Preventing Cross-Site Scripting Attacks by Combining Classifiers

Fawaz A. Mereani[1,2] and Jacob M. Howe[1]
[1]*City, University of London, Northampton Square, London, United Kingdom*
[2]*Umm AL-Qura University, Makkah, Saudi Arabia*
{*fawaz.mereani, j.m.howe*}*@city.ac.uk*

Keywords: Cascading Classifiers, Stacking Ensemble, Cross-Site Scripting

Abstract: Cross-Site Scripting (XSS) is one of the most popular attacks targeting web applications. Using XSS attackers can obtain sensitive information or obtain unauthorized privileges. This motivates building a system that can recognise a malicious script when the attacker attempts to store it on a server, preventing the XSS attack. This work uses machine learning to power such a system. The system is based on a combination of classifiers, using cascading to build a two phase classifier and the stacking ensemble technique to improve accuracy. The system is evaluated and shown to achieve high accuracy and high detection rate on a large real world dataset.

1 Introduction

Attackers exploit vulnerabilities in web applications in order to perform malicious actions such as obtaining sensitive information or gaining unauthorized privileges, for example administrator access. This makes the protection of these applications an important topic in computer security. A number of attack types target web application vulnerabilities, for example SQL injection and Cross-Site Scripting (XSS), and these remain in the top ten vulnerabilities listed in the Open Web Application Security Project (OWASP) (OWASP, 2017). Victims can be affected by XSS attacks that modify a webpage, steal cookies, capture clipboard contents, perform dynamic downloads and other attacks vectors (Raman, 2008). The traditional defence systems for web applications are based on a database of signatures that describe known attacks (Ariu and Giacinto, 2011). The most common cause of XSS vulnerabilities in web applications is the weakness of verification of the user's input or the environment (Rocha and Souto, 2014), where such vulnerabilities are discovered by attackers and exploited either on the client side or the server side.

This work shows how machine learning can be used to incorporate a verification stage into a web application with dynamic content. Machine learning is used to build classifiers which in combination can be used to detect and prevent attacks. The target is to prevent persistent or stored XSS which is when a malicious script is injected into a web application through input points and stored in the application database. Then on every visit to the page, the script will be executed on the user's browser (Kirda et al., 2009; Malviya et al., 2013; Williams et al., 2018).

In previous work, machine learning has been used to detect XSS attacks based on using one stage classifiers (Likarish et al., 2009; Khan et al., 2017; Mereani and Howe, 2018). In these works, the classifiers have been shown to be successful at classifying scripts as malicious or benign. However, in a web application the input might well not be a script at all, but plain text. Initial experiment showed that plain text was not always correctly classified. This paper investigates two ways of combining classifiers in order to improve the prevention of XSS. Firstly, the stacking ensemble technique is used to build a classifier to differentiate between malicious and benign scripts. Secondly, this is used together with a classifier to distinguish between user input that is plain text or script in order to build a cascading classifier for user input.

The contributions of this paper as follows:

- the feature space for detecting XSS from (Mereani and Howe, 2018) is developed
- five diverse classifiers, including a neural network, are used to recognise XSS attacks
- a stacking classifier to recognise XSS attacks is built from these five base level classifiers
- a decision tree classifier for distinguishing plain text and scripts is built
- a cascading classifier combining the two classifiers above is built to verify user input to a web application

- the various classifiers are evaluated.

The rest of this paper is organised as follows: Section 2 gives an overview of ensemble techniques, and Section 3 discusses related work on detecting XSS, and on using ensemble techniques for preventing attacks. Section 4 describes the datasets, the features used by the classifiers, and the classifiers' optimisation. Section 5 gives, then discusses, the experimental results, and conclusions are given in Section 6.

2 Ensemble Techniques

Ensemble techniques can be defined as combining multiple models to produce more stable, accurate and powerful predictive results than using a single model. Ensemble techniques have proved successful, for example, the winner of the Netflix competition (Salakhutdinov et al., 2007) implemented a powerful filtering algorithm by using an ensemble method, whilst the winner of KDD 2009 also used ensemble methods (Niculescu-Mizil et al., 2009). A wide range of ensemble-based algorithms have been developed under different names: bagging (Breiman, 1996), random forests, composite classifier systems (Dasarathy and Sheela, 1979), mixture of expert (Jacobs et al., 1991; Jordan and Jacobs, 1994), stacked generalization (Wolpert, 1992), consensus aggregation (Wolpert, 1992), combination of multiple classifiers (Xu et al., 1992), dynamic classifier selection (Woods et al., 1997), classifier fusion (Bloch, 1996), committee of neural networks (Drucker et al., 1994), classifier ensembles (Kuncheva, 2004), among others. The most prominent ensemble algorithms are:

Bagging Also called Bootstrap Aggregation (Breiman, 1996; Dietterich, 2002), Bagging creates a series of instances of the same type of classifier, each built by independently training on a random sample of the training dataset. The overall classification is achieved by combining the results of the instances, for example by majority voting. This method is suitable for high variance with low bias models.

Boosting This is a class of iterative approaches for generating a classifier. A first classifier is developed, and then subsequent classifiers trained to avoid misclassifications from previous iterations (Cao et al., 2010). The ensembles of weak classifiers of boosting methods are suitable for high bias with low variance models.

Stacking Also called stacked generalization (Wolpert, 1992) combines multiple classification models by using these different learning algorithms to train base level models. The base level is the first phase in the stacking process, the second phase is to use the outputs from the first phase classifiers as the input to another, independent, classifier which performs the final classification. The base level uses a standard dataset of examples abstracted to feature vectors and class labels to build the classifiers.

The meta classifier in the second phase of stacking uses the outputs of the base level to build its training dataset, the base level outputs being the features (Džeroski and Ženko, 2004; Zhang and Ma, 2012).

Cascading Classifiers Cascading generalization (Gama and Brazdil, 2000) is another method of combining classifiers. Cascading classifiers involves multiple classifiers; the output of the first classifier will be used in the next classifier (Zhao and Ram, 2004) and so on, concatenating several classifiers in a cascade. This method has been used to improve classification accuracy and to reduce complexity (Baig et al., 2014).

3 Related Work

3.1 Ensemble and Computer Security

Ensemble techniques, including cascading and stacking, have been used in a number of contexts for preventing exploits of vulnerabilities in networks or web applications. This section gives a brief overview of relevant work using multiple classifiers.

Cascaded classifiers are used in (Khor et al., 2012) to detect network intrusion. A first classifier classifies the input as Normal, Denial of Service, or Probing attacks and a second classifier classifies the inputs as Normal, Remote to Local, or User to Root attacks. This allows the J48 and Bayesian Network (BN) classifiers used to work with more balanced data, leading to an increase in detection rates (94.8% using J48-BN and 94.2% using BN-J48). In (Xiang et al., 2008) a multiple-level hybrid classifier based on decision tree classifiers and Bayesian clustering methods is used to implement an intrusion detection system. It includes four stages of classification, the first three stages to distinguish between attacks and the final stage to classify the attack to specific types. Different features are used for each stage. Their approach achieves a 96.80% detection rate. The stacking of SVM with 9 other machine learning algorithms (BayesNet, AdaBoost, Logistic, IBK, J48, Random Forest, JRip,

OneR and SimpleCart) is studied in the context of intrusion detection systems in (Chand et al., 2016). As in the previous works, the NSL-KDD dataset was used. Experiments compare the performance of using SVM as a meta-classifier together with the other classifiers, against a benchmark of using SVM only. The best classifier proves to be SVM stacked with Random Forest, which achieves 97.50% accuracy with 97.60% precision, better than using SVM only which achieved 91.81% accuracy and 91.70% precision.

In (Ariu et al., 2011) a model to detect attacks against a web server depending on HTTP payload structure is proposed. The payload is analysed by five different Hidden Markov Models (HMM) ensembles. The outputs are then used as features for a one-class classifier analysis. Experiments used two datasets containing normal HTTP requests, the first collected from requests towards the website of Georgia Tech (GT), the second dataset collected from requests toward the website of the authors' department at the University of Cagliari (DIEE). The dataset for attacks came from (Perdisci et al., 2009). The results of the experiment were AUC averages of 84.7% with DIEE dataset and 82.7% with GT dataset. A tool called VEnsemble is presented in (Goel et al., 2016), which used ensemble techniques for Vulnerability Assessment and Penetration Testing (VAPT). VEnsemble works by scanning the target (be it system, software, or network) using a variety of VAPT tools, then converting their outputs to numerical form, calculating weights based on VAPT tool accuracy and calculating a final result based on majority voting.

3.2 Machine Learning and XSS

Single classifier approaches have been used to detect XSS attacks using machine learning, achieving good results. (Likarish et al., 2009) evaluated a number of classifiers namely Naïve Bayes, ADTree, SVM, and RIPPER to detect obfuscated scripts (as a proxy for malicious). Their best results used an SVM classifier that achieved a 92% precision rate. (Khan et al., 2017) evaluated a collection of classifiers, Naïve Bayes, SVM, k-NN, and Decision Trees. Their approach achieved a 90.70% precision rate in recognising malicious scripts with the k-NN classifier. Both of these approaches used a small dataset of malicious instances. In (Mereani and Howe, 2018) SVM, k-NN, and Random Forests classifiers were evaluated using a larger dataset, with the best result achieved being a 96.79% precision with the k-NN classifier.

4 Methodology

The aim of this work is to take user input which might be either normal text, or a benign script, or a malicious script and in the case that the input is normal text or a benign script allow this to be stored on the server, whilst if the input is a malicious script it will be quarantined. This will be done by using a combination of classifiers, in two phases. The first phase will determine whether or not input is normal text or a script. The second phase will determine whether those inputs classified as scripts in the first phase are benign or malicious. This second phase will itself be built as a combination of five classifiers. Figure 1 illustrates the system.

4.1 Datasets

A combination of two datasets is used in this work, and this combined dataset is used in the two phases above. To give a real world simulation, the combined dataset is itself divided into two. One part (the training dataset) will be used to tune parameters and train the classifiers. The second part (the testing dataset) will be used to evaluate the classifiers once built.

4.1.1 Text Type Dataset

Normal text was obtained from the LARA dataset (Wang et al., 2011) which contains hotel and product reviews from Amazon.com and TripAdvisor. The normal text was divided into 4,972 instances for the training dataset and 4,096 instances for testing dataset, each instance labelled as normal text.

4.1.2 Scripts Datasets

The scripts dataset was obtained from (Mereani and Howe, 2018) and contains 30,027 scripts, with 15,029 labelled as benign and 14,998 labelled as malicious. These are divided into a training set of 10,027 scripts (5,029 benign and 4,998 malicious) and a testing set of 20,000 scripts (10,000 each of benign and malicious). This dataset comes from a variety of sources and was prepared for experiment by removing extra spaces and lines, then lowercasing all letters.

4.1.3 Datasets for Training and Testing

The first phase classifier to determine if user input is normal text or a script was trained using the combination of the training datasets for text type and for scripts, a total of 14,999 instances (4,972 labelled as

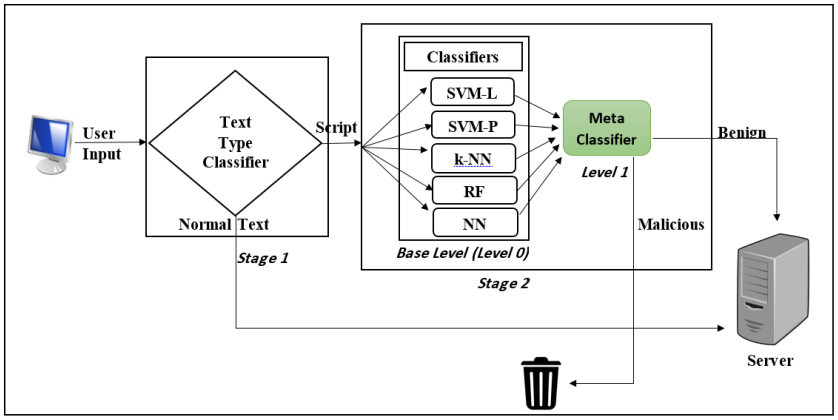

Figure 1: XSS Preventing System.

normal text and 10,027 as scripts). For testing, the remaining combined 24,096 instances were used (4,096 labelled as normal text and 20,000 labelled as scripts).

The second phase base level classifiers were trained on the combined text type instances and script instances. That is, 10,001 instances that are benign (either benign scripts or normal text) and 4,998 malicious scripts. For testing again all 24,096 instances are used, including the normal text instances (labelled as benign). The meta level classifiers are trained and tested on the labelled output from the base level classifiers, that is the 0/1 vectors of base level predictions.

4.2 Selecting Features

Features were selected for each stage separately.

4.2.1 Text Type Features

The features were selected to differentiate between inputs that are normal text or scripts. Inputs are composed of letters, numbers, and punctuation symbols (including parentheses). In order to distinguish between normal text and scripts the structure of the text was taken into consideration; normal text is expected to contain a lower percentage of punctuation symbols than scripts. Table 1 gives the six text type features that are used for the first phase classifier. All features take a value between 0 and 1, that represents the proportion of the text described by that feature.

4.2.2 Script Features

Features in this stage are categorized into two groups, 1) non-alphanumeric characters, and 2) alphanumeric. Adapting the 59 features given in (Mereani and Howe, 2018) (categorized as structural and behavioural), in total 62 features are considered as input for the base level classifiers.

Table 1: Text Type Features, Proportions.

Features	Description
Letter	Proportion of letters.
Number	Proportion of numbers.
Space	Proportion of spaces.
Punctuation	; , : . = ( ) [ ] { } < > / “ ” ‘
Special Character	! $? _ &
Operations	+ - * ^

4.2.2.1 Non-alphanumeric Features These are the non-alphanumeric characters that can be used in scripts. The XSS attacker can use techniques to trick a signature based protection system, such as using extra spaces or unnecessary symbols inside a script. In addition, there are combinations of symbols that might also be used in malicious scripts. An example of using symbol combination is (” ><). This combination will close the previous tag and open a new one containing a malicious script. The full non-alphanumeric characters are given in Table 2, including symbol combinations that might be used in attacks. This work has diverged from (Mereani and Howe, 2018) by adding apostrophe to the symbols group, and double slash and &lt to the symbols combinations.

Table 2: Non-alphanumeric Features.

Features Group	Terms
Symbols	&, %, /, \, +, ’, ?, !, ;, #, =, [,], $, (,), ^, *, , , -, <, >, @, _ , :, {, }, ˜, ., space, |, ¦, ”, ‘
Combinations	” ><, ’ ” ><, [], ==, &# , //, &lt

4.2.2.2 Alphanumeric Features Alphanumeric features includes commands and functions that might be used in XSS attacks, where the attackers use a range of functions or commands in their attacks. For

example, when using de-obfuscated functions, or adding commands within HTML tags. This work has removed one feature (createelement) from those given in (Mereani and Howe, 2018) and furthermore has added two new features, the proportion of letters, and the proportion of numbers within the script.

The features used in the current work are (0/1) values for sixty of the features, that is, within the script, either the feature exists or it does not. The remaining two features, the proportion of letters and of numbers within the script, are valued between 0 and 1. Table 3 gives the alphanumeric features.

Table 3: Alphanumeric Features.

Features	Description
Readability	Is the script readable.
Objects	document, window, iframe, location
Events	Onload, Onerror.
Methods	String.fromCharCode, Search.
Tags	DIV, IMG, script.
Attributes	SRC, Href, Cookie.
Reserved	Var .
Functions	eval().
Protocol	HTTP.
External File	.js file.
Letters	ASCII Code between 97 and 122.
Numbers	ASCII Code between 48 and 57.

4.3 Classifiers

A range of classifiers are used across the two phases of this work: one for the first phase, both base level and meta level classifiers for the second phase. The two phases are then cascaded together.

4.3.1 Text Type Classifier

For the first phase, text type classification, the Decision Tree (DT) classifier was used to create a model using the training dataset. The DT was optimized by tuning the MaxNumSplits parameter to control the maximum number of decision splits on a branch.

4.3.2 Base Level Classifiers

The motivation for using stacking is to combine classifiers so that diversity amongst them can be taken advantage of in order to improve the performance of predictions. To this end, in the second phase, five different kinds of classifier were used for determining if a script is malicious or benign: support vector machines (SVM) with a linear kernal (SVM-L), SVM with a polynomial kernal (SVM-P), k-NN, Random Forests (RF) and a feed forward Neural Network (NN). Working with the training set, SVM-L was tuned by adjusting BoxConstraint to control the maximum penalty of misclassification, and SVM-P tuned by setting OutlierFraction to determine the proportion of outliers in the instances. The k-NN classifier was tuned by adjusting k, the number of neighbours. For the RF classifier the number of trees was tuned. The NN classifier was tuned by determining the number of hidden layer neurons (Units) within the network, and the train function to update the weight and bias values.

4.3.3 Meta Level Classifiers

The meta level classifier is independent of the base level classifier. This work investigates five choices for the meta level classifier, these again being SVM with linear and polynomial kernals, k-NN, RF and NN. The meta training dataset (the outputs of the base level classifiers) is used as an input to create Meta classifiers. SVM-L is again tuned by adjusting BoxConstraint, SVM-P is again tuned by adjusting OutlierFraction, k-NN tuned by adjusting the number of neighbours, RF tuned by adjusting the number of trees, and NN tuned by adjusting the number of hidden layer neurons (Units), and the train function.

5 Results

Experiments were conducted using MatLab R2016b, with the experiments focused on the performance of DT, SVM, k-NN, RF, and NN classifiers in both phases using the datasets and features described in Section 4. The classifiers were tuned using the training dataset, and evaluation of the optimised classifiers is performed using five fold cross validation with the training dataset. The entire training dataset is then used to train the final classifiers for each stage and these are evaluated with the testing dataset (which has not been used in tuning and training). Timing experiments are also given, and this section concludes with a discussion of the results presented.

5.1 Classification Performance

5.1.1 Classifier Optimisation

Classifiers were optimised during training, leading to the models used for evaluation, as follows. In the text type stage DT was optimised by setting the MaxNumSplit parameter to 30. In the script classification stage the base level classifiers were optimised as follows: SVM-L was optimised by setting the BoxConstraint

parameter to 0.07. SVM-P was optimised by setting the OutlierFraction parameter to 0.1. k-NN was optimised by setting NumNeighbors to 1. RF was optimised by setting the number of trees to 20. NN was optimised by setting the number of hidden units to be 10, and the train function to be "trainbr", that is using Bayesian regularisation to minimize a combination of squared errors and weights. The same settings are used with meta level classifiers except for k-NN, which is optimised by setting NumNeighbors to 100.

5.1.2 Cross Validation

This section gives the results of the evaluation of the tuned classifiers on the training data. The descriptive statistics are averaged across the five folds.

5.1.2.1 Text Type Classification Table 4 gives the results for the DT text classifier in the first stage.

Table 4: Decision Tree Text Type Classifier Evaluation.

Accuracy	Precision	Sensitivity	Specificity
99.73%	99.63%	99.57%	99.82%

5.1.2.2 Script Classification: Base Level The five optimised base level classifiers (SVM-L, SVM-P, k-NN, RF, and NN) were each evaluated and the results on the training dataset are given in Table 5.

Table 5: Base Level Classifiers Evaluation.

Classifiers	Acc	Prec	Sens	Spec
SVM-L	97.73%	95.60%	97.52%	95.81%
SVM-P	99.15%	95.48%	98.36%	99.25%
k-NN	99.20%	98.67%	98.91%	99.34%
RF	99.32%	98.62%	99.13%	98.32%
NN	98.80%	97.75%	98.64%	98.87%

5.1.2.3 Script Classification: Meta Level The five base level classifiers are then stacked. The outputs from the five classifiers provide the features for the meta classifier. Five choices of meta classifier are investigated, again choosing SVM-L, SVM-P, k-NN, RF, and NN. The results for the performances of these five stackings are given in Table 6.

5.1.3 Testing Performance

The final classifiers are produced by training them using the entire training dataset. These classifiers are then evaluated on the testing dataset (which has not been used in the tuning and training).

Table 6: Meta Level Classifiers Evaluation.

Classifiers	Acc	Prec	Sens	Spec
SVM-L	99.25%	98.63%	99.13%	99.32%
SVM-P	99.11%	99.28%	98.08%	99.63%
k-NN	99.28%	98.85%	98.98%	99.43%
RF	99.19%	98.71%	98.84%	99.36%
NN	99.22%	98.71%	98.95%	99.36%

5.1.3.1 Testing the Text Classifier The final DT classifier for determining text type was tested with the text dataset of scripts and normal text. The first stage gave the results given in Table 7.

Table 7: Confusion Matrix of Text Type Testing.

	Text	Code
Text	4090	6
Code	0	20000

5.1.3.2 Testing Script Classifiers The base level classifiers' performance with the base testing dataset (including scripts and normal text) is given in Table 8.

Table 8: Base Level Testing Performance.

Classifiers	Acc	Prec	Sens	Spec
SVM-L	99.75%	99.65%	99.74%	99.75%
SVM-P	99.78%	99.83%	99.66%	99.87%
k-NN	99.88%	99.87%	99.85%	99.90%
RF	99.92%	99.93%	99.89%	99.95%
NN	99.80%	99.86%	99.66%	99.90%

The five meta level classifiers each using the output of the five base level classifiers are tested. The performance of these classifiers is given in Table 9.

Table 9: Meta Level Testing Performance.

Classifiers	Acc	Prec	Sens	Spec
SVM-L	99.92%	99.89%	99.93%	99.92%
SVM-P	99.93%	99.96%	99.88%	99.97%
k-NN	99.88%	99.80%	99.92%	99.85%
RF	99.90%	99.81%	99.95%	99.86%
NN	99.89%	99.81%	99.94%	99.86%

5.1.4 Entire System

The testing dataset (to simulate a real world attack) was used to test the entire cascading system performance. To investigate the classifiers working together, the dataset containing normal text, malicious

and benign scripts was used as input to the first stage, with the Decision Tree used to classify the data as text or script. Then, all predictions classified as a script in the first stage (20,006 in this experiment) are used in the second stage as an input (the remaining 4,090 scripts were classified as normal text, hence benign in the first stage). This stage uses a stacking classifier with SVM with polynomial kernal as the meta level classifier over the five base levels (SVM-L, SVM-P, k-NN, RF and NN) in order to determine whether the scripts are benign or malicious. Table 10 shows the confusion matrix of both stages (where the rows are the real result and the columns those given by the classifier). Table 11 shows the results of the entire system.

Table 10: Entire System Confusion Matrix.

First Stage			Second Stage		
	Text	Code		XSS	Benign
Text	4090	6	XSS	9988	4
Code	0	20000	Benign	12	10002

Table 11: Entire System Performance.

Stages	Acc	Prec	Sens	Spec
1st	99.97%	99.85%	100%	99.97%
2nd	99.92%	99.96%	99.88%	99.96%

5.2 Timing Performance

The goal of this work is for the overall classifier to be a layer between the user entering input and its being entering into the database of a web application. Hence the classification needs to be executed quickly enough that the performance of the website is not impacted. To this end, the time taken by the classifiers was evaluated. Table 12 details the system time taken (in seconds) by each stage when the testing set of 24,096 scripts was classified, as in Section 5.1.4. This includes the time taken by the DT classifier for the first stage, the time taken by each of the base level classifiers to return results, and the time taken by the meta level classifier.

Table 12: Time Performance (secs).

Classifier	Timing	Classifier	Timing
DT	0.0040	RF	0.2761
SVM-L	0.1471	NN	0.0290
SVM-P	0.1048	Meta SVM-P	0.0055
k-NN	5.6169		

The overall time of 6.1834 seconds is dominated by the cost of the k-NN classifier. Whilst the cost for each classification remains small, this mismatch motivates calculating the classifier performance for the entire system, omitting the k-NN classifier. The experiment from Section 5.1.4 has been repeated after removing the k-NN classifier from the stacking ensemble. Table 13 shows the results and Table 14 shows the confusion matrices for each meta classifier.

Table 13: Meta Level Testing Performance Without k-NN.

Classifiers	Acc	Prec	Sens	Spec
SVM-L	99.92%	99.93%	99.92%	99.93%
SVM-P	99.85%	99.85%	99.79%	99.85%
RF	99.86%	99.95%	99.78%	99.94%
NN	99.86%	99.95%	99.78%	99.94%

Table 14: Meta Level Confusion Matrices Without k-NN.

SVM-L			SVM-P		
	XSS	Ben.		XSS	Ben.
XSS	9993	7	XSS	9985	15
Ben.	8	9998	Ben.	21	9985
RF			NN		
	XSS	Ben.		XSS	Ben.
XSS	9995	5	XSS	9995	5
Ben.	22	9984	Ben.	22	9984

5.3 Discussion

Once the classifiers have been tuned on training data, the five fold cross-validation gives good results for both phases of the cascading classifier. The DT classifier gives excellent results for text classification with accuracy of 99.73%. The base level classifiers for determining if an instance is benign or malicious all perform well, and as conjectured each of the meta-level stacking classifiers improves on the base level results.

The final classifiers resulting from training using the whole training dataset give strong results when evaluated using the testing dataset (not used in the training and tuning of the classifiers) to simulate a real world deployment. All of the base level classifiers give over 99% accuracy, precision, sensitivity and specificity. With the base level classifiers already working so well, there is little room for stacking to improve performance, however, a small improvement can be observed, with the SVM-P classifier giving the best results. Hence this is used as the meta-level classifier in the cascading classifier for end-to-end testing.

The first phase of the cascading classifier to distinguish between normal text and scripts gives 100%

sensitivity, meaning that no scripts are classified as text, hence all scripts are passed to the second phase. This is exactly the desired behaviour. A small number of normal text instances are misclassified as scripts and passed to the second phase. However, as it has already been established that script classification can correctly classify these instances as benign this is not problematic. The second phase, using the stacked classifier (with SVM-P as the meta level classifier), classifies scripts with high accuracy and precision.

When examining the time required for each meta-level classifier it was found that k-NN was considerably more expensive than other classifiers. This motivated the study of results without k-NN in the stacking ensemble. Without k-NN, the SVM-P meta-classifier performs a little less well, with accuracy reduced from 99.92% to 99.85%. In security higher accuracy is better even if the time is longer, as long as the application continues to work smoothly. It is also worth observing that in this instance, the SVM-L meta-classifier performance is best, illustrating that with all the meta-classifiers performing well it is hard to determine an overall best choice.

6 Conclusion

This paper has demonstrated a system that contains two stages, each stage containing a classifier doing a different job. The first stage classifies user input either as normal text or as a script, with accuracy of up to 99.97%. The second stage depends on a stacked classifier based on SVM-L, SVM-P, k-NN, RF, and NN. This stacked classifier with SVM-P as the meta classifier gives high accuracy when applied to a large real attack dataset. The entire system cascading the two stages together achieved high precision (99.96%) for defending web applications from XSS.

This final accuracy result improves on the results with the same dataset from (Mereani and Howe, 2018) (where precision was 96.79%). This is partly because of an improved feature set, and partly because of the use of stacking. It is important to note that cascading allows the classifier to be used in a real web application where the script classifier is preceded by the text classifier. A proof of concept website incorporating the cascading classifier as a security layer has been constructed, and with a single user operates successfully. The cost of the k-NN classifier in the base level of the ensemble classifier is the dominating cost of the overall classifier. This was not problematic at the proof of concept stage. Future work is to test the performance of websites involving such a cascading classifier by performing large scale load testing.

Some scripts are still misclassified. Misclassification occurs for malicious scripts encrypted using Base64 encoding. Base64 uses letters more than numbers or signs. This result hints that adding to the ensemble a classifier aimed at this type of encryption would add to the power of a combined classifier.

In conclusion this work demonstrates that combining classifiers can lead to better overall classification by incorporating diversity into the classification process, and that a security layer can be incorporated into web applications for detecting XSS attacks.

REFERENCES

Ariu, D. and Giacinto, G. (2011). A modular architecture for the analysis of HTTP payloads based on multiple classifiers. In *Multiple Classifier Systems*, volume 6713 of *LNCS*, pages 330–339. Springer.

Ariu, D., Tronci, R., and Giacinto, G. (2011). HMM-Payl: An intrusion detection system based on Hidden Markov Models. *Computers & Security*, 30(4):221–241.

Baig, A., Bouridane, A., Kurugollu, F., and Albesher, B. (2014). Cascaded multimodal biometric recognition framework. *IET Biometrics*, 3(1):16–28.

Bloch, I. (1996). Information combination operators for data fusion: a comparative review with classification. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, 26(1):52–67.

Breiman, L. (1996). Bagging predictors. *Machine Learning*, 24(2):123–140.

Cao, D.-S., Xu, Q.-S., Liang, Y.-Z., Zhang, L.-X., and Li, H.-D. (2010). The boosting: A new idea of building models. *Chemometrics and Intelligent Laboratory Systems*, 100(1):1–11.

Chand, N., Mishra, P., Krishna, C. R., Pilli, E. S., and Govil, M. C. (2016). A comparative analysis of SVM and its stacking with other classification algorithm for intrusion detection. In *Advances in Computing, Communication, & Automation*, pages 1–6. IEEE.

Dasarathy, B. V. and Sheela, B. V. (1979). A composite classifier system design: concepts and methodology. *Proceedings of the IEEE*, 67(5):708–713.

Dietterich, T. G. (2002). Ensemble learning. *The handbook of brain theory and neural networks*, 2:110–125.

Drucker, H., Cortes, C., Jackel, L. D., LeCun, Y., and Vapnik, V. (1994). Boosting and other ensemble methods. *Neural Computation*, 6(6):1289–1301.

Džeroski, S. and Ženko, B. (2004). Is combining classifiers with stacking better than selecting the best one? *Machine Learning*, 54(3):255–273.

Gama, J. and Brazdil, P. (2000). Cascade generalization. *Machine Learning*, 41(3):315–343.

Goel, J. N., Asghar, M. H., Kumar, V., and Pandey, S. K. (2016). Ensemble based approach to increase vulnerability assessment and penetration testing accuracy. In *Innovation and Challenges in Cyber Security*, pages 330–335. IEEE.

Jacobs, R. A., Jordan, M. I., Nowlan, S. J., and Hinton, G. E. (1991). Adaptive mixtures of local experts. *Neural Computation*, 3(1):79–87.

Jordan, M. I. and Jacobs, R. A. (1994). Hierarchical mixtures of experts and the EM algorithm. *Neural Computation*, 6(2):181–214.

Khan, N., Abdullah, J., and Khan, A. S. (2017). Defending malicious script attacks using machine learning classifiers. *Wireless Communications and Mobile Computing*, 2017(5360472):9 pages.

Khor, K.-C., Ting, C.-Y., and Phon-Amnuaisuk, S. (2012). A cascaded classifier approach for improving detection rates on rare attack categories in network intrusion detection. *Applied Intelligence*, 36(2):320–329.

Kirda, E., Jovanovic, N., Kruegel, C., and Vigna, G. (2009). Client-side cross-site scripting protection. *Computers & Security*, 28(7):592–604.

Kuncheva, L. I. (2004). Classifier ensembles for changing environments. In *Multiple classifier systems*, volume 3077 of *LNCS*, pages 1–15. Springer.

Likarish, P., Jung, E., and Jo, I. (2009). Obfuscated malicious Javascript detection using classification techniques. In *Malicious and Unwanted Software (MALWARE)*, pages 47–54. IEEE.

Malviya, V. K., Saurav, S., and Gupta, A. (2013). On Security Issues in Web Applications through Cross Site Scripting (XSS). In *Asia-Pacific Software Engineering Conference*, pages 583–588. IEEE.

Mereani, F. A. and Howe, J. M. (2018). Detecting Cross-Site Scripting Attacks Using Machine Learning. In *Advanced Machine Learning Technologies and Applications*, volume 723 of *AISC*, pages 200–210. Springer.

Niculescu-Mizil, A., Perlich, C., Swirszcz, G., Sindhwani, V., Liu, Y., Melville, P., Wang, D., Xiao, J., Hu, J., Singh, M., Shang, W. X., and Zhu, Y. F. (2009). Winning the KDD cup orange challenge with ensemble selection. In *International Conference on KDD-Cup*, pages 23–34. JMLR.org.

OWASP (2017). OWASP Top 10 - 2017 rc1. `www.owasp.org`. Accessed: 26/4/2018.

Perdisci, R., Ariu, D., Fogla, P., Giacinto, G., and Lee, W. (2009). McPAD: A multiple classifier system for accurate payload-based anomaly detection. *Computer Networks*, 53(6):864 – 881.

Raman, P. (2008). *JaSPIn: JavaScript based Anomaly Detection of Cross-site scripting attacks*. PhD thesis, Carleton University, Ottawa.

Rocha, T. S. and Souto, E. (2014). ETSSDetector: a tool to automatically detect Cross-Site Scripting vulnerabilities. In *Network Computing and Applications*, pages 306–309. IEEE.

Salakhutdinov, R., Mnih, A., and Hinton, G. (2007). Restricted Boltzmann machines for collaborative filtering. In *International Conference on Machine Learning*, pages 791–798. ACM.

Wang, H., Lu, Y., and Zhai, C. (2011). Latent aspect rating analysis without aspect keyword supervision. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 618–626. ACM.

Williams, J., Manico, J., and Mattatall, N. (2018). Cross-site Scripting (XSS). https://www.owasp.org/index.php/Cross-site_Scripting_(XSS). Accessed: 26/4/2018.

Wolpert, D. H. (1992). Stacked generalization. *Neural Networks*, 5(2):241–259.

Woods, K., Kegelmeyer, W. P., and Bowyer, K. (1997). Combination of multiple classifiers using local accuracy estimates. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(4):405–410.

Xiang, C., Yong, P. C., and Meng, L. S. (2008). Design of multiple-level hybrid classifier for intrusion detection system using Bayesian clustering and decision trees. *Pattern Recognition Letters*, 29(7):918 – 924.

Xu, L., Krzyzak, A., and Suen, C. Y. (1992). Methods of combining multiple classifiers and their applications to handwriting recognition. *IEEE Transactions on Systems, Man, and Cybernetics*, 22(3):418–435.

Zhang, C. and Ma, Y. (2012). *Ensemble machine learning: methods and applications*. Springer.

Zhao, H. and Ram, S. (2004). Constrained cascade generalization of decision trees. *IEEE Transactions on Knowledge and Data Engineering*, 16(6):727–739.