Alert classification of web application attacks
using Bayesian networks to classify alerts from anomaly based intrusion detection systems

Schellekens, C.H.

Award date:
2014

TECHNISCHE UNIVERSITEIT EINDHOVEN
Department of Mathematics and Computer Science

Master's Thesis

Alert Classification of Web Application Attacks

Using Bayesian Networks to Classify Alerts from Anomaly Based Intrusion Detection Systems

by
C.H. Schellekens

December 13, 2013

Supervisors:
Prof. dr. Sandro Etalle
Dr. Damiano Bolzoni (SecurityMatters)

Abstract

Intrusion detection systems are used in an operational IT environment to strengthen the security. Even with firewalls, system hardening, patch management and other preventive security controls, intrusions might still occur because of remaining weaknesses. IDS detect intrusions from predefined signatures or by detecting anomalies in behaviour from users, network, applications or systems. Signature-based IDS are dependend on defined patterns, rules or signatures that are developed for know attacks. New attacks, such as zero-day attacks or changing signatures of attacks are typically not detected by signature-based IDS. Anomaly-based IDS detect intrusions by detecting abnormal behaviour such as unexpected network communication, service requests or user behaviour. Because of the non-signature approach an anomaly-based IDS can also detect previously unseen attacks. If the number of false alerts can be kept low, anomaly-based IDS can reach good accuracy and detect even unknown attacks. However, because only anomalies are detected, an operator will have to analyse the anomaly event manually to verify and classify an alert. This leaves the usability of anomaly-based IDS at a low level, even when a high accuracy is achieved.

This thesis presents an experiment to classify anomaly alerts automatically through supervised machine learning. The experiment is performed on web application attacks, with anomaly alerts received from the SilentDefense anomaly-based intrusion detection system. After considering several machine learning methods, naïve Bayes is selected for the experiment. A naïve Bayesian network is built from attack features that are extracted from the anomaly event data. The accuracy of the network is tested with two different datasets. Incremental learning is briefly regarded. Also a threshold is introduced to prevent misclassifications.

The experimental results show that it might very well be possible to classify anomalies with a high TP rate and a low misclassification rate, using simple (stateless) features and a self-learning Naïve Bayesian network. A TP-rate of over 90%, as formulated in the research goal, might be achievable in practice for any well-defined attack class. A FP-rate of 1% or less can even be reached with the use of a threshold.

Preface

This thesis ends a period of five years of part-time study on information security technology at the Eindhoven University of Technology, or better, at the Kerckhoffs Institute, the collaboration on computer security between the University of Twente, the Radboud University Nijmegen, and the Eindhoven University of Technology.

I want to thank Sandro Etalle for offering me this research project which was a very good opportunity for me to combine theoretical models with a practical application in computer security. I also want to thank Daminano Bolzoni for helping me out during the project and for asking the questions that are most important to answer. I also want to thank my employer and collegues that helped me to make this possible.

Most of all, I want to thank my wife Christel and my children Bas en Janneke for letting me spend almost all of my spare time, weekends and holidays studying, for years in a row. I know that I have neglected them regularly for the last couple of years, but still they did not complain very often.

I am glad that I started this study program five years ago, but I am also glad to be able to round it of now. With all the attention given to information security at the moment, I am looking forward to making myself more useful in this field of expertise.

Table of Contents

1. Introduction

This master's thesis describes a research project that investigates whether it is possible to determine the concrete attack class of alerts from an anomaly based intrusion detection system with supervised machine learning.

The thesis starts with some backgrounds and a definition of the research question in chapter 2. The general approach that is taken in the research is also explained in this chapter.
Chapter 3 gives an overview of intrusion detection systems (IDS), explaining different aspects of IDS in general. It also describes differences between signature-based IDS and anomaly based IDS, and some information on the SilentDefense IDS that was used in the experiment is given.
Chapter 4 discusses some well-known methods in supervised machine learning and it is explained why Bayesian Networks, or more precise Naïve Bayes is chosen as the machine learning method for the experiment.
In chapter 5, attack classes are defined that will have to be recognized by the Bayesian classifier, and features are selected for these classes that should make correct classification possible.
Chapter 6 discusses the process of data collection to obtain a labelled data set for learning and testing. It also describes the process of feature extraction from the resulting data set, and the setup of the analysis towards a Bayesian classifier is explained.
In chapter 7 the resulting Bayesian network is presented, and the results with this classifier are analysed and discussed.
Finally, in chapter 8, conclusions are drawn and some ideas for future research and implementation of the proposed method are given.

2. Problem Statement

2.1. Background

Intrusion detection systems (IDS) add a layer of security behind perimeter defence mechanisms such as firewalls. With intrusion detection the behaviour of systems and users is analysed to detect intrusions that have occurred in spite of all other security mechanisms in place. The approach that is taken in IDS is typically signature based or anomaly based. In the signature-based approach signatures are developed for known attacks. New events are compared with the list of signatures to be able to identify an attack that has occurred. This approach typically gives a low false positive rate (false alerts), but has a general disadvantage that it can only detect known attacks. Anomaly-based intrusion detection tries to find evidence for intrusions by finding deviations from what can be defined as normal system or user behaviour. This normal behaviour can be defined in rules or in a different type of static or dynamic model. The main advantage of anomaly-based IDS is that it can detect previously unknown attacks. A shortcoming in the usability of anomaly-based IDS is that, even in a well-tuned system with a low false positive rate, an alert will only tell that an anomaly has occurred, not what kind of attack has occurred. Automized post-classification of alerts could improve on the usability of anomaly based IDS.

2.2. Research Question

In this research it is investigated whether models in supervised machine learning can be used to classify alerts from an anomaly based intrusion detection system.

The main research question that will be answered is:

> **“Is it possible to classify alerts from anomaly based intrusion detection systems with sufficient accuracy, using supervised machine learning?”**

For ‘sufficient accuracy’ the aim is to be able to classify over 90% of all alerts correctly. This would mean for example that if 30 alerts are received per day, 27 would be classified correctly and 3 would be presented incorrectly to the security monitoring operator. For practicle usability, this would result in having a usable classification system.

2.3. Approach

The research will focus on web application attacks only such as Cross-site Scripting and SQL injection. For the research an experimental setup will be used with the anomaly based IDS *SilentDefense* monitoring a webserver. Data sets of anomaly alerts will be collected from real-world attacks from external malicious attackers and from test-patterns for web application attacks that will be used to simulate attacks on the monitored system. All attacks that are registered as an anomaly will be classified manually by the operator of the IDS and labelled according to defined attack classes. Any false positive of the IDS will not be included in the datasets because it cannot be labelled for the supervised machine learning approach. With the obtained datasets a supervised machine learning method, that will have to be selected first, can be trained and tested. Results can then be analysed to be able to answer the research question.

In the research it is chosen not to use any analysis results from the IDS that delivers the alerts. The IDS could give extra information about the anomaly such as in which part of the event data the anomaly is found, or what the deviation from normal behaviour is. These details could be included in the model to improve classification results. The reason not to include this extra information is that the research tries to find a general solution that will work with any IDS. Including the extra details to improve the classification model would blur the results, making it more difficult to draw general conclusions from the research.

To categorize attacks, features from network events will be used to build a machine-learning model. A network-based approach will be used, not a host-based approach, so features from internal system behaviour will not be included. With this approach a system independent model can be created usable for all web systems. Also, only single events will be used. No relations between requests, or any comparison to a learned normal behaviour will be included in the model. This approach is chosen to be able to create a light model that is independent of any implementation features and independent of having to remember or keep track of events over time. Goal is however to create a model fit for incremental learning that can automatically improve over time.

3. Intrusion Detection Systems

Intrusion detection systems are used in an operational IT environment to strengthen the security. The need for IDS comes from the fact that it is very hard to prevent intrusions completely by placing firewalls, updating all systems and software regularly, and strengthening authentication mechanisms. Even then, malicious intrusions might occur because of other problems such as unsafe behaviour by end-users infecting internal systems, misconfigurations in firewalls or servers, or internal attacks from within the guarded perimeters. Unsafe behaviour by end-users often is a violation of the company's security policies, but happens nonetheless. Some examples are: users that visit non-work-related malicious websites, users unaware of security hazards such as phishing, users that extend the network infrastructure with insecure wireless access points, and users that bring in malware on infected USB drives. More examples can be given, but the lesson learned is that monitoring the operational IT environment is necessary to detect and respond to such breaches in security. IDS plays an important role in this monitoring process.

IDS only detects intrusions, hence the name. An extension to this functionality is when the IDS is able to not only detect intrusions, but can block further intrusive behaviour once the intrusion is detected. This form of IDS is referred to as intrusion prevention (IPS). Prevention of further intrusion is a desirable feature, since detection alone will not solve the problem at hand. However, automatically blocking of further activity from a user, system or process would worsen the consequences in case of a false alert. The prevention mechanism can also be abused explicitly by attackers to cause a denial of service. For these reasons, IDS is often preferred without automatic preventive action taken by the system.

The performance quality of IDS can be measured by the error rates that occur during detection. An IDS might raise a false alarm now and then, when analysis shows that a malicious event has occurred while in fact this was not the case. Such a false alert is referred to as a False Positive (FP). Another type of error is a False Negative (FN), where no alert is raised by the IDS, where it should have. Next to these two types of errors also True Positives and True Negatives can be defined. A True Positive is a correct alert, raised by the IDS in response to malicious activity. A True Negative is the event where non-malicious activity is analyzed by the IDS and no alert is raised.

IDS is categorized into different types based on how intrusions are distinguished from normal behaviour and whether the analysis is done within a host or on the network. The latter differentiation categorizes IDS into host-based (HIDS) or network-based (NIDS) intrusion detection. In NIDS network communication is analysed for malicious content without knowing the effect of the content within the receiving systems. The attacker's behaviour is analysed, not the attack result within a host, although NIDS can analyse the external effect from internal behaviour by studying resulting communication from hosts. In host-based intrusion detection (HIDS), intrusions are monitored from within a computer system. HIDS can therefore not only analyse network communication from and to the host, but also monitor behaviour from the operating system and behaviour from applications. The approach taken to differentiate normal behaviour from intrusions is identified as signature-based or an anomaly-based. The characteristics of both approaches are described in the coming subparagraph and are essential for the reason this research experiment is performed.

3.1. Signature-based versus Anomaly-based IDS

Signature-based IDS detect intrusions by specific signatures that are defined for concrete attack patterns. A signature defines unique evidence for a known attack. An example signature from the well-known NIDS Snort (Roesch, M., 1999) is displayed below. Snort's detection engine uses detection rules that specify signatures based on communication parameters and packet contents. The rule that is displayed in the example tries to detect outgoing communication from an internal host infected with a Buterat Trojan.

```
alert tcp $HOME_NET any -> $EXTERNAL_NET $HTTP_PORTS (msg:"MALWARE-CNC
Win.Trojan.Buterat outbound connection"; flow:to_server,established; content:"From|3A|";
http_header; content:"Via|3A|"; http_header; urilen:13;
pcre:"/^\x2f\d{3}\x2f\d{3}\x2ehtml$/U"; metadata:policy balanced-ips drop, policy
security-ips drop, ruleset community, service http;
reference:url,www.virustotal.com/file/90fb793d1fd7245b841ca4b195e3944a991d97d8540
90729062d700fe74553e5/analysis/; classtype:trojan-activity; sid:25269; rev:2;)
```

rule parts:

rule-header describing protocol characteristics of the traffic to be analyzed
alert message that will be generated if the rule matches the traffic that is analyzed
signature to be matched

Figure. An example of a signature-based IDS rule from Snort.

The rule analyses outbound HTTP traffic only, which can be seen in the rule's header (alert tcp $HOME_NET any -> $EXTERNAL_NET $HTTP_PORTS). The header contains variable definitions that are specified in configuration files for the monitored network environment. After the rule-header follow rule-details where, in this rule, first is defined to check for the existence of HTTP headers "From:" and "Via:" to limit the scope of analysed HTTP traffic to HTTP requests containing these headers. The signature then detects an HTTP request-URI with an exact length of 13 characters and the content matching the regular expression "/^\x2f\d{3}\x2f\d{3}\x2ehtml$/U". This regular expression identifies a URI of the form "/*ddd*.html", where *ddd* is a 3 digit numerical string.

The approach of defining exact unique signatures for known attacks gives good results for known attacks resulting in a low False Positive rate for the system. However, on new attacks, variations of an attack, or obfuscated attacks, a signature-based IDS will not detect the attack anymore. A typical problem with signature-based IDS is therefore a high False Negative rate.

Anomaly-based IDS detect intrusions by differentiating normal activity from abnormal activity in a more statistical sense of the term *normal*. An example could be a definition for a company's user-LAN where employees only work during the daytime. If outbound network communication would then be detected at night from this network, this could be identified as an anomaly. An anomaly-based IDS defines a profile of what is normal behavior from users, hosts, applications, or networking services. This profile can be pre-defined in static rules by an expert, or it can be learned automatically by monitoring normal behavior during a well-chosen time-period.

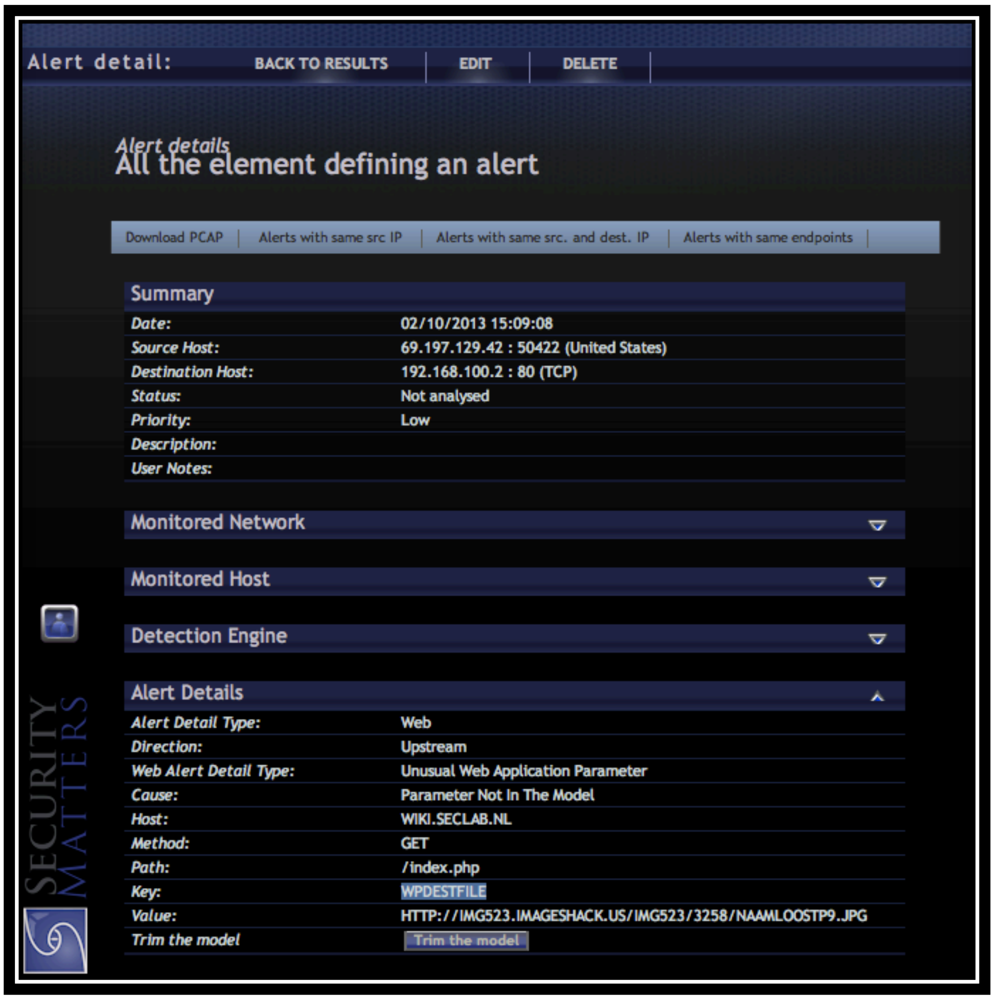
The major advantage of anomaly-based IDS is that it can also detect previously unknown attacks. For anomaly-based IDS false positives are generally a bigger problem to overcome. For the example used where user communication at night is defined as an anomaly, a false positive could occur when employees have to work late.

An anomaly based IDS will typically not conclude what type of attack has occurred exactly, but merely that an anomaly has occurred that indicates a malicious intrusion. An anomaly-based IDS will raise an alert that an anomaly has occurred, and can include details from the event data and how the event deviates from normal behavior. What exact type of attack has occurred however, is not known to the IDS, because no attack definitions exist in the IDS' profile, only a definition of what is normal or abnormal behavior. A major disadvantage of anomaly-based IDS is therefore that an alert has to be manually inspected and classified by a human operator, leaving the usability of anomaly-based IDS at a low level.

The amount of related work on automatic classification appears to be limited. Similar research has been done by Bolzoni (2009), resulting in Panacea, an automatic classification system with an overall performance of up to 92% correctly classified attacks in automatic mode, with over 75% correct classifications for any attack class.

3.2. SilentDefense

For the research experiment, the anomaly-based NIDS *SilentDefense Web* is used (SecurityMatters, 2013). SilentDefense Web, from here on referred to as SilentDefense, monitors web applications by first building a profile of normal HTTP communication to and from a web application. This profile contains characteristics of normal communication such as format and values of request parameters in combination with HTTP protocol characteristics. For parameters the value type and length are defined and with pattern descriptions and whitelists a more exact definition of normal parameter values can be given. If a parameter has an abnormal type, value or distribution, or is unknown, an alert will be raised. Also, if requests are done to unknown locations or when unusual HTTP request details occur, an alert will be raised. If the learning period for profiling is taken long enough to be able to profile all normal use, SilentDefense has a low FP-rate. During operation the detection profile can still be manually extended or the system can temporarily re-enter learning mode for additional learning. An example of an alert raised by SilentDefense is displayed below.



Summary

Date:	02/10/2013 15:09:08
Source Host:	69.197.129.42 : 50422 (United States)
Destination Host:	192.168.100.2 : 80 (TCP)
Status:	Not analysed
Priority:	Low
Description:	
User Notes:	

Alert Details

Alert Detail Type:	Web
Direction:	Upstream
Web Alert Detail Type:	Unusual Web Application Parameter
Cause:	Parameter Not In The Model
Host:	WIKI.SECLAB.NL
Method:	GET
Path:	/index.php
Key:	WPDESTFILE
Value:	HTTP://IMG523.IMAGESHACK.US/IMG523/3258/NAAMLOOSTP9.JPG
Trim the model	Trim the model

Figure. Alert from SilentDefense

The reason that the alert in the example was raised is that the parameter WPDESTFILE used in the request is not in the model that was profiled for the web application that this request was sent to. Even though this malicious request, which is a Remote File Inclusion attempt, is detected by SilentDefense, the attack class itself cannot be concluded by the anomaly-based IDS.

4. Supervised Machine Learning

4.1. Machine Learning

Machine learning is a form of artificial intelligence where predictive models for a certain task are constructed from data. In Supervised Machine Learning a training set with classified instances is used to develop a model with which new instances can be classified. Alternatives to supervised machine learning are unsupervised machine learning, where hidden patterns are discovered in unlabelled data, and reinforcement learning, where reward signals are used to discover the best action. Different approaches are available for supervised learning. Well-known methods are Decision Trees, Artificial Neural Networks, Bayesian Networks, Instance Based Learning and Support Vector Machines. Characteristics of these methods are described in the following paragraph.

4.2. Classification Algorithms

4.2.1. Decision trees

Decision trees are trees that classify instances by choosing a path from the root node down the tree based on feature values. Each node in the tree represents a feature and each branch a value or value range that the feature can have. Each leaf of the tree represents a class that the instance falls into once it reaches the end of its path down the tree. As an example a decision tree is shown below to decide if a guitar is a Fender or a Gibson guitar. In this case it is based on five features.

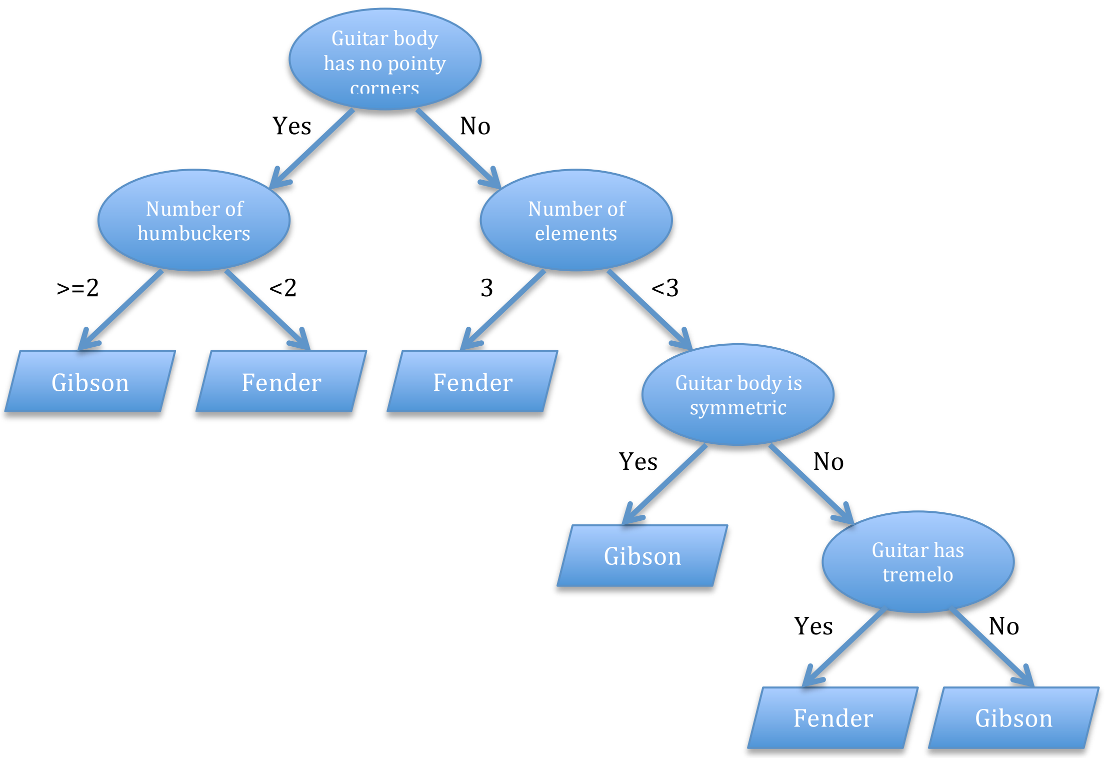


Figure. Decision tree to classify (standard issue) Fender and Gibson guitars

Building a decision tree is done with heuristics of the training data. For example, the feature that separates the training data best will be the root node. This process can recursively be continued to build the tree further. A standard issue that will occur when building a decision tree is overfitting because building the tree can be continued until it perfectly fits the training data. To prevent overfitting the training algorithm has to be stopped in time, and post-pruning can be applied where the tree is simplified by removing parts that contribute too little to the performance.

4.2.2. Artificial Neural Networks

The naming of Artificial Neural Networks (ANN's) is inspired by Biological Neural Networks (BNN's) such as the brain. In BNN's neurons, or nerve cells, process input signals, based on previously learned patterns, to activate output if the combination of all inputs surpasses a certain threshold. This output is then passed to following neurons in a larger neural network. An ANN is also a system of interconnected neurons that compute a weighted output from inputs, also with the use of a threshold.
The building blocks of an ANN are perceptrons, a single neuron with feature values as input. The output is then determined as the sum of the weighted input values. The output value is passed through a threshold function to determine if the output is 1 or 0, classifying the input into one of two classes.

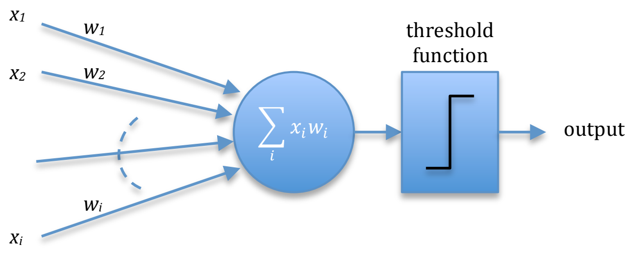


Figure. Perceptron structure

With a perceptron, only linearly separable data sets can be classified. With multiple layers of perceptrons more complex learning networks can be created. An ANN is a multi-layer perceptron where multiple layers of nodes are connected to determine eventual output. An ANN has an input layer, one or more internal (hidden) layers and an output layer.

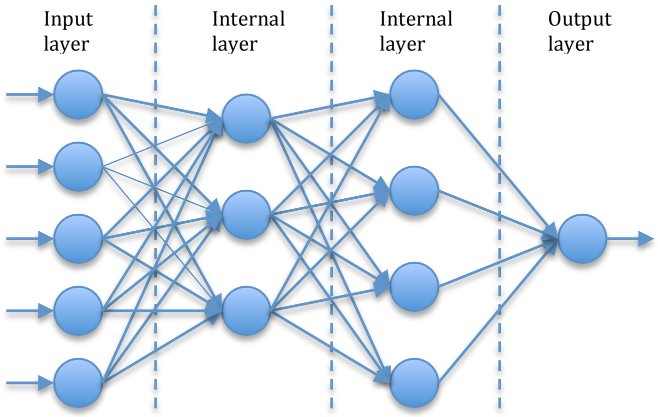


Figure. Artificial Neural Network

Training a network is done by setting initial weights to zero, or a low random value, or if possible to good guesses for weight values to speed up the learning process. The weights are then adjusted by repeatedly going through the network with instances of the training set, comparing the outcome with the labelled data. When the outcome value does not match the label value, the weight values are adjusted slightly towards the desired output and steps are repeated. Training time for neural networks in general is usually long compared to most other supervised machine learning methods.

4.2.3. Bayesian Networks

A Bayesian network is a Directed Acyclic Graph (DAG) as a representation of a probabilistic distribution. Each node in the graph is a random variable X_i and each edge holds a probabilistic dependency between two nodes. This dependency is specified by a conditional probability table for each node, holding the relations with the node's parent.
A Bayesian Classifier is a Bayesian Network for a certain classification. It has a node C that represents the class variable, and a node X_i for each feature in the model.
In practical applications of Bayesian networks, not necessarily all possible relations are modeled in order to keep the model compact and therefore usable. The simplest form of a Bayesian network is the simple Bayesian classifier (SBC), often referred to as Naive-Bayes (NB). In NB complete independence of all features is assumed. This leads to a model as pictured in the figure below.

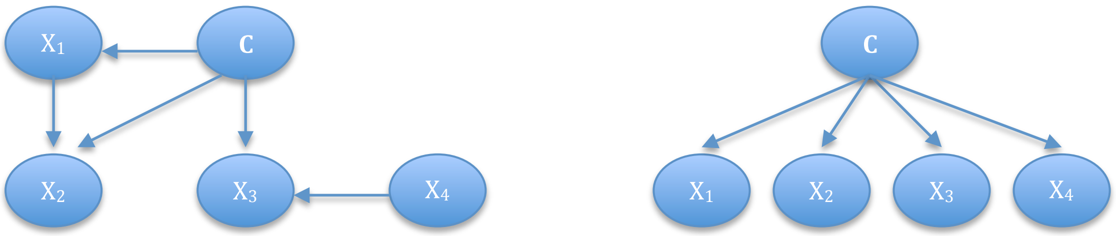


Figure. General Bayesian network versus Naïve Bayesian network.

Even though the independence assumption in NB is almost always untrue in the real world, NB can have a very good performance, and often is competitive with other classifiers, as demonstrated and analyzed by for example Langley, Iba, & Thompson (1992), and Friedman, Geiger, & Goldszmidt (1997).

Bayesian networks are based on Bayes' Theorem on conditional probabilities:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

To stick to the earlier example of Fender and Gibson guitars the training and operation of a Bayesian Network can be explained. Suppose that someone tells you that he or she has seen a very nice guitar and that it had a tremolo system. Given the right probabilities we can now calculate the chance that this guitar was a Fender guitar. Suppose that we know that 75% of all Fender guitars have tremolo systems, so $P(Tremolo|Fender)$, or $P(T|F) = 0.75$. Also it is known that 10% of all Gibson guitars have tremolo systems, so $P(T|G) = 0.1$. An estimate could be that the rate of Fender guitars versus Gibson guitar is 50/50, and we known that this particular guitar was one of the two. The probability that this guitar was a Fender guitar can now be calculated as:

$$P(F|T) = \frac{P(T|F)P(F)}{P(T)} = \frac{P(T|F)P(F)}{P(T|F)P(F) + P(T|G)P(G)} = \frac{0.75 \cdot 0.5}{0.75 \cdot 0.5 + 0.1 \cdot 0.5} \approx 0.88$$

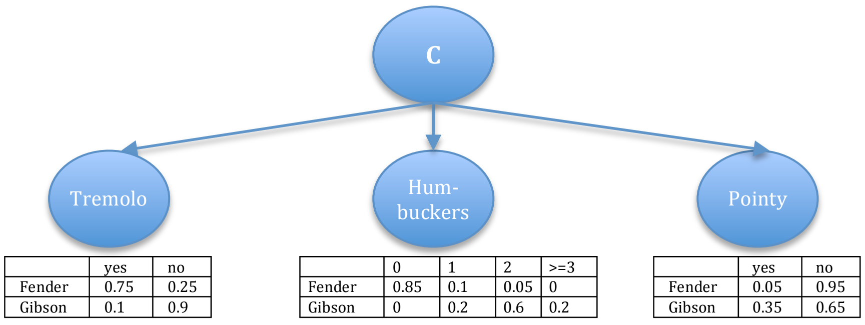
In the learning phase of building a Bayesian classifier the occurrence of the chosen features is counted in the instances in the training data set. From these frequency counts the probabilities for the classifier are calculated. In the above example our training data could have consisted of 100 Fender guitars and 100 Gibson guitars. To get the probabilities in the example, 75 Fender guitars and 10 Gibson guitars would need to have had a tremolo system.

In a larger Bayesian network probability tables can be calculated for all feature nodes from the frequency counts in the training data. Suppose that we have also counted the number of humbuckers on each guitar, and whether the guitar body has 'pointy corners' or round edges only.

	Tremolo	*Humbuckers*	*Pointy corners*
Fender	75x	1:10x, 2:5x	5x
Gibson	10x	1:20x, 2:60x, 3:20x	35x

Table. Guitar feature frequencies

Assuming complete independence of these features, a Naïve Bayesian network classifier can be defined with conditional probability tables (CPT).



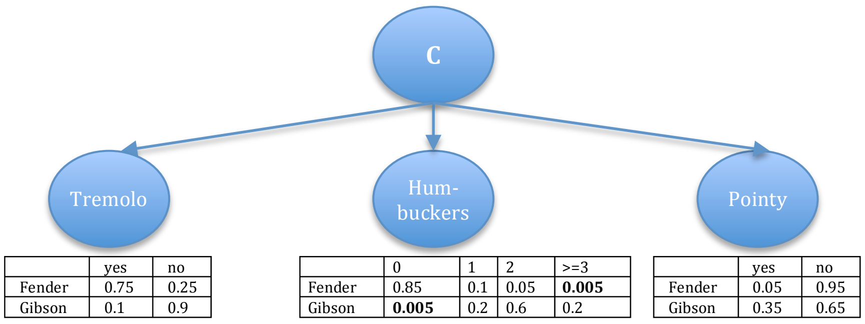
	yes	no
Fender	0.75	0.25
Gibson	0.1	0.9

	0	1	2	>=3
Fender	0.85	0.1	0.05	0
Gibson	0	0.2	0.6	0.2

	yes	no
Fender	0.05	0.95
Gibson	0.35	0.65

Figure. Example Naïve Bayesian network.

A standard problem in Bayesian Network is how to deal with probabilities of 0, because this would always result in a total probability outcome of 0 in calculations, erasing all information from other features. A correction has to be used to prevent this, for example by starting with an initial frequency count of 0.5. Several methods are available for correction, such as a Laplace estimator where 1 is added to both numerator and denominator values in the probability calculation. If, for the sake of simple demonstration, we only use a 0.5 frequency count instead of 0, we can correct the example classifier for the fact no Fender guitar has 3 humbuckers, and all Gibson guitars have at least 1 humbucker:



	yes	no
Fender	0.75	0.25
Gibson	0.1	0.9

	0	1	2	>=3
Fender	0.85	0.1	0.05	**0.005**
Gibson	**0.005**	0.2	0.6	0.2

	yes	no
Fender	0.05	0.95
Gibson	0.35	0.65

Figure. Example Naïve Bayesian network with correction for 0-probabilities.

With this Naïve Bayesian Network the probability that a new sample is a Fender or a Gibson guitar can be calculated. The probability of a sample guitar being e.g. a Gibson (class G) is:

$$P(G|f_1, f_2, f_3) = \frac{P(f_1, f_2, f_3|G) \cdot P(G)}{P(f_1, f_2, f_3)}$$

with f_1, f_2, f_3 being the values for the selected features.

Because of the independence assumption:

$$P(f_1, f_2, f_3|G) = P(f_1|G) \cdot P(f_2|G) \cdot P(f_3|G)$$

and:

$$P(f_1, f_2, f_3) = P(f_1, f_2, f_3|G) \cdot P(G) + P(f_1, f_2, f_3|F) \cdot P(F)$$

$$= P(f_1|G) \cdot P(f_2|G) \cdot P(f_3|G) \cdot P(G) + P(f_1|F) \cdot P(f_2|F) \cdot P(f_3|F) \cdot P(F)$$

resulting in:

$$P(G|f_1, f_2, f_3) = \frac{P(f_1|G) \cdot P(f_2|G) \cdot P(f_3|G) \cdot P(G)}{P(f_1|G) \cdot P(f_2|G) \cdot P(f_3|G) \cdot P(G) + P(f_1|F) \cdot P(f_2|F) \cdot P(f_3|F) \cdot P(F)}$$

With this formula, probabilities are easy to calculate, because all needed values are available in the classifier. If we take the following guitar as a new sample, our classifier can predict the make of this guitar from the defined features only (ignoring the fact that it clearly says Gibson on the head of the guitar):

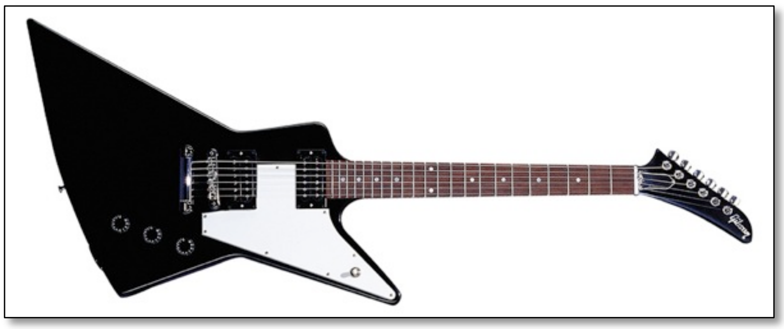


Figure. Sample guitar

The feature values for this sample are that it has no tremolo system, 2 humbuckers and it does have 'pointy corners'. The probabilities for this sample then are:

$$P(Gibson|'no, 2, yes') = \frac{0.9 \cdot 0.6 \cdot 0.35 \cdot 0.5}{0.9 \cdot 0.6 \cdot 0.35 \cdot 0.5 + 0.25 \cdot 0.05 \cdot 0.05 \cdot 0.5} = 0.997$$

$$P(Fender|'no, 2, yes') = \frac{0.25 \cdot 0.05 \cdot 0.05 \cdot 0.5}{0.25 \cdot 0.05 \cdot 0.05 \cdot 0.5 + 0.9 \cdot 0.6 \cdot 0.35 \cdot 0.5} = 0.003$$

So, the classifier seems pretty convinced that the sample is a Gibson, which is correct in this case.

4.2.4. Instance Based Learning

With instance based learning a new instance is compared to instances in the available training data. Instance based learning is also referred to as lazy-learning, because generalization of training data is delayed until an actual classification has to be done. Examples of instance based learning algorithms are the Nearest Neighbour type of algorithms such as k-Nearest Neighbour (kNN). With kNN an instance that has to be classified is compared to the instances in the training data. It looks for the *k* training instances that resemble the new instance most. The class prediction of the new instance will then be the class label that occurs most frequent in these *k* training instances. If *n* features are used to describe a sample, instances are placed in an *n*-dimensional space. For a new instance, the distance to training instances can then be calculated as the Euclidean distance, but relative distance calculations and extra weighting schemes are also used to improve the results.

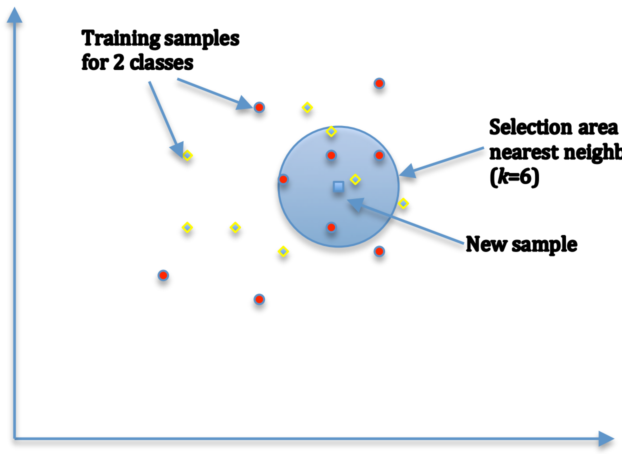

Training samples for 2 classes
Selection area for *k* nearest neighbours (*k*=6)
New sample

Figure. Illustration of k-nearest neighbour principle

Consequences of the lazy-learning approach are that the complete training data set has to be stored and that the speed of classifying instances is slow compared to most other methods.

4.2.5. Support Vector Machines

A Support Vector Machine (SVM) is a classifier that separates instances of two classes in a high-dimensional feature space by a separating hyperplane. SVM are the most recent supervised machine learning method, defined by Cortes & Vapnik (1995). They explain the idea of a support vector network as input vectors that are non-linearly mapped to a very high dimension feature space, in such a way that a linear decision surface can be constructed. In the basic model the two classes have to be completely separable and the optimal hyperplane is chosen in such a way that the margin between the two classes is maximized. Data points that lie on the margin boundary are called the support vectors and the resulting classifier is defined by a linear combination of these points. New data points can then be mapped into the same high dimensional space, placing them on one side of the hyperplane, which determines their predicted class.

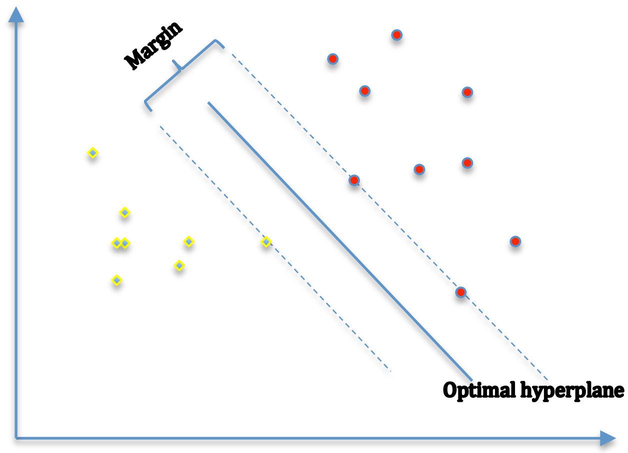


Figure. A Support Vector Machine (SVM) in 2-dimensional space

To prevent having to calculate in a possibly billions or higher dimensional space, a so-called kernel trick is performed, avoiding the need for explicit mapping to the high dimension through the use of a kernel function. With this kernel function the needed computations can still be done in the original feature space making calculations feasible.

4.3. Algorithm Selection

With an overview of different supervised machine learning methods, the choice is made which algorithm to use for the experiment. The chosen method must be able to handle multi-class problems, because we are not looking for a true/false prediction, but a classification of multiple attack categories such as Cross-site Scripting, SQL injection, Path Traversal, etcetera. The algorithm selected should also be able to deal with 'more than a few' features, probably dozens. These features are sometimes binary (e.g. Was the HTTP request RFC compliant?), sometimes discrete/categorical (e.g. what HTTP response status code is returned?), and sometimes more continuous (e.g. how long was the HTTP request?). Also, it is hard to estimate whether all chosen features are always relevant, or relevant for all attack classes. This goes especially for different operational situations if the model would be used in practise, and for future developments in attacks. The chosen algorithm must therefore be robust towards irrelevant features. Also for an operational application of the chosen algorithm, incremental learning must be possible and easy, because you would want continuous learning with each new event handled in an operational situation.

A nice overview and comparison of supervised machine learning methods is given by Kotsiantis, Zaharakis, & Pintelas (2007). It discusses differences, strengths, and weaknesses of the different methods. It shows for example that all methods have a way to be able to deal with multi-class problems even if the basic method is fit for binary decisions only, such as SVM.

Speed of classification might differ between methods, but this is not really a selection criterion in the setting of this research. Only detected alerts will have to be classified. The real time performance criterion is therefore for the anomaly based IDS that has to analyse all traffic real-time, and not for the classification system that only has to classify in the much rarer event of an anomaly alert. This will typically only happen several times a day for a well-tuned IDS, or maybe tens or hundreds of times a day if the guarded systems are under heavier attack.

SVMs and neural networks perform best on continuous features, although discrete features can be rescaled to a continuous representation. Both these methods need a large data set for training to achieve good prediction accuracy. Also speed of training and incremental learning is typically rather slow, especially for neural networks. Speed of training is not very relevant for the research experiment, but in an operational situation retraining might be needed when changes are required to the operational classifier, for example when new features or new attacks are added over time. But, in the incidental occasion that retraining is needed, a waiting time of half an hour or so should still be no problem.

Methods that are best fit for incremental learning are Naïve Bayes, and kNN, as an instance based learning algorithm. For kNN however, it also states that it is very sensitive to irrelevant features, and noise in feature values.

Requirement	Decision Tree	ANN	(Naïve) Bayesian Network	kNN	SVM
Speed of classification	+	+	+	--	+
Speed of training	+	-	+	++	-
Fit for incremental learning	-	+	++	++	-
Accuracy in general	+/-	+	-	+/-	++
Scalability to number of features	+	-	++	++	-
Tolerance to irrelevant attributes	+	-	+/-	+/-	++
Fit for multi-class problems	+	-	+	-	-
Comprehensibility	++	--	+	-	--

Scale: --, -, +/-, +, ++

Table. Comparitive overview of characteristics of supervise machine learning methods

The accuracy of Naïve Bayes in general is less compared to more sophisticated methods because the assumption of independence of features is almost always untrue distorting the simplified probability calculations used. Also, the problem at hand is a rather fuzzy problem where a probabilistic approach, as with NB, seems to fit best. An example to explain this ‘fuzziness’ is a feature such as the length of a request parameter in an HTTP query string or a POST parameter. It is not as straightforward that a XSS attack will have a certain parameter length, but in general, the length of parameter values will increase if an attacker tries to include complete malicious scripts in a request parameter. The probability of a request being a XSS attack might therefore increase with a longer parameter length, but trying to put this in a decision tree might be difficult. A feature might be relevant in general but not in all cases. In a probabilistic approach, a full Bayesian Network could become too complex with a lot of features, but a Naïve Bayesian classifier would keep the model simple. And even though the independence assumption of NB would not be completely realistic, dealing with all features independently could still give good outcomes, as NB does in general with these violations of the real dependencies.

For the research experiment, Naïve Bayes is chosen because the probabilistic approach seems to fit the problem’s characteristics well and because it is computationally fast and incremental learning is easy with NB. Decision trees are not used because the problem at hand does not seem to be very linear. Artificial Neural Networks and Support Vector Machines are not used because they do not scale very well to a higher number of features and the comprehensibility of both methods is low. kNN is not used because as a lazy learning method the speed of classification is low.

Using a full Bayesian network instead of NB, or even introducing limited dependency relations between features in the network, might be difficult because clear dependency relations could be hard to define. Also, the independence assumption in NB is estimated not to be a problem because, even though features of attacks might have some dependency relation, regarding the features completely independent of each other can still result in a correct best guess estimate of the attack class.

5. Attack Classes and Features

Before an experiment can be done, first the attack classes have to be defined. Using classes such as Cross-site Scripting and SQL injection seems evident, but which other classes for web application attacks should be used. How about other injection techniques like LDAP injection or XPath Injection? Are those types of attack separate attack classes or do we need a broader class-definition than SQL injection. Also, for the experiment, not all types of attacks have to be covered, but the aim is to define and test for the top-5 attack classes in web application attacks.

5.1. Attack Classes

5.1.1. Choosing attack classes

A good source for the major attack classes could be OWASP, the Open Source Web application Security Project, well known for their OWASP Top Ten Application Security Risks. The top-3 in this OWASP list, both in the last definitive 2010 version, as well as in the 2013 release candidate for the OWASP Top Ten, has the same risks, although the second and third place have switched places. In the 2013 version the top-3 is as follows:

1. Injection, containing different injection flaws like SQL injection, OS command injection and LDAP injection where.
2. Broken Authentication and Session Management, containing vulnerabilities regarding authentication and session management.
3. Cross-site Scripting (XSS), where an attacker can inject scripts that will eventually be executed in a victim's browser.

If we consider detectability for this top-3, the injection attack class is a very broad class, containing different types of more concrete attacks, each having different discriminating features. The attack class in second place, broken authentication and session management, also is a diverse class, and more a class of vulnerabilities than attacks.

Another source, to determine the frequencies with which vulnerabilities are exploited, is the national vulnerability database (NVD) that is kept by NIST, the US National Institute of Standards and Technology. NVD vulnerabilities are traceable by their CVE (Common Vulnerability and Exposures) number. Examples of CVE categories used for labelling vulnerabilities are XSS, CSRF, buffer errors, code injection, Path Traversal, SQL Injection and OS command Injection. These vulnerabilities are not kept for web applications specifically but for computer systems in general. Of the 5000+ vulnerabilities registered in 2012 the following 'market shares' of the possibly web application related categories are listed in the table below.

Category	**Total**	**Buffer errors**	**XSS**	**SQL injection**	**CSRF**	**Code injection**	**Path Traversal**	**OS command injection**
#Occurrences	5289	724	720	238	153	136	118	14
Percentage of total attacks	100%	13.7%	13.6%	4.5%	2.9%	2.6%	2.2%	0.3%

Table. CVE vulnerabilities over 2012

Buffer errors, e.g. stack or heap based Buffer Overflows, have a large percentage of the CVE vulnerabilities, but are not web application specific. Also code injection is not web application specific. However, both are a web application threat because Buffer Overflows and code injection attacks are possible and do occur towards websites over the HTTP protocol.

Next to having understandable attack classes it is also important that classes are detectable. To get detectable attack classes we consider previous work on detecting web application attacks to find out what attack classes have discriminating features. Kruegel, Vigna, and Robertson (2005) used features like attribute length, and the use of non-printable characters, to detect attack classes Buffer Overflow, Directory Traversal and XSS. Buffer Overflow attacks are not often directly aimed at web applications, because Buffer Overflow attacks are a typical problem for programs written in C and C++. However, programming libraries or webserver components might be written in C or C++. It can then happen that web application input can cause Buffer Overflow problems when the input is eventually used in such a function or component.

Other attack classes that have been considered are: Local and Remote File Injection (LFI/RFI), Cross-Site Request Forgery (CSRF), Reconnaissance, (D)DoS, Brute Force or Dictionary Password attacks.
These classes have been dropped for the experiment for different reasons. LFI/RFI was dropped because there was no file upload possibility in the webservers used for testing, so occurrences or attempts of this attack are expected to be low. CSRF is dropped because that type of attack is hard to detect since the requests themselves in a CSRF attack are valid requests. In general, checking the Referer Http header ("Cross-Site Request Forgery (CSRF) Prevention Cheat Sheet", 2011) will detect part of the CSRF-attacks. However, the Referer header is not always present, e.g. when a URL-link is clicked in an email message (Wharton, 2007), leading to a high False Negative rate. Also, the fact that the referer is different does not necessarily mean a CSRF attempt is made, leading to a high False Positive rate. DDoS attacks are dropped because these are not 'content attacks', meaning that the attack is not in the contents of a separate HTTP-request, but in the amount of traffic towards a web application, server or IT network. A DDoS attack can be detected by monitoring the number of requests that is received within a short time period (seconds), but a chosen restriction in this experiment is that conclusions must be drawn from single requests only. For password attacks it also applies that this can only be concluded from combining multiple requests, although a single attempt might be detectable from the error response that is returned after a failed log in attempt. Single failures alone however will also lead to false positives.

For the experiment attack classes are chosen in such a way that:

- Attack classes are commonly well-know and therefore understandable in security monitoring operations and reporting,
- Attack classes have a high risk profile and therefore detection is important for companies,
- Attack classes seem to have clear discriminating features so detection might be possible.

Eventually, for the experiment, five attack classes have been chosen. The resulting classes that are defined for the experiment are:

- XSS, still very high on the (OWASP) list of web application risks and CVE statistics,
- SQL Injection, also high on the OWASP list in the more general class of Injection risks. I will define discriminating features for SQL injection only, but this might overlap with other types of injection like XPath injection.
- Buffer Overflows, because it might have very different discriminating features than the other attack classes. Also, explicit detection and registration of this attack class, when it occurs, might be important for security monitoring.
- Path Traversal, because of its discriminating features and it might also catch (part of the) Command Injection attempts that are left out in the definition of the SQL injection class compared to the larger OWASP Injection class.
- Reconnaissance attacks, to see if it is possible to define and detect the broad range of scanning attacks that precede a more intrusive attack. Detection of this class might be difficult because it could be a class that is hard to define unambiguously. The experiment might therefore be unsuccessful for this class, i.e. low True Positive rates and/or high False Positives rates.

For these five classes it is investigated what features can be used to classify attacks, and for all classes except Buffer Overflows an experiment is set up to test if successful detection is possible with a supervise learning approach. Buffer Overflow attacks are eventually left out of the experiment because during the test period it appeared that no Buffer Overflow alerts were received. This attack class is analysed for its features but extraction of these features is not implemented for the experiment.

5.1.2. Cross-site Scripting

In a Cross-site Scripting attack, often referred to as XSS, an attacker indirectly offers malicious client-side scripting code to a victim's browser to be executed. This code can be injected in a genuine website, for example in a forum post, or it can be included in a request query that the victim then has to use in order to trigger the attack. The by the victim activated malicious scripting code is then run in the security context of the victim's browser, having access to for example the victim's HTTP cookie data.

XSS can be classified as reflected, stored or DOM based. Reflected XSS is a non-persistent form of XSS, where a malicious script is included in, for example, a URL request query or in a cookie parameter. A victim is then tricked into using the URL, or the URL is hidden in a web page, for example in a hidden iframe[1] that is manipulated by the attacker. The server-side vulnerability in reflected XSS is that the malicious input is not detected and that it is returned (i.e. reflected) directly in the response. This causes the malicious script to be executed in the victim's browser environment. In stored XSS, an attacker injects a malicious script into a non-malicious but vulnerable web application, where the script is then stored within the web application. A victim that visits the web application later will then receive the injected script, again causing it to be executed in the victim's browser. Injection of scripts is possible in for example a forum or guestbook. The third class of XSS, DOM based XSS, is defined more recently (Klein, 2005) than the other two classes. DOM is the Document Object Model, an object oriented representation of documents such as HTML documents. With DOM, the document's contents and properties can be handled by programming languages such as JavaScript. Properties of an HTML document include HTTP information such as document.url and document.referrer. In a DOM based XSS attack, the DOM environment is influenced causing malicious behavior.

To circumvent detection of a XSS attack, attackers can obfuscate the recognizable attack patterns. An example where too straightforward detection did not work is the 2005 MySpace, or Samy XSS-worm, that infected MySpace. Myspace filtered all occurrences of "<script>" and "javascript". However, the Myspace worm used "java\nscript", with a newline character in between "java" and "script". As described by Wassermann, and Zhendong (2008), Internet Explorer concatenates strings broken by newlines and allows javascript in cascading style sheets. Also, Internet Explorer allows for script code in cascading style sheet tags, so the worm could include script as follows:

```
<div style="background:url('java\nscript: ...')">
```

Because of these two obfuscation techniques, the MySpace worm could circumvent straightforward detection, but remained functional.

Features of XSS attacks have been researched by others. In a recent study (Nunan, Souto, Santos & Feitosa, 2012) features were categorized in three groups: obfuscation based, suspicious patterns, and HTML/JavaScript schemes. Obfuscation based features were the use of alternative encodings such as hexadecimal or Base64 encodings, and the URL length because obfuscated code tends to be longer. Suspicious patterns were the use of multiple domain names in one URL, use of multiple special characters such as "<" to circumvent XSS filtering, and the use of keywords such as banking, redirect, root, password, commonly found in attacks. Their last category, HTML/JavaScript schemes were typical HTML-tags (e.g. <script>, <iframe>), HTML properties (e.g. href, HTTP-equiv), event handlers (e.g. onclick, onmouseover), DOM objects (e.g. Windows, Document), and other properties such as Cookie, Referrer or JavaScript methods such as write(), alert(), or fromCharCode(). In their experiment, Nunan et al. (2012) reached a detection rate up to 99% with a False Alarm rate as low as 0.22% using Naive Bayes. With the use of another statistical machine learning method, Support Vector Machines (SVM) even higher detection rates where achieved.

[1] An iframe is an HTML element; a hidden iframe is an iframe that is not displayed to the user by a web browser.

Wang, Mao, & Lee (2010) mention the use of HTML entity codes, URL encoding, Base64, and double encoding as frequently used obfuscation techniques in XSS attacks.

Johns, Engelmann, and Posegga (2008) used separate detection for reflective XSS and for stored XSS. Detection of reflective XSS was done, for example, by comparing HTTP request and response messages containing longer JavaScript code. They achieved a rate of zero false negatives, and no false-positives for 80% of the web applications used in their experiment.

In the detection of XSS attack class I will gather characteristics of these types of XSS attacks, and build the Bayesian detection model with these characteristics. First, all input is analyzed for the use of encodings, and then decoded thoroughly. After the decoding process, indications of suspicious patterns, tags, properties and event handler are analyzed.

5.1.3. SQL injection

With SQL injection an attacker is able to inject SQL keywords and operators, changing the intended SQL query that is executed in the web application's server side source code. Normally, a user would provide input that is used as a value in an expression, which is part of an SQL query. An attacker is able to escape the intended expression, extending the SQL query with malicious goals.

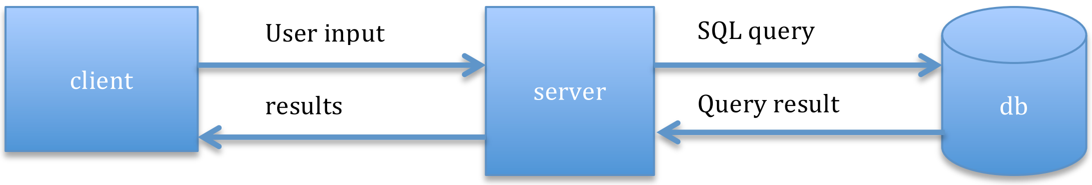


Figure. SQL injection

An example of SQL injection is when a web application requires a login, and the user has to provide a user-id and a password. This input would then typically be passed to the webapplication as POST-parameters where the login attempt is validated by checking the password in the database. The resulting vulnerable source code in a php webapplication could be:

$query = "SELECT * FROM users WHERE user='" + $_POST["userid"] + "' AND password='" + $_POST["password"] + "'";

An attacker could enter a userid "*hacker*" and a password "*idontknow' OR 'x'='x*". This would then result in the following query:

SELECT * FROM users WHERE user='hacker' AND password='idontknow' OR 'x'='x'

resulting in bypassing the authentication check, because logically this WHERE clause evaluates to being true always. An attacker could also enter a userid "admin'-- " resulting in the the query:

SELECT * FROM users WHERE user='admin'-- AND password=''

where '--' is the comment operator in SQL, resulting in removing the password check from the WHERE clause.

In SQL injection, as also mentioned with XSS, obfuscation is used by attackers to circumvent detection. An example of obfuscation using source code comments is from the ModSecurity SQL Injection challenge to bypass SQL injection filter detection. ModSecurity is a rule-based open source web application firewall. One of the successful attempts to bypass all filters ("SQL Injection Challenge: Lessons Learned", 2011) was with the following request:

http://www.modsecurity.org/testphp.vulnweb.com/artists.php?artist=0+div+1+union%23foo*%2F*bar%0D%0Aselect%23foo%0D%0A1%2C2%2Ccurrent_user

This request uses an, at that time undetected combination of MySQL comments. The SQL payload decodes to:

0 div 1 union#foo*/*bar
select#foo
1,2,current_user

This is executed as:

0 div 1 union select 1,2,current_user

So, while circumventing detection rules, the interpretation of the obfuscated query still leads to successful execution of the query.

In the previous example, the SQL injection attack query was added to an HTTP query parameter. Note that SQL injection can be successful in any input that is used in an SQL query. This input can come from GET query parameters, POST message data and also from cookie or other header contents.

Halfond, Viegas, and Orso (2006), and Halfond, Orso, and Manolios (2008) describe different types of SQL injection attacks and existing detection techniques. The different types of attacks and their characteristics are related to the attack intent of the attacker. The different attack intents defined are to:

- Identify injectable parameters,
- Perform database fingerprinting,
- Determine the database schema,
- Extract data, add or modify data,
- Perform denial of service,
- Evade detection,
- Bypass authentication,
- Execute remote commands and
- Perform privilege escalation.

For these intents, Halfond, et al (2006, 2008), describe the following different types of SQL injection:

- Tautologies, where an attacker transforms the WHERE clause into a tautology.
- Illegal/Logicallly incorrect queries, to cause exceptions, giving information about the type and structure of the backend database system.
- Union Query attacks, where an attacker can change the returned dataset by using a UNION SELECT construction.
- Piggy-Backed queries, where an extra query is added, e.g. with an input "Jack'; drop table users --" changing a query string into a double query like: SELECT * FROM users WHERE user='Jack'; drop table users -- <*rest of the original query*>
- Stored Procedures, where an attacker is able to exploit stored procedures, a mechanism often mentioned as the solution against SQL injection. This mechanism however can also be vulnerable to SQL injection attacks.
- Inference, where an attacker infers the successfulness of an attempt from the response behavior of the system. The noted behavior can be on different aspects, for example (lack of) continuation in the website, or changes in timing for successful or unsuccessful attempts. Examples of inference and other advanced SQL injection attacks can be found in (Anley, 2002).
- Alternate encodings, where different encoding of characters, e.g. hexadecimal or Unicode encoding, is used to evade detection mechanisms.

In the detection of the SQL injection attack class I will gather characteristics of these types of SQL injection attacks, and build the Bayesian detection model with these characteristics. First, all input is analyzed for the use of encodings, then decoded thoroughly. After the decoding process, indications of tautologies, SQL queries and typical SQL injection patterns are analyzed.

5.1.4. Buffer Overflows

In a Buffer Overflow attack, an attacker is able to run code (native CPU instructions) within a vulnerable computer process. The code can be injected by the attacker in the process' address space, or the attacker can jump to code already present on the system and available to the computer process. The vulnerability that makes the attack possible, a buffer overflow vulnerability, determines the name of this attack category. Different types of Buffer Overflows can be distinguished based on the location of the buffer in memory. Cowan, Wagle, Pu, Beattie & Walpole (2000) define stack based buffers, heap based buffers, and static data area buffers. An alternative approach, to injecting malicious code, used by attackers is to use code that is already available to the process. The attacker then only needs to be able to set certain parameters and to make the process jump to the particular code. An often mentioned example of this injectionless type of attack is the Return-to-libc attack method (Designer, S., 1997). This type of attack is developed and used to circumvent stack memory protection against Buffer Overflow attacks, where the stack memory is set to non-executable memory.
The school-book example of a Buffer Overflow attack is a stack-based Buffer Overflow attack, first described by AlephOne (1996), who's real name is Elias Levy, in the infamous article "Smashing The Stack For Fun and Profit". In the typical stack Buffer Overflow attack an attacker injects malicious code into a vulnerable buffer that is located in stack memory. The attacker then overflows the buffer until the stack frame's return address is reached. This return address is then overwritten with the starting location of the injected code, causing the process to jump to that location. This happens in the process' execution flow when the return address is read from the stack frame and replaced in the processor's instruction pointer register.

Since Buffer Overflows attack data typically contain computer processor instructions and computer memory addresses, the percentage of non-printable characters will often be high compared to normal attribute values in HTTP requests.
Robertson, Vigna, Keugel & Kemmerer (2006) mention the use of binary value as ASCII values greater than 0x80 as a general characteristic of Buffer Overflow attacks. According to Wang, Pan, Liu & Zhu (2010), "buffer overflow attacks typically contain executables whereas legitimate client requests never contain executables in most Internet services". Their SigFree application level detection, a signature free Buffer Overflow detection tool, looks for executable code in input.

Another typical characteristic of Buffer Overflow attacks is the use of extra nop (no operation) instructions as a so-called nop-sled or -sledge before the actual exploit code. A nop-sledge is used by attackers to increase the chance of a successful attack. With a nop-sledge placed before the injected code, the jump to the starting memory address of the injected code can be less precise, solving the problem that an attacker might not know the exact memory location within the stack memory where the injected code is copied. Toth & Kruegel (2002) focus on detecting such a nop-sledge as the detection mechanism for Buffer Overflows. The typical nop instruction in an Intel processor is the instruction with opcode 0x90, however other instructions can also be used with the same *no operation* effect. Toth & Kreugel state that for Intel IA32 architecture alone over 50 single-byte opcodes can be used as a nop instruction. Also multi-byte opcodes with arguments can be used by an attacker if applied carefully. In fact, any instruction that has no negative side effect in the attack can be used by an attacker as a nop instruction. For example, if a CPU register EBX is not used during the attack, the attacker could include adding a random value to the register's current contents. An attacker can use any combination of operations in a nop-sledge, making straightforward detection of nop-sledges impossible in advanced attacks. Toth & Kruegel use *abstract execution* to test for valid nop instructions. With abstract execution, the byte values of input are evaluated and interpreted to test if it can be represented as a sequence of consecutive valid instructions. For this, the byte sequences must be syntactically valid instructions and all used memory addresses must be accessible by the process. A nop-sledge is then recognizable by the length of executable sequences, which are long compared to those in legitimate requests.

In the reconnaissance phase of a Buffer Overflow attack, or in testing for Buffer Overflow vulnerabilities, fuzzing is often used. When fuzzing for Buffer Overflows and learning the correct memory locations for an attack, an attacker will often start with injection of test strings with increasing length. If a Buffer Overflow vulnerability exists, eventually a test string will overflow a return address or other vital memory values, causing a crash or different behaviour of the vulnerable computer process. Vulnerability scanners also use fuzzing to detect vulnerabilities. A single- or multi-byte pattern repetition detection could be used for detection of the typical application of Buffer Overflow fuzzing.

Hsu, Guo, & Chiueh, (2006) detect repeating memory addresses in their network based detection system called Nebula. Nebula detects the memory address that is injected in an attack as the starting location of the malicious code to be executed. It is stated that this memory address is often repeated "to accommodate differences in the address of the target control-sensitive data structure due to different combinations of compiler, loader, operating system, and command-line arguments". These (repeating) memory addresses should be located in the stack region or text region of process memory for respectively code injection attacks or return-to-libc attacks.

Features that can be used in the experiment are the number of non-printable characters, and repeating patterns of different lengths and formats. Also the abstract execution detection mechanism of Toth & Kruegel (2002), although rather complex to implement, could give good success rates.

5.1.5. Path Traversal

In a Path Traversal, or Directory Traversal attack, an attacker tries to access files and folders on the webserver that are outside the web root folder. Path Traversal normally tries to navigate out of the virtual webfolder towards the server's system folders using relative paths. A typical example of such a Path Traversal attack:

http:// www.example.com/vulnerable.php?search=../../../../../../../../../etc/passwd

However, also absolute paths can be used in Path Traversal attempts:

http://www.example.com/?search=C%3A%2Finetpub%2Fwwwroot%2Fglobal.asa

which will (url-)decode to:

http://www.example.com/?search=C:/inetpub/wwwroot/global.asa

Obfuscation and encoding is often used in Path Traversal attempts to avoid detection. URL-encoding and double-encoding might also be used to obfuscate suspicious path patterns in the HTTP request query to the user:

http://www.example.com/scripts/..%255c../winnt/system32/attrib.exe?c:\*.*

Here, double URL-encoding is used, which means that the percentage character in the URL-encoding is also URL-encoded. This request, in a first decoding step, decodes to:

http://www.example.com/scripts/..%5c../winnt/system32/attrib.exe?c:\*.*

and in a second decoding step to:

http://www.example.com/scripts/../../winnt/system32/attrib.exe?c:\*.*

If, in the detection process only one round of decoding is performed, the Path Traversal pattern will not be revealed and therefore probably not detected.

A special case of Path Traversal obfuscation in Microsoft's IIS webserver was discovered in 2000. In the Unicode support so-called overlong representation of characters were supported. Because of this characters '/' and '\' could be represented in URL-encoded overlong Unicode representation as respectively '*%c0%af*' and '*%c1%9c*'. An example use in a Path Traversal attempt in an HTTP POST request:

POST /index.php/?action=submitlogin&returnto=Speciaal:Zoeken& title=..%c0%af..%c0%af..%c0%af..%c0%af..%c0%af..%c0%af..%c0%af..%c0%afetc/passwd &type=login HTTP/1.1

Such patterns will not be decoded in a URL-decoding process, but an IIS webserver would interpret it as a directory path.

5.1.6. Reconnaissance

During testing, acting as an IDS administrator handling event alerts, there were anomaly alerts that could not be placed in a standard category of for example XSS, or SQL injection attacks, but were not false positives either. For example, when a request is done to identify certain functionality such as WordPress content management system, which is not present on the guarded webserver, it was labelled as reconnaissance. Also, header-less HTTP-request attempts, null-scans, and other types of scans were labeled as Reconnaissance, sometimes without being able to determine the (malicious) purpose of the Reconnaissance.

To show for the diversity of alerts that were labeled as Reconnaissance attacks some examples are given next with an explanation.

Attempt to connect to radmin Trojan.

Radmin is a remote administration tool, not necessarily malicious. However, it can be installed by a Trojan to give remote access to for example a Command & Control server controlling the infected computer remotely. The alert was given on binary data received over port 80 as displayed in the figure below. Since the server is not infected and radmin is not running on the server, this alert was labeled as a Reconnaissance, in this case a random scan for radmin.

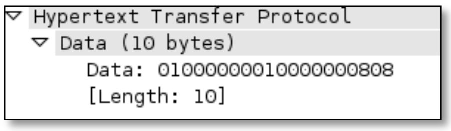

```
▽ Hypertext Transfer Protocol
  ▽ Data (10 bytes)
      Data: 01000000010000000808
      [Length: 10]
```

Figure. Binary data over http

Snort has a rule defined for this specific event, the rule with sid 12373.

```
alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"BACKDOOR radmin 3.0 runtime detection - initial connection"; flow:to_server,established; content:"|01 00 00 00 01 00 00 00 08 08|"; depth:10; flowbits:set,Radmin3.0_conn_detection; flowbits:noalert; classtype:trojan-activity; sid:12373; rev:1;)
```

Figure. Snort rule for radmin Trojan

W00tw00t

The w00tw00t Reconnaissance scan is a malformed HTTP Get-request. The request below is the complete request, so no host- or other headers are included:

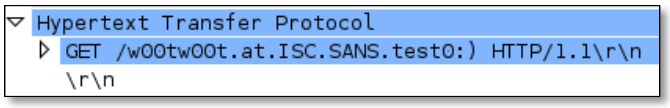
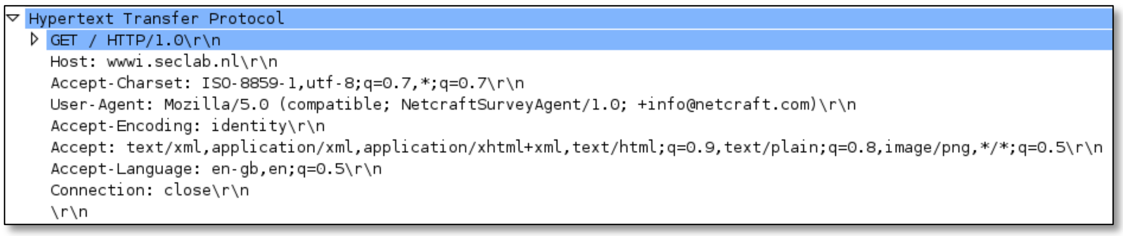

```
▽ Hypertext Transfer Protocol
  ▷ GET /w00tw00t.at.ISC.SANS.test0:) HTTP/1.1\r\n
    \r\n
```

Figure. Malformed HTTP request

Because of the absolute path in the request URI, the host-header is required to specify the hostname. Therefore this request is not conform the HTTP standard, causing an HTTP return status code 400 Bad request.

HTTP/1.0 requests and invalid host-value

The current version of the HTTP protocol is 1.1 since 1996. The only alternative is HTTP/1.0, but this is normally not used any more by browsers, although some tools such as the well-known *wget* utility use http version 1.0. In the example request below http version 1.0 is used by a Netcraft bot.



```
▽ Hypertext Transfer Protocol
  ▷ GET / HTTP/1.0\r\n
    Host: wwwi.seclab.nl\r\n
    Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7\r\n
    User-Agent: Mozilla/5.0 (compatible; NetcraftSurveyAgent/1.0; +info@netcraft.com)\r\n
    Accept-Encoding: identity\r\n
    Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5\r\n
    Accept-Language: en-gb,en;q=0.5\r\n
    Connection: close\r\n
    \r\n
```

Figure. HTTP/1.0 request

Another characteristic in this request is the incorrect Host-header value *wwwi.seclab.nl*. The request originates from the Netcraft internet survey exploring the internet, keeping statistics such as the number of active websites, and webserver market shares. It is not really malicious behavior but is not a normal website request either.

Characteristics of the Reconnaissance attack class

Because the malicious requests that were labeled as Reconnaissance attacks vary a lot it has no shared concrete attack features. However, shared characteristics might come from the validity of the HTTP requests and error codes returned. Features that were extracted for the category of Reconnaissance attacks are:

1. HTTP/1.0 requests, which appears typical for scans, malicious or non-malicious.
2. Error return codes, such as 400 Bad request response, 404 Not Found, 405 Method not allowed are a typical signal for Reconnaissance attacks.
3. A host-value containing an IP-address instead of using the domain name for the website.
4. An incorrect host-value (either as domain name or ip-address).

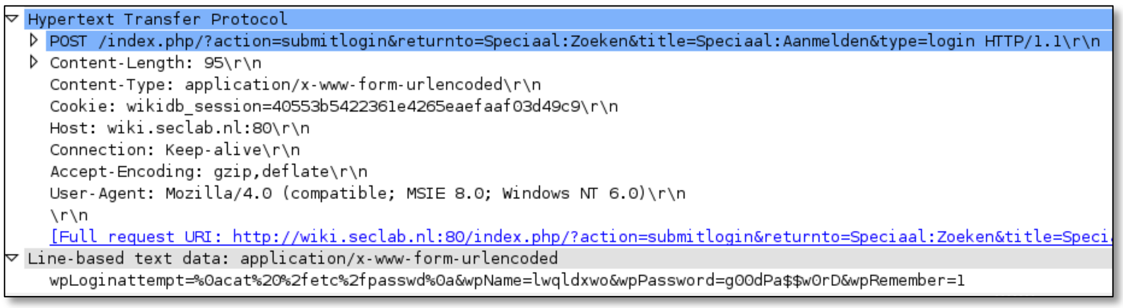
5.1.7. Other

Any attack that cannot be labelled as one of the five attack categories is labelled as ‘other’. In fact, this then becomes a sixth category that also must be detected by the Bayesian classifier. It is a strange class because it cannot be defined. It will be interesting to find out how a Bayesian classifier copes with an undefined class.

To show for the diversity of alerts that were labeled as ‘Other’, some examples are given next with an explanation.

Command injection attack

The request below was part of an attack that consisted of multiple attempts with command injection type of attack attempts.



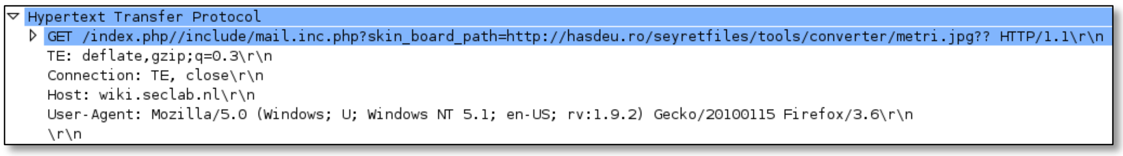
```
▽ Hypertext Transfer Protocol
  ▷ POST /index.php/?action=submitlogin&returnto=Speciaal:Zoeken&title=Speciaal:Aanmelden&type=login HTTP/1.1\r\n
  ▷ Content-Length: 95\r\n
    Content-Type: application/x-www-form-urlencoded\r\n
    Cookie: wikidb_session=40553b5422361e4265eaefaaf03d49c9\r\n
    Host: wiki.seclab.nl:80\r\n
    Connection: Keep-alive\r\n
    Accept-Encoding: gzip,deflate\r\n
    User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.0)\r\n
    \r\n
    [Full request URI: http://wiki.seclab.nl:80/index.php/?action=submitlogin&returnto=Speciaal:Zoeken&title=Speci
▽ Line-based text data: application/x-www-form-urlencoded
    wpLoginattempt=%0acat%20%2fetc%2fpasswd%0a&wpName=lwqldxwo&wpPassword=g00dPa$$w0rD&wpRemember=1
```

Figure. Command injection attack

The HTTP request itself is a valid login request, but the value of the wpLoginattempt parameter contains URL-encoded character values of non-printable binary characters decoded to a command injection attempt of “cat /etc/passwd”.

Remote File Inclusion

In the request below an attempt is made to include a remote file metri.jpg which probably is not a jpeg picture file, but in fact is a malicious executable file or script.

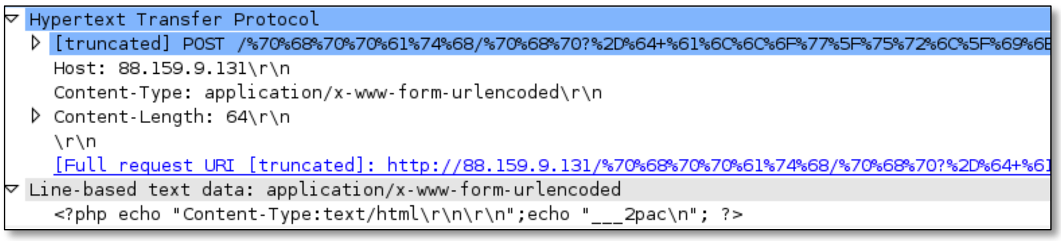


```
▽ Hypertext Transfer Protocol
  ▷ GET /index.php//include/mail.inc.php?skin_board_path=http://hasdeu.ro/seyretfiles/tools/converter/metri.jpg?? HTTP/1.1\r\n
    TE: deflate,gzip;q=0.3\r\n
    Connection: TE, close\r\n
    Host: wiki.seclab.nl\r\n
    User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.9.2) Gecko/20100115 Firefox/3.6\r\n
    \r\n
```

Figure. Remote File Inclusion attack

Exploit attempt

This request shows a first step in the exploit of a vulnerability in the webhosting management tool Plesk.



```
▽ Hypertext Transfer Protocol
  ▷ [truncated] POST /%70%68%70%70%61%74%68/%70%68%70?%2D%64+%61%6C%6C%6F%77%5F%75%72%6C%5F%69%6E
    Host: 88.159.9.131\r\n
    Content-Type: application/x-www-form-urlencoded\r\n
  ▷ Content-Length: 64\r\n
    \r\n
    [Full request URI [truncated]: http://88.159.9.131/%70%68%70%70%61%74%68/%70%68%70?%2D%64+%61
▽ Line-based text data: application/x-www-form-urlencoded
    <?php echo "Content-Type:text/html\r\n\r\n";echo "___2pac\n"; ?>
```

Figure. Exploit attempt Plesk vulnerability

The POST request is a heavily URL-encoded request:

```
POST
/%70%68%70%70%61%74%68/%70%68%70?%2D%64+%61%6C%6C%6F%77%5F%75%72%6C%5F%69%
6E %63%6C%75%64%65%3D%6F %6E+%2D%64+%73%61%66%65%5F%6D%6F%64%65%3D%6F%66%66
+%2D%64+%73%75%68%6F%73%69%6E%2E %73%69%6D%75%6C%61%74%69%6F%6E%3D%6F%6E+%
2D%64+%64%69%73%61%62%6C%65%5F%66%75%6E %63%74%69%6F%6E%73%3D%22%22+%2D%64
+%6F%70%65%6E%5F%62%61%73%65%64%69%72%3D%6E%6F %6E%65+%2D%64+%61%75%74%6F%
5F%70%72%65%70%65%6E%64%5F%66%69%6C%65%3D%70%68%70%3A %2F%2F%69%6E%70%75%7
4+%2D%6E HTTP/1.1
```

The request decodes to:

```
/phppath/php?-d allow_url_include=on -d safe_mode=off -d suhosin.simulation=on -d
disable_functions="" -d open_basedir=none -d auto_prepend_file=php://input -n T
```

The POST data simply echo's a string to be able to detect if the vulnerability exists as a first step in the further attack if the webserver turns out to be vulnerable.

```
<?php echo "Content-Type:text/html\r\n\r\n";echo "__2pac\n"; ?>
```

These different attacks show that no overlapping features can be defined, and in further research, clear attack classes should be chosen for the other types of attacks. For this experiment, the detection must come from the fact that these attacks do not match any other attack class.

5.2. Attack features

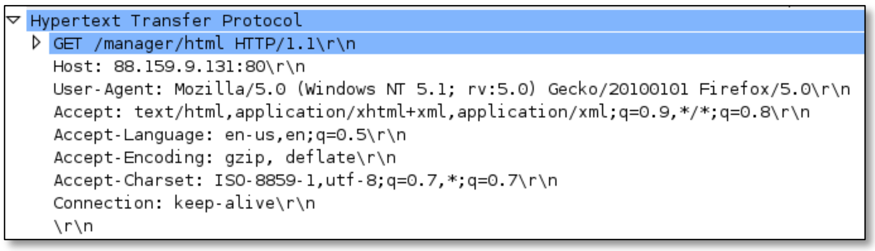
From the analysis of the defined attack classes attack features are selected. The alerts' attack features will be extracted from HTTP query parameters, HTTP headers and POST data. In total, 24 features are defined. An overview and short description of all features is listed below. The justification for most of these features comes from the analysis of the attack features in the attack classes as described in the previous paragraph. After describing the features an overview is given of the expected contribution of the features to the attack classes.

5.2.1. Feature overview

For the 24 chosen features a description is given, the value type and expected values are given, and if relevant expectations or limitations are described.

URL length.

The URL length is defined as the total length in bytes of the request URI in the HTTP request. This length is taken before any decoding is done.



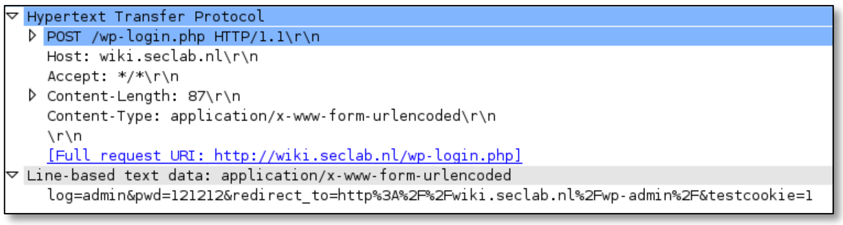
```
▽ Hypertext Transfer Protocol
  ▷ GET /manager/html HTTP/1.1\r\n
    Host: 88.159.9.131:80\r\n
    User-Agent: Mozilla/5.0 (Windows NT 5.1; rv:5.0) Gecko/20100101 Firefox/5.0\r\n
    Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n
    Accept-Language: en-us,en;q=0.5\r\n
    Accept-Encoding: gzip, deflate\r\n
    Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7\r\n
    Connection: keep-alive\r\n
    \r\n
```

Figure. Example where the URL length is 13 characters

The URL length in the example is 13 characters which is the length of the request URI '/manager/html'. The expectation is that the URL length increases when XSS scripts are included. This will be amplified when obfuscation techniques are used. With SQL injection attacks the length might also increase, but to a lesser extent. Path Traversals normally do not need extra lengthy parameter values, unless the traversals are overdone. Part of the reconnaissance attacks have a shorter length in stead of a longer length, but this will then also be differentiating for that attack class. Other attacks can have a broader value range for the URL length because different types of attacks are in this class.
Of course it should be noted that URI's by themselves can be of arbitrary length, so a lot of noise is expected in this value too. It also should be noted that the attack could very well not be in the request URI at all, but for example in the POST data or header-values, leading to more noise.

Maximum parameter length.

The maximum parameter length is defined as the maximum length in bytes of any of the parameters found in the request-URI, headers, or POST data. It is similar to the URL length, but more specific and also measures the length of header values and post data parameters.



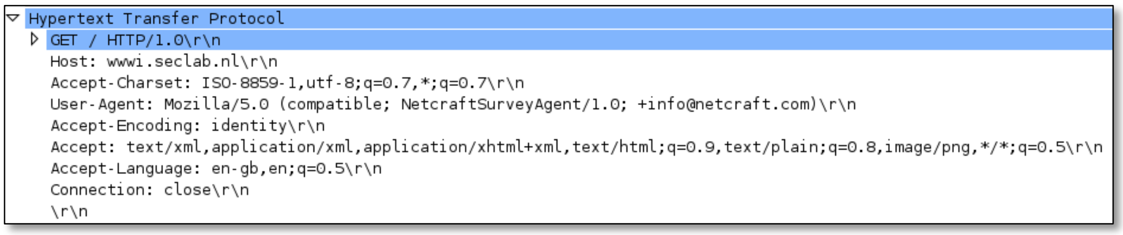
```
▽ Hypertext Transfer Protocol
  ▷ POST /wp-login.php HTTP/1.1\r\n
    Host: wiki.seclab.nl\r\n
    Accept: */*\r\n
  ▷ Content-Length: 87\r\n
    Content-Type: application/x-www-form-urlencoded\r\n
    \r\n
    [Full request URI: http://wiki.seclab.nl/wp-login.php]
▽ Line-based text data: application/x-www-form-urlencoded
    log=admin&pwd=121212&redirect_to=http%3A%2F%2Fwiki.seclab.nl%2Fwp-admin%2F&testcookie=1
```

Figure. Example where an irrelevant maximum parameter value is taken

In the example the longest parameter is the header Content-Type, not the POST parameter redirect_to that contains the intrusive data. As shown by the example the wrong value can be taken for this feature if a longer parameter value exists than the parameter with the intrusion data. This parameter therefore only contributes to the analysis for long attack values and gives noise in stead of extra contribution to the attack characteristics for shorter values. Note that if alert information from the SilentDefense IDS would have been used, only the parameter that caused the alert could be used for this feature's value.

HTTP version

The HTTP-version in the request, normally 'HTTP/1.1' because that is the standard since 1999. The only other valid option is 'HTTP/1.0'. As explained in the previous paragraph, reconnaissance scans sometimes use HTTP version 1.0. For other attack classes this version value is not expected.

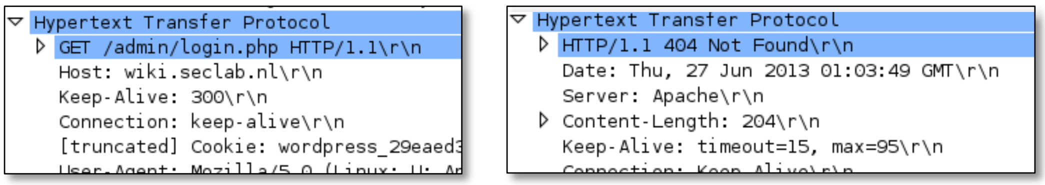


```
▽ Hypertext Transfer Protocol
  ▷ GET / HTTP/1.0\r\n
    Host: wwwi.seclab.nl\r\n
    Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7\r\n
    User-Agent: Mozilla/5.0 (compatible; NetcraftSurveyAgent/1.0; +info@netcraft.com)\r\n
    Accept-Encoding: identity\r\n
    Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5\r\n
    Accept-Language: en-gb,en;q=0.5\r\n
    Connection: close\r\n
    \r\n
```

Figure. HTTP/1.0 request

HTTP status code response

The return code in the HTTP response. The possible return code values are defined for five classes of responses:
1xx – Informational; This code is not seen a lot and not specific for any of the attacks.
2xx – Succes; Mainly code 200 OK is expected for most attack attempts.
3xx – Redirection; The client must undertake action to complete a request, e.g. go to another URI.
4xx – Client Error; Client seems to have erred, and receives e.g. an Authentication error
5xx – Server Error; Server failed to fulfil a valid request.



```
▽ Hypertext Transfer Protocol
  ▷ GET /admin/login.php HTTP/1.1\r\n
    Host: wiki.seclab.nl\r\n
    Keep-Alive: 300\r\n
    Connection: keep-alive\r\n
    [truncated] Cookie: wordpress_29eaed3
```

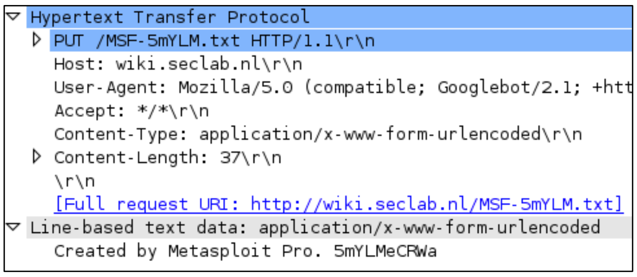
```
▽ Hypertext Transfer Protocol
  ▷ HTTP/1.1 404 Not Found\r\n
    Date: Thu, 27 Jun 2013 01:03:49 GMT\r\n
    Server: Apache\r\n
  ▷ Content-Length: 204\r\n
    Keep-Alive: timeout=15, max=95\r\n
```

Figure. Request and Error response HTTP return code 404

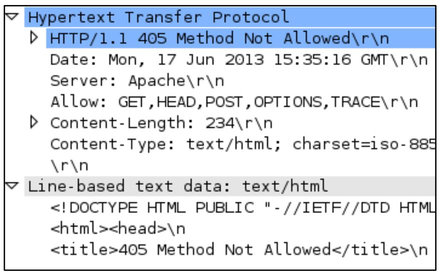
Error codes or expected Reconnaissance attacks more often than for other attack types. As can be seen in the example an HTTP 404 return code is returned on a Reconnaissance scan for a Wordpress login script. Reconnaissance attacks are expected to generate most HTTP errors because this class contains invalid requests and requests to non-existing resources. Other attacks such as Path Traversals, SQL injection and Other attacks might sometimes generate an error. XSS is not expected to generate a lot of errors.

HTTP request method

The HTTP request method that was used, e.g. GET, or HEAD, is also defined as a feature, mainly for Reconnaissance attacks. GET and POST requests are expected mostly for all attack classes. Only for Reconnaissance some use of other request methods is expected.



```
▽ Hypertext Transfer Protocol
  ▷ PUT /MSF-5mYLM.txt HTTP/1.1\r\n
    Host: wiki.seclab.nl\r\n
    User-Agent: Mozilla/5.0 (compatible; Googlebot/2.1; +htt
    Accept: */*\r\n
    Content-Type: application/x-www-form-urlencoded\r\n
  ▷ Content-Length: 37\r\n
    \r\n
    [Full request URI: http://wiki.seclab.nl/MSF-5mYLM.txt]
▽ Line-based text data: application/x-www-form-urlencoded
    Created by Metasploit Pro. 5mYLMeCRWa
```



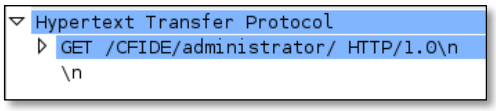
```
▽ Hypertext Transfer Protocol
  ▷ HTTP/1.1 405 Method Not Allowed\r\n
    Date: Mon, 17 Jun 2013 15:35:16 GMT\r\n
    Server: Apache\r\n
    Allow: GET,HEAD,POST,OPTIONS,TRACE\r\n
  ▷ Content-Length: 234\r\n
    Content-Type: text/html; charset=iso-885
    \r\n
▽ Line-based text data: text/html
    <!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML
    <html><head>\n
    <title>405 Method Not Allowed</title>\n
```

Figure. A PUT request and response error

In the example a PUT request is made to the webserver, which is not allowed, hence the return code 405 Method not allowed.

HTTP request has no CRLF ending

After the request line and headers, an HTTP request must end with a carriage return (CR) and a line feed (LF) according to the HTTP standard. Non-compliant requests might indicate requests generated by tools or incomplete malicious scripts. These requests are expected mainly with Reconnaissance attacks and maybe with Other type of attacks.



```
▽ Hypertext Transfer Protocol
  ▷ GET /CFIDE/administrator/ HTTP/1.0\n
    \n
```

Figure. An HTTP request that is non-compliant to the HTTP standard.

Extra URL encodings

Some reserved characters have to be encoded in the URL in an HTTP request when they have a special meaning in the URL.

!	#	$	&	‘	(	)	*	+	,	/	:	;	=	?	@	[	]
%21	%23	%24	%26	%27	%28	%29	%2A	%2B	%2C	%2F	%3A	%3B	%3D	%3F	%40	%5B	%5D

Table. Reserved characters and their URL encoded values.

Attackers often encode extra characters for obfuscation. URL encoding is applied where this is not required to obfuscate the attack. This unnecessary URL encoding is therefore suspicious. Unnecessary URL encoding is expected in all attack classes except for scans. The amount of extra URL encodings is counted, which might give a different distribution for different types of attacks. An example of the use of unnecessary URL encodings in a XSS attack:

POST
/index.php/?action=submitlogin&returnto=Speciaal:Zoeken&title=Speciaal:Aanmelden&type=%27%22%28%29%26%251%3cScRiPt%20%3eprompt%28933278%29%3c%2fScRiPt%3e HTTP/1.1\r\n

Which URL-decodes to:

POST
/index.php/?action=submitlogin&returnto=Speciaal:Zoeken&title=Speciaal:Aanmelden&type='"()&%1<ScRiPt >prompt(933278)</ScRiPt>
HTTP/1.1\r\n

Double encodings

The use of %25 as the URL encoded percent character is often used by attackers in an attempt to circumvent detection. If the ‘%’ of a URL encoded character is also URL encoded, two rounds of decoding are necessary to be able to detect an attack pattern. Some detection mechanism might not do this correctly. An example:

```
GET /index.php/Speciaal:Zoeken?
search=%253CIMG%2BSRC%253D%2522javascript%253Aalert%2528%2527XSS%2527%2
529%253B%2522%253E&
fulltext=Zoeken HTTP/1.1
```

Which, in the first decoding round, decodes to:

```
GET /index.php/Speciaal:Zoeken?
search=%3CIMG+SRC%3D%22javascript%3Aalert%28%27XSS%27%29%3B%22%3E&
fulltext=Zoeken HTTP/1.1
```

And in the second round to:

```
GET /index.php/Speciaal:Zoeken?
search=<IMG SRC="javascript:alert('XSS');">&
fulltext=Zoeken HTTP/1.1
```

Double encodings are expected in XSS attacks, but also in SQL injection and Path Traversal attacks.

Unicode, Base64, and Hex encoding of characters

Other types of obfuscation through encoding are also extracted as features, such as overlong Unicode representations, Base64 encoding of attack code, or Hex encoded text characters. Unicode can be used by attackers the hide attacks in request URI's or request message. The use of Base64 encoding was mentioned as an obfuscation technique in XSS attacks, but could be used in other attack forms also. The use of hex encoded data, especially of printable characters is very suspicious. All these encodings indicate possible malicious intent. The amount of encoding is counted for these features, which might lead to a different distribution for the attack classes. An example where hex encoded characters are used in a XSS attack:

```
GET /index.php/Speciaal:Zoeken?search=pindakaas&
go=eval('\x61\x6c\x65\x72\x74(1)') HTTP/1.1\r\n
```

Which decodes to:

```
GET /index.php/Speciaal:Zoeken?search=pindakaas&go=eval('alert(1)') HTTP/1.1\r\n
```

Malicious Tags, Suspicious XSS patterns, HTML entity codes, Javascript methods, event handlers, Suspicious DOM objects

These features are used to detect XSS mainly. HTML Tags or specific XSS patterns that are often used in XSS attacks. HTML tags to look for are *<script>*, *<javascript>*, *<iframe>*, and attributes such as *http-equiv*, and *lowsrc*. Suspicious patterns were mentioned from other research in the previous paragraph to occur frequently in XSS attacks, e.g. "*xss*", use of double angle brackets '*<<*' as an extra form of obfuscation against detection. Also the use of html comments is seen as a way to obfuscate an XSS attack. HTML entity names (e.g. *&lt*) and codes (e.g. *&#69*, or *&#x45*) are normally used to encode reserved characters in HTML. In an attack it can be used as an extra form of obfuscation, also applied when not required. Suspicious javascript methods are e.g. *fromCharCode()*, *alert()*, *eval()*. Javascript event handlers such as *onMouseOver* or *onLoad* are sometimes used in XSS attacks. If present in request parameters, these event handlers are suspicious. Also the occurrence of DOM objects and properties that are sometimes used in XSS attacks are suspicious when present in request parameters, such as DOM objects *Windows* and *Location*, or DOM object properties such as *Cookie* and *Referrer*.

An example with some of these features:

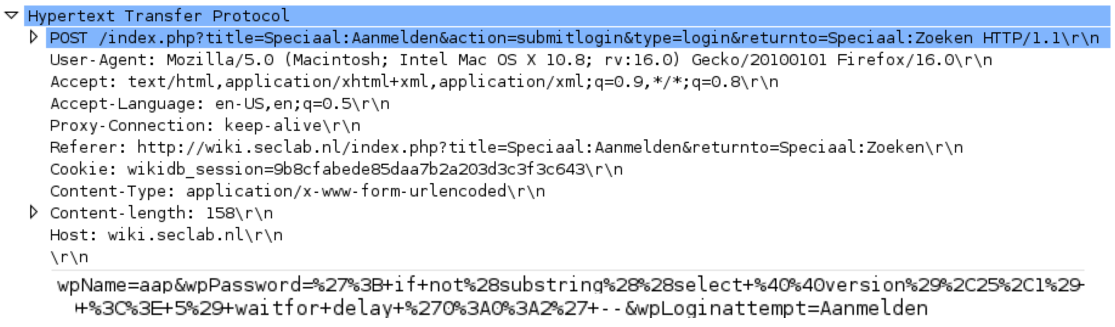
```
GET /index.php/Speciaal:Zoeken?
search=<META HTTP-EQUIV="Set-Cookie"
Content="USERID=&lt;SCRIPT&gt;alert('XSS')&lt;/SCRIPT&gt;">& go=OK HTTP/1.1
```

After decoding of HTML entity codes:

GET /index.php/Speciaal:Zoeken?
search=<META HTTP-EQUIV="Set-Cookie"
Content="USERID<SCRIPT>alert('XSS'</SCRIPT>">&
go=OK HTTP/1.1

Tautologies, SQL queries, SQL injection patterns

Specifically for SQL injection, features that are extracted are tautologies, SQL queries and specific SQL injection patterns. Tautologies are used in SQL injection attacks to adapt the logic of a WHERE clause in an SQL query. SQL queries are also registered and the use of keywords that are typical for SQL injection attacks, e.g. TABLESPACE, granted_role, Postgress pg_-structures, or stored procedure sp_-structures. For an operational implementation these features can be categorized to the type of database used. If the database type is known it can be configured for the classification to make query detection more precise. SQL injection patterns are typical patterns for these attacks such as the use of SQL comments. The number of occurrences for the different features in the request data are extracted after thoroughly decoding and de-obfuscating the data first.



```
▽ Hypertext Transfer Protocol
  ▷ POST /index.php?title=Speciaal:Aanmelden&action=submitlogin&type=login&returnto=Speciaal:Zoeken HTTP/1.1\r\n
    User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.8; rv:16.0) Gecko/20100101 Firefox/16.0\r\n
    Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n
    Accept-Language: en-US,en;q=0.5\r\n
    Proxy-Connection: keep-alive\r\n
    Referer: http://wiki.seclab.nl/index.php?title=Speciaal:Aanmelden&returnto=Speciaal:Zoeken\r\n
    Cookie: wikidb_session=9b8cfabede85daa7b2a203d3c3f3c643\r\n
    Content-Type: application/x-www-form-urlencoded\r\n
  ▷ Content-length: 158\r\n
    Host: wiki.seclab.nl\r\n
    \r\n
     wpName=aap&wpPassword=%27%3B+if+not%28substring%28%28select+%40%40version%29%2C25%2C1%29
       +%3C%3E+5%29+waitfor+delay+%270%3A0%3A2%27+--&wpLoginattempt=Aanmelden
```

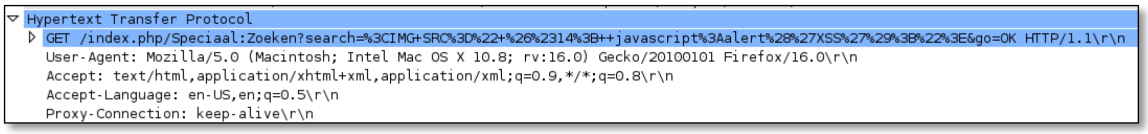
Figure. SQL injection attempt

In this request the POST data decodes to the following data, containing an SQL injection attempt:

wpName=aap&wpPassword='; if not(substring((select @@version),25,1) <> 5)
waitfor delay '0:0:2' --&wpLoginattempt=Aanmelden

Non-printable characters

Non-printable characters where mainly included as a possible indication of Buffer Overflow attacks and exploit code. However, non-printable characters can also be included as encoded characters, such as as an html-entity code in the example below.



```
▽ Hypertext Transfer Protocol
  ▷ GET /index.php/Speciaal:Zoeken?search=%3CIMG+SRC%3D%22+%26%2314%3B++javascript%3Aalert%28%27XSS%27%29%3B%22%3E&go=OK HTTP/1.1\r\n
    User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.8; rv:16.0) Gecko/20100101 Firefox/16.0\r\n
    Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n
    Accept-Language: en-US,en;q=0.5\r\n
    Proxy-Connection: keep-alive\r\n
```

Figure. Use of a non-printable html entity encoded character.

Which decodes to the following parameter value for the search parameter, showing the html-entity code &#14 for a non-printable character, which is the decimal ASCII character 14, the shift-out control character.

?search=<IMG SRC=" &#14; javascript:alert('XSS');">

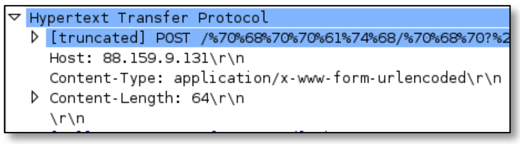
For this feature, the number of non-printable characters are counted in the request data.

Path Traversals

For Path Traversals data is analysed for different dot-dot-slash variations. The number of dot-dot-slash patterns or variations are counted. To differentiate short or absolute Path Traversals from other requests that might also contain the same patterns, a relative count is used, the percentage of the length of the value that is made up of the defined Path Traversal patterns.

IP host value or Incorrect host value

The Host header-value sometimes contains the IP-address of the website in stead of the domain name. This is not incorrect, but it is abnormal, because in normal use a webserver request uses the domain name of the webserver. The Host value can also be incorrect, neither the correct domain name nor a correct ip-address is then used as the host-value. Pen-testing tools and scanners, but also bots, sometimes use incorrect values. This feature's value can be True or False.



```
▽ Hypertext Transfer Protocol
  ▷ [truncated] POST /%70%68%70%70%61%74%68/%70%68%70?%2
    Host: 88.159.9.131\r\n
    Content-Type: application/x-www-form-urlencoded\r\n
  ▷ Content-Length: 64\r\n
    \r\n
```

Figure. Example where the ip-address of the website is used as host value

5.2.2. Expected feature contribution to attack classes

In the data set, these 24 features will be analysed and used to train and operate a Bayesian classifier with. The expectation is that most attributes have larger contribution to one attack class, and that the contribution to other classes will be either absent or the frequency or amount of occurrence will be much lower. The table below gives an overview of the expected contribution of all features to all attack classes.

Attack class: / Feature:	XSS	SQL injection	Path Traversal	Recon-naissance	Other
URL length	+	+/-	-	-	+/-
Maximum parameter length	+	+/-	-	-	+/-
HTTP version	-	-	-	+/-	-
HTTP response status code	2xx	2xx	2xx / 4xx	2xx / 3xx	all
HTTP request method	GET/POST	GET/POST	GET/POST	all	GET/POST
No CRLF ending	-	-	-	+/-	-
Extra URL encodings	+	+/-	+	-	+/-
Double encodings	+	+/-	+	-	-
Unicode	+/-	-	+/-	-	+/-
Base64 Encodings	+	-	-	-	+/-
Hexcodes	+/-	+/-	-	-	+/-
Malicious HTML tags	+	-	-	-	-
Suspicious XSS patterns	+	-	-	-	-
Html entity codes	+	-	-	-	-
Malicious use of Javascript methods	+	-	-	-	-
Malicious use of event handlers	+	-	-	-	-
Malicious use of DOM objects	+	-	-	-	-
Tautologies	-	+	-	-	-
SQL queries	-	+	-	-	-
SQL injection patterns	-	+	-	-	-
Non-printable characters	-	-	-	+/-	+/-
Path Traversals	-	-	+	-	+
IP address host value	-	-	-	+	+/-
Wrong host	-	-	-	+	+/-

Table. Expected contribution of the selected features to the defined attack classes

6. Data Collection

To be able to collect and analyse attack features, a dataset with labelled anomalies must be gathered from an anomaly based IDS.

6.1. System Setup

In the setup for the experiment the anomaly based IDS SilentDefense is used to collect anomaly alerts. SilentDefense was set up to guard the webserver hosting http://wiki.seclab.nl. During a training period the IDS learned to distinguish anomalies from normal traffic. After the training period was over, SilentDefense went into operation, analysing all requests to the website in real-time. Whenever SilentDefense came up with an alert, it was analysed what attack class the alert was in, labelling the alert with that class manually. Attacks where received from real attackers, but also from test patterns using the penetration tool ZAP (Zed Attack Proxy) from OWASP (OWASP, 2012). After the test period, the labelled attack data is exported from the IDS and the http request and response data is parsed from the binary network captures. From this parsed data the attack features are extracted and written to a data format called arff that is used in the weka workbench (Hall, Frank, Holmes, Pfahringer, Reutemann, & Witten, 2009) for training and testing a Bayesian classifier.

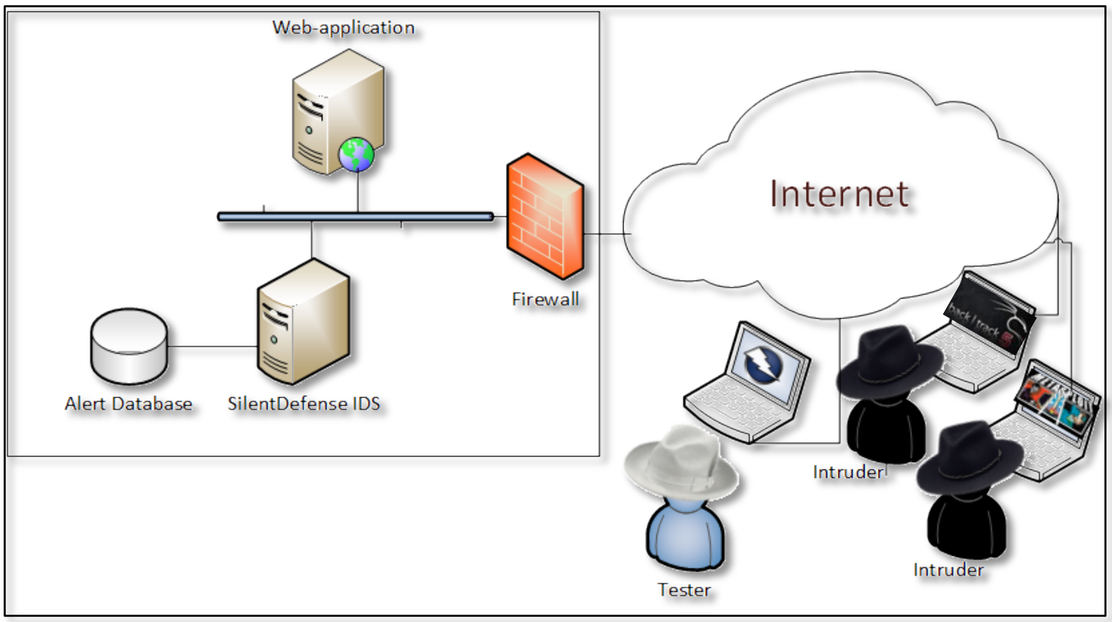


Figure. System Setup for the experiment with the SilentDefense IDS, the monitored Web application, 'white hat' testing, and real-world 'black hat' intrusions received from the Internet. Manually labelled alerts are stored in the Alert Database with the alert's event data.

6.2. Datasets

To be able to analyse the effectiveness of the selected features two datasets are collected. The datasets are obtained from SilentDefense IDS, monitoring a live webserver running the website wiki.seclab.nl. a good dataset is required. The goal was to obtain a dataset from running the SilentDefense IDS on an operational website. This selected webserver for the experiment was the www.fontys.nl website, the #837 website in the Netherlands according to "Alexa, the Web Information Company" (2012). However, the website was being migrated to a new backend system during the data collection phase of the experiment, and monitoring the website during this period unfortunately was not permitted. Therefore an alternative website was selected, wiki.seclab.nl, which was under my own administrative control, but had a very light load. The data set obtained during the test period eventually had about 200 alerts. This was too small for a valid experiment so the dataset had to be extended with realistic data. This was done with attack patterns from security incidents and test patterns. The test patterns were added to request-parameters, header and post parameters with OWASP's penetration testing tool ZAP (Zed Attack Proxy). Attack patterns were used from google's Fuzzdb project (Muntner, 2012), an attack and discovery patterns database for application fuzz testing. Also attack patterns from the ModSecurity SQL injection challenge ("SQL Injection Challenge: Lessons Learned", 2011) were used.

During the test period it also appeared that no Buffer Overflow alerts were received. Even with the help of the Metasploit framework and vulnerability scanners, only a couple of realistic events could be generated. Eventually the sample size for this class was too low to be able to include this class in the experiment. The Buffer Overflow attack class was therefore left out of the further experiment.

The resulting dataset had 1440 alerts, manually classified into attack classes. The number of alerts per attack class is listed in the table below.

Attack class	#alerts
XSS	344
SQLi	503
Path Traversal	466
Reconnaissance	57
Other	70
Total:	1440

Table. Number of events per attack class in dataset 1

The 'other' class contains the rest of the attacks, consisting of alerts on Remote File Inclusion (45), Password attacks (4), Command Injection (3), targeted probes for vulnerabilities or infections (6), and unknown attacks (12) where it could not be decided in which category the attack should be placed.

For further testing, a second dataset was collected from attacks during a later timeframe and tests with different attack patterns from OWASP's Xenotix XSS Exploit Framework (OWASP 2013), SQL Power Injector (Larouche, 2007) and Uniscan (Rocha, 2012).

Attack class	#alerts
XSS	1246
SQLi	181
Path Traversal	215
Reconnaissance	37
Other	12
Total:	1691

Table. Number of events per attack class in dataset 2

The dataset has a rather low percentage of real-world attacks, only 49 malicious attempts were collected during the second, shorter period of collecting attack samples for the second dataset. The samples collected where in the classes Reconnaissance and Other attacks only. All alerts in other attack classes were generated by test patterns from the mentioned test tools.

6.3. Feature Extraction

The collected datasets can be used for training a Bayesian network and for testing. From the collected data, the selected features must be extracted by parsing the HTTP messages and adding the attack labels that where manually added by an IDS operator. Parsing the HTTP messages starts with parsing the message for message-type, HTTP-headers, query-parameters and message content, according to the HTTP standard as defined in RFC2616 (Fielding, 1999). From these data fields the attack features can be extracted and written to a file in arff format. This file can then be used to analyse the features with a Bayesian classifier in the Weka data mining software (Hall, Frank, Holmes, Pfahringer, Reutemann, & Witten, 2009).

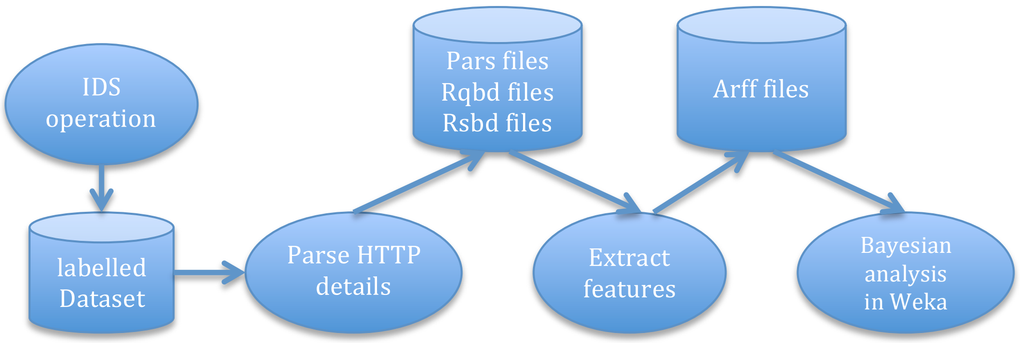


Figure. Collecting alerts and extracting features

6.3.1. Parsing http messages

Alerts are triggered by anomalies in HTTP requests. As defined in RFC2068 HTTP is a request/response protocol. To be able to parse all details correctly the protocol standard must be studied in detail. A client sends a request to the server in the form of a request method, URI, and protocol version, followed by a MIME-like message containing request modifiers, client information, and possible body content over a connection with a server. The response contains a status, meta-information like date/timestamps, and possibly the response body. Examples of an HTTP request and a response is given in the figures below.

```
GET http://www.seclab.nl/ HTTP/1.1
Host: www.seclab.nl
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.8; rv:16.0) Gecko/20100101 Firefox/16.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Proxy-Connection: keep-alive
```

Figure. Example HTTP-request

```
HTTP/1.1 200 OK
Date: Sat, 30 Mar 2013 08:12:25 GMT
Server: Apache
Last-Modified: Wed, 19 Sep 2007 19:23:18 GMT
ETag: "2916c9-136-f84f9580"
Accept-Ranges: bytes
Content-Length: 310
Content-Type: text/html; charset=UTF-8

<html>
<head>
<title>Fontys Security lab</title>
</head>

<body>
<h1>Security Lab</h1>
<h2>Fontys Hogeschool ICT</h2>
<p>Hier komt binnenkort de website van het Fontys security lab.</p>
<p>Bekijk voor meer informatie over onze opleidingen <a
href="http://www.fontys.nl/">www.fontys.nl</a>.</p>
</body>
</html>
```

Figure. Example HTTP-response

The HTTP-Request

An HTTP-request message from a client to a server consists of a request-line, header(s) and the request-body. In between these parts are a carriage return (CR) and a line feed (LF), hexadecimal ascii-values respectively *0d* and *0a*, here and in the RFC standard noted as *CRLF*.

> ***Request =*** ***request-line CRLF***
> ***headers CRLF***
> ***CRLF***
> ***body***

The request-line holds the request method to a certain resource, e.g.:

> **'GET http://wiki.seclab.nl/index.php/Hoofdpagina HTTP/1.1'**

so the format of the request-line is as follows:

> ***Request-Line = Method SP Request-URI SP HTTP-Version CRLF***

The HTTP-version string normally is 'HTTP/1.1' because that has been the version since 1999. The only valid alternative value is 'HTTP/1.0'.

In the 1.1 version, 8 standard request methods are available:
- GET
- POST
- HEAD
- OPTIONS
- PUT
- DELETE
- TRACE
- CONNECT

Use of other methods is possible, e.g. Webdav, an extension of HTTP, defines 7 extra methods, and RFC5789 defines the PATCH method as an alternative for the PUT method, to modify an existing HTTP resource.

The Request-URI, or Uniform Resource Identifier identifies the resource upon which to apply the request. The URI can be in one of four options:

> *Request-URI = "*" | absoluteURI | abs_path | authority*

The first and last options for the request-URI are special options:

- **'*'**
  means that the request does not apply to a particular resource, but to the server itself, e.g.: OPTIONS * HTTP/1.1
- **authority**
  is only used by the CONNECT method and is a hostname and portnumber, e.g.: CONNECT server.example.com:80 HTTP/1.1

The most common form is the abs_path form to identify a resource on a server. The abs_path then is the absolute path to the resource on the server. The network location of the server is then given in the Host header field. An example:

> *GET /pub/WWW/TheProject.html HTTP/1.1*
> *Host: www.w3.org*

The absoluteURI is required when the request is made to a proxy. The URI must then be a complete location of the resource, as in this example:

> *GET HTTP://www.w3.org/pub/WWW/TheProject.html HTTP/1.1*

A complete HTTP url, with host and possibly port-number is then in the following format:

> "HTTP:" "//" host [":" port] [abs_path ["?" query]]

If the port is left out, the default HTTP-port 80 is assumed.
The host-part in a URI is case-insensitive and characters can be put in a %HEX encoding. The following three URI's are therefore equivalent:

HTTP://abc.com:80/~smith/home.html
HTTP://ABC.com/%7Esmith/home.html
HTTP://ABC.com:/%7esmith/home.html

The query string that follows after the "?" contains data that is passed to web applications such as CGI programs, or PHP scripts, for example:

HTTP://wiki.seclab.nl/index.php?**12436**

With the main use of query strings, in HTML forms, the query string is composed of one or a series of field-value pairs, separated by "&", for example as follows:

HTTP://wiki.seclab.nl/index.php?**field1=value1&field2=v2&field3=v3**

Request headers

The request line in an http-request can be followed by one or more headers. Standard headers are defined for requests and responses, but non-standard headers are also allowed and commonly used, e.g. X-Requested-with and X-XSS-Protection. Non-standard headers were conventionally prefixed with "X-", but this was deprecated by rfc 6648, Saint-Andre (2012). Headers can also be defined in other standards. Well-known examples are the Set-Cookie and Cookie header fields, currently defined by the standard rfc 6265, Barth (2011)

Standard headers can be of different types. There are general header fields, request header fields and entity header fields.
General header fields only apply to the message sent, and are used in both request and response requests. Examples of general header fields are e.g. cache control, date and transfer-encoding.
Request header fields are only used in a request, and, as stated in the HTTP/1.1 standard rfc 2616 (Fielding, 1999), "allow the client to pass additional information about the request, and about the client itself, to the server" (p. 37). Examples of request headers are Host, User-Agent and Accept-Encoding.
Entity header fields contain information about the resource carried in the message, or in the body of the message. The body is referred to as *entity* in the standard. Examples of entity headers are Content-Encoding, Content-Type, Expires, and Last-Modified.

If a request or response has multiple headers, separation of headers is through standard ending with a carriage return (CR) and a line feed (LF). These are the hexadecimal ascii-values respectively *0d* and *0a*, or as escape sequences *\r* and *\n*.
The end of the header fields is indicated by an empty line, so only the *CRLF* line-ending. This is then possibly followed by a message-body.

A header field consists of a header name, followed by a colon (":") and the field value. Field names are case-insensitive and may be preceded by one or more white spaces or horizontal tabs. A single white space is preferred by the standard. Header fields can be extended over multiple lines.

Request body

The request body is referred to as entity, or entity-body. The body is a sequence of bytes, an octet string, with the data type and format specified in the header fields Content-Type and Content-Encoding. The length of the body can be specified in the Content-Length header field, but other options are possible as defined in paragraph 4.4 in the HTTP/1.1 standard rfc 2616 (p.32).

Typical contents of the request body is POST data, the data that is sent with a POST-request.

The HTTP-Response

A response message is the server's answer to an HTTP-request. A response contains a status-line, and possibly header(s) and a message-body. Characteristics of headers and message-body are the same as described with the request-characteristics, with the response body typically being the requested web page.

The status-line contains the HTTP-version, a Status-Code and a Reason-Phrase. The format is as follows:

Status-Line = HTTP-Version SP Status-Code SP Reason-Phrase CRLF

The status-code is a 3-digit result code with five possible values for the first digit:
1xx Informational, 2xx Success, 3xx Redirection, 4xx Client Error, 5xx Server Error. The HTTP standard contains recommended values, e.g. 200 OK. The Reason-Phrase is a textual explanation of the result code and is intended for human users.

Next to the use of general-headers and entity-headers in a HTTP-response, nine specific response headers are defined in the HTTP/1.1 standard, e.g. ETag, Retry-After, and Server.

Parsing alert's network event data

The event data from the alerts is in tcpdump libpcap format (Tcpdump, 2012). To get the HTTP message parts, a C-program was written, based on a libpcap demonstration sniffer program sniffex from tcpdump (Carstens, 2002). The source code of the HTTP parser is added to the report in Appendix A. The program reads all pcap data, determines if it is an HTTP message, and parses the request/response line, headers and the message body. This data is written to a file for further handling in the following self-defined pars-format:

alertid: SP <alertid>
type: SP Request|Response
Requestline | responseline: SP <data>
* parameter: SP <querystringparameters>
* (header: SP <data>)
nobody|(body: SP <body_filename>)
attackclass: SP <attackclassname>

It was chosen not to write the parsed data to a database but to text files, because it is only an intermediate format, input for further analysis. Text files can easily be transferred to and read on different types of computer systems and by programming languages for import and further analysis without having to install or configure any tooling. Request and Response message-body data, which can be binairy, is written to respectively .rqbd and .rsbd files.

6.3.2. Feature extraction process

In the feature extraction process, first the standard HTTP characteristics are extracted. These standard characteristics are the used HTTP request method, HTTP version, request URL length, invalid request ending, returned HTTP response status code. Also an incorrect request ending, where no CRLF is found at the end of a request is registered as a feature for malformed requests uncompliant to the HTTP standard. The steps in the feature extraction process and the outcomes are also displayed in the figure below.

The further process of feature extraction is done per parameter (query parameters, headers, POST data parameters). Steps taken are to decode data first, counting all encoding features. After decoding, the data is cleaned from possible extra obfuscations like tabs and newlines. For most of the remaining feature extraction the cleaned parameter data is used. From this cleaned data all other features, such as the maximum parameter length, number of Path Traversals, etcetera, are extracted.

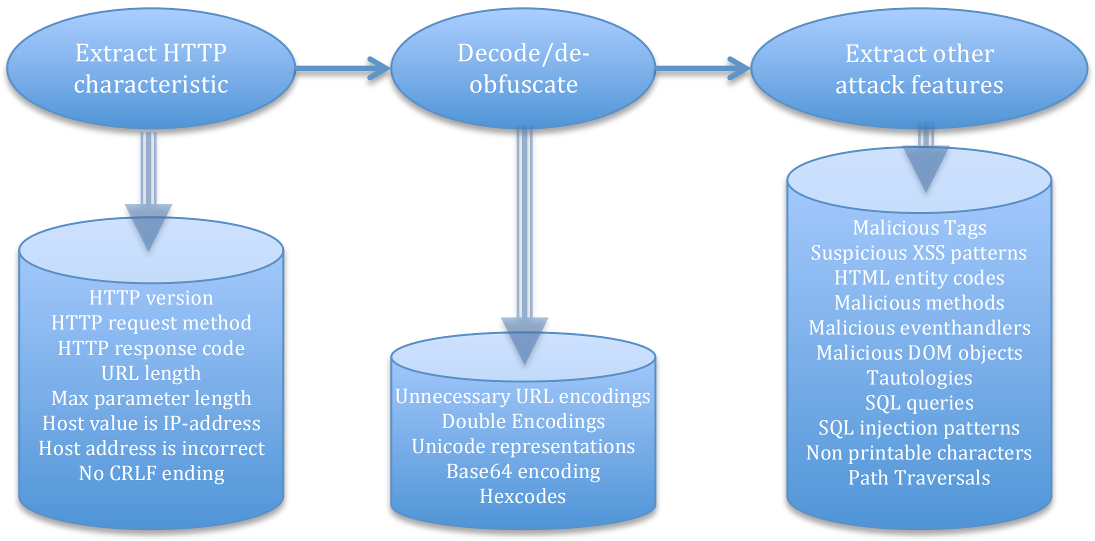


Figure. Process of feature extraction from dataset

6.3.3. Bayesian analysis

The extracted features are written in a text file according to the Attribute Relation File Format ARFF (University of Waikato, 2002). An ARFF file is an ASCII text file that describes a list of instances sharing a set of attributes. The relation and attributes are described in the header, followed by the data. Part of the ARFF file for dataset 1 is displayed in the figure below.

```
% Title: attack features dataset1
%
% Creator: Casper Schellekens
% Creation Date: Fri October 25, 2013 13:04:34
%
@RELATION all_attack_classes

@ATTRIBUTE urllength numeric
@ATTRIBUTE urlencodingsextra numeric
@ATTRIBUTE doubleencodings numeric
@ATTRIBUTE unicode numeric
@ATTRIBUTE htmlentitycodes numeric
@ATTRIBUTE base64encodings numeric
@ATTRIBUTE hexcodes numeric
@ATTRIBUTE maxparamlength numeric
@ATTRIBUTE maltags numeric
@ATTRIBUTE susppatterns numeric
@ATTRIBUTE malmethods numeric
@ATTRIBUTE maleventhandlers numeric
@ATTRIBUTE malobjects numeric
@ATTRIBUTE httpversion {1.0, 1.1, other}
@ATTRIBUTE statuscode numeric
@ATTRIBUTE requestmethod {GET, POST, OPTIONS, HEAD, PUT, DELETE, TRACE, CONNECT, PATCH,
WebDAV, Invalid}
@ATTRIBUTE tautologies numeric
@ATTRIBUTE sqlqueries numeric
@ATTRIBUTE sqlipatterns numeric
@ATTRIBUTE nonprintable numeric
@ATTRIBUTE pathtraversals numeric
@ATTRIBUTE iphost {true, false}
@ATTRIBUTE wronghost {true, false}
@ATTRIBUTE nocrlfend {true, false}
@ATTRIBUTE attackclass {xss, sqli, pathtraversal, scan, other}
```

```
@DATA
86,4,0,0,0,0,0,93,0,0,0,0,1,1.1,200,POST,0,0,0,0,0,false,false,false,sqli
87,121,11,0,0,0,0,72,22,4,0,0,70,1.1,200,POST,0,0,0,0,0,false,false,false,xss
74,1,6,0,0,0,0,93,0,0,0,0,0,1.1,200,GET,0,0,0,0,70,false,false,false,pathtraversal
224,0,0,0,52,0,0,184,2,2,0,0,0,1.1,200,GET,0,0,0,0,0,false,false,false,xss
90,4,0,0,0,0,0,93,0,0,0,0,1,1.1,200,POST,3,0,5,0,0,false,false,false,sqli
```

Figure. ARFF data file for dataset 1 (only five instances out of 1440 displayed)

Weka is a collection of machine learning methods for data mining tasks (Hall, Frank, Holmes, Pfahringer, Reutemann, & Witten, I., 2009). The Weka workbench also supports Bayes Network learning using various search algorithms and estimators (Bouckaert, 2004). The search algorithm, to find probabilistic dependency relations, is not very relevant because a naïve Bayes network is used, where all nodes have one and the same parent, the node for the attack class.

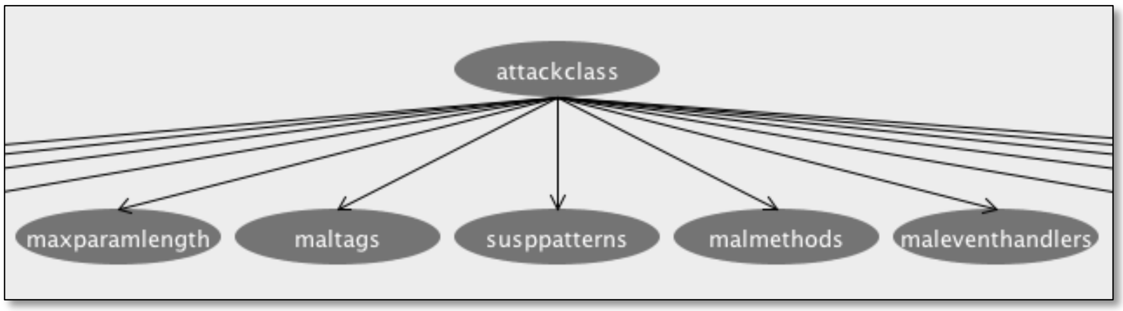


Figure. Naïve Bayes Network structure where all nodes are related to one and the same parent, the attack class in this case.

For building the probability tables for the nodes in the Naïve Bayesian Network, the SimpleEstimator is available. This estimator uses an initial count which eventually is set to 0.01 for the best results in the experiment. The initial count prevents having zero probabilities which would wipe out all information from other feature values.

7. Results & Discussion

7.1. DATASET 1

With the help of the Weka workbench a Naïve Bayesian network is built. With the tooling it is possible to go through learning mode to set the probability distribution tables for the nodes in the Bayesian network, and test with cross-validation on the obtained dataset of 1440 alerts. Building a Bayesian network with all features gives an overall weighted TP rate of 96.94%, and a weighted average FP rate per attack class of 0.9%. In the table below the results per attack class are also listed.

Attack class	TP rate	FP rate
XSS	95.3%	0.4%
SQL injection	99.4%	1.1%
Path Traversal	99.6%	0.3%
Reconnaissance	86%	1%
Other	78.6%	0.9%
weighted average	*96.94%*	*0.9%*

Table. TP/FP rates for attack classes

It shows that the well-defined attack classes can be recognized very well, with XSS a bit harder to detect, but XSS still has a TP rate of 95.3%. The somewhat less defined class of reconnaissance scores 86%. The remaining or 'other' attacks are undefined and are therefore less recognizable with a TP rate of 78.6%. FP rates per class vary from 0.3% to 1.1%. Note that the total FP rate in this experiment is: 100% - TP rate= 3.06%, because there is no larger class of non-malicious requests as included in FP-rate calculations in intrusion detection systems.

Before going into more detail on the results and the Bayesian network, the contribution of the separate features is investigated in the next paragraph.

7.2. Feature selection

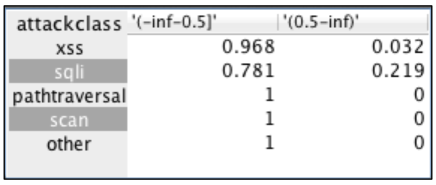
Not all features will contribute evenly to the achieved results. Features can give zero gain to the results or features could even influence the results in a negative way. For the achieved results we can show the contribution to results, also called the gain to the results. We do so by removing an attribute and calculate the differences in result without that attribute. The gain to the TP rate is listed below for all attributes. The FP rate was influenced with a maximum of 0.1%.

Attribute	TP rate without the attribute	Gain to TP rate
Max parameter length	94.58%	2.09%
Path Traversals	96.11%	0.83%
Tautologies	96.11%	0.83%
Malicious tags	96.39%	0.55%
Suspicious patterns	96.39%	0.55%
SQL queries	96.53%	0.41%
Extra URL encodings	96.60%	0.34%
Malicious methods	96.67%	0.27%
Malicious event handlers	96.74%	0.20%
URL length	96.81%	0.13%
Unicode	96.88%	0.06%
Hexcodes	96.88%	0.06%
Status code response	96.88%	0.06%
HTTP version	96.88%	0.06%
Double encodings	96.94%	0%
SQL injection patterns	96.94%	0%
Html entity codes	96.94%	0%

HTTP request method	96.94%	0%
Wrong host	96.94%	0%
Base64 Encodings	96.94%	0%
Non-printable characters	96.94%	0%
IP address host value	96.94%	0%
No CRLF ending	96.94%	0%
Malicious objects	97.01%	-0.07%

Table. Attribute gain of all attributes separately removed from the network

Some results are unexpected. For example, the contribution of the feature 'SQL injection patterns' is zero. This does not necessarily mean that this feature is not discriminating for the class of SQL Injection, but that the feature is covered by other features already. That the feature is discriminating for the class of SQL injection can be seen in the probability distribution table for this feature.



attackclass	'(-inf-0.5]'	'(0.5-inf)'
xss	0.968	0.032
sqli	0.781	0.219
pathtraversal	1	0
scan	1	0
other	1	0

Table. Probability distribution table for the feature 'SQL injection patterns'

Some attributes do not contribute at all to the results. The attributes with no positive contribution can be removed from the model, resulting in 14 features that give the following, slightly changed results:

Attack class	TP rate	FP rate
XSS	95.1%	0.3%
SQL injection	99.6%	1.4%
Path Traversal	99.8%	0.5%
Reconnaissance	87.7%	1.2%
Other	75.7%	0.5%
Overall	*96.9%*	*0.8%*

Table. TP/FP rates for attack classes

7.3. Bayesian network

The resulting Bayesian network classifier is displayed below. Each node contributes to the Attack class, independently of other nodes.

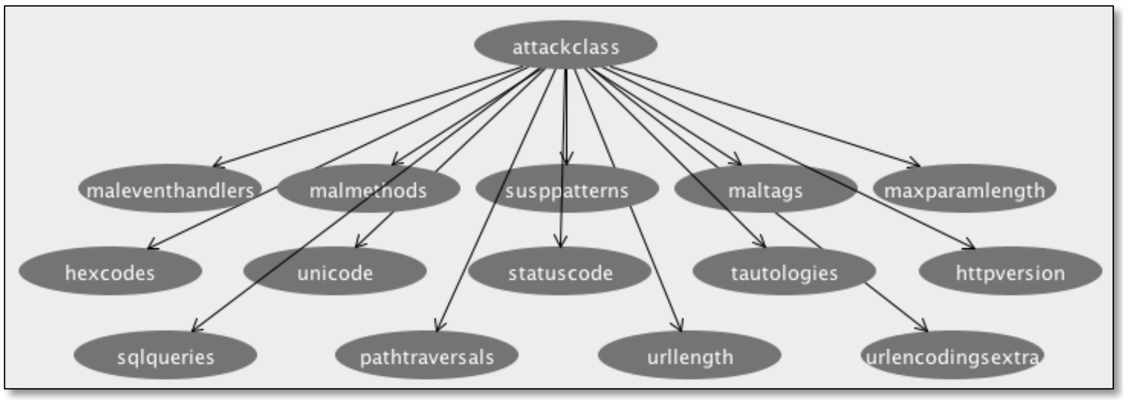
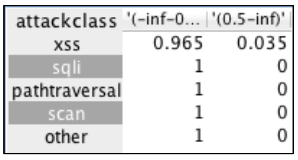


Figure. Resulting Bayesian Network

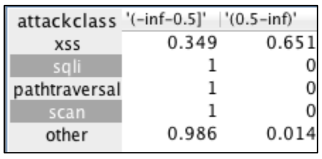
Each node has a probability distribution table with the learned probabilities. Below the probability distribution tables for the nodes in the Bayesian network are given.

Malicious Event handlers:



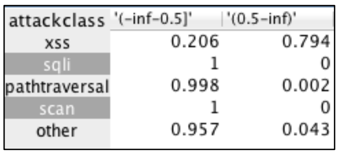
attackclass	'(-inf-0...	'(0.5-inf)'
xss	0.965	0.035
sqli	1	0
pathtraversal	1	0
scan	1	0
other	1	0

Malicious Javascript methods:



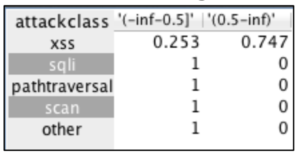
attackclass	'(-inf-0.5]'	'(0.5-inf)'
xss	0.349	0.651
sqli	1	0
pathtraversal	1	0
scan	1	0
other	0.986	0.014

Suspicious XSS patterns:



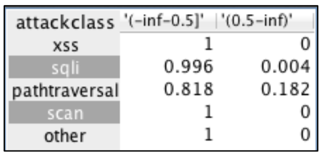
attackclass	'(-inf-0.5]'	'(0.5-inf)'
xss	0.206	0.794
sqli	1	0
pathtraversal	0.998	0.002
scan	1	0
other	0.957	0.043

Malicious html tags:



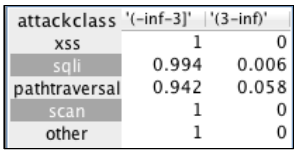
attackclass	'(-inf-0.5]'	'(0.5-inf)'
xss	0.253	0.747
sqli	1	0
pathtraversal	1	0
scan	1	0
other	1	0

Unicode:



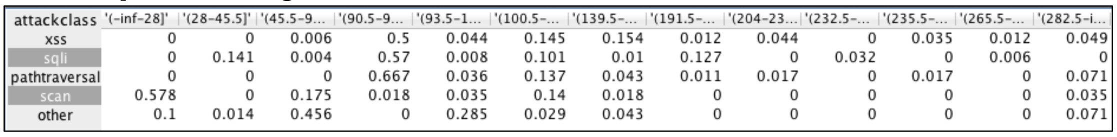
attackclass	'(-inf-0.5]'	'(0.5-inf)'
xss	1	0
sqli	0.996	0.004
pathtraversal	0.818	0.182
scan	1	0
other	1	0

Hexcodes:



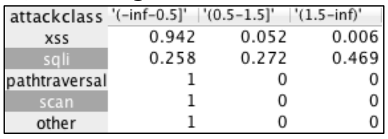
attackclass	'(-inf-3]'	'(3-inf)'
xss	1	0
sqli	0.994	0.006
pathtraversal	0.942	0.058
scan	1	0
other	1	0

Maximum parameter length:



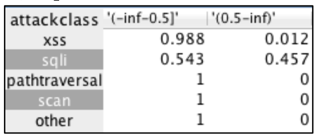
attackclass	'(-inf-28]'	'(28-45.5]'	'(45.5-9...	'(90.5-9...	'(93.5-1...	'(100.5-...	'(139.5-...	'(191.5-...	'(204-23...	'(232.5-...	'(235.5-...	'(265.5-...	'(282.5-i...
xss	0	0	0.006	0.5	0.044	0.145	0.154	0.012	0.044	0	0.035	0.012	0.049
sqli	0	0.141	0.004	0.57	0.008	0.101	0.01	0.127	0	0.032	0	0.006	0
pathtraversal	0	0	0	0.667	0.036	0.137	0.043	0.011	0.017	0	0.017	0	0.071
scan	0.578	0	0.175	0.018	0.035	0.14	0.018	0	0	0	0	0	0.035
other	0.1	0.014	0.456	0	0.285	0.029	0.043	0	0	0	0	0	0.071

SQL tautologies:



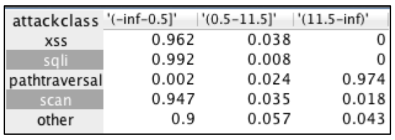
attackclass	'(-inf-0.5]'	'(0.5-1.5]'	'(1.5-inf)'
xss	0.942	0.052	0.006
sqli	0.258	0.272	0.469
pathtraversal	1	0	0
scan	1	0	0
other	1	0	0

SQL queries:



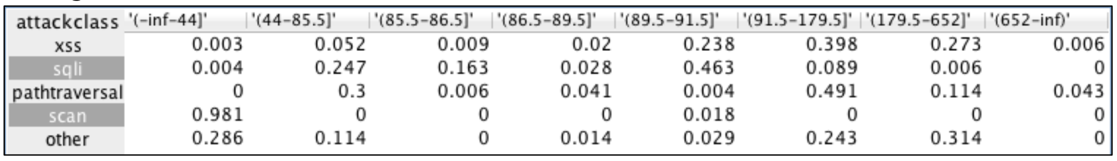
attackclass	'(-inf-0.5]'	'(0.5-inf)'
xss	0.988	0.012
sqli	0.543	0.457
pathtraversal	1	0
scan	1	0
other	1	0

Path Traversals:



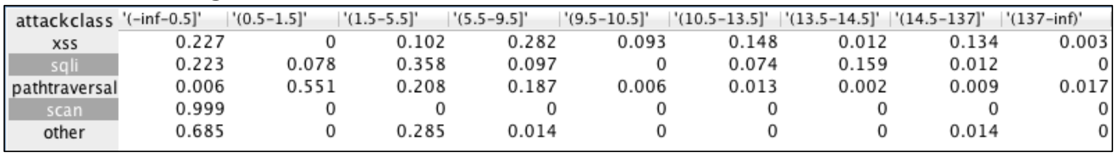
attackclass	'(-inf-0.5]'	'(0.5-11.5]'	'(11.5-inf)'
xss	0.962	0.038	0
sqli	0.992	0.008	0
pathtraversal	0.002	0.024	0.974
scan	0.947	0.035	0.018
other	0.9	0.057	0.043

Urllength:



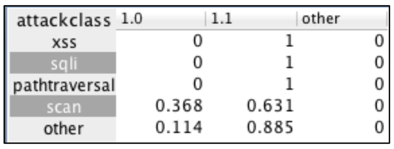
attackclass	'(-inf-44]'	'(44-85.5]'	'(85.5-86.5]'	'(86.5-89.5]'	'(89.5-91.5]'	'(91.5-179.5]'	'(179.5-652]'	'(652-inf)'
xss	0.003	0.052	0.009	0.02	0.238	0.398	0.273	0.006
sqli	0.004	0.247	0.163	0.028	0.463	0.089	0.006	0
pathtraversal	0	0.3	0.006	0.041	0.004	0.491	0.114	0.043
scan	0.981	0	0	0	0.018	0	0	0
other	0.286	0.114	0	0.014	0.029	0.243	0.314	0

extra URL encodings



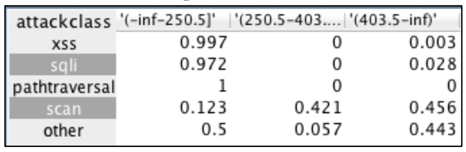
attackclass	'(-inf-0.5]'	'(0.5-1.5]'	'(1.5-5.5]'	'(5.5-9.5]'	'(9.5-10.5]'	'(10.5-13.5]'	'(13.5-14.5]'	'(14.5-137]'	'(137-inf)'
xss	0.227	0	0.102	0.282	0.093	0.148	0.012	0.134	0.003
sqli	0.223	0.078	0.358	0.097	0	0.074	0.159	0.012	0
pathtraversal	0.006	0.551	0.208	0.187	0.006	0.013	0.002	0.009	0.017
scan	0.999	0	0	0	0	0	0	0	0
other	0.685	0	0.285	0.014	0	0	0	0.014	0

HTTP version:



attackclass	1.0	1.1	other
xss	0	1	0
sqli	0	1	0
pathtraversal	0	1	0
scan	0.368	0.631	0
other	0.114	0.885	0

HTTP status response code:



attackclass	'(-inf-250.5]'	'(250.5-403....	'(403.5-inf)'
xss	0.997	0	0.003
sqli	0.972	0	0.028
pathtraversal	1	0	0
scan	0.123	0.421	0.456
other	0.5	0.057	0.443

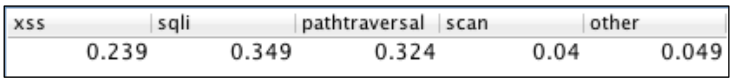
Figure. Probability Distribution Tables for Bayesian network

7.4. Fine-tuning the network

The dataset is made up of real-world attacks and test vectors to increase the dataset. The percentage of test vectors is large, over 85% of the dataset used for learning the Bayesian network. Because of this the distribution of the different attack classes is not realistic, because separate test-vector sets for the different attack classes where obtained regardless of real-world attack rates. Also the distribution of feature values might have some unnatural distributions because of the test vectors used.

7.4.1. Attack class distribution

In the experimental set-up used, the monitored web application was one with a light load. The real world attacks were used as test data but most contents of the first dataset came from test vectors. Therefore, the distribution of the different attack frequencies in the datasets is not very realistic. But this rate does influence the outcome of the probability calculations in the Bayesian network for new events. The distribution of dataset 1 is displayed in the table below.



xss	sqli	pathtraversal	scan	other
0.239	0.349	0.324	0.04	0.049

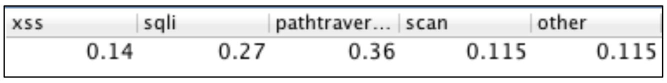
Figure. Attack class distribution

To get a more realistic distribution, a recent overview of web application attacks is needed. This overview is found in Imperva's latest Web Application Attack report (Imperva, 2013). This annual report is based on monitoring and analysing internet traffic to 70 web applications during a period of six months. The results for the category of web applications other then retail applications, which the web application used in the experiment can be classified in, are displayed below. The categories used do resemble the attack classes defined for the experiment, but no reconnaissance attack class is used.

Attack class	**Attack rate**
XSS	14%
SQL injection	27%
Path Traversal	36%
Combined remaining results	23%

Table. Imperva's attack type ratio's over first half of 2013

Without any further information on Reconnaissance and other type of attacks a safe 50/50 split is used for the remaining attacks, based on the experienced real world attacks in the experiment. This leads to the following attack class distribution that can be used as a more realistic distribution in the Bayesian network.

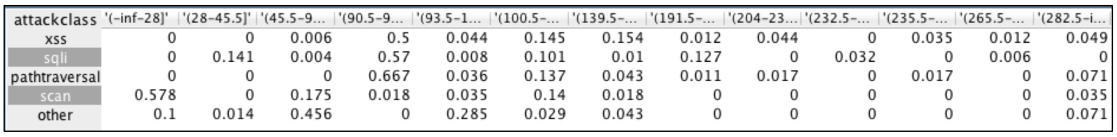


xss	sqli	pathtraver...	scan	other
0.14	0.27	0.36	0.115	0.115

Figure. Adapted attack class distribution

7.4.2. Feature value distribution and overfitting

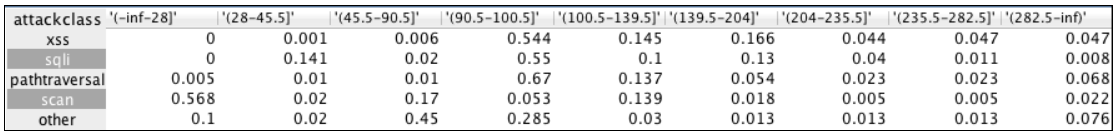
Even though the dataset consists of over 1400 alerts, frequencies of parameter value ranges can have an unnatural distribution because of the large part of test-vectors used. Especially when a learned probability of a value range is almost zero while not realistically explainable, the probability table should be adjusted manually to prevent new alerts (outside the learning dataset) from being misclassified. An example can be found in the network node for maximum parameter length. With 13 separate ranges of lengths the training algorithm has come up with a very precise probability distribution that (over)fits the training data very well, but is not necessarily very fit for the real world.



attackclass	'(-inf-28]'	'(28-45.5]'	'(45.5-9...	'(90.5-9...	'(93.5-1...	'(100.5-...	'(139.5-...	'(191.5-...	'(204-23...	'(232.5-...	'(235.5-...	'(265.5-...	'(282.5-i...
xss	0	0	0.006	0.5	0.044	0.145	0.154	0.012	0.044	0	0.035	0.012	0.049
sqli	0	0.141	0.004	0.57	0.008	0.101	0.01	0.127	0	0.032	0	0.006	0
pathtraversal	0	0	0	0.667	0.036	0.137	0.043	0.011	0.017	0	0.017	0	0.071
scan	0.578	0	0.175	0.018	0.035	0.14	0.018	0	0	0	0	0	0.035
other	0.1	0.014	0.456	0	0.285	0.029	0.043	0	0	0	0	0	0.071

Table. Probability Distribution Table for maximum parameter length

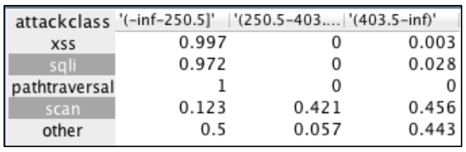
Apparently, in the dataset used for learning, almost no reconnaissance attack or scan has a maximum parameter length from 28 to 46. The probability displayed is 0, but in fact is 0.000175. A low probability for certain values of course is possible, but in this case plenty of alerts have a maximum parameter length just below or above that value. It does not seem very realistic that that would be a real-world distribution for the maximum parameter length. If, with this Bayesian network model, an alert would occur with a parameter length within that range, it would probably not be classified as a scan. A solution could be to raise the overall initial count that is used to prevent zero values in the probability tables. However, this would decrease the overall performance of the network. The solution chosen is to merge columns with a small range, because that is a typical example of overfitting unless a very good reason for that can be found. If the merged columns still show an unnatural distribution that cannot be explained from the real world those probability values will be distributed more fluently. The result for the probability distribution table is displayed below.



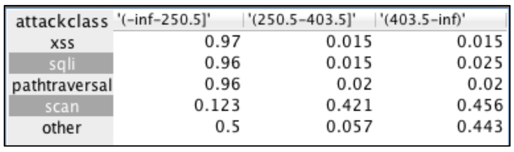
attackclass	'(-inf-28]'	'(28-45.5]'	'(45.5-90.5]'	'(90.5-100.5]'	'(100.5-139.5]'	'(139.5-204]'	'(204-235.5]'	'(235.5-282.5]'	'(282.5-inf)'
xss	0	0.001	0.006	0.544	0.145	0.166	0.044	0.047	0.047
sqli	0	0.141	0.02	0.55	0.1	0.13	0.04	0.011	0.008
pathtraversal	0.005	0.01	0.01	0.67	0.137	0.054	0.023	0.023	0.068
scan	0.568	0.02	0.17	0.053	0.139	0.018	0.005	0.005	0.022
other	0.1	0.02	0.45	0.285	0.03	0.013	0.013	0.013	0.076

Table. Adapted Probability Distribution Table for maximum parameter length

Adapting the table will lower the performance for the training data, but the aim is to prevent unreasonable misclassifications in new events, so in the operation of the Bayesian classifier.
For similar reasons the PDT is adapted for HTTP status response code. Here, return codes above 200 were not present for Path Traversal, which is not realistic. A Path Traversal can also result in for example a 403 Forbidden, or 404 Not Found.



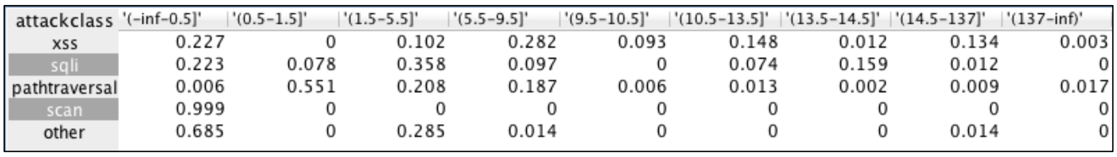
attackclass	'(-inf-250.5]'	'(250.5-403....	'(403.5-inf)'
xss	0.997	0	0.003
sqli	0.972	0	0.028
pathtraversal	1	0	0
scan	0.123	0.421	0.456
other	0.5	0.057	0.443



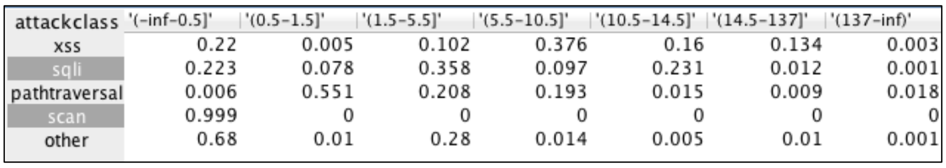
attackclass	'(-inf-250.5]'	'(250.5-403.5]'	'(403.5-inf)'
xss	0.97	0.015	0.015
sqli	0.96	0.015	0.025
pathtraversal	0.96	0.02	0.02
scan	0.123	0.421	0.456
other	0.5	0.057	0.443

Table. Original vs. Adapted PDT for HTTP response code

The PDT's for the total URL length and extra URL encodings are similar to the parameter length PDT. The PDT's are rather detailed, and have probabilities of almost zero for values that could very well happen in the real world. Again to prevent overfitting on the learning data, columns are merged and probability values are distributed more fluently where needed. In the original table for extra URL encodings for example, a separate column exists for exactly 10 encodings. This is a clear sign of overfitting and the column is merged with the previous range. The next columns are also combined and some zero approximations are replaced by slightly higher and more realistic values. The same is done for the PDT of URL length.

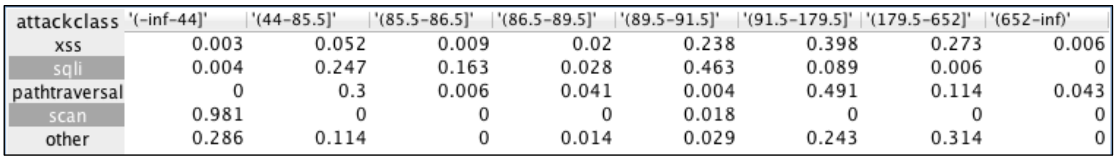


attackclass	'(-inf-0.5]'	'(0.5-1.5]'	'(1.5-5.5]'	'(5.5-9.5]'	'(9.5-10.5]'	'(10.5-13.5]'	'(13.5-14.5]'	'(14.5-137]'	'(137-inf)'
xss	0.227	0	0.102	0.282	0.093	0.148	0.012	0.134	0.003
sqli	0.223	0.078	0.358	0.097	0	0.074	0.159	0.012	0
pathtraversal	0.006	0.551	0.208	0.187	0.006	0.013	0.002	0.009	0.017
scan	0.999	0	0	0	0	0	0	0	0
other	0.685	0	0.285	0.014	0	0	0	0.014	0

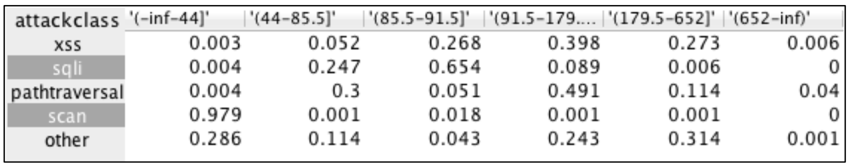


attackclass	'(-inf-0.5]'	'(0.5-1.5]'	'(1.5-5.5]'	'(5.5-10.5]'	'(10.5-14.5]'	'(14.5-137]'	'(137-inf)'
xss	0.22	0.005	0.102	0.376	0.16	0.134	0.003
sqli	0.223	0.078	0.358	0.097	0.231	0.012	0.001
pathtraversal	0.006	0.551	0.208	0.193	0.015	0.009	0.018
scan	0.999	0	0	0	0	0	0
other	0.68	0.01	0.28	0.014	0.005	0.01	0.001

Table. Original vs. Adapted PDT for Extra URL encodings



attackclass	'(-inf-44]'	'(44-85.5]'	'(85.5-86.5]'	'(86.5-89.5]'	'(89.5-91.5]'	'(91.5-179.5]'	'(179.5-652]'	'(652-inf)'
xss	0.003	0.052	0.009	0.02	0.238	0.398	0.273	0.006
sqli	0.004	0.247	0.163	0.028	0.463	0.089	0.006	0
pathtraversal	0	0.3	0.006	0.041	0.004	0.491	0.114	0.043
scan	0.981	0	0	0	0.018	0	0	0
other	0.286	0.114	0	0.014	0.029	0.243	0.314	0



attackclass	'(-inf-44]'	'(44-85.5]'	'(85.5-91.5]'	'(91.5-179....	'(179.5-652]'	'(652-inf)'
xss	0.003	0.052	0.268	0.398	0.273	0.006
sqli	0.004	0.247	0.654	0.089	0.006	0
pathtraversal	0.004	0.3	0.051	0.491	0.114	0.04
scan	0.979	0.001	0.018	0.001	0.001	0
other	0.286	0.114	0.043	0.243	0.314	0.001

Table. Original vs. Adapted PDT for URL length

7.4.3. Testing with the adapted probability distributions

To find out the consequences of the adaptions, the data set has to be tested with the adapted attack class distribution and adapted probability distributions. In the weka workbench it is possible to see and adapt the Bayesian network with the Bayes net editor. However, the changed model cannot be reused for testing directly in the weka GUI. Therefore a self-written implementation of the Bayesian classifier is coded that reads the exported model from weka and calculates predictions and overall performance metrics from a test set. If we use the first dataset and run it as test data with the adapted model, the following results are obtained. It shows that results have not changed a lot from the adaptions made. XSS, SQL injection and Path Traversal classes performed a little worse, but Reconnaissance and other attacks were predicted even better with the adapted model.

Attack class	**TP rate**	**FP rate**
XSS	94.4%	0.1%
SQL injection	99.0%	0.7%
Path Traversal	99.7%	0.3%
Reconnaissance	92.9%	0.9%
Other	78.5%	0.9%
Overall	*96.9%*	*0.5%*

Table. TP/FP rates with adapted probability distributions

7.5. Introducing a threshold

A way of predicting whether minor fluctuations in the learning data are exaggerated too much in an overfitted model, might be to test whether the predictions of the model are very certain or if there are a lot of close calls, improving the performance on a lucky basis. If these 'lucky shots' are taken out of the TP result rate, the rate will drop and this could be a better estimate of the predictive quality of the network in practice. Also, and more important for an implementation of the Bayesian classification in practice, if such a threshold margin is applied in practice, the number of incorrectly classified alerts will be lower. If the calculated probability for the predicted class is below a pre-defined threshold, the output to the user can be presented with a warning that the correctness of the outcome is uncertain. Assuming that the confidence of the model in an incorrectly classified alert will often be lower than for correctly classified alerts, introducing this threshold could improve on the misclassifications rate a lot. Below, the number of correct predictions and the misclassification rate is listed for different estimation probability- or "lucky guess"-thresholds.

Required Estimation Certainty	**#events below certainty level. Displayed is: #events (correct/incorrect)**	**Resulting TP rate (Correct estimates)**	**Total FP rate**	**Average FP rate per class**
>95%	116 (85/31)	91.0%	0.9%	0.1%
>90%	92 (65/27)	92.4%	1.2%	0.2%
>80%	53 (28/25)	95.0%	1.3%	0.2%

Table. TP/FP rates with "lucky guess"-threshold

This means, that if we would use a model where not only the attack class is chosen with the highest probability, but we would not accept guesses below an estimation certainty of 90%, then the TP rate would become 92.4% with our dataset instead of our result of 96.9%.
Also, with this threshold, the total FP or misclassification rate would go down from 3.1% to 1.2%, which is a welcome improvement. The exact number of misclassifications per class is shown below. For example, no XSS estimates are incorrect and only 5 SQL injection estimates are incorrect.

Estimate: **Class:**	**XSS**	**SQLi**	**PT**	**Recon.**	**Other**	**TP-rate**	**FP-rate**
XSS	322	5	2	0	0	0.978	0.000
SQL injection	0	462	0	0	0	1.000	0.003
Path Traversal	0	0	462	0	0	1.000	0.001
Reconnaissance	0	0	0	43	0	1.000	0.006
Other	0	0	0	10	42	0.807	0.000

Table. (Mis)classifications per attack class with the use of a threshold of 90%

In total only 17 out of the 1440 alerts will be misclassified, but for 92 alerts it is concluded that classification is uncertain. Of these uncertain guesses, 65 are correct and 27 are incorrect. So, introducing a threshold would improve on the misclassification rate considerably, although the TP rate would be lower as well.

7.6. DATASET 2

Another way to test the classifier completely independent of the training data is by using a new, independend data set. Even though cross-validation is used in the training set, all validation and testing is done within the same larger data set. Introducing new samples, and testing these with the earlier established classifier will be a good test for independed performance of the model and possible overfitting.

7.6.1. Analysis

The second dataset is tested with the Bayesian model with adapted propability distributions and no threshold. The overall TP-rate of 94.2% is not as good as with the first dataset, but still ok. However, the performance of the different classes is very diverse as can be seen in the table below. Especially the TP-rate for Path Traversal attacks is unexpectedly low.

Attack class	TP rate	FP rate
XSS	99.1%	0.1%
SQL injection	94.4%	0.7%
Path Traversal	72.0%	0.0%
Reconnaissance	59.4%	0.3%
Other	75.0%	4.6%
Weighted avg.	*94.2%*	*0.8%*

Table. TP/FP rates for second dataset

XSS performs very well with a 99.1% TP-rate and only 0.1% FP-rate, which means that only 2 alerts where misclassified as XSS. The performance of XSS is much higher than with the first dataset (94.4%) that was also used for learning. SQL injection has opposite performance results when comparing dataset 1 and dataset 2 results. With dataset 2 a TP-rate of 94.4% is achieved which was 99% with dataset 1. The attack class of Path Traversal attempts has a low score with a TP-rate of only 72%, and the classes with a less clearer definition, Reconnaissance and Other, only score a TP-rate of respectively 59.5% and 75%. To be able to analyse this in more detail the classifications are displayed below.

Estimate: **Class:**	**XSS**	**SQLi**	**PT**	**Recon.**	**Other**	**TP-rate**	**FP-rate**
XSS	**1235**	8	0	0	3	0.991	0.001
SQL injection	0	**171**	1	3	6	0.944	0.007
Path Traversal	0	0	**155**	3	57	0.720	0.000
Reconnaissance	1	2	0	**22**	12	0.594	0.003
Other	1	2	0	0	**9**	0.750	0.046

Table. Detailed alert classifications of dataset 2

In each class the misclassifications can be analysed to be able to conclude why these alerts are misclassified.

XSS misclassifications

Of the 1246 alerts that were labelled as XSS, 11 were misclassified. 8 were misclassified as SQL injection, and 3 as Other. Most SQL injection classifications, 7 out of 8, come from a consecutive series of test patterns where each alert was based on multiple HTTP GET requests were XSS attacks were hidden in URI request parameters or in header values. An example request:

```
GET /index.php/Speciaal:Zoeken?search=pindakaas&
        go=eval('%5C141%5C154%5C145%5C162%5C164%5C50%5C61%5C51') HTTP/1.1
```

During the process of extracting features the relevant request parameter is decoded in two steps. First the URL encoded characters are decoded, then the resulting hex encoded characters are decoded:

```
eval('\x61\x6c\x65\x72\x74(1)')
eval('alert(1)')
```

What is noted is that this attempt has the typical javascript alert() method, often used in search for XSS vulnerabilities. However, it misses any HTML tags such as <script> tags to make this attack potentially successful. With no feature score on malicious tags the calculated probability that this is a XSS attack drops, leading to a misclassification. The misclassifications of Other have the same problem, but calculation came out on Other because of other feature values.
With a threshold, 7 out of the 11 misclassifications would be labelled as uncertain.

SQL injection misclassifications

Of the 181 alerts that were labelled as SQL injection, 10 were misclassified. 6 were misclassified as Other, 3 as Reconnaissance, and 1 as Path Traversal. Examples of misclassified alerts:

```
GET /index.php/Speciaal:Zoeken?
  search=%26lt%3B%26gt%3B%26quot%3B%27%25%3B%29%28%26amp%3B%2B&go=OK HTTP/1.1
GET /index.php/Speciaal:Zoeken?search=*%7C&go=OK HTTP/1.1
GET /index.php?feed=rss&title=Speciaal:RecenteWijzigingen+AND+1=2 HTTP/1.1
```

The first two payloads are decoded as <>"'%;)(&+ and *| which are not your typical SQL injection attempts, but an injection of special characters, probably to see if this has any effect such as an error returned by the handling of the resulting SQL query. However, these inputs do not give any score in the feature extraction process, leading to the misclassifications.
The 'AND 1=2' value cannot result in a SQL tautology because it logically evaluates to *False*, also leading to a misclassification.
With a threshold, 6 out of the 10 misclassifications would be labelled as uncertain.

Path Traversal misclassifications

Of the 215 alerts that were labelled as Path Traversal, 60 were misclassified, which is a lot, 28% to be more exact. 57 were misclassified as Other, and 3 as Reconnaissance. To be able to understand the reason for misclassification the Bayesian probability calculation is studied for a correctly classified instance and a misclassified instance.
The request for the correctly classified instance is:

```
GET /index.php?title=../../../../../../../../../../etc/passwd%00&feed=rss HTTP/1.1
```

The request for the misclassified instance is:

```
GET /index.php?title=Speciaal:RecenteWijzigingen&
    feed=//..%255c..%255c..%255c..%255c..%255c..%255c..%255c..%255..%255..%255cetc/passwd HTTP/1.1
```

Which is decoded in the feature extraction process to:

```
feed=//..\..\..\..\..\..\..%5..%5..\etc/passwd
```

The misclassified instance is clearly a Path Traversal attempt, so it is strange that it is misclassified as another attack. In the table below the Bayesian calculations are displayed based on the extracted feature values with probabilities used for the event's values and intermediate results in the calculations.
The misclassification estimate is in **red**, the non-determining feature values are greyed-out, determining features are marked in *yellow*, and the typical Path Traversal score is marked in green.

Classification	**Correctly Classified as Path Traversal**		**Misclassified as Other**	
Feature values	69,0,0,0,106,0,0,0,0,1.1,200,0,0,109		130,0,0,0,106,0,0,0,0,1.1,500,0,0,134	
Probability estimates per class	xss sqli pt scan other 0.000 0.000 0.926 0.000 0.072		xss sqli pt scan other 0.000 0.000 0.322 0.006 **0.671**	
	PT calculation	Other calculation	PT calculation	Other calculation
Urllength	0.3	0.114	0.491	0.243
Urlencodings	**0.0060*=0.00180	*0.68=0.07752	**0.0060*=0.00294	*0.68=0.16524
Unicode	*0.81758=0.00147	*0.99985=0.07750	*0.81758=0.00240	*0.99985=0.16521
Hexcodes	*0.94204=0.00138	*0.99985=0.07749	*0.94204=0.00226	*0.99985=0.16519
Maxparamlength	*0.137=0.00018	*0.03=0.00232	*0.137=0.00031	*0.03=0.00495
Maltags	**0.99997=0.00018*	**0.99985=0.00232*	**0.99997=0.00031*	**0.99985=0.00495*
Susppatterns	**0.99783=0.00018*	**0.95701=0.00222*	**0.99783=0.00031*	**0.95701=0.00474*
Malmethods	**0.99997=0.00018*	**0.98557=0.00219*	**0.99997=0.00031*	**0.98557=0.00467*
Maleventhandlers	**0.99997=0.00018*	**0.99985=0.00219*	**0.99997=0.00031*	**0.99985=0.00467*
Httpversion	*0.99995=0.00018	*0.88547=0.00194	*0.99995=0.00031	*0.88547=0.00413
Statuscode	*0.96=0.00017	*0.49992=0.00097	**0.02*=0.00001	*0.44281=0.00183
Tautologies	**0.99995=0.00017*	**0.99971=0.00097*	**0.99995=0.00001*	**0.99971=0.00183*
Sqlqueries	**0.99997=0.00017*	**0.99985=0.00097*	**0.99997=0.00001*	**0.99985=0.00183*
Pathtraversals	*0.97420=0.00016	*0.04298=0.00004	*0.97420=0.00001	*0.04298=0.00007

Table. Probability calculations of correctly vs. incorrectly classified Path Traversal attack

What becomes clear when viewing the detailed calculation is that the Path Traversal feature value cannot make up for the returned status code probability used. Remember that the Probability Distribution Table for the returned status code was already adapted on non-2xx values, because the learning data caused no or hardly any error return codes. In dataset 2 however, 67 Path Traversal attempts generated an HTTP error value of 404 or 500, causing all Path Traversal misclassifications.

As a test we set the probability for a higher return code to 10% in the PDT for status return code. As described earlier in this chapter, the PDT for error return code has three columns with value ranges (0-250.5], (250.0-403.], (403.5-∞). When testing dataset 2 with this change gives a much better performance as can be viewed in the table below.

Estimate: **Class:**	**XSS**	**SQLi**	**PT**	**Recon.**	**Other**	**TP-rate**	**FP-rate**
XSS	1235	8	0	0	3	0.991	0.001
SQL injection	0	171	1	3	6	0.944	0.007
Path Traversal	0	0	**203**	3	9	**0.994**	0.000
Reconnaissance	1	2	0	22	12	0.594	0.003
Other	1	2	0	0	9	0.750	0.017
					Avg.	**0.970**	**0.004**

Table. Improved results of dataset 2 when changing the PDT of returned status code

Now the overall performance increases to a TP-rate of 97%, with only effects on the problem at hand. TP-rate for Path Traversal attacks increases to 99.4%, and only 12 alerts will be misclassified any more. Note that of course, tweaking probabilities for one dataset is very risky and does not guarantee any general improvement for the model in the real world, but the improvement here is remarkable nonetheless. In fact, this tweaking would be very similar to what incremental learning would do to the probability distribution table, so the adaptation seems justified. If incremental learning would be used, probabilities are updated with new events. If, in this case, a larger relative part of Path Traversal attacks does receive response error codes, the PDT's would gradually adapt to this 'new situation'. Note that in the case of misclassifications the attack class values will have to be corrected manually by an operator in order to improve the model.

Reconnaissance and Other misclassifications

In the two classes of Reconnaissance and Other attacks the second dataset is not classified very well. There is no need to analyse individual misclassifications, because this result merely shows what is expected; To be able to classify alerts, clear class definitions of attacks are needed. Both Reconnaissance and Other attacks consist of so many different types of attacks or malicious attempts that these alerts cannot be recognized from one class definition.

7.6.2. Threshold

The second data set can also be tested with a threshold. The resulting performance is displayed in the table below.

Required Estimation Certainty	**#events below certainty level. Displayed is: #events (correct/incorrect)**	**Resulting TP rate (Correct guesses)**	**Total FP rate**	**Average FP rate per class**
>95%	288 (199/89)	82.4%	0.6%	0.1%
>90%	155 (70/85)	90.0%	0.9%	0.1%
>80%	131 (56/75)	90.8%	1.4%	0.2%

Table. TP/FP rates for second dataset with threshold

With a threshold of 90% 155, which is over 9% of all alerts, will be presented as uncertain classifications of which over half would be incorrect. Overall the alerts that then would be classified correctly would be 90% and 0.9%, or 14 alerts would be misclassified. For a more detailed view on the misclassifications the results are displayed in more detail below.

Estimate: Class:	**XSS**	**SQLi**	**PT**	**Recon.**	**Other**	**TP-rate**	**FP-rate**
XSS	1229	4	0	0	3	0.991	0.001
SQL injection	0	122	0	0	0	0.994	0.007
Path Traversal	0	0	146	0	1	0.994	0.000
Reconnaissance	0	0	0	18	8	0.594	0.003
Other	0	1	0	0	7	0.750	0.017
					Avg.	**0.900**	**0.009**

In the three well-defined attack classes the total number of misclassifications would go down to 8 out of 1642 alerts, with a TP-rate of over 99% for all three classes. The classification performance of the two somewhat vaguer defined classes remains low with TP-rates of 59.4% and 75% for Reconnaissance and Other attacks respectively.

8. Conclusions

The overall performance of the Bayesian network on a first dataset, using the selected features for four attack classes (XSS, SQLi, Path Traversal, Reconnaissance) and a category of all remaining attacks, has a TP rate of 96.9% with an average FP rate per class of 0.9%, or total misclassification rate of 3.1%. For the well defined attack classes XSS, SQLi and Path Traversal, the average performance is good or very good with TP rates of respectively 95.3%, 99.4% and 99.6% and FP rates of 0.4%, 1.1%, 0.3%. These results are achieved with a dataset of 1440 attacks with ten-fold cross validation. After removing features that were not contributing to the performance, a Bayesian classifier was built on 14 attack features.

After adapting the Bayesian network slightly for unrealistic test distributions the attack class distribution was set to a more realistic distribution and the Probability Distribution Tables for four of the nodes were simplified to prevent overfitting. Also, unrealistically low probability values for ranges of feature values were increased a bit without influencing the learned network too much. With these changes the performance of the network remained stable and individual attack class TP/FP rates did not deteriorate.

To prevent misclassifications a threshold is introduced where an alert classification is presented as uncertain if the probability of the estimated class is below the threshold. If the threshold is set to 90%, meaning that the network must be at least 90% sure before a guess is accepted, the overall TP rate is still 92.4%, but the total FP or misclassification rate drops from 3.1% to 1.2%, which is a welcome improvement because it would mean that out of 1440 alerts, only 17 would be misclassified.

First results with the second dataset using the Bayesian model that was learned from the first dataset were a bit less impressive. Overall performance of 94.2% was still good, but only the classes for XSS and SQL injection performed well with respectively a 99.1% and 94.4% TP-rate. Path Traversal only scores a 72% TP-rate, but this can be addressed to the fact that in dataset 1 all PT attempts had a HTTP return code of 200. In dataset 2 however, 67 PT attempts generated an error code. If probabilities for the error return code PDT would be changed a bit as if incremental learning was used, the TP-rate for Path Traversal attacks would climb to 99.4% giving an overall weighted TP-rate of 97%.

Using a threshold on dataset 2 has a similar effect as with dataset 1. The TP-rate would go down to 90%, but only 14 alerts out of a total of 1691 alerts would be misclassified. In operational security monitoring it will be preferable to present "unsure how to classify this alert" to the operator over having a rather high misclassification rate. The classification result can then still be presented to the user as a best guess, leaving the final decision up to the operator. Also, with any manual overrides, the Bayesian network probability tables can be refined automatically and continuously through incremental learning, resulting in a self-improving system.

The experimental results show that it might very well be possible to classify anomalies with a high TP rate and a low misclassification rate, using simple features and a self-learning Naïve Bayesian network. A TP-rate of over 90%, as formulated in the research goal, might be achievable in practice for any well-defined attack class. A FP-rate of 1% or less might even be achievable by using a threshold.

Future research could be to define a full set of clearly scoped attack classes and to implement the Bayesian classifier in an operational situation. A possible improvement on classification can be achieved by integrating anomaly details from the IDS such as in what part of the data the anomaly was detected exactly. This could make the feature extraction more precise, possibly improving the results. What has not been tested yet in this experiment is how the model reacts on false positives from the IDS. Classification of alerts that later turn out to be false positives can be confusing, so maybe classification should be presented as a guess until approved by an operator. The effects of incremental learning would also be interesting to study. For the eventual quality of the Bayesian network with incremental learning it should be studied how the model stabilizes after a longer period of incremental learning.

Bibliography

1. AlephOne (1996), “Smashing The Stack For Fun And Profit”, Phrack 49th Ed. File 14th of 16, Phrack.org
2. Alexa, the Web Information Company. (2012). Retrieved December 21, 2012, from http://www.alexa.com/siteinfo/fontys.nl
3. Anley, C. (2002). Advanced SQL injection in SQL server applications. *White paper, Next Generation Security Software Ltd*.
4. Barth, A. (2011). RFC 6265, HTTP State Management Mechanism
5. Bolzoni, D. (2009). Revisiting anomaly-based network intrusion detection systems. University of Twente.
6. Bouckaert, R. R. (2004). *Bayesian network classifiers in weka*. Department of Computer Science, University of Waikato.
7. Carstens, T. (2005). Sniffex.c (ver.0.1.1) [Computer Software]. Retrieved September 2012, from http://www.tcpdump.org/sniffex.c.
8. Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine learning*, *20*(3), 273-297.
9. Cowan, C., Wagle, F., Pu, C., Beattie, S., & Walpole, J. (2000). Buffer Overflows: Attacks and defenses for the vulnerability of the decade. In *DARPA Information Survivability Conference and Exposition, 2000. DISCEX'00. Proceedings* (Vol. 2, pp. 119-129). IEEE.
10. Cross-Site Request Forgery (CSRF) Prevention Cheat Sheet. (2011). Retrieved June 14, 2013, from https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)_Prevention_Cheat_Sheet.
11. Designer, S. Getting around non-executable stack (and fix). 1997. *Bugtraq mailing list, http://www.securityfocus.com/archive/1/7480*, Aug. 1997.
12. Dewan Md. Farid, Nouria Harbi, & Mohammad Zahidur Rahman, Combining Naïve Bayes and Decision Tree for Adaptive Intrusion Detection, International Journal of Network Security & Its Applications, Vol. 2, No. 2, April 2010, pp. 12-25.
13. Eichelberger, R. K., & Sheng, V. S. (2013, June). Does One-Against-All or One-Against-One Improve the Performance of Multiclass Classifications?. In Twenty-Seventh AAAI Conference on Artificial Intelligence.
14. Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P. & Berners-Lee, T. (1999). RFC 2616, Hypertext Transfer Protocol -- HTTP/1.1
15. Friedman, N., Geiger, D., & Goldszmidt, M. (1997). Bayesian network classifiers. *Machine learning*, *29*(2-3), 131-163.
16. Halfond, W. G., Orso, A., & Manolios, P. (2008). WASP: Protecting Web applications using positive tainting and syntax-aware evaluation. *Software Engineering, IEEE Transactions on*, *34*(1), 65-81.
17. Halfond, W. G., Viegas, J., & Orso, A. (2006, March). A classification of SQL-injection attacks and countermeasures. In *Proceedings of the IEEE International Symposium on Secure Software Engineering, Arlington, VA, USA* (pp. 13-15).
18. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., & Witten, I. H. (2009). The WEKA data mining software: an update. *ACM SIGKDD Explorations Newsletter*, *11*(1), 10-18.
19. Hsu, F. H., Guo, F., & Chiueh, T. C. (2006, December). Scalable network-based Buffer Overflow attack detection. In Architecture for Networking and Communications systems, 2006. ANCS 2006. ACM/IEEE Symposium on (pp. 163-172). IEEE.
20. Imperva (2013). Imperva Webapplication Attack Report – Edition #4 – July 2013. Retrieved October 14, 2013, from http://www.imperva.com/docs/HII_Web_Application_Attack_Report_Ed4.pdf.
21. Johns, M.; Engelmann, B.; Posegga, J. (2008). XSSDS: Server-Side Detection of Cross-site Scripting Attacks, *Computer Security Applications Conference, 2008. ACSAC 2008. Annual* , vol., no., pp.335,344
22. Klein, A. (2005). DOM based Cross-site Scripting or XSS of the third kind. *Web Application Security Consortium, Articles*, *4*.
23. Kotsiantis, S. B., Zaharakis, I. D., & Pintelas, P. E. (2007). Supervised machine learning: A review of classification techniques.

24. Langley, P., Iba, W., & Thompson, K. (1992, July). An analysis of Bayesian classifiers. In *AAAI* (Vol. 90, pp. 223-228).
25. Larouche, F. (2007). SQL Power Injector (ver.2.1) [Computer Software]. Retrieved September 2012, from http://www.sqlpowerinjector.com.
26. Kruegel, C., Vigna, & G., Robertson, W. (2005). A Multi-model Approach to the Detection of Web-based Attacks. Computer Networks 48(5), 717–738
27. ModSecurity SQL Injection Challenge: Lessons Learned. (2011). Retrieved November 11, 2012, from http://blog.spiderlabs.com/2011/07/modsecurity-SQL-injection-challenge-lessons-learned.html.
28. Muntner, A. (2012). FuzzDB, Attack and Discovery Pattern Database for Application Fuzz Testing. Retrieved November 2012, from https://code.google.com/p/fuzzdb/.
29. Nunan, A.E., Souto, E., dos Santos, E.M., & Feitosa, E. (2012). "Automatic classification of Cross-site Scripting in web pages using document-based and URL-based features," *Computers and Communications (ISCC), IEEE Symposium on* , vol., no., pp.000702,000707
30. OWASP (2012). OWASP Zed Attack Proxy (1.4.1) [Computer Software]. Retrieved November 2012, from https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project.
31. OWASP (2013). OWASP Xenotix XSS Exploit Framework. Retrieved September 2013, from https://www.owasp.org/index.php/OWASP_Xenotix_XSS_Exploit_Framework.
32. Rish, I. (2001, August). An empirical study of the naive Bayes classifier. In *IJCAI 2001 workshop on empirical methods in artificial intelligence* (Vol. 3, No. 22, pp. 41-46).
33. Robertson, W. K., Vigna, G., Kruegel, C., & Kemmerer, R. A. (2006, February). Using Generalization and Characterization Techniques in the Anomaly-based Detection of Web Attacks. In *NDSS*.
34. Rocha, D.P. (2012). Uniscan (ver.2.0) [Computer Software]. Retrieved September 2012, from http://sourceforge.net/projects/uniscan/.
35. Saint-Andre, P., Crocker, D., & Nottingham, M., (2012). RFC 6648, Deprecating the "X-" Prefix and Similar Constructs in Application Protocols
36. SecurityMatters (2013). SilentDefense Web. Last visited October 2013, at http://www.secmatters.com/products-web.
37. Tcpdump (2012). Tcpdump & Libpcap (ver.4.3.0) [computer software]. Retrieved September 2012, from http://www.tcpdump.org.
38. Toth, T., & Kruegel, C. (2002, January). Accurate Buffer Overflow detection via abstract pay load execution. In *Recent Advances in Intrusion Detection* (pp. 274-291). Springer Berlin Heidelberg.
39. University of Waikato (2002). ARFF data format. Last visited October 2013, at http://weka.wikispaces.com/ARFF.
40. Wang, X., Pan, C. C., Liu, P., & Zhu, S. (2010). Sigfree: A signature-free Buffer Overflow attack blocker. *Dependable and Secure Computing, IEEE Transactions on*, *7*(1), 65-79.
41. Wang, Y. H., Mao, C. H., & Lee, H. M. (2010). Structural Learning of Attack Vectors for Generating Mutated XSS Attacks. *arXiv preprint arXiv:1009.3711*.
42. Wassermann, G., & Zhendong Su. (2008). Static detection of Cross-site Scripting vulnerabilities, *Software Engineering*, 2008. ICSE '08. ACM/IEEE 30th International Conference on , vol., no., pp.171,180, 10-18
43. Wharton, D. (2007). XSRF: Checking HTTP Referer Header Is Not Enough. Retrieved June 14, 2013, from http://www.secureworks.com/cyber-threat-intelligence/blog/research/21009/.

Appendix A. HTTP parser source code

```
/****************************************************************************
 * http_parser.c
 * author: Casper Schellekens
 *
 * Tool to parse http message parts request-/responseline, header(s),
 * message body.
 * This program is based on a tcpdump's demo program of TCP/IP packet
 * capture using libpcap. Below the copyright terms to reuse this source
 * code.
 *
 ***************************************************************************
 *
 ***************************************************************************
 *
 * sniffex.c
 *
 * Sniffer example of TCP/IP packet capture using libpcap.
 *
 * Version 0.1.1 (2005-07-05)
 * Copyright (c) 2005 The Tcpdump Group
 *
 * This software is intended to be used as a practical example and
 * demonstration of the libpcap library; available at:
 * http://www.tcpdump.org/
 *

****************************************************************************
 *
 * This software is a modification of Tim Carstens' "sniffer.c"
 * demonstration source code, released as follows:
 *
 * sniffer.c
 * Copyright (c) 2002 Tim Carstens
 * 2002-01-07
 * Demonstration of using libpcap
 * timcarst -at- yahoo -dot- com
 *
 * "sniffer.c" is distributed under these terms:
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 * 4. The name "Tim Carstens" may not be used to endorse or promote
 *    products derived from this software without prior written permission
 *
 * THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS ``AS IS'' AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE
 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
 * THE POSSIBILITY OF SUCH DAMAGE.
 * <end of "sniffer.c" terms>
 *
 * This software, "sniffex.c", is a derivative work of "sniffer.c" and is
 * covered by the following terms:
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Because this is a derivative work, you must comply with the
 *    "sniffer.c" terms reproduced above.
 * 2. Redistributions of source code must retain the Tcpdump Group copyright
 *    notice at the top of this source file, this list of conditions and the
 *    following disclaimer.
 * 3. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 * 4. The names "tcpdump" or "libpcap" may not be used to endorse or promote
 *    products derived from this software without prior written permission.
 *
 * THERE IS ABSOLUTELY NO WARRANTY FOR THIS PROGRAM.
 * BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY
 * FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW.  EXCEPT WHEN
 * OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES
 * PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER
 * EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
 * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.  THE
 * ENTIRE RISK AS
 * TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU.  SHOULD THE
 * PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING,
 * REPAIR OR CORRECTION.
 *
 * IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING
 * WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR
 * REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR
 * DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL
 * DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING
 * BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR
 * LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO
 * OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS
 * BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.
 * <end of "sniffex.c" terms>
 * ***************************************************************************/

#define APP_NAME          "http_parser, based on sniffex"
#define APP_DESC          "Sniffer example using libpcap"
#define APP_COPYRIGHT     "Copyright (c) 2005 The Tcpdump Group"
#define APP_DISCLAIMER    "THERE IS ABSOLUTELY NO WARRANTY FOR THIS
PROGRAM."

#include <pcap.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

/* capture live or offline from pcap file */
#define PCAP_LIVE 1
#define PCAP_OFFLINE 2

/* default snap length (maximum bytes per packet to capture) */
#define SNAP_LEN 1518

/* ethernet headers are always exactly 14 bytes [1] */
```

```
#define SIZE_ETHERNET 14

/* Ethernet addresses are 6 bytes */
#define ETHER_ADDR_LEN     6

/* Ethernet header */
struct sniff_ethernet {
        u_char  ether_dhost[ETHER_ADDR_LEN];    /* destination host address
*/
        u_char  ether_shost[ETHER_ADDR_LEN];    /* source host address */
        u_short ether_type;                     /* IP? ARP? RARP? etc */
};

/* IP header */
struct sniff_ip {
        u_char  ip_vhl;                 /* version << 4 | header length >> 2
*/
        u_char  ip_tos;                 /* type of service */
        u_short ip_len;                 /* total length */
        u_short ip_id;                  /* identification */
        u_short ip_off;                 /* fragment offset field */
        #define IP_RF 0x8000            /* reserved fragment flag */
        #define IP_DF 0x4000            /* dont fragment flag */
        #define IP_MF 0x2000            /* more fragments flag */
        #define IP_OFFMASK 0x1fff       /* mask for fragmenting bits */
        u_char  ip_ttl;                 /* time to live */
        u_char  ip_p;                   /* protocol */
        u_short ip_sum;                 /* checksum */
        struct  in_addr ip_src,ip_dst;  /* source and dest address */
};
#define IP_HL(ip)               (((ip)->ip_vhl) & 0x0f)
#define IP_V(ip)                (((ip)->ip_vhl) >> 4)

/* TCP header */
typedef u_int tcp_seq;

struct sniff_tcp {
        u_short th_sport;               /* source port */
        u_short th_dport;               /* destination port */
        tcp_seq th_seq;                 /* sequence number */
        tcp_seq th_ack;                 /* acknowledgement number */
        u_char  th_offx2;               /* data offset, rsvd */

#define TH_OFF(th)      (((th)->th_offx2 & 0xf0) >> 4)
        u_char  th_flags;
        #define TH_FIN  0x01
        #define TH_SYN  0x02
        #define TH_RST  0x04
        #define TH_PUSH 0x08
        #define TH_ACK  0x10
        #define TH_URG  0x20
        #define TH_ECE  0x40
        #define TH_CWR  0x80
        #define TH_FLAGS (TH_FIN|TH_SYN|TH_RST|TH_ACK|TH_URG|TH_ECE|TH_CWR)
        u_short th_win;                 /* window */
        u_short th_sum;                 /* checksum */
        u_short th_urp;                 /* urgent pointer */
};

/* HTTP message definitions */
#define NO_HTTP  0
#define HTTP_REQ 1
#define HTTP_RES 2

#define NO_REQLINE 0
#define REQLINE    1
```

```
#define FALSE 0
#define TRUE  1

/* pcap file to be read */
char *pcapfile = NULL;

/* function declarations */
/* check_http and parse_http are by author C.Schellekens */
/* got_packet is by sniffex author and slightly adapted by C.Schellekens */
/* print_payload, print_hex_ascii_line, print_app_banner, print_app_usage */
/* are from author sniffex.c */

int
check_http(const char *payload, int len);

void
parse_http(const char *payload, int len, int type);

void
got_packet(u_char *args, const struct pcap_pkthdr *header, const u_char
*packet);

void
print_payload(const u_char *payload, int len);

void
print_hex_ascii_line(const u_char *payload, int len, int offset);

void
print_app_banner(void);

void
print_app_usage(void);


/*
 * app name/banner
 */
void
print_app_banner(void)
{

     printf("%s - %s\n", APP_NAME, APP_DESC);
     printf("%s\n", APP_COPYRIGHT);
     printf("%s\n", APP_DISCLAIMER);
     printf("\n");

return;
}

/*
 * print help text
 */
void
print_app_usage(void)
{

     printf("Usage: %s [interface]\n", APP_NAME);
     printf("\n");
     printf("Options:\n");
     printf("    interface    Listen on <interface> for packets.\n");
     printf("\n");

return;
}
```

```
/*
 * print data in rows of 16 bytes: offset   hex   ascii
 *
 * 00000   47 45 54 20 2f 20 48 54  54 50 2f 31 2e 31 0d 0a   GET /
HTTP/1.1..
 */
void
print_hex_ascii_line(const u_char *payload, int len, int offset)
{

     int i;
     int gap;
     const u_char *ch;

     /* offset */
     printf("%05d   ", offset);

     /* hex */
     ch = payload;
     for(i = 0; i < len; i++) {
       printf("%02x ", *ch);
       ch++;
       /* print extra space after 8th byte for visual aid */
       if (i == 7)
              printf(" ");
     }
     /* print space to handle line less than 8 bytes */
     if (len < 8)
       printf(" ");

     /* fill hex gap with spaces if not full line */
     if (len < 16) {
       gap = 16 - len;
       for (i = 0; i < gap; i++) {
              printf("   ");
       }
     }
     printf("   ");

     /* ascii (if printable) */
     ch = payload;
     for(i = 0; i < len; i++) {
       if (isprint(*ch))
              printf("%c", *ch);
       else
              printf(".");
       ch++;
     }

     printf("\n");

return;
}

/*
 * print packet payload data (avoid printing binary data)
 */
void
print_payload(const u_char *payload, int len)
{

     int len_rem = len;
     int line_width = 16;            /* number of bytes per line */
     int line_len;
     int offset = 0;                 /* zero-based offset counter */
	const u_char *ch = payload;

	if (len <= 0)
	  return;

	/* data fits on one line */
	if (len <= line_width) {
	  print_hex_ascii_line(ch, len, offset);
	  return;
	}

	/* data spans multiple lines */
	for ( ;; ) {
	  /* compute current line length */
	  line_len = line_width % len_rem;
	  /* print line */
	  print_hex_ascii_line(ch, line_len, offset);
	  /* compute total remaining */
	  len_rem = len_rem - line_len;
	  /* shift pointer to remaining bytes to print */
	  ch = ch + line_len;
	  /* add offset */
	  offset = offset + line_width;
	  /* check if we have line width chars or less */
	  if (len_rem <= line_width) {
		/* print last line and get out */
		print_hex_ascii_line(ch, len_rem, offset);
		break;
	  }
	}

return;
}

/*
 * dissect/print packet
 */
void
got_packet(u_char *args, const struct pcap_pkthdr *header, const u_char
*packet)
{
	static int count = 1;                   /* packet counter */
	/* declare pointers to packet headers */
	const struct sniff_ethernet *ethernet;  /* The ethernet header [1] */
	const struct sniff_ip *ip;              /* The IP header */
	const struct sniff_tcp *tcp;            /* The TCP header */
	const char *payload;                    /* Packet payload */

	int size_ip;
	int size_tcp;
	int size_payload;
	int http_type;

	printf("\nPacket number %d:\n", count);
	count++;

	/* define ethernet header */
	ethernet = (struct sniff_ethernet*)(packet);

	/* define/compute ip header offset */
	ip = (struct sniff_ip*)(packet + SIZE_ETHERNET);
	size_ip = IP_HL(ip)*4;
	if (size_ip < 20) {
	  printf("   * Invalid IP header length: %u bytes\n", size_ip);
	  return;
	}
```

```
     /* print source and destination IP addresses */
     printf("       From: %s\n", inet_ntoa(ip->ip_src));
     printf("         To: %s\n", inet_ntoa(ip->ip_dst));

     /* determine protocol */
     switch(ip->ip_p) {
       case IPPROTO_TCP:
              printf("   Protocol: TCP\n");
              break;
       case IPPROTO_UDP:
              printf("   Protocol: UDP\n");
              return;
       case IPPROTO_ICMP:
              printf("   Protocol: ICMP\n");
              return;
       case IPPROTO_IP:
              printf("   Protocol: IP\n");
              return;
       default:
              printf("   Protocol: unknown\n");
              return;
     }

     /*
      *  OK, this packet is TCP.
      */

     /* define/compute tcp header offset */
     tcp = (struct sniff_tcp*)(packet + SIZE_ETHERNET + size_ip);
     size_tcp = TH_OFF(tcp)*4;
     if (size_tcp < 20) {
       printf("   * Invalid TCP header length: %u bytes\n", size_tcp);
       return;
     }

     printf("   Src port: %d\n", ntohs(tcp->th_sport));
     printf("   Dst port: %d\n", ntohs(tcp->th_dport));

     /* define/compute tcp payload (segment) offset */
     payload = (u_char *)(packet + SIZE_ETHERNET + size_ip + size_tcp);

     /* compute tcp payload (segment) size */
     size_payload = ntohs(ip->ip_len) - (size_ip + size_tcp);

     /*
      * Print payload data; it might be binary, so don't just
      * treat it as a string.
      */
     if (size_payload > 0) {
       printf("   Payload (%d bytes):\n", size_payload);
       print_payload(payload, size_payload);
       /* check if http request or response */
       http_type = check_http(payload, size_payload);
       if (http_type != NO_HTTP) {
              if (http_type == HTTP_REQ) printf("HTTP Request msg
                                                   found!!\n");
              if (http_type == HTTP_RES) printf("HTTP Response msg
                                                   found!!\n");
              /* parse all parts of the http message */
              parse_http(payload, size_payload, http_type);
       }
     }
return;
}
```

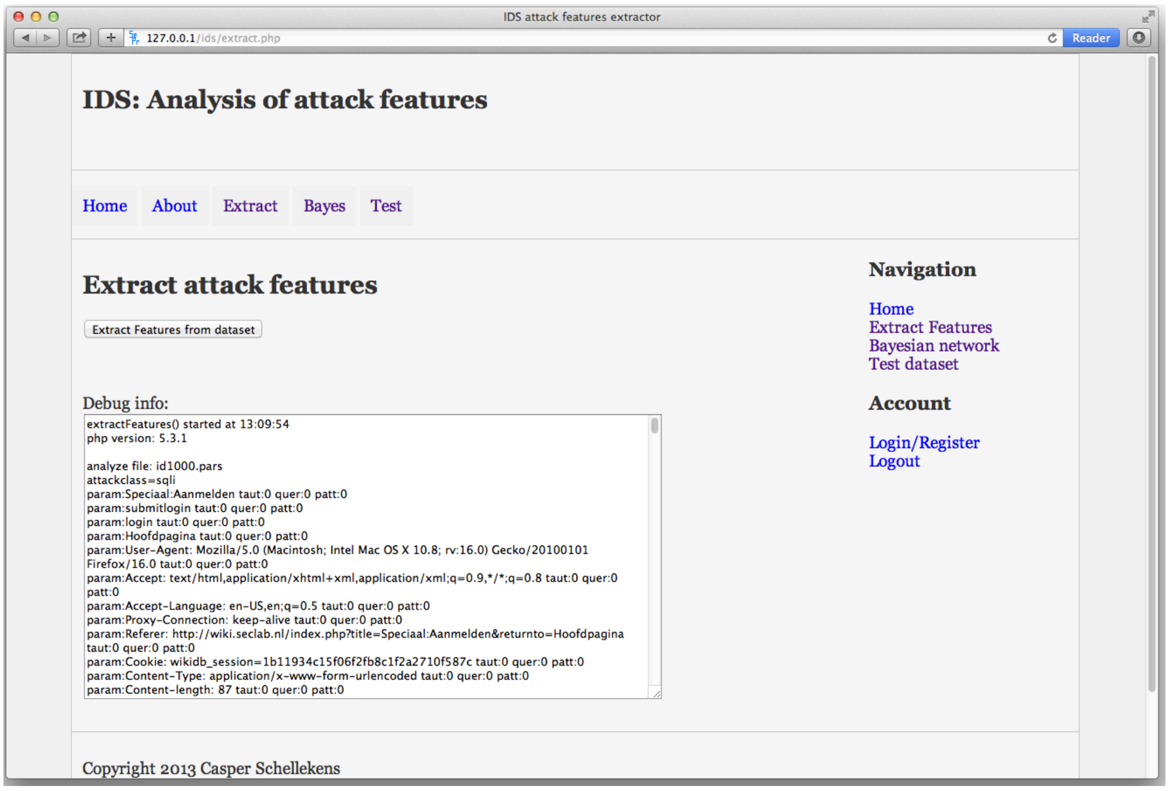
```
int
check_http(const char *payload, int len)
{
      int http_type = NO_HTTP;
      char * pch;
      char * pch2;
      char strpart[5]; //teststring to find "HTTP"
      if (strlen((char *)payload) > 8) {
        if (strncmp(payload,"HTTP", 4) == 0) http_type = HTTP_RES;
        else {
          /* if not a HTTP response, check if HTTP request, */
          /* so first line must end with HTTP version */
          pch = strstr(payload, "\r\n");
          if (pch == NULL) {
            /* try ending '\n\n' even though non-compliant to standard, */
            /* because sometimes used in scans */
            pch = strstr(payload, "\n\n");
               if (pch != NULL) {
                 printf("CRCR found at %d!!!\n", pch-payload+1);
                 pch2 = strstr(payload, "\n");
                 if (pch2 != NULL && pch2 < pch) pch = pch2;
               }
               else {
                 /* try ending '\n' even though non-compliant to standard, */
                 /* because sometimes used in scans */
                 pch = strstr(payload, "\n");
               }
            }
            if (pch != NULL) {
               printf("CRLF found at %d!!\n", pch-payload+1);
               strncpy(strpart, pch-8, 4);
               if (strncmp(strpart,"HTTP", 4) == 0) http_type = HTTP_REQ;
               else {
                 strncpy(strpart+4, "", 1);
                 printf("no HTTP found but: %s", strpart);
               }
            }
          }
      }
      return http_type;
}

void
parse_http(const char *payload, int len, int type)
{
  int pos = 0;      /* relative position in tcp payload data */
  char * pch;       /* pointer to position in tcp payload data */
  char * pch2;
  char * line;      /* one line in the http message */
  int linenr = 0; /* keeps track of number of lines dealt with */
  FILE *fp, *fpb;
  char fnw[50], fnb[50], fn[50]; /* file name to write to */
  int i,j,k;
  char alertid[10];
  char * param;
  char * pfn;
  int param_end = FALSE; /* whether end of request query parameter is found
*/
  int uncompliant = FALSE;
  int singleend = FALSE; /* if line ending is CR iso CRLF */
  const char *html_entity[15] = {"quot", "apos", "amp", "lt", "gt", "nbsp",
                                 "copy", "laquo", "raquo", "not", "times",
                                 "divide", "Tab", "NewLine", "colon"};
  const int html_entity_codes = 15;

  line = malloc(100);
  strncpy(fnw, pcapfile, strlen(pcapfile));
  strncpy(fnw+strlen(pcapfile)-5, ".pars", 5);
  fnw[strlen(pcapfile)]= '\0';
  fp = fopen(fnw, "ab");
  if (fp == NULL) {
    fprintf(stderr, "file <%s> could not be opened for writing,
quitting..\n",
      fnw);
    exit(0);
  }
  /* add alertid, only once at beginning of file */
  if (ftell(fp) == 0) {
    strncpy( alertid, "0\0", 2);
    j = 0;
    for (i=0; i<strlen(pcapfile); i++) {
      /* check if numeric */
      if (pcapfile[i] >= 48 && pcapfile[i] <= 57) {
        strncpy(alertid+j, pcapfile+i, 1);
        j++;
      }
    }
    alertid[j]='\0';
    fprintf(fp, "alertid: %s\n", alertid);
  }
  pch = strstr(payload+pos, "\r\n");
  if (pch == NULL) {
    /* try ending '\n\n' even though non-compliant to standard, */
    /* because sometimes used in scans */
    pch = strstr(payload+pos, "\n\n");
      if (pch != NULL) {
        pch2 = strstr(payload+pos, "\n");
        if (pch2 < pch) {
          pch = pch2;
          singleend = TRUE;
        }
      }
      else {
        pch = strstr(payload+pos, "\n");
        if (pch != NULL) singleend = TRUE;
      }
      if (pch != NULL) {
        uncompliant = TRUE;
        printf("uncompliant request ending CRCR/CR found!\n");
      }
     }
     while (pch != NULL && (pch-(payload+pos)) != 0) {
       line = realloc(line, pch-(payload+pos)+1);
       strncpy(line, payload+pos, pch-(payload+pos) );
       line[pch-(payload+pos)] = '\0';
       printf("request line %d: %s\n", linenr, line);
       if (linenr == 0) { //first line is the request/response line
         if (type == HTTP_RES) {
           fprintf(fp, "type: Response\n");
           fprintf(fp, "responseline: %s\n", line);
         }
         if (type == HTTP_REQ) {
           fprintf(fp, "type: Request\n");
           fprintf(fp, "requestline: %s\n", line);
           for (i=0; i<strlen(line); i++) {
             if (line[i] == '?') {
             /* parameters follow */
             /* make sure enough memory is reserved for parameters */
             param = malloc(strlen(line));
             i++;
             while (line[i] != 32) {    /* space is end of query */
               j=0;
```

```
              /* param name until '=' normally, */
              /* sometimes only value is given */
              while(line[i] != 61 && line[i] != 38 && line[i] != 32) {
                param[j] = line[i];
                i++;j++;
              }
              param[j] = '\0';
              if (line[i] == 61) {
                /* after '=' comes value, */
                /* otherwise it was the value already */
                /* not the parameter name */
                fprintf(fp, "parameter: %s = ", param);
                j=0;i++;
                param_end = FALSE; //until proven TRUE
                while(param_end == FALSE){
                  /* parameter value until '&' or space */
                  while(line[i] != 38 && line[i] != 32) {
                    param[j] = line[i];
                    i++;j++;
                  }
                  if (line[i+1] != 35) {
                    param_end = TRUE;
                    for (k=0; k<html_entity_codes; k++) {
                      if (strncmp(line+i+1, html_entity[k],
                           strlen(html_entity[k])) == 0) {
                               param_end = FALSE;
                        param[j] = line[i];
                        i++;j++;
                        k = html_entity_codes;
                      }
                    }
                  }
                  else {
                    param[j] = line[i];
                    i++;j++;
                  }
                }
                param[j] = '\0';
              }
              else {
                /* write parameter keyword without parameter name, */
                /* because there is none! */
                fprintf(fp, "parameter: = ");
              }
              fprintf(fp, "%s\n", param);
              if (line[i] == 38) i++;   /* skip '&' to get to next parameter
*/
            }
            free(param);
          }
        }
      }
      if (uncompliant) {
        fprintf(fp, "crending: true\n");
        uncompliant = FALSE;
      }
    }
    else { //else it should be a header
      fprintf(fp, "header: %s\n", line);
      if (uncompliant) {
        fprintf(fp, "crending: true\n");
        uncompliant = FALSE;
      }
    }
    /* skip de '\r\n' to get to new line beginning */
    if (!singleend) pos += pch-(payload+pos)+2;
```

```
      else {
        /* skip de '\n' to get to new line beginning */
        pos += pch-(payload+pos)+1;
        singleend = FALSE;
      }
      printf("new position = %d\n", pos);
      linenr++;
      pch = strstr(payload+pos, "\r\n");
      if (pch == NULL) {
        /* try ending '\n\n' even though non-compliant to standard, */
        /* because sometimes used in scans */
        pch = strstr(payload+pos, "\n\n");
        if (pch != NULL) {
          pch2 = strstr(payload+pos, "\n");
          if (pch2 < pch) {
            pch = pch2;
            singleend = TRUE;
          }
        }
        else {
          pch = strstr(payload+pos, "\n");
          if (pch != NULL) singleend = TRUE;
        }
        if (pch != NULL) uncompliant = TRUE;
      }
    }
    printf("len - pos = %d - %d = %d\n", len, pos, len-pos);
    if (len - pos >0) {
      /* otherwise it is just the message ending */
      if (!(strncmp(payload+pos,"\r\n", 2) == 0 && len - pos ==2)) {
        if (type == HTTP_REQ || type == HTTP_RES) {
          strncpy(fnb, fnw, strlen(fnw)+1);
          if (type == HTTP_REQ) strncpy(fnb+strlen(fnw)-5, ".rqbd", 5);
          if (type == HTTP_RES) strncpy(fnb+strlen(fnw)-5, ".rsbd", 5);
          /* only copy the filename, not the foldername */
          if (strrchr(fnw,'/')!= NULL) pfn = strrchr(fnw,'/')+1;
          else pfn = fnw;
          strncpy(fn, pfn, strlen(fnw)+1);
          if (type == HTTP_REQ) strncpy(fn+strlen(pfn)-5, ".rqbd", 5);
          if (type == HTTP_RES) strncpy(fn+strlen(pfn)-5, ".rsbd", 5);
          printf("bodyfile: %s (fnlength:%d), pars-file: %s (fnlength:%d)\n",
                  fnb, strlen(fnb), fnw, strlen(fnw));
        }
        else {
          fprintf(stderr, "bodyfile not opened because msg type unknown,
                   quitting..\n");
          exit(0);
        }
        fpb = fopen(fnb, "ab");
        if (fpb == NULL) {
          fprintf(stderr, "file <%s> could not be opened for writing,
                   quitting..\n", fnb);
          exit(0);
        }
        /* header part should be ended with \r\n */
        if (strncmp(payload+pos,"\r\n", 2) == 0) {
          fprintf(fp, "body: %s\n",fn);
          fwrite(payload+pos+2, 1, len-pos-2, fpb);
        }
        else {
          fprintf(fp, "error: http message body does not comply to
                       standard\n");
          fprintf(fp, "errdata: %s\n",fn);
          fwrite(payload+pos, 1, len-pos, fpb);
        }
        fclose(fpb);
      }
      else {
        fprintf(fp, "no_body \n");
      }
    }
    else {
      fprintf(fp, "no_body \n");
    }
    fprintf(fp, "eom \n\n");
    fclose(fp);
    return;
  }


  int main(int argc, char **argv)
  {
      char *dev = NULL;                   /* capture device name */
      /* default is live capture, */
      /* alternative is PCAP_OFFLINE from pcap file */
      int capture = PCAP_LIVE;
      char errbuf[PCAP_ERRBUF_SIZE];      /* error buffer */
      pcap_t *handle;                     /* packet capture handle */
      /* default filter expression [3] for parsing http messages*/
      char filter_exp[] = "tcp port 80";
      struct bpf_program fp;        /* compiled filter program (expression) */
      bpf_u_int32 mask;             /* subnet mask */
      bpf_u_int32 net;              /* ip */
      int num_packets = -1;         /* number of packets to capture */

      print_app_banner();

      /* check for capture device name on command-line */
      if (argc == 2) {
        dev = argv[1];
      }
      else if (argc == 3) {
        dev = argv[1];
        pcapfile = argv[2];
        capture = PCAP_OFFLINE;
      }
      else if (argc > 3) {
        fprintf(stderr, "error: unrecognized command-line options\n\n");
        print_app_usage();
        exit(EXIT_FAILURE);
      }
      else {
        /* find a capture device if not specified on command-line */
        dev = pcap_lookupdev(errbuf);
        if (dev == NULL) {
          fprintf(stderr, "Couldn't find default device: %s\n", errbuf);
          exit(EXIT_FAILURE);
        }
      }

      if (capture == PCAP_OFFLINE) {
        handle = pcap_open_offline(pcapfile, errbuf);
      }
      else {
        /* get network number and mask associated with capture device */
        if (pcap_lookupnet(dev, &net, &mask, errbuf) == -1) {
               fprintf(stderr, "Couldn't get netmask for device %s: %s\n",
                  dev, errbuf);
               net = 0;
               mask = 0;
        }
```

```
        /* print capture info */
        printf("Device: %s\n", dev);
        printf("Number of packets: %d\n", num_packets);
        printf("Filter expression: %s\n", filter_exp);

        /* open capture device */
        handle = pcap_open_live(dev, SNAP_LEN, 1, 1000, errbuf);
        if (handle == NULL) {
                fprintf(stderr, "Couldn't open device %s: %s\n", dev, errbuf);
                exit(EXIT_FAILURE);
        }

        /* make sure we're capturing on an Ethernet device [2] */
        if (pcap_datalink(handle) != DLT_EN10MB) {
                fprintf(stderr, "%s is not an Ethernet\n", dev);
                exit(EXIT_FAILURE);
        }
      }
      /* compile the filter expression */
      if (pcap_compile(handle, &fp, filter_exp, 0, net) == -1) {
        fprintf(stderr, "Couldn't parse filter %s: %s\n",
            filter_exp, pcap_geterr(handle));
        exit(EXIT_FAILURE);
      }

      /* apply the compiled filter */
      if (pcap_setfilter(handle, &fp) == -1) {
        fprintf(stderr, "Couldn't install filter %s: %s\n",
            filter_exp, pcap_geterr(handle));
        exit(EXIT_FAILURE);
      }

      /* now we can set our callback function */
      pcap_loop(handle, num_packets, got_packet, NULL);

      /* cleanup */
      pcap_freecode(&fp);
      pcap_close(handle);

      printf("\nCapture complete.\n");

return 0;
}
```

Appendix B. Feature Extractor source code


IDS attack features extractor
127.0.0.1/ids/extract.php
Reader
IDS: Analysis of attack features
Home
About
Extract
Bayes
Test
Extract attack features
Extract Features from dataset
Navigation
Home
Extract Features
Bayesian network
Test dataset
Account
Login/Register
Logout
Copyright 2013 Casper Schellekens

Debug info:

```
extractFeatures() started at 13:09:54
php version: 5.3.1

analyze file: id1000.pars
attackclass=sqli
param:Speciaal:Aanmelden taut:0 quer:0 patt:0
param:submitlogin taut:0 quer:0 patt:0
param:login taut:0 quer:0 patt:0
param:Hoofdpagina taut:0 quer:0 patt:0
param:User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.8; rv:16.0) Gecko/20100101
Firefox/16.0 taut:0 quer:0 patt:0
param:Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8 taut:0 quer:0
patt:0
param:Accept-Language: en-US,en;q=0.5 taut:0 quer:0 patt:0
param:Proxy-Connection: keep-alive taut:0 quer:0 patt:0
param:Referer: http://wiki.seclab.nl/index.php?title=Speciaal:Aanmelden&returnto=Hoofdpagina
taut:0 quer:0 patt:0
param:Cookie: wikidb_session=1b11934c15f06f2fb8c1f2a2710f587c taut:0 quer:0 patt:0
param:Content-Type: application/x-www-form-urlencoded taut:0 quer:0 patt:0
param:Content-length: 87 taut:0 quer:0 patt:0
```

Figure. User interface of the feature extractor tool

Class_features.php:

```php
<?php
const REQTYPE = 'Request';
const RESTYPE = 'Response';

class AttackFeatures {
   // property declarations: attack features for XSS
   public $urllength = 0;
   public $urlencodingsextra = 0;
   public $doubleencodings = 0;
   public $unicode = 0;
   public $htmlentitycodes = 0;
   public $bas64encodings = 0;
   public $hexcodes = 0;
   public $maxparamlength = 0;
   public $maltags = 0;
   public $susppatterns = 0;
   public $malmethods = 0;
   public $maleventhandlers = 0;
   public $malobjects = 0;
   public $httpversion = '1.1';
   public $statuscode = '?';
   // Request method values: GET, POST, OPTIONS, HEAD, PUT, DELETE, TRACE,
   // CONNECT, PATCH, WebDAV, Invalid}
   public $requestmethod = '?';
   public $tautologies = 0;
   public $sqlqueries = 0;
   public $sqlipatterns = 0;
   public $nonprintable = 0;
   public $pathtraversals = 0;
   public $iphost = 'false';
   public $wronghost = 'false';
   public $nocrlfend = 'false';
   public $alertid = '?';
   public $attackclass = '?';

} // end of class Features

class HttpMsgParsed {
   // headers, query parameters from http request or response
   public $alertid = 0;
   public $attackclass = '?';
   public $httpversion = '1.1';  // HTTP version: 1.0 or 1.1
   public $type;                 // HTTP message type: Request or Response
   public $requestline;          // HTTP request line
   public $urllength;
   public $responseline;         // HTTP response line
   public $method;               // HTTP request method: GET, POST, etc.
   public $statuscode = '?';     // HTTP response status code
   public $paramvals = array();  // Query parameter values
   public $headers = array();    // HTTP message headers
   public $postparams = array(); // Post parameters
   public $requestbody;          // HTTP request message body
   public $responsebody;         // HTTP response message body
   // requestline and headers should end with CRLF ('\n\r'),
   // scanners and malware might be uncompliant
   public $nocrlfend = 'false';
   public $rfcerror = 'false';   // data was not compliant to HTTP standard
}

?>
```

class_extractor.php:

```
<?php
include('class_features.php');

class Extractor {
   // property declarations
   private $basepars = 'data1/';
   private $basearff = 'arff/';
   // file names for .pars file and arff file to resp. read and write
   private $fnr, $fnw;
   // file handles for .pars file and arff file to resp. read and write
   private $fhr, $fhw;
   private $ext = "pars";

   public function extractFeatures() {
      $retval = 0;
      $exclude_list = array(".", "..");
      $dir_list;
      $file;

      //date_default_timezone_set('UTC');
      set_time_limit(0);
      date_default_timezone_set('Europe/Amsterdam');
      $_SESSION['debug'] .= "extractFeatures() started at "
                             .date('H:i:s')."\n";
      $_SESSION['debug'] .= "php version: " . phpversion() . "\n";
      self::createArff();
      $dir_list = array_diff(scandir($this->basepars), $exclude_list);
      foreach ($dir_list as $file) {
         $pi = pathinfo($this->basepars.$file);
         if (is_file($this->basepars.$file) &&
              $pi['extension'] == $this->ext) {
            $reqmsg = new HttpMsgParsed();
            $resmsg = new HttpMsgParsed();
            $feat = new AttackFeatures();
            $this->fnr = $file;
            $_SESSION['debug'] .= "\nanalyze file: " . $file . "\n";
            self::readParsFile($file, $reqmsg, $resmsg);
            // Buffer Overflows were taken out of the experiment
            // because of low occurrence rate.
            if ($reqmsg->attackclass != 'bufferoverflow') {
               self::getAttackFeatures($reqmsg, $resmsg, $feat);
               self::writeToArff($feat);
            }
            else {
               $_SESSION['debug'] .= "bufferoverflow: skipped because ”.
                                     "left out of the experiment\n";
            }
         }
      }
      $_SESSION['debug'] .= "extractFeatures() done at ".
                              date('H:i:s') . "\n";

      $fh = fopen('arff/debug.txt', 'w') or
             die('Cannot open file: arff/debug.txt');
      $data =  $_SESSION['debug'];
      fwrite($fh, $data);
      fclose($fh);

      return $retval;

   } // end of method extractFeatures()

   private function writeToArff($feat) {
      $this->fhw = fopen($this->fnw, 'a') or
             die('Cannot open file:  ' . $this->fnw);
      $data = $feat->urllength . ",".
              $feat->urlencodingsextra . ",".
              $feat->doubleencodings . ",".
              $feat->unicode . ",".
              $feat->htmlentitycodes . ",".
              $feat->bas64encodings . ",".
              $feat->hexcodes . ",".
              $feat->maxparamlength . ",".
              $feat->maltags . ",".
              $feat->susppatterns . ",".
              $feat->malmethods . ",".
              $feat->maleventhandlers . ",".
              $feat->malobjects . ",".
              $feat->httpversion . ",".
              $feat->statuscode . ",".
              $feat->requestmethod . ",".
              $feat->tautologies . ",".
              $feat->sqlqueries . ",".
              $feat->sqlipatterns . ",".
              $feat->nonprintable . ",".
              $feat->pathtraversals . ",".
              $feat->iphost . ",".
              $feat->wronghost . ",".
              $feat->nocrlfend . ",".
              $feat->alertid . ",".
              $feat->attackclass . "\n".
              "% " . $feat->alertid . "\n";

      fwrite($this->fhw, $data);
      fclose($this->fhw);
   }

   private function readParsFile($f, &$reqmsg, &$resmsg) {
```

```
$retval = 0;
// each line in pars file starts with a keyword
// (except for the body, that can run over multiple lines)
$keyword;
$type;
$class;

//$_SESSION['debug'] .= "readParsFile() has started\n";
$this->fhr = fopen($this->basepars.$f, "r") or
       die("cannot open file: " . $f);
while (!feof($this->fhr)) {
   $line = fgets($this->fhr);
   //$_SESSION['debug'] .= "line: ".$line;
   $keyword = strstr($line, " ", true);
   switch ($keyword) {
      case "alertid:":
         $reqmsg->alertid = intval(strstr($line, " ", false));
         $resmsg->alertid = intval(strstr($line, " ", false));
         break;
      case "type:":
         $type = trim(strstr($line, " ", false));
         if ($type == REQTYPE) $reqmsg->type = $type;
         if ($type == RESTYPE) $resmsg->type = $type;
         break;
      case "requestline:":
         $reqmsg->requestline = trim(strstr($line, " ", false));
         $method = trim(strstr($reqmsg->requestline, " ", true));
         $httpversion = trim(strstr(trim(strstr($reqmsg->requestline,
                             " ", false)), " ", false));
         $reqmsg->urllength = strlen(trim(strstr(trim(strstr(
                     $reqmsg->requestline, " ", false)), " ", true)));
         $reqmsg->httpversion = trim(strstr($httpversion, "/", false),
                                        "/");
         if ($reqmsg->httpversion != "1.0" &&
              $reqmsg->httpversion != "1.1") {
            $reqmsg->httpversion = "other";
         }
         switch ($method) {
            case "GET":
            case "POST":
            case "OPTIONS":
            case "HEAD":
            case "PUT":
            case "DELETE":
            case "TRACE":
            case "CONNECT":
            case "PATCH":
               $reqmsg->method = $method;
               break;
            case "PROPFIND":
            case "PROPPATCH":
            case "MKCOL":
            case "COPY":
            case "MOVE":
            case "LOCK":
            case "UNLOCK":
               $reqmsg->method = "WebDAV";
               break;
            default:
               $reqmsg->method = "Invalid";
         }
         break;
      case "responseline:":
         $resmsg->responseline = trim(strstr($line, " ", false));
         //skip http version part
         $skipped = trim(strstr($resmsg->responseline, " ", false));
```

```
    if (preg_match("/\d+/", $skipped, $matched) == 1) {
       $resmsg->statuscode = $matched[0];
    }
    break;
 case "parameter:":
    if ($type == REQTYPE) {
       $reqmsg->paramvals[] = trim(strstr($line, "=", false),
         " =\n");
    }
    break;
 case "header:":
    if ($type == REQTYPE) {
       $reqmsg->headers[] = trim(strstr($line, " ", false));
       // also list header as input parameter so it can be
       // evaluated for attack features
       $reqmsg->paramvals[] = trim(strstr($line, " ", false));
    }
    if ($type == RESTYPE) $resmsg->headers[] = trim(strstr($line,
                                                  " ", false));
    if (strstr($line, "Referer") != FALSE || strstr($line,
                                           "Origin") != FALSE) {
    }
    break;
 case "attackclass:":
    $class = trim(strstr($line, " ", false));
    //$_SESSION['debug'] .= "attack class: ". $class . "\n";
    if ($class == 'xss') {
       $reqmsg->attackclass = 'xss';
       //$reqmsg->attackclass = 'other';
     }
    else if (strstr($class, "sql") != false) {
       $reqmsg->attackclass = 'sqli';
       //$reqmsg->attackclass = 'other';
    }
    else if ( (strstr($class, "path") != false) ||
                    (strstr($class, "pt")!=false) ) {
       $reqmsg->attackclass = 'pathtraversal';
       //$reqmsg->attackclass = 'other';
    }
    else if (strstr($class, "buffer") != false) {
       $reqmsg->attackclass = 'bufferoverflow';
       //$reqmsg->attackclass = 'other';
    }
    else if (strstr($class, "scan") != false) {
       $reqmsg->attackclass = 'scan';
       //$reqmsg->attackclass = 'other';
    }
    else if (strstr($class, "password") != false) {
       $reqmsg->attackclass = 'other';
    }
    else if (stristr($class, "rfi") != false) {
       $reqmsg->attackclass = 'other';
    }
    else if (strstr($class, "exploit") != false) {
       $reqmsg->attackclass = 'other';
    }
    else if (strstr($class, "probe") != false) {
       $reqmsg->attackclass = 'other';
    }
    else if (strstr($class, "unknown") != false) {
       $reqmsg->attackclass = 'other';
    }
    else if ( (strstr($class, "cmdi") != false) ||
                    (strstr($class, "ci")!=false) ) {
       $reqmsg->attackclass = 'other';
    }
```

```
            else {
               $reqmsg->attackclass = 'other';
            }
            $_SESSION['debug'] .= "attackclass=" . $reqmsg->attackclass .
                                                                 "\n";
            break;
         case "crending:":
            $reqmsg->nocrlfend = 'true';
            break;
         case "errdata:":
            if ($type == REQTYPE) $reqmsg->rfcerror = 'true';
            if ($type == RESTYPE) $resmsg->rfcerror = 'true';
            // no break because body still has to be handled
         case "body:":
            $fnb = trim(strstr($line, " ", false));
            if ($fhb = fopen($this->basepars.$fnb, "rb")) {
               $content = fread($fhb, filesize($this->basepars.$fnb));
               if ($type == REQTYPE) {
                  $reqmsg->requestbody = $content;
                  if ($reqmsg->method == 'POST') {
                     self::parsePostParams($reqmsg);
                  }
               }
               else if ($type == RESTYPE) {
                  $resmsg->responsebody = $content;
               }
               else {
                  $_SESSION['debug'] .= "ERROR in alert ".
                  $parsmsg->alertid. ": BODY FOUND BEFORE MSG TYPE” .
                                                      “IS DEFINED\n";
               }
               break;
            }
            else {
               $_SESSION['debug'] .= "Body file <".
                            $this->basepars.$fnb .">  cannot be opened\n";
            }
            break;
         case "eom":
            break;
         default:
            break;
      }
   }
   fclose($this->fhr);
   return $retval;
} // end of method readParsFile()


private function parsePostParams(&$reqmsg) {
   $paramname;
   $done = false;
   $value;
   $rest = $reqmsg->requestbody;
   while ($done == false) {
      $paramname = strstr($rest, "=", true);
      if ($paramname != false) {
         $paramname = trim($paramname);
         $rest = trim(strstr($rest, "=", false), "= \t\n\r\0\x0B");
         $value = strstr($rest, "&", true);
         if ($value != false) {
            $reqmsg->paramvals[] = trim($value);
            $rest = trim(strstr($rest, "&", false), "& \t\n\r\0\x0B");
         }
         else {
            // last parametervalue, parse and exit loop
            if (trim($rest) != "") {
               $reqmsg->paramvals[] = trim($rest);
            }
            $done = true;
         }
      }
      else {
         $value = strstr($rest, "&", true);
         if ($value != false) {
            // seems that we have a value without a param name,
            // store as value to be analyzed
            if (trim($value) != "") {
               $reqmsg->paramvals[] = trim($value);
            }
            $rest = trim(strstr($rest, "&", false), "= \t\n\r\0\x0B");
         }
         else {
            // no standard delimiters, store (rest of) request body
            // as a param value to be analysed, and exit loop
            if (trim($rest) != "") {
               $reqmsg->paramvals[] = trim($rest);
            }
            $done = true;
         }
      }
   }
}

private function getAttackFeatures(&$reqmsg, &$resmsg, &$feat) {
   $retval = 0;
   $input;
   $feat->urllength = $reqmsg->urllength;
   $feat->alertid = $reqmsg->alertid;
   $feat->attackclass = $reqmsg->attackclass;
   $feat->requestmethod = $reqmsg->method;
   $feat->nocrlfend = $reqmsg->nocrlfend;
   $feat->httpversion = $reqmsg->httpversion;
   $feat->statuscode = $resmsg->statuscode;
   $index = 0;
   foreach ($reqmsg->paramvals as $param) {
      $i=0;
      $param_length = strlen($param);
      if ($param_length > $feat->maxparamlength) {
         $feat->maxparamlength = $param_length;
         if ($feat->alertid==26493) $_SESSION['debug'] .=
                                   "LONGEST id26493 param: " . $param;
      }
      // normalizing parameter, more than one round if needed,
      // untill stable
      while ($param != $input && $i<10) {
         $input = $param;
         $feat->doubleencodings += self::findDoubleEncodings($param);
         $feat->urlencodingsextra +=
                         self::findExtraUrlEncodings($param, $reqmsg);
         $feat->unicode += self::findUnicode($param);
         $feat->htmlentitycodes += self::findHtmlEntityCodes($param);
         $feat->hexcodes += self::findHexcodedPrintable($param);
         $i++;
      }
      // remove tabs, carriage returns, line feeds, etc.
      $cleaned = self::cleanParameter($param);
      if (strlen($cleaned) > 0) {
         // look for html tags: <script>, <javascript>, <iframe>, <meta>,
         // <div>; and properties: href, http-equiv, lowsrc
         $feat->maltags += self::findMalTags($cleaned);
         // look for suspicious text patterns like: xss, <<, <"<<, root,
         // passwd, banking, hacker, evil
         $feat->susppatterns += self::findSuspXssPatterns($cleaned);
         // suspicious javascript methods: fromCharCode, alert, write,
          // getElementsByTagName, eval
         $feat->malmethods += self::findMalMethods($cleaned);
         $feat->maleventhandlers += self::findMalEventHandlers($cleaned);
         // DOM objects: Windows, Location, Document; DOM object
         // properties: Cookie, Referrer, InnerHtml
         $feat->malobjects += self::findMalObjects($cleaned);
         //find indications of tautologies like 'or 1=1
         $feat->tautologies += self::findSqliTautologies($cleaned);
         //find sql keywords and query structures
         $feat->sqlqueries += self::findSqlQueries($cleaned);
         //find sql patterns indicating malicious behaviour
         $feat->sqlipatterns += self::findSqliPatterns($cleaned);
         $feat->pathtraversals += self::findPathTraversals($cleaned,
                                                              $reqmsg);
      }
       //find non printable characters in normalized (decoded) parameter
      $feat->nonprintable += self::findNonPrintable($param);
      if (stripos(trim($param), "Host:") !== false){
         $ip_address = 'false';
         $feat->wronghost = self::checkHostHeader($param, $ip_address);
         $feat->iphost = $ip_address;
      }
      //$_SESSION['debug'] .= "#cycles done to normalize parameter: " .
      //                                                          $i . "\n";
      $index++;
   }

   return $retval;
} // end of method getAttackFeatures()

private function checkHostHeader($p, &$ip_address){
   $retval = 'false';
   $dummy = array();
   $count = preg_match_all('@\b(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9]
                            [0-9]?)\.){3}(?:25[0-5]|2[0-4][0-9]|[01]?
                            [0-9][0-9]?)\b@', $p, $dummy);
   if ($count == 1){ //test for ip_address
      $ip_address = 'true';
      //check if ip-address is correct (88.159.9.131)
      if (preg_match_all('@88\.159\.9\.131@', $p, $dummy) != 1){
         $retval = 'true';
      }
   }
   elseif (preg_match_all('@wiki\.seclab\.nl|www\.seclab\.nl@', $p,
                                                       $dummy) != 1){
      $retval = 'true';
   }
   return $retval;
}

private function findPathTraversals($c, $m) {
   //pre: string is cleaned from tabs, newlines etc.
   $retval = 0;
   $score = 0;
   $dummy = array();
   $matches = array();
   $slash_and_dotless = "";

   //calculate relative amount of pt patterns ../ or ..\ in param value
   $score = preg_match_all('@\.+/|\.+\\\\@', $c, $dummy);
   //$score = preg_match_all('@\.\./|\.\.\\\\@', $c, $dummy);
   //also take the special (undecoded) utf encodings into account
   $score += preg_match_all('@\.\.%u@i', $c, $matches);
   $temp = round(($score / (strlen($c) / 3)) * 100);
   $retval += $temp;
   //remove (back)slashes and '.' for further relative counting
   // of features
   $slash_and_dotless = preg_replace('@[/\.\\\\]@','',$c, -1, $count);
   //calculate percentage of patterns in parameter value relative to
   //slash_and_dotless param
   $score = preg_match_all('@C:\\\\|C:/|inetpub|wwwroot|/var/www@i', $c,
   $matches); //look for root folders
   if ($score > 0) {
      $temp = round(( strlen($matches[0][0]) /
                 (strlen($slash_and_dotless)+1) ) * 100);
      $retval += $temp;
   }
   if ($score > 0) $_SESSION['debug'] .= "pt's s2 relative score: ".
                 $temp . " from (1st match): " . $matches[0][0] . "\n";

   //also take absolute paths for linux systems
   $score = preg_match_all('@/etc|etc/|/etc/paswd|/etc/shadow|
              /user|user/|/s?bin|s?bin/|/usr|usr/|/lib|lib/|/home|home/|
              /var|var/|/mnt|mnt/|/dev|dev/@i', $c, $matches);
   if ($score > 0) {
     //plus 1 to compensate for a slash in the pattern
      $temp = round(( strlen($matches[0][0]) /
                         (strlen($slash_and_dotless)+1) ) * 100);
      $retval += $temp;
   }

   //look for just a filename in root: root folder plus some indication
   //of following file
   $score = preg_match_all('@^(/|C:\\\\).{1,20}[/\\\\\.]@i', $c,
                                $matches); //look at start of param only
   if ($score > 0) {
     // patternlenght relative to whole parameter length
      $temp = round( (strlen($matches[0][0]) / strlen($c) ) * 100);
      $retval += $temp;
   }
   return $retval;
} // end of method findPathTraversals()


private function findNonPrintable($p) {
   // pre: string is not cleaned from any special characters,
   // e.g. vertical tabs etc.
   $retval = 0;
   $dummy = array();

   $retval += preg_match_all('/[\x00-\x08\x0B\x0C\x0E-\x1F\x7F-\xFF]/u',
                                                         $p, $dummy);
   return $retval;
} // end of method findNonPrintable()


private function findSqliPatterns($c) {
   // pre: string is cleaned from tabs, newlines etc.
   $retval = 0;
   $matches = array();

   $r = preg_match_all('@(password|user.?name|admin|null).*(#|--)@', $c,
                                                               $matches);
   $retval += $r;
   // find /*/ or /*****/ and /* comment */, non-greedy (.+? vs .+) to
   // find multiple patterns iso largest pattern
   $r = preg_match_all('@(/\*+?/|/\*.+?\*/)@', $c, $matches);
   //if ($r > 0) $_SESSION['debug'] .= "1st pattern s2: ".
   //$matches[0][0]. "\n";
   $retval += $r;
   $r = preg_match_all('@CONCAT\(|waitfor delay|pg_|sp_|substring\(|
          user\(\)|row_count\(|schema\(|benchmark\(@i', $c, $matches);
   $retval += $r;

   $d = preg_replace('/ /','',$c, -1, $count); //remove all spaces
   if (strlen($d) < 16) { //check for short blind sqli such as ';
      //count number of sqli special chars
      $r = preg_match_all('@[\'=#\;]|(/\*)|(--)@', $d, $matches);
      // if over 40% of parameterlength special chars then count number
      // of special chars
      if ( ($r / strlen($d)) > 0.4) {
         //$_SESSION['debug'] .= "1st pattern s4: ". $matches[0][0].
          //"\n";
         $retval += $r;
      }
   }
   return $retval;
} // end of method findSqliPatterns()


private function findSqlQueries($c) {
   //pre: string is cleaned from tabs, newlines etc.
   $retval = 0;
   $matches = array();

   $r = preg_match_all('@UNION|SELECT +COUNT|DISTINCT|SELECT.+FROM|
          SELECT \*|ROWNUM|INSERT +INTO|DELETE +FROM|
          TABLESPACE|\@\@VERSION|table_name|column_name|
          granted_role|(sys\..*(user|table|column)|
          SELECT.*\(.*\)|SELECT.+;|CREATE.+USER)@i', $c, $matches);
   $retval += $r;

   //keywords by themselves give too high FP-rate, but ended with comment
   //sign much lower FP, and probably still a low FN
   $r = preg_match_all('@(CREATE |EXEC[ \(]|GRANT |REVOKE |DROP |
          COUNT ?\(|ALTER +TABLE|RENAME +(TABLE|INDEX)|GROUP BY|
          VALUES\(|HAVING .{0,5}[=><]|TRUNCATE).+(#|--|;)@i',
          $c, $matches);
   $retval += $r;
   $r = preg_match_all('@, ?null ?,@i', $c, $matches);
   $retval += $r;
   return $retval;
} // end of method findSqlQueries()


private function findSqliTautologies($c) {
   // pre: string is cleaned from tabs, newlines etc.
   $retval = 0;
   $matches = array();

   $r = preg_match_all('@(\') *(OR|AND|#|--!>|;)@i', $c, $matches);
   //e.g. 'OR 1=1;  e.g. '#, exclude hmtml comment end -->
   if ($r > 0) $_SESSION['debug'] .= "1st taut s1: ". $matches[0][0].
          "\n";
   $retval += $r;

   $r = preg_match_all('@; *(#|--|/\*)@', $c, $matches); //e.g. ;#
   if ($r > 0) $_SESSION['debug'] .= "1st taut s2: ". $matches[0][0].
          "\n";
   $retval += $r;
   $r = preg_match_all('@([0-9]{1,5}) ?[<|>]?= ?\1@', $c, $matches);
   if ($r > 0) $_SESSION['debug'] .= "1st taut s3: ". $matches[0][0].
          "\n";
   $retval += $r;
   $r = preg_match_all('@([0-9a-zA-Z]{1,5})\' ?[<|>]?= ?\'\1@', $c,
           $matches); //e.g. a'='a, as' <= 'as
   if ($r > 0) $_SESSION['debug'] .= "1st taut s3a: ". $matches[0][0].
           "\n";
   $retval += $r;
   $r = preg_match_all('@[ \'\)] *(OR|AND) *((pg_)?sleep\(|
           waitfor|benchmark\(|length\()@i', $c, $matches);
   $retval += $r;
   $r = preg_match_all('@[ \'\)] *(OR|AND)(.{2,10}[=><])@i', $c,
           $matches); //e.g. 1) OR 2>1
   if ($r > 0) $_SESSION['debug'] .= "1st taut s4: ". $matches[0][0].
           "\n";
   $retval += $r;
   //comment -- at the end of a parameter value
   $r = preg_match_all('@--$@', $c, $matches);
   if ($r > 0) $_SESSION['debug'] .= "1st taut s5: ". $matches[0][0].
           "\n";
   $retval += $r;
   //$_SESSION['debug'] .= "number tautologies found: ". $retval. "\n";
   return $retval;
} // end of method findSqliTautologies()


private function findMalObjects($c) {
   //pre: string is cleaned from tabs, newlines etc.
   $retval = 0;
   $dummy = array();
   // look for DOM objects: Windows, Location, Document;
   // DOM object properties: Cookie, Referrer, InnerHtml
   $retval += preg_match_all('/Windows|Location|Document|Cookie|Referrer|
           InnerHtml/i', $c, $dummy);
   return $retval;
} // end of method findMalObjects()


private function findMalEventHandlers($c) {
   //pre: string is cleaned from tabs, newlines etc.
   $retval = 0;
   $dummy = array();
   //look for suspicious javascript eventhandlers:
   //onmouseover, onclick, onload
   $retval += preg_match_all('/onmouseover|onclick|onload/i', $c,
           $dummy);
   return $retval;
} // end of method findMalEventHandlers()


private function findMalMethods($c) {
   //pre: string is cleaned from tabs, newlines etc.
   $retval = 0;
   $dummy = array();
   //look for suspicious javascript methods: fromCharCode, alert, write,
   //getElementsByTagName, eval
   $retval += preg_match_all('/fromCharCode|alert|write\(|
           getElementsByTagName|eval/i', $c, $dummy);
   return $retval;
} // end of method findMalMethods()


private function findSuspXssPatterns($c) {
   //pre: string is cleaned from tabs, newlines etc.
   $retval = 0;
   $dummy = array();
   $c = preg_replace('/ /','',$c, -1, $count); //remove all spaces
   //look for suspicious text patterns like: xss, <<, <"<<, banking,
   //hacker, evil
   $retval += preg_match_all('/xss|<<|<.<|<..<|bank|hack|evil/i', $c,
          $dummy);
   $retval += preg_match_all('/<\!--|-->/', $c, $dummy);
    return $retval;
} // end of method findSuspXssPatterns()


private function findMalTags($c) {
   //pre: string is cleaned from tabs, newlines etc.
   $retval = 0;
   $dummy = array();
   // clean up some more
   $c = preg_replace('/[ ;.,-\/\#\'\"\*\\\?]/', '', $c);
   // look for html tags: <script>, <javascript>, <iframe>, <meta>, <div>
   $retval += preg_match_all('/script|javascript|iframe|meta|<div/i', $c,
          $dummy);  // just 'div' gives false positives, use '<div'
   //look for html tag properties: href, http-equiv, lowsrc
   $retval += preg_match_all('/href|http-equiv|lowsrc/i', $c, $dummy);

   return $retval;
} // end of method findMalTags()


private function cleanParameter($p) {
   $cleanstring = preg_replace('/[\n\r\t\v\e\f\a\b]/','',$p, -1, $count);
   return $cleanstring;
} // end of method cleanParameter()

private function findHexcodedPrintable(&$p) {
   $retval = 0;
   $out = $p;
   $count = 0;
   // replace hex encoded characters in call-back function
   $out = preg_replace_callback('/([0\\\\])(x)([2-7][0-f])/',
                 "Extractor::HexDecode",$out, $limit = -1, $count);
   if ($out != $p) {
      $retval = strlen($p) - strlen($out);
      $p = $out;
   }

   return $retval;
} // end of method findHexcodedPrintable()


private function findHtmlEntityCodes(&$p) {
   $retval = 0;
   $out = $p;
   $temp = $p;
   $count = 0;

   do {
      $temp = $out;
      $out = html_entity_decode($temp, ENT_QUOTES);
      // remove trailing zero's for decimal html entity codes;
      // I've seen attacks and is not decoded otherwise
      $out = preg_replace('/&#[0]+/','&#',$out, $limit = -1, &$count);
      $out = preg_replace('/(&#)([0-9]+)(;?)/','$1$2;',$out, $limit = -1,
           &$count);  //add ';' if not present;
      // html encoded hex chars are not replaced by html_entity_decode so
      // we'll do this ourselves.
      $out = html_entity_decode($out, ENT_QUOTES);    //try again
      // remove trailing zero's for hex html entity codes
      $out = preg_replace('/(&#x)([0]+)([0-9A-Fa-f]{2})/','$1$3',$out);
      // replace htmlhexencoded characters in call-back function
      $out = preg_replace_callback('/(&#)(x)([0-9A-Fa-f]{2})(;?)/',
          "Extractor::HexDecode",$out, $limit = -1, $count);
   } while ($out != $temp);
```

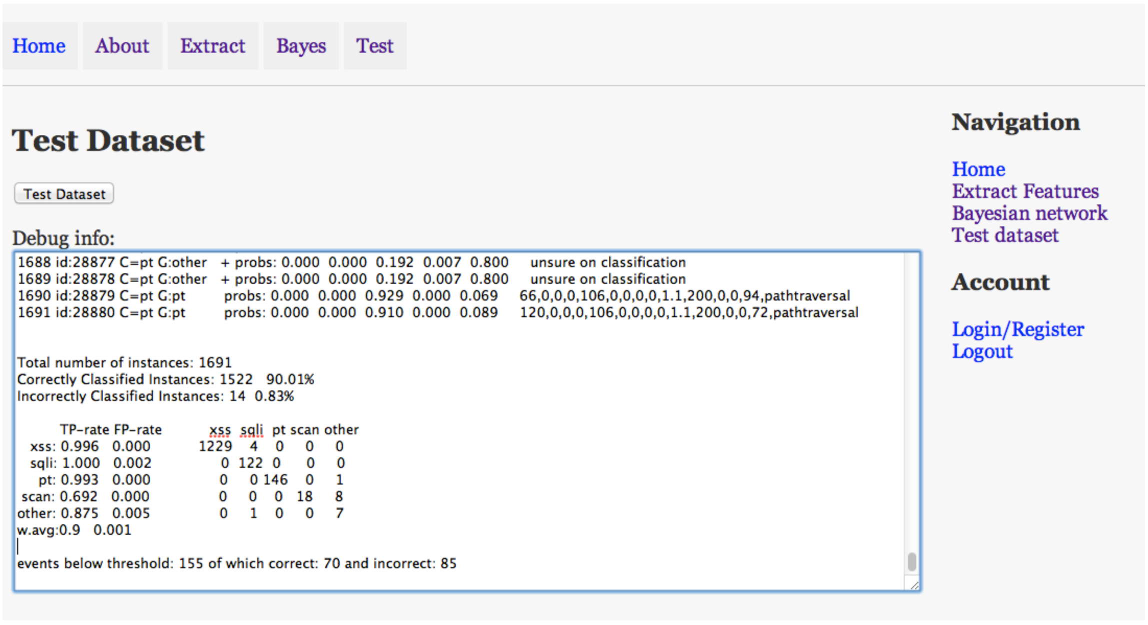
```
      if ($out != $p) {
         //take difference in length as count for html entity codes. Note
         //that this is different (more) then the #entitycodes replaced.
         $retval = strlen($p) - strlen($out);
         //$_SESSION['debug'] .= "BEFORE html_entity_decode: ". $p . "\n";
         //$_SESSION['debug'] .= "AFTER  html_entity_decode: ". $out . "\n";
         $p = $out;
      }
      return $retval;
   } // end of method findHtmlEntityCodes()

   function HexDecode($s) {
      //used as call back function in pregreplace to replace hexencoded
      //characters
      $r = chr(hexdec($s[3]));
      return $r;
   } // end of callback function HexDecode()

   private function findUnicode(&$p) {
      $retval = 0;
      $out = utf8_decode($p);
      if ($out != $p) {
         $retval = strlen($p) - strlen($out);
         $p = $out;
      }
      return $retval;
   } // end of method findUnicode()

   private function findExtraUrlEncodings(&$p, $reqmsg) {
      $retval = 0;
      $matches = array();
      $found = preg_match_all("/[%][0-9A-Fa-f]{2}/", $p, $matches);
      if ( $found > 0) {
         foreach ($matches[0] as $enc) {
            $enc = trim($enc, "%");
            $enc = intval($enc, 16);

            if (strpbrk(chr($enc), '><)(\'"/') != false) {
               $retval++;
            }
         }
         $p = urldecode($p);
      }
      return $retval;
   } // end of method findExtraUrlEncodings()


   private function findDoubleEncodings(&$p) {
      $retval = 0;
      $count = 0;
      $p = str_replace("%25", "%", $p, $count);
      while ($count > 0) {
         $retval+=$count;
         $count = 0;
         $p = str_replace("%25", "%", $p, $count);
      }
      return $retval;
   } // end of method findDoubleEncodings


   private function createArff() {
      $this->fnw = $this->basearff . date('YmdHis') . preg_replace('@/@',
              '', $this->basepars) . ".arff";
      $this->fhw = fopen($this->fnw, 'w') or die('Cannot open file:  ' .
              $this->fnw);
      $data =  "% Title: attack features dataset ". $basepars. "\n".
```

```
               "%\n".
               "% Creator: Casper Schellekens\n".
               "% Creation Date: " . date('D F j, Y H:i:s') . "\n".
               "%\n".
               "@RELATION all_attack_classes\n".
               "\n".
               "@ATTRIBUTE urllength numeric\n".
               "@ATTRIBUTE urlencodingsextra numeric\n".
               "@ATTRIBUTE doubleencodings numeric\n".
               "@ATTRIBUTE unicode numeric\n".
               "@ATTRIBUTE htmlentitycodes numeric\n".
               "@ATTRIBUTE base64encodings numeric\n".
               "@ATTRIBUTE hexcodes numeric\n".
               "@ATTRIBUTE maxparamlength numeric\n".
               "@ATTRIBUTE maltags numeric\n".
               "@ATTRIBUTE susppatterns numeric\n".
               "@ATTRIBUTE malmethods numeric\n".
               "@ATTRIBUTE maleventhandlers numeric\n".
               "@ATTRIBUTE malobjects numeric\n".
               "@ATTRIBUTE httpversion {1.0, 1.1, other}\n".
               "@ATTRIBUTE statuscode numeric\n".
               "@ATTRIBUTE requestmethod {GET, POST, OPTIONS, HEAD, PUT,
                     DELETE, TRACE, CONNECT, PATCH, WebDAV, Invalid}\n".
               "@ATTRIBUTE tautologies numeric\n".
               "@ATTRIBUTE sqlqueries numeric\n".
               "@ATTRIBUTE sqlipatterns numeric\n".
               "@ATTRIBUTE nonprintable numeric\n".
               "@ATTRIBUTE pathtraversals numeric\n".
               "@ATTRIBUTE iphost {true, false}\n".
               "@ATTRIBUTE wronghost {true, false}\n".
               "@ATTRIBUTE nocrlfend {true, false}\n".
               "@ATTRIBUTE attackclass {xss, sqli, pathtraversal, scan,
                     other}\n".
               "\n".
               "@DATA\n";

      fwrite($this->fhw, $data);
      fclose($this->fhw);

   } // end of method createArff()
} // end of class Extractor

?>
```

Appendix C. (partial) ARFF file data set

```
% Title: attack features dataset
%
% Creator: Casper Schellekens
% Creation Date: Fri October 25, 2013 13:04:34
%
@RELATION all_attack_classes

@ATTRIBUTE urllength numeric
@ATTRIBUTE urlencodingsextra numeric
@ATTRIBUTE doubleencodings numeric
@ATTRIBUTE unicode numeric
@ATTRIBUTE htmlentitycodes numeric
@ATTRIBUTE base64encodings numeric
@ATTRIBUTE hexcodes numeric
@ATTRIBUTE maxparamlength numeric
@ATTRIBUTE maltags numeric
@ATTRIBUTE susppatterns numeric
@ATTRIBUTE malmethods numeric
@ATTRIBUTE maleventhandlers numeric
@ATTRIBUTE malobjects numeric
@ATTRIBUTE httpversion {1.0, 1.1, other}
@ATTRIBUTE statuscode numeric
@ATTRIBUTE requestmethod {GET, POST, OPTIONS, HEAD, PUT, DELETE, TRACE, CONNECT, PATCH,
WebDAV, Invalid}
@ATTRIBUTE tautologies numeric
@ATTRIBUTE sqlqueries numeric
@ATTRIBUTE sqlipatterns numeric
@ATTRIBUTE nonprintable numeric
@ATTRIBUTE pathtraversals numeric
@ATTRIBUTE iphost {true, false}
@ATTRIBUTE wronghost {true, false}
@ATTRIBUTE nocrlfend {true, false}
@ATTRIBUTE attackclass {xss, sqli, pathtraversal, scan, other}


@DATA
86,4,0,0,0,0,0,93,0,0,0,0,1,1.1,200,POST,0,0,0,0,0,false,false,false,sqli
87,121,11,0,0,0,0,72,22,4,0,0,70,1.1,200,POST,0,0,0,0,0,false,false,false,xss
74,1,6,0,0,0,0,93,0,0,0,0,0,1.1,200,GET,0,0,0,0,70,false,false,false,pathtraversal
224,0,0,0,52,0,0,184,2,2,0,0,0,1.1,200,GET,0,0,0,0,0,false,false,false,xss
90,4,0,0,0,0,0,93,0,0,0,0,1,1.1,200,POST,3,0,5,0,0,false,false,false,sqli
12,0,0,0,0,0,0,0,0,0,0,0,0,1.0,404,GET,0,0,0,0,0,false,false,false,scan
97,0,0,0,0,0,0,397,0,0,0,0,0,1.1,404,POST,4,0,0,0,0,false,false,false,other
107,0,0,0,0,0,0,397,0,0,0,0,0,1.1,200,POST,4,0,0,0,0,false,false,false,other
1,0,0,0,0,0,0,0,0,0,0,0,0,1.0,301,GET,0,0,0,0,0,false,false,false,scan
1,0,0,0,0,0,0,11,0,0,0,0,0,1.1,400,GET,0,0,0,0,0,false,false,false,scan
35,0,0,0,0,0,0,84,0,0,0,0,0,1.1,200,GET,0,0,0,0,0,false,false,false,scan
27,0,0,0,0,0,0,18,0,0,0,0,0,1.1,404,POST,0,0,0,0,0,true,false,true,scan
27,0,0,0,0,0,0,119,0,0,0,0,15,1.1,200,GET,0,0,0,0,135,false,false,false,scan
67,2,0,0,0,0,0,121,4,0,2,0,5,1.1,200,GET,0,0,2,0,0,false,false,false,xss
67,2,0,0,0,0,0,222,4,0,2,0,5,1.1,200,GET,0,0,2,0,0,false,false,false,xss
67,12,0,0,0,0,0,99,7,0,3,0,10,1.1,304,GET,0,0,2,0,0,false,false,false,xss
89,17,0,0,0,0,0,222,8,0,4,0,8,1.1,200,GET,1,0,2,0,0,false,false,false,xss
89,40,0,0,0,0,0,120,14,0,7,0,16,1.1,304,GET,2,0,2,0,0,false,false,false,xss
80,21,0,0,0,0,0,121,10,0,5,0,9,1.1,200,GET,1,0,2,0,0,false,false,false,xss
88,30,0,0,0,0,0,222,14,0,7,0,11,1.1,200,GET,1,0,2,0,0,false,false,false,xss
92,69,0,0,0,0,0,120,26,0,13,0,22,1.1,304,GET,2,0,2,0,0,false,false,false,xss
69,0,0,0,0,0,0,106,0,0,0,0,0,1.1,200,GET,0,0,0,1,109,false,false,false,pathtraversal
73,0,0,0,0,0,0,106,0,0,0,0,0,1.1,200,GET,0,0,0,1,96,false,false,false,pathtraversal
74,0,0,0,0,0,0,106,0,0,0,0,0,1.1,200,GET,0,0,0,1,92,false,false,false,pathtraversal
```

Appendix D. Bayesian Network source code



```
1688 id:28877 C=pt G:other  + probs: 0.000  0.000  0.192  0.007  0.800    unsure on classification
1689 id:28878 C=pt G:other  + probs: 0.000  0.000  0.192  0.007  0.800    unsure on classification
1690 id:28879 C=pt G:pt      probs: 0.000  0.000  0.929  0.000  0.069    66,0,0,0,106,0,0,0,0,1.1,200,0,0,94,pathtraversal
1691 id:28880 C=pt G:pt      probs: 0.000  0.000  0.910  0.000  0.089    120,0,0,0,106,0,0,0,0,1.1,200,0,0,72,pathtraversal


Total number of instances: 1691
Correctly Classified Instances: 1522   90.01%
Incorrectly Classified Instances: 14  0.83%

        TP-rate FP-rate        xss  sqli  pt scan other
   xss: 0.996   0.000         1229   4    0    0    0
  sqli: 1.000   0.002            0  122   0    0    0
    pt: 0.993   0.000            0    0 146    0    1
  scan: 0.692   0.000            0    0   0   18    8
other: 0.875   0.005            0    1   0    0    7
w.avg:0.9   0.001

events below threshold: 155 of which correct: 70 and incorrect: 85
```

Figure. User interface of the Test Dataset tool

Class_testdataset.php:

```php
<?php
session_start();

class TestDataSet {
   // property declarations
   private $base = 'arff/';
   private $fnr;        // file names for bayes network model file to read
   private $fhr;        // file handle
   private $filename = '20131025_ds2_set14.arff';
   private $BN;

   //constructor
   function __construct($bn) {
      $this->BN = $bn;
      //test with realistic population shares
      $this->BN[0]->popshare[1]= "0.14";
      $this->BN[0]->popshare[2]= "0.27";
      $this->BN[0]->popshare[3]= "0.36";
      $this->BN[0]->popshare[4]= "0.115";
      $this->BN[0]->popshare[5]= "0.115";
   }
   // method implementations
   public function Test() {
      //set_time_limit(0);
      $values = array();
      $matches = array();
      $correct = array();
      $incorrect = array();
      $correct[total] = 0;
      $correct[xss] = 0;
      $correct[sqli] = 0;
      $correct[pt] = 0;
      $correct[scan] = 0;
      $correct[other] = 0;
      $incorrect[total] = 0;
```

```
$incorrect[xss] = 0;
$incorrect[sqli] = 0;
$incorrect[pt] = 0;
$incorrect[scan] = 0;
$incorrect[other] = 0;
$threshold = 0.9; // lucky guess threshold
$unsure[correct] = 0;
$unsure[incorrect] = 0;

for ($i=1;$i<=5;$i++) $guesses['xss'][$i] = 0;
for ($i=1;$i<=5;$i++) $guesses['sqli'][$i] = 0;
for ($i=1;$i<=5;$i++) $guesses['pt'][$i] = 0;
for ($i=1;$i<=5;$i++) $guesses['scan'][$i] = 0;
for ($i=1;$i<=5;$i++) $guesses['other'][$i] = 0;

date_default_timezone_set('Europe/Amsterdam');
$_SESSION['debug3'] .= "Test() started at ". date('H:i:s') . "\n";
$_SESSION['debug3'] .= "php version: " . phpversion() . "\n";

$this->fhr = fopen($this->base.$this->filename, "r") or
       die("cannot open file: " . $f);
while (!feof($this->fhr)) {
   $line = fgets($this->fhr);
   if (preg_match('$^@DATA$', $line) == 1){
      //read data and calculate best guess
      $inst = 0;
      $line = fgets($this->fhr);
      while (!feof($this->fhr)) {
         $inst++;
         $line2 = fgets($this->fhr); //get comment line with alert id
         preg_match('@^% (.+)@', $line2, $matches2);
         $alertid = $matches2[1];
         if (preg_match('$^%$', $line) == 0 && !feof($this->fhr)) {
            preg_match_all('@(.+?),@', $line, $matches,
                     PREG_SET_ORDER);
            $i=1;
            foreach ($matches as $m) {
               $values[$i] = $m[1];
               $i++;
            }
            preg_match_all('@(.+?)(,|$)@', $line, $matches,
                     PREG_SET_ORDER);
            foreach ($matches as $m) {
               $class = $m[1];
               $i++;
            }
            if ($class == 'pathtraversal') $class = 'pt';
            for ($s=0;$s<4-strlen($inst);$s++)
                     $_SESSION['debug3'] .= " ";
            $_SESSION['debug3'] .= $inst. " id:". $alertid. " C=".
                     $class;
            $g = self::CalculateProbabilities($values, $class);
            $guess = $g[attackclass];
            if (round($g[certainty],2) <= $threshold) {
               if ($class == $guess) $unsure[correct]++;
               else $unsure[incorrect]++;
               $_SESSION['debug3'] .=
                           "     unsure on classification\n";
            }
            else {
               $_SESSION['debug3'] .= "     ". $line;
               if ($class == $guess) $correct[total]++;
               else $incorrect[total]++;
               if ($class == 'xss') {
                  if ($class == $guess) $correct['xss']++;
                  else $incorrect['xss']++;
                  switch ($guess) {
                  case "xss":
                     $guesses['xss'][1]++;
                     break;
                  case "sqli":
                     $guesses['xss'][2]++;
                     $fp['sqli']++;
                     break;
                  case "pt":
                     $guesses['xss'][3]++;
                     $fp['pt']++;
                     break;
                  case "scan":
                     $guesses['xss'][4]++;
                     $fp['scan']++;
                     break;
                  case "other":
                     $guesses['xss'][5]++;
                     $fp['other']++;
                     break;
                  }
               }
               if ($class == 'sqli') {
                  if ($class == $guess) $correct['sqli']++;
                  else $incorrect['sqli']++;
                  switch ($guess) {
                  case "xss":
                     $guesses['sqli'][1]++;
                     $fp['xss']++;
                     break;
                  case "sqli":
                     $guesses['sqli'][2]++;
                     break;
                  case "pt":
                     $guesses['sqli'][3]++;
                     $fp['pt']++;
                     break;
                  case "scan":
                     $guesses['sqli'][4]++;
                     $fp['scan']++;
                     break;
                  case "other":
                     $guesses['sqli'][5]++;
                     $fp['other']++;
                     break;
                  }
               }
               if ($class == 'pt') {
                  if ($class == $guess) $correct['pt']++;
                  else $incorrect['pt']++;
                  switch ($guess) {
                  case "xss":
                     $guesses['pt'][1]++;
                     $fp['xss']++;
                     break;
                  case "sqli":
                     $guesses['pt'][2]++;
                     $fp['sqli']++;
                     break;
                  case "pt":
                     $guesses['pt'][3]++;
                     break;
                  case "scan":
                     $guesses['pt'][4]++;
                     $fp['scan']++;
                     break;
```

```
                  case "other":
                     $guesses['pt'][5]++;
                     $fp['other']++;
                     break;
                  }
               }
               if ($class == 'scan') {
                  if ($class == $guess) $correct['scan']++;
                  else $incorrect['scan']++;
                  switch ($guess) {
                     case "xss":
                     $guesses['scan'][1]++;
                     $fp['xss']++;
                     break;
                  case "sqli":
                     $guesses['scan'][2]++;
                     $fp['sqli']++;
                     break;
                  case "pt":
                     $guesses['scan'][3]++;
                     $fp['pt']++;
                     break;
                  case "scan":
                     $guesses['scan'][4]++;
                     break;
                  case "other":
                     $guesses['scan'][5]++;
                     $fp['other']++;
                     break;
                  }
               }
               if ($class == 'other') {
                  if ($class == $guess) $correct['other']++;
                  else $incorrect['other']++;
                  switch ($guess) {
                  case "xss":
                     $guesses['other'][1]++;
                     $fp['xss']++;
                     break;
                  case "sqli":
                     $guesses['other'][2]++;
                     $fp['sqli']++;
                     break;
                  case "pt":
                     $guesses['other'][3]++;
                     $fp['pt']++;
                     break;
                  case "scan":
                     $guesses['other'][4]++;
                     $fp['scan']++;
                     break;
                  case "other":
                     $guesses['other'][5]++;
                     break;
                  }
               }
            }
            $line = fgets($this->fhr);
         }
      }
   }
}
$_SESSION['debug3'] .= "\n\nTotal number of instances: " . $inst .
       "\n";
$_SESSION['debug3'] .= "Correctly Classified Instances: ".
       $correct[total] . "   ".
```

```
               round($correct[total]/$inst*100, 2) ."%\n";
        $_SESSION['debug3'] .= "Incorrectly Classified Instances: ".
               $incorrect[total] . "  ".
               round($incorrect[total]/$inst*100, 2) ."%\n";
        $_SESSION['debug3'] .= "\n";
        $_SESSION['debug3'] .= "          TP-rate FP-rate       " .
               "xss sqli pt scan other\n";
        $_SESSION['debug3'] .= "   xss: " .
               bcdiv($correct['xss'],($correct['xss']+$incorrect['xss']),3);
        $_SESSION['debug3'] .= "   " . bcdiv($fp['xss'],$inst,3). "       ";
        for ($i=1;$i<=5;$i++) $_SESSION['debug3'] .= "    " .
               $guesses['xss'][$i];
        $_SESSION['debug3'] .= "\n";
        $_SESSION['debug3'] .= "   sqli: " .
               bcdiv($correct['sqli'],
               ($correct['sqli']+$incorrect['sqli']),3);
        $_SESSION['debug3'] .= "   " . bcdiv($fp['sqli'],$inst,3). "       ";
        for ($i=1;$i<=5;$i++) $_SESSION['debug3'] .= "    ".
               $guesses['sqli'][$i];
        $_SESSION['debug3'] .= "\n";
        $_SESSION['debug3'] .= "     pt: " .
               bcdiv($correct['pt'],($correct['pt']+$incorrect['pt']),3);
        $_SESSION['debug3'] .= "   " . bcdiv($fp['pt'],$inst,3). "       ";
        for ($i=1;$i<=5;$i++) $_SESSION['debug3'] .= "    ".
               $guesses['pt'][$i];
        $_SESSION['debug3'] .= "\n";
        $_SESSION['debug3'] .= " scan: " .
               bcdiv($correct['scan'],
               ($correct['scan']+$incorrect['scan']),3);
        $_SESSION['debug3'] .= "   " . bcdiv($fp['scan'],$inst,3). "       ";
        for ($i=1;$i<=5;$i++) $_SESSION['debug3'] .= "    ".
               $guesses['scan'][$i];
        $_SESSION['debug3'] .= "\n";
        $_SESSION['debug3'] .= "other: " .
               bcdiv($correct['other'],
               ($correct['other']+$incorrect['other']),3);
        $_SESSION['debug3'] .= "   " . bcdiv($fp['other'],$inst,3). "       ";
        for ($i=1;$i<=5;$i++) $_SESSION['debug3'] .= "    ".
               $guesses['other'][$i];
        $_SESSION['debug3'] .= "\n";
        $wfp = bcmul($fp['xss'],$this->BN[0]->popshare[1], 10);
        $wfp += bcmul($fp['sqli'],$this->BN[0]->popshare[2], 10);
        $wfp += bcmul($fp['pt'],$this->BN[0]->popshare[3], 10);
        $wfp += bcmul($fp['scan'],$this->BN[0]->popshare[4], 10);
        $wfp += bcmul($fp['other'],$this->BN[0]->popshare[5], 10);
        $wfp = bcdiv($wfp,$inst,3);
        $_SESSION['debug3'] .= "w.avg:" . round($correct[total]/$inst, 3) .
        "   " . $wfp . "\n";
        $unsure[total] = $unsure[correct] + $unsure[incorrect];
        $_SESSION['debug3'] .= "events below threshold: ". $unsure[total];
        $_SESSION['debug3'] .= " of which correct: " . $unsure[correct].
               " and incorrect: " . $unsure[incorrect];
        $_SESSION['debug3'] .= "\n";
        fclose($this->fhr);
        return $retval;
    } // end of method readParsFile()

    private function CalculateProbabilities($values, $C){
        //$type;  //type of value integer or string
        $guess = array();
        $pexss = self::CalcECXss($values, $C);
        $pesqli = self::CalcECSqli($values, $C);
        $denom = bcadd($pexss, $pesqli, 50);
        $pept = self::CalcECPt($values, $C);
        $denom = bcadd($denom, $pept, 50);
        $pescan = self::CalcECScan($values, $C);
```

```
      $denom = bcadd($denom, $pescan, 50);
      $peother = self::CalcECOther($values, $C);
      $denom = bcadd($denom, $peother, 50);
      if (floatval($denom) <= 0) {
         $_SESSION['debug3'] .=
                     "DENOMINATOR ZERO?! values of parts denominator:\n";
         $_SESSION['debug3'] .=
                     $pexss."\n".$pesqli."\n".$pept."\n".$pescan."\n".
                     $peother."\n";
      }
      else {
         $pxsse = bcdiv($pexss, $denom, 3);
         $psqlie = bcdiv($pesqli, $denom, 3);
         $ppte = bcdiv($pept, $denom, 3);
         $pscane = bcdiv($pescan, $denom, 3);
         $pothere = bcdiv($peother, $denom, 3);
         $guess[attackclass] = 'xss';
         $highest = $pxsse;
         if ($psqlie > $highest){
            $highest = $psqlie;
            $guess[attackclass] = 'sqli';
         }
         if ($ppte > $highest){
            $highest = $ppte;
            $guess[attackclass] = 'pt';
         }
         if ($pscane > $highest){
            $highest = $pscane;
            $guess[attackclass] = 'scan';
         }
         if ($pothere > $highest){
            $highest = $pothere;
            $guess[attackclass] = 'other';
         }
         $guess[certainty] = $highest;
         $_SESSION['debug3'] .= " G:". $guess[attackclass];
         for ($s=0;$s<10-strlen($C.$guess[attackclass]);$s++)
             $_SESSION['debug3'] .= " ";
         if ($guess[attackclass] != $C) $_SESSION['debug3'] .= "+";
         else $_SESSION['debug3'] .= "  ";
         $_SESSION['debug3'] .= " probs: ". $pxsse . "  " . $psqlie . "  " .
                     $ppte . "  " . $pscane . "  " . $pothere;

      }
      return $guess;
   }

   private function CalcECXss($values, $class) {
   // calculate P(event|xss)*P(xss)
      $ni = 0;
      $p; //probability
      $p_fs_xss = '1';
      foreach ($values as $v) {
         $ni++;
         if ($v === '1.0' || $v === '1.1') ;
         else if ($v === '1' || $v === '0') $v = intval($v);
         else if (intval($v) > 0) $v = intval($v);
         if (is_int($v)) {
            if ($this->BN[$ni]->rangestart[1] == 'all')
              $p = $this->BN[$ni]->xss[1];
            else {
               for ($c=1;$c<=$this->BN[$ni]->columns; $c++){
                  if ($this->BN[$ni]->rangestart[$c] == 'inf') {
                     if ($v <= $this->BN[$ni]->rangeend[$c]) {
                        $p = $this->BN[$ni]->xss[$c];
                     }
                  }
                  else if ($this->BN[$ni]->rangeend[$c] == 'inf') {
                     if ($v > $this->BN[$ni]->rangestart[$c]) {
                        $p = $this->BN[$ni]->xss[$c];
                     }
                  }
                  else {
                     if (($v <= $this->BN[$ni]->rangeend[$c]) &&
                           ($v > $this->BN[$ni]->rangestart[$c])) {
                        $p = $this->BN[$ni]->xss[$c];
                     }
                  }
               }
            }
         }
         else {
         // value is not numeric so must fall in named category tablecolumn
         // in PDT
            if ($this->BN[$ni]->rangestart[1] == 'all')
               $p = $this->BN[$ni]->xss[1];
            else {
               for ($c=1;$c<=$this->BN[$ni]->columns; $c++){
                  if ($this->BN[$ni]->rangestart[$c] === $v) {
                     $p = $this->BN[$ni]->xss[$c];
                     $c = $this->BN[$ni]->columns + 1;
                  }
               }
            }
         }
         if (bccomp('0', bcmul($p_fs_xss, $p, 50), 50) == 0){
            $_SESSION['debug3'] .= "bcmul=0 found mul was: ".$p_fs_xss .
                    ' $p=' . $p. "\n";
            $_SESSION['debug3'] .= "value: " . $v . " prev col1xss: ".
                    $this->BN[$ni-1]->xss[1]."\n";
         }
         $p_fs_xss = bcmul($p_fs_xss, $p, 50);
         if (bccomp('0', $p_fs_xss, 50) == 0) $_SESSION['debug3'] .=
             "P()=0 found on value ".$v." node:".$ni."\n";
      }
      $p_fs_xss = bcmul($p_fs_xss, $this->BN[0]->popshare[1], 50);
      return $p_fs_xss;

   } //end of funtion CalcECXss()

   private function CalcECSqli($values, $class) {
   // calculate P(event|sqli)*P(sqli)
      $ni = 0;
      $p; //probability
      $p_fs_sqli = '1';
      foreach ($values as $v) {
         $ni++;
         if ($v === '1.0' || $v === '1.1') ;
         else if ($v === '1' || $v === '0') $v = intval($v);
         else if (intval($v) > 0) $v = intval($v);
         if (is_int($v)) {
            if ($this->BN[$ni]->rangestart[1] == 'all')
               $p = $this->BN[$ni]->sqli[1];
            else {
               for ($c=1;$c<=$this->BN[$ni]->columns; $c++){
                  if ($this->BN[$ni]->rangestart[$c] == 'inf') {
                     if ($v <= $this->BN[$ni]->rangeend[$c])
                        $p = $this->BN[$ni]->sqli[$c];
                  }
                  else if ($this->BN[$ni]->rangeend[$c] == 'inf') {
                     if ($v > $this->BN[$ni]->rangestart[$c])
                        $p = $this->BN[$ni]->sqli[$c];
                  }
                  else {
                     if ($v <= $this->BN[$ni]->rangeend[$c] &&
                           $v > $this->BN[$ni]->rangestart[$c])
                        $p = $this->BN[$ni]->sqli[$c];
                  }
               }
            }
         }
         else {
            // value is not numeric so must fall in named category
            // tablecolumn in PDT
            if ($this->BN[$ni]->rangestart[1] == 'all')
               $p = $this->BN[$ni]->sqli[1];
            else {
               for ($c=1;$c<=$this->BN[$ni]->columns; $c++){
                  if ($this->BN[$ni]->rangestart[$c] === $v) {
                     $p = $this->BN[$ni]->sqli[$c];
                  }
               }
            }
         }
         $p_fs_sqli = bcmul($p_fs_sqli, $p, 50);
      }
      $p_fs_sqli = bcmul($p_fs_sqli, $this->BN[0]->popshare[2], 50);
      return $p_fs_sqli;
   } //end of funtion CalcECSqli()

   private function CalcECPt($values, $class) {
   // calculate P(event|pt)*P(pt)
      $ni = 0;
      $p; //probability
      $p_fs_pt = '1';
      foreach ($values as $v) {
         $ni++;
         if ($v === '1.0' || $v === '1.1') ;
         else if ($v === '1' || $v === '0') $v = intval($v);
         else if (intval($v) > 0) $v = intval($v);
         if (is_int($v)) {
            if ($this->BN[$ni]->rangestart[1] == 'all')
               $p = $this->BN[$ni]->pt[1];
            else {
               for ($c=1;$c<=$this->BN[$ni]->columns; $c++){
                  if ($this->BN[$ni]->rangestart[$c] == 'inf') {
                     if ($v <= $this->BN[$ni]->rangeend[$c])
                        $p = $this->BN[$ni]->pt[$c];
                  }
                  else if ($this->BN[$ni]->rangeend[$c] == 'inf') {
                     if ($v > $this->BN[$ni]->rangestart[$c])
                        $p = $this->BN[$ni]->pt[$c];
                  }
                  else {
                     if ($v <= $this->BN[$ni]->rangeend[$c] &&
                           $v > $this->BN[$ni]->rangestart[$c])
                        $p = $this->BN[$ni]->pt[$c];
                  }
               }
            }
         }
         else {
         // value is not numeric so must fall in named category tablecolumn
         // in PDT
            if ($this->BN[$ni]->rangestart[1] == 'all')
               $p = $this->BN[$ni]->pt[1];
            else {
               for ($c=1;$c<=$this->BN[$ni]->columns; $c++){
                  if ($this->BN[$ni]->rangestart[$c] === $v) {
                     $p = $this->BN[$ni]->pt[$c];
                  }
               }
            }
         }
         $p_fs_pt = bcmul($p_fs_pt, $p, 50);
      }
      $p_fs_pt = bcmul($p_fs_pt, $this->BN[0]->popshare[3], 50);
      //$_SESSION['debug3'] .= "\n";
      return $p_fs_pt;
   } //end of funtion CalcECPt()


   private function CalcECScan($values, $class) {
   // calculate P(event|scan)*P(scan)
      $ni = 0;
      $p; //probability
      $p_fs_scan = '1';
      foreach ($values as $v) {
         $ni++;
         if ($v === '1.0' || $v === '1.1') ;
         else if ($v === '1' || $v === '0') $v = intval($v);
         else if (intval($v) > 0) $v = intval($v);
         if (is_int($v)) {
            if ($this->BN[$ni]->rangestart[1] == 'all')
               $p = $this->BN[$ni]->scan[1];
            else {
               for ($c=1;$c<=$this->BN[$ni]->columns; $c++){
                  if ($this->BN[$ni]->rangestart[$c] == 'inf') {
                     if ($v <= $this->BN[$ni]->rangeend[$c])
                        $p = $this->BN[$ni]->scan[$c];
                  }
                  else if ($this->BN[$ni]->rangeend[$c] == 'inf') {
                     if ($v > $this->BN[$ni]->rangestart[$c])
                        $p = $this->BN[$ni]->scan[$c];
                  }
                  else {
                     if ($v <= $this->BN[$ni]->rangeend[$c] &&
                           $v > $this->BN[$ni]->rangestart[$c])
                        $p = $this->BN[$ni]->scan[$c];
                  }
               }
            }
         }
         else {
         // value is not numeric so must fall in named category tablecolumn
         // in PDT
            if ($this->BN[$ni]->rangestart[1] == 'all')
               $p = $this->BN[$ni]->scan[1];
            else {
               for ($c=1;$c<=$this->BN[$ni]->columns; $c++){
                  if ($this->BN[$ni]->rangestart[$c] === $v) {
                     $p = $this->BN[$ni]->scan[$c];
                  }
               }
            }
         }
         $p_fs_scan = bcmul($p_fs_scan, $p, 50);
      }
      $p_fs_scan = bcmul($p_fs_scan, $this->BN[0]->popshare[4], 50);

      return $p_fs_scan;
   } //end of funtion CalcECScan()
   private function CalcECOther($values, $class) {
   // calculate P(event|other)*P(other)
      $ni = 0;
      $p; //probability
      $p_fs_other = '1';
      foreach ($values as $v) {
         $ni++;
         if ($v === '1.0' || $v === '1.1') ;
         else if ($v === '1' || $v === '0') $v = intval($v);
         else if (intval($v) > 0) $v = intval($v);
         if (is_int($v)) {
            if ($this->BN[$ni]->rangestart[1] == 'all')
               $p = $this->BN[$ni]->other[1];
            else {
               for ($c=1;$c<=$this->BN[$ni]->columns; $c++){
                  if ($this->BN[$ni]->rangestart[$c] == 'inf') {
                     if ($v <= $this->BN[$ni]->rangeend[$c])
                        $p = $this->BN[$ni]->other[$c];
                  }
                  else if ($this->BN[$ni]->rangeend[$c] == 'inf') {
                     if ($v > $this->BN[$ni]->rangestart[$c])
                        $p = $this->BN[$ni]->other[$c];
                  }
                  else {
                     if ($v <= $this->BN[$ni]->rangeend[$c] &&
                           $v > $this->BN[$ni]->rangestart[$c])
                        $p = $this->BN[$ni]->other[$c];
                  }
               }
            }
         }
         else {
          // value is not numeric so must fall in named category tablecolumn
          // in PDT
            if ($this->BN[$ni]->rangestart[1] == 'all')
               $p = $this->BN[$ni]->other[1];
            else {
               for ($c=1;$c<=$this->BN[$ni]->columns; $c++){
                  if ($this->BN[$ni]->rangestart[$c] === $v) {
                     $p = $this->BN[$ni]->other[$c];
                  }
               }
            }
         }
         $p_fs_other = bcmul($p_fs_other, $p, 50);
      }
      $p_fs_other = bcmul($p_fs_other, $this->BN[0]->popshare[5], 50);
      return $p_fs_other;
   } //end of funtion CalcECOther()

} // end of class Extractor

?>
```