding=koi8 HTTP/1.1
0.5   0.665 GET /~andrei/dictionary/index.cgi?English=ping-wing+path&Encoding=koi8 HTTP/1.1
0.5   0.665 GET /~andrei/dictionary/index.cgi?English=did+not&Encoding=koi8 HTTP/1.1
0.5   0.665 GET /~andrei/dictionary/index.cgi?English=art+not+war&Encoding=koi8 HTTP/1.1
0.5   0.665 GET /~andrei/dictionary/index.cgi?English=crowding+out&Encoding=koi8 HTTP/1.1
0.5   0.665 GET /~andrei/dictionary/index.cgi?English=max+out&Encoding=koi8 HTTP/1.1
0.5   0.665 GET /~andrei/dictionary/index.cgi?English=equity+share+capital&Encoding=koi8 HTTP/1.1
0.5   0.665 GET /~andrei/dictionary/index.cgi?English=Take+out+&Encoding=koi8 HTTP/1.1
0.5   0.665 GET /~andrei/dictionary/index.cgi?English=fall+out&Encoding=koi8 HTTP/1.1
0.5   0.665 GET /~andrei/dictionary/index.cgi?English=authorize+&Encoding=koi8 HTTP/1.1
0.5   0.665 GET /~andrei/dictionary/index.cgi?English=maximize+out&Encoding=koi8 HTTP/1.1
0.5   0.665 GET /~andrei/dictionary/index.cgi?English=peace+out&Encoding=koi8 HTTP/1.1
0.5   0.665 GET /~andrei/dictionary/index.cgi?English=ring+out&Encoding=koi8 HTTP/1.1
0.5   0.665 GET /~andrei/dictionary/index.cgi?English=body+mist&Encoding=koi8 HTTP/1.1
0.5   0.665 GET /~andrei/dictionary/index.cgi?English=body+gaurd&Encoding=koi8 HTTP/1.1
0.5   0.665 GET /~andrei/dictionary/index.cgi?English=root+canal&Encoding=koi8 HTTP/1.1
0.5   0.665 GET /~andrei/dictionary/index.cgi?English=Projects&Encoding=koi8 HTTP/1.1
0.5   0.665 GET /~andrei/dictionary/index.cgi?English=John&Encoding=koi8 HTTP/1.1
0.5   0.665 GET /~andrei/dictionary/index.cgi?English=work+out&Encoding=koi8 HTTP/1.1
0.5   0.665 GET /~andrei/dictionary/index.cgi?English=skim+out&Encoding=koi8 HTTP/1.1
0.5   0.665 GET /~andrei/dictionary/index.cgi?English=not+right+now&Encoding=koi8 HTTP/1.1
0.5   0.665 GET /~andrei/dictionary/index.cgi?English=not+possible&Encoding=koi8 HTTP/1.1
0.5   0.665 GET /~andrei/dictionary/index.cgi?English=not+in+my+life&Encoding=koi8 HTTP/1.1
0.5   0.665 GET /~andrei/dictionary/index.cgi?English=Not&Encoding=koi8 HTTP/1.1
0.5   0.665 GET /~andrei/dictionary/index.cgi?English=not+choose&Encoding=koi8 HTTP/1.1
0.5   0.665 GET /~andrei/dictionary/index.cgi?English=+not+against&Encoding=koi8 HTTP/1.1
0.5   0.665 GET /~andrei/dictionary/index.cgi?English=leave+out&Encoding=koi8 HTTP/1.1
```

```
GET http://www.helllabs.com.ua/c
```

Figure 3: Examples of false alarms in February log.

runner.com'). It uses N-gram clustering followed by a three dimensional visual display of the clusters. The input data can be recorded network traffic or general text logs, such as reports from intrusion detection systems, or other logs that have no connection to computer security at all. While the display of the clusters is easier to interpret than the Kohonen map approach cited above, the visualisation here is solely based on the output of the system. As such it enables the operator to get a grasp of the classification the system makes, but does not lend much insight into *how* the system reached the conclusion it did.

The main difference between these works and ours is that they rely on clustering techniques combined with a visual display of the clusters, without visualising the actual clustering process itself. While the output of the system is presented visually, it is still difficult for the operator to determine exactly *why* the output appears the way it does. This does not lend much insight into the learning process of the system, and hence does not provide insight into how that process could be controlled. Furthermore, they typically aren't interactive to nearly the same degree, the operator has little possibility of interactively controlling the learning of the system, being instead presented with the more or less final output of the system. This is perhaps natural given the lack of presentation of the internal state of the system.

A literature search has not turned up any attempts at visualising the state of Bayesian classifiers.

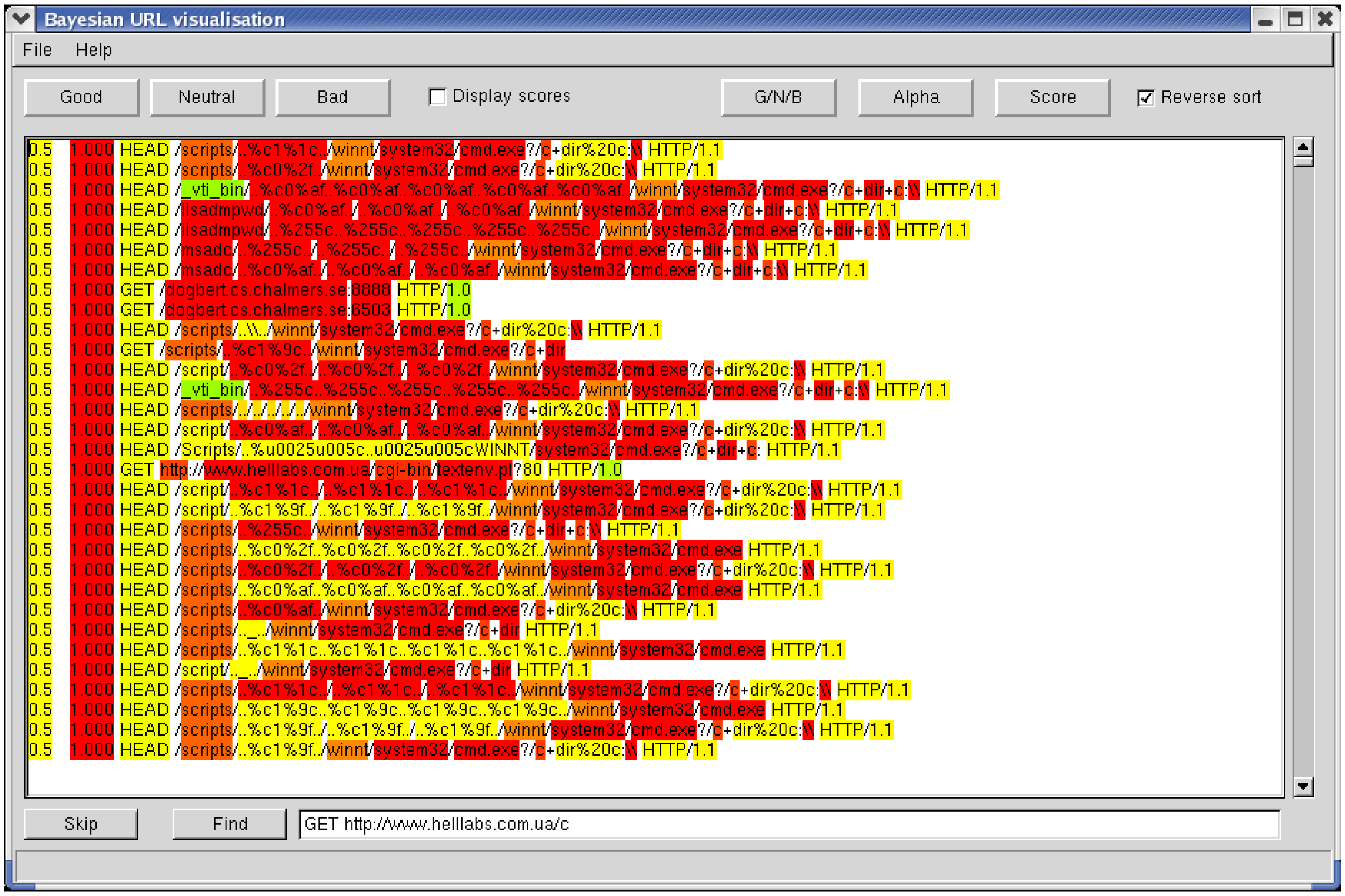
9. FUTURE WORK

A first step is to develop or gain access to other corpora of log data that contains realistic and known intrusive and benign behaviour, and to apply our tool to such data. An investigation of how visualisation could be applied to other detection techniques, especially more advanced Bayesian classifiers that take the order of the tokens into account, but also undirected self learning systems, is also planned.

10. CONCLUSIONS

We have developed an intrusion detection tool based on Bayesian classification. The tool—*Bayesvis*—provides the user with interactive visual feedback on the state of the learning process, and as such the user can both ascertain the quality of the output (by viewing the process that give rise to the alarms) and selectively train the system until it has reached a sufficient level of learning.

The tool was tested on our own corpora consisting of four months worth of access requests to a fairly large university department web server, using a *Train until no false positives* strategy. The tool proved successful. The Bayesian detector was somewhat successful in correctly classifying requests as intrusive or benign, and the visualisation made the limitations of the detector and the training readily apparent to operator who could evaluate the quality of the output and re-train the IDS interactively as necessary.



```
0.5   1.000 HEAD /scripts/..%c1%1c../winnt/system32/cmd.exe?/c+dir%20c:\ HTTP/1.1
0.5   1.000 HEAD /scripts/..%c0%2f../winnt/system32/cmd.exe?/c+dir%20c:\ HTTP/1.1
0.5   1.000 HEAD /_vti_bin/..%c0%af..%c0%af..%c0%af..%c0%af..%c0%af../winnt/system32/cmd.exe?/c+dir+c:\ HTTP/1.1
0.5   1.000 HEAD /iisadmpwd/..%c0%af../..%c0%af../..%c0%af../winnt/system32/cmd.exe?/c+dir+c:\ HTTP/1.1
0.5   1.000 HEAD /iisadmpwd/..%255c..%255c..%255c..%255c..%255c../winnt/system32/cmd.exe?/c+dir+c:\ HTTP/1.1
0.5   1.000 HEAD /msadc/..%255c../..%255c../..%255c../winnt/system32/cmd.exe?/c+dir+c:\ HTTP/1.1
0.5   1.000 HEAD /msadc/..%c0%af../..%c0%af../..%c0%af../winnt/system32/cmd.exe?/c+dir+c:\ HTTP/1.1
0.5   1.000 GET /dogbert.cs.chalmers.se:8888 HTTP/1.0
0.5   1.000 GET /dogbert.cs.chalmers.se:6503 HTTP/1.0
0.5   1.000 HEAD /scripts/..\\../winnt/system32/cmd.exe?/c+dir%20c:\ HTTP/1.1
0.5   1.000 GET /scripts/..%c1%9c../winnt/system32/cmd.exe?/c+dir
0.5   1.000 HEAD /script/..%c0%2f../..%c0%2f../..%c0%2f../winnt/system32/cmd.exe?/c+dir%20c:\ HTTP/1.1
0.5   1.000 HEAD /_vti_bin/..%255c..%255c..%255c..%255c..%255c../winnt/system32/cmd.exe?/c+dir+c:\ HTTP/1.1
0.5   1.000 HEAD /scripts/../../../../../winnt/system32/cmd.exe?/c+dir%20c:\ HTTP/1.1
0.5   1.000 HEAD /script/..%c0%af../..%c0%af../..%c0%af../winnt/system32/cmd.exe?/c+dir%20c:\ HTTP/1.1
0.5   1.000 HEAD /Scripts/..%u0025u005c..u0025u005cWINNT/system32/cmd.exe?/c+dir+c: HTTP/1.1
0.5   1.000 GET http://www.helllabs.com.ua/cgi-bin/textenv.pl?80 HTTP/1.0
0.5   1.000 HEAD /script/..%c1%1c../..%c1%1c../..%c1%1c../winnt/system32/cmd.exe?/c+dir%20c:\ HTTP/1.1
0.5   1.000 HEAD /script/..%c1%9f../..%c1%9f../..%c1%9f../winnt/system32/cmd.exe?/c+dir%20c:\ HTTP/1.1
0.5   1.000 HEAD /scripts/..%255c../winnt/system32/cmd.exe?/c+dir+c:\ HTTP/1.1
0.5   1.000 HEAD /scripts/..%c0%2f..%c0%2f..%c0%2f..%c0%2f../winnt/system32/cmd.exe HTTP/1.1
0.5   1.000 HEAD /scripts/..%c0%2f../..%c0%2f../..%c0%2f../winnt/system32/cmd.exe?/c+dir%20c:\ HTTP/1.1
0.5   1.000 HEAD /scripts/..%c0%af..%c0%af..%c0%af..%c0%af../winnt/system32/cmd.exe HTTP/1.1
0.5   1.000 HEAD /scripts/..%c0%af../winnt/system32/cmd.exe?/c+dir%20c:\ HTTP/1.1
0.5   1.000 HEAD /scripts/.._../winnt/system32/cmd.exe?/c+dir HTTP/1.1
0.5   1.000 HEAD /scripts/..%c1%1c..%c1%1c..%c1%1c..%c1%1c../winnt/system32/cmd.exe HTTP/1.1
0.5   1.000 HEAD /script/.._../winnt/system32/cmd.exe?/c+dir HTTP/1.1
0.5   1.000 HEAD /scripts/..%c1%1c../..%c1%1c../..%c1%1c../winnt/system32/cmd.exe?/c+dir%20c:\ HTTP/1.1
0.5   1.000 HEAD /scripts/..%c1%9c..%c1%9c..%c1%9c..%c1%9c../winnt/system32/cmd.exe HTTP/1.1
0.5   1.000 HEAD /scripts/..%c1%9f../..%c1%9f../..%c1%9f../winnt/system32/cmd.exe?/c+dir%20c:\ HTTP/1.1
0.5   1.000 HEAD /scripts/..%c1%9f../winnt/system32/cmd.exe?/c+dir%20c:\ HTTP/1.1
```

```
GET http://www.helllabs.com.ua/c
```

Figure 4: Generalised detection of Unicode attacks.

Furthermore, the training itself proved to not be unreasonably time consuming if one discounts the time taken to identify malicious examples to train the system on, a task that can be carried out by external experts and amortised over several installations as is the case with signature based IDS today.

11. REFERENCES

[1] S. Axelsson. The base-rate fallacy and the difficulty of intrusion detection. *ACM Transactions on Information and System Security (TISSEC)*, 3(3):186–205, 2000.

[2] S. Axelsson. Visualization for intrusion detection: Hooking the worm. In *The proceedings of the 8th European Symposium on Research in Computer Security (ESORICS 2003)*, volume 2808 of *LNCS*, Gjøvik, Norway, 13–15 Oct. 2003. Springer Verlag.

[3] L. Girardin and D. Brodbeck. A visual approach for monitoring logs. In *The Proceedings of the 12th Systems Administration Conference (LISA '98)*, pages 299–308, Boston, Massachusetts, USA, 6–11 Dec. 1998. The USENIX Association.

[4] T. Kohonen. *Self-Organizing Maps*, volume 30 of *Springer Series in Information Sciences*. Springer Verlag, Third edition, 2001. ISBN 3–540–67921–9, ISSN 0720–678X.

[5] W. Lee and D. Xiang. Information-theoretic measures for anomaly detection. In *IEEE Symposium on Security and Privacy*, Oakland, California, USA, 14–16 May 2001. IEEE.

[6] R. P. Lippmann, I. Graf, S. L. Garfinkel, A. S. Gorton, K. R. Kendall, D. J. McClung, D. J. Weber, S. E. Webster, D. Wyschogrod, and M. A. Zissman. The 1998 DARPA/AFRL off-line intrusion detection evaluation. The First Workshop on Recent Advances in Intrusion Detection (RAID-98), Lovain-la-Neuve, Belgium, 14–16 Sept. 1998.

[7] M. Ramadas, S. Ostermann, and B. Tjaden. Detecting anomalous network traffic with self-organizing maps. In *Proceedings of the Sixth International Symposium on Recent Advances in Intrusion Detection*, LNCS, Pittsburgh, PA, USA, 8–10 Sept. 2003. Springer Verlag.

[8] J. Rasmussen, K. Duncan, and J. Leplat, editors. *New Technology and Human Error (New Technologies and Work)*. John Wiley & Sons, Mar. 1987.

[9] E. R. Tufte. *The Visual Display of Quantitative Information*. Graphics Press, second edition, May 2001. ISBN 0–96–139214–2.

[10] C. D. Wickens and J. G. Hollands. *Engineering Psychology and Human Performance*. Prentice Hall, third edition, Sept. 1999. ISBN 0–32–104711–7.