An Analysis and Implementation of Methods for High Speed Lexical Classification of Malicious URLs

Shaun Egan
Dept. Computer Science
Rhodes University
Grahamstown, South Africa
Email: g10e4008@campus.ru.ac.zal

Barry Irwin
Dept. Computer Science
Rhodes University
Grahamstown, South Africa
Email: b.irwin@ru.ac.za

***Abstract*—Several authors have put forward methods of using Artificial Neural Networks (ANN) to classify URLs as malicious or benign by using lexical features of those URLs. These methods have been compared to other methods of classification, such as blacklisting and spam filtering, and have been found to be as accurate. Early attempts proved to be as highly accurate. Fully featured classifications use lexical features as well as lookups to classify URLs and include (but are not limited to) blacklists, spam filters and reputation services. These classifiers are based on the Online Perceptron Model, using a single neuron as a linear combiner and used lexical features that rely on the presence (or lack thereof) of words belonging to a bag-of-words. Several obfuscation resistant features are also used to increase the positive classification rate of these perceptrons. Examples of these include URL length, number of directory traversals and length of arguments passed to the file within the URL. In this paper we describe how we implement the online perceptron model and methods that we used to try to increase the accuracy of this model through the use of hidden layers and training cost validation. We discuss our results in relation to those of other papers, as well as other analysis performed on the training data and the neural networks themselves to best understand why they are so effective. Also described will be the proposed model for developing these Neural Networks, how to implement them in the real world through the use of browser extensions, proxy plugins and spam filters for mail servers, and our current implementation. Finally, work that is still in progress will be described. This work includes other methods of increasing accuracy through the use of modern training techniques and testing in a real world environment.**

Index terms: Frameworks, Phishing, Malicious URLs, Classifiers

I. Introduction

As shown in Aaron and Rasmussen [1], the number of unique phishing and malicious URLs have dropped in the period of the first half and second half of 2011. While these statistics would appear to indicate a decline in these kinds of attacks, they continue to indicate the large volume of phishing attacks that occur every month, with a total of 83083 attacks recorded for the second half of 2011 [1]. In the first half of 2011, the month of March had 26402 phishing attacks reported to the Anti Phishing Work Group alone [2], accounting for 18.81% of the total for the first half of the year. Phishing in general, consists of several different categories, all of which an attacker will send a message containing a URL, often by email, to a website that is intended to mislead the user into thinking that it originates from a legitimate source and the visited site is legitimate. The goal of this is to trick the user into entering private data such as credit card details which are then stolen. The complexity of these attacks can range from simple messages to high fidelity web sites that are made to look like the legitimate service.

The focus of this paper is primarily on detecting (and mitigating the threat from) phishing attacks of their various forms, namely phishing and targeted phishing (spear phishing). However, the research may apply directly to other kinds of malicious web addresses where the URL may be used to identify intent other than what is indicated such as sites hosting drive-by downloads where users unknowingly install malware [3]. Section II describes methods in which attackers try to mask the intentions of their website in terms of the URL that is used. *Current Methodologies*, section III discusses related work and the the contemporary approach to mitigating threats from phishing attacks. In the section IV, we discuss solutions proposed by other authors and how we intend to implement them using the Normandy framework discussed in section VII. Section V describes some geolocation research that was performed on the malicious training data and offers some explanations for the results. Section VI describes the basic operation of an Online Perceptron, our implementation of the Online Perceptron model, its strengths and problems, as well as suggesting modifications for improvement of this model. Section VII describes the framework currently being built to automate the process of data collection, network training and distribution of these networks to clients on any platform. It will also discuss the benefits of such a platform. Finally, section VIII highlights the conclusions made in this paper.

II. Obfuscation

Obfuscation, in the context of phishing, is the method where by attackers try to mask the intent of the website by creating a URL that appears to be legitimate or hide the destination completely from the end user. These URLs try

to deceive the user as well as hide the intent of the URL from automated detection, attempting to cause them to enter confidential information used for logging into a service, or causing the user to enter credit card details using fraudulent sales mechanisms. This is done by using one or more of several approaches. These range from miss-spelling parts of the URL, replacing the domain with an IP address or using a port number.

Miss-spelled domains are effective at a glance, but many users may detect them if they read the URL carefully. IP addresses are used in replacement of a domain, and hide the destinations intent completely and may generally be viewed as malicious. Port numbers are well hidden in a URL as they may be used on a benign websites address. This may move the user to a completely separate webserver or website. These obfuscation methods are outlined in more detail in Egan et al [4], Egan et al [5] and in Ma et al [3].

Often, security professionals and aware users are able to spot, often at a glance, malicious URLs before following them. The URL may not have any specific feature that definitively identifies it as malicious, but several contributing factors. This is justification for creating a classifier that may be trained to identify these URLs using artificial intelligence.

III. Current Methodologies

There are two main approaches to detecting malicious websites, focussed on detecting the nature of the address before or after following the link. The method following the latter approach is to use a content scanner, which downloads the webpage and analyses the data and classifies it from there. The primary problem for this approach is that it requires the user to download content before it can be classified, at which point the damage may have already been done in terms of malware already installed.

A. Modern Approaches

Several methods exist for detecting malicious websites without having to follow the URL. These methods include blacklists, spam filters and reputation services. Each of these are described in detail in Egan et al [4]. Blacklists can be very accurate, some with manually vetted input, and are a highly reliable source of information used to determine whether a URL points to malicious content. They suffer from being unable to adapt quickly. If no user has previously visited the phishing site, it may not have been reported to blacklisting services or may still be in a queue of URLs still to be vetted. Blacklists require the client side to perform a lookup each time a URL is visited, and in a South African context, this may be very slow with network latencies and also has a small impact on bandwidth. This adds to latency already required to fetch the webpage which lowers the useability of a blacklist implementation.

Spam filters are used within email servers to try and detect spam by using a series of heuristics. While these are relatively effective, without constant updating they quickly become irrelevant as obfuscation trends change and generally apply heuristic rules to the content of emails, rather than identifying URLs specifically. They are largely ineffective at detecting URLs without the use of a blacklist and will often employ a blacklist to mitigate this. As a result they tend to inherit the shortfalls of blacklists, although not useability since filtering can be done before the end user requests mail. A third approach used today are collectively called Reputation Services. These are offered by many security companies and create rating images next to any links displayed in search results. This works by having users rate websites as they use them. While this approach is useful in theory, it will only work for large sites which have large numbers of users. Smaller, but still legitimate, sites will have no rating and users will be unable to tell whether they are benign or malicious.

IV. Proposed solution

Using the shortfalls of current techniques described above, the following criteria are set for the next generation of malicious intent classifiers:

- Speed - Little or no perceptible overhead should be added to the user experience, achieving transparency until the user needs to be notified.
- Accuracy - An accuracy comparable to current methods of classification should be achieved.
- Adaptability - The solution should not be effected by changing trends or new sites that have not been previously visited.
- Automation - A completely automated system removes human error and keeps classification method up to date.

Le and Ma[6], [3] proposed Artificial Neural Networks (ANNs) as a method of classifying malicious URLs. As is shown later in this paper, ANNs within the proper framework meet all of these criteria. They improve adaptability as they classify URLs with insight rather than simply checking if a URL appears in a database. Usability is also improved as classification can be done on the client side in a lightweight fashion. This eliminates the need for lookups which add latency and is deployable in a number of ways:

- Browser plugin for single user or personal applications
- Proxy plugin for large institutions where a browser plugin may not be feasible
- Mail server plugin for detecting incoming attacks via email

V. Analysis of phishing data

As part of the data gathering done for this research, Phishtank [7] was used as the primary source of malicious URLs used for training data. The site Phishtank is a manually vetted blacklist whereby people may submit URLs. A total of 13867 malicious URLs we collected over a period of two weeks on which the following analysis was performed.

Firstly, each of the URLs was geolocated where possible using the MaxMind GeoLocation Library available from [8]. The number of hosts that were successfully located is 12084. The figure 1 shows unique locations of phishing hosts found

within the data. Note that this was performed without following redirects due to limitations in the API that was used.

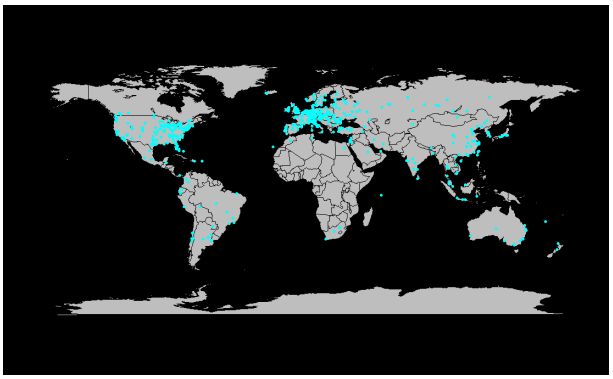

Figure 1. Unique locations of phishing hosts.

This data was then used to find countries in which these hosts reside and a heatmap was generated and is shown in figure 2.

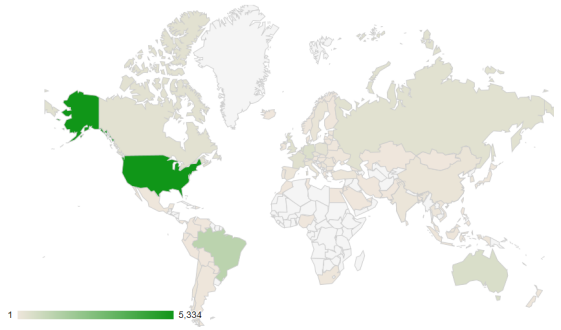

Figure 2. Heatmap showing population density of phishing hosts per country.

This heatmap shows an overwhelming portion of the hosts are found in the United States which may be due to its perceived wealthy economy or the advanced nature of e-commerce within its economy. This bias may also be due to the fact that Phishtank has predominately English speaking users. For a clearer image as to the rest of the data, figure 3 is shown without the United States input data.

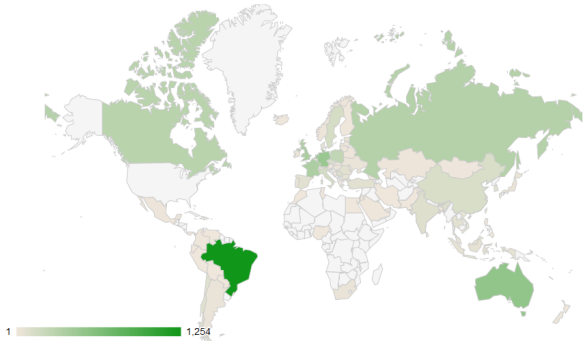

Figure 3. Heatmap showing population density of phishing hosts per country excluding US data.

Figure 3 gives a much clearer view of the population density of phishing hosts per country. It is interesting to note that not only first world countries have high densities of phishing hosts. This may be a combination of factors. Countries with lower technical expertise may suffer from higher numbers of compromised hosts, inflating their numbers. The sheer volume of hosts available in first world countries may contribute to the high densities of compromised hosts within those countries.

The countries with the six highest population densities are shown below.

Country	# of hosts	% of total
United States	5334	0.44
Brazil	1254	0.10
Australia	524	0.04
Netherland	508	0.04
Germany	490	0.04
United Kingdom	390	0.03

The table shows the six highest population densities for phishing hosts by country. Again it seems to indicate that first world economies have the highest rate of phishing hosts in the context of the [7] training data, with the United States accounting for 44

VI. Neural Networks

In both Ma et al [3] and Le et al [6] the proposed method of learning to identify malicious URLs is to use Artificial Neural Networks with varying degrees of complexity. These neural networks can be trained as classifiers to identify malicious URLs if given the correct inputs and training data. Neural networks are a mathematical model that try to mimic the way the human brain works by modelling neurons and interconnecting synapses. These may have a single hidden layer of a single neuron, such as the perceptron model, or multiple hidden layers with many neurons per layer. Both papers, however, suggest using the perceptron model as the base for the classification method.

A. Perceptron Model

The perceptron model has two observable layers. The first has many nodes, called inputs. These nodes represent the input data to the neural network. The second layer has a single neuron and is connected to the input layer by a series of weighted synapses. These weights are randomly initialised. Another important element is the threshold or bias applied to the neuron and its purpose is discussed in a later section and is also randomly initialised.

The figure 4 shows the logical layout of a basic perceptron, including the input layer, connecting synapses and the neuron within the hidden layer.

In our implementation, the perceptron is created using 17336 inputs. The majority of these are made up by a bagof-words created from the malicious and benign training data. The URLs in this training data are kept separate and are broken to find unique words within the URLs. Words are considered unique within the portion of the URL they are found. URLs are

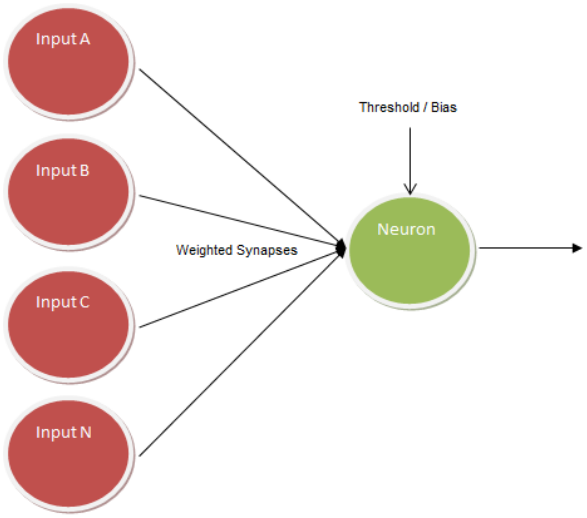

Figure 4. Basic Layout of a perceptron

broken down into domain, path, file and arguments sections of the URL. Each of these words, from both benign and malicious lists, are used as binary inputs into the neural network. In addition, there are several lexical features which are also used as inputs to the neural network and act as obfuscation resistant metrics.

The synapses are all randomly initialised to a value within the range of -0.5 to 0.5. The bias for the neuron is initialised using the same method. This sets the network into a random state which helps in the learning process.

1) Data sources: The first step in generating a perceptron model is to gather the training and testing data required. The malicious data that we used was sourced from Phishtank, while the benign data was sourced from Open Directory [9].

2) Pre Training Formatting: In addition to the bag of words used as input, 20 obfuscation resistant lexical features are also used as inputs. These inputs include, but are not limited to, the length of the URL, number of arguments passed, number of directory traversals as well as the length of the domain name. Each of these inputs are normalised to a range of 0 to 1. The value 0 represents the length of the shortest value seen within the training data, while 1 represents the largest value seen within the same data, for both malicious and benign training data.

Some features may simply not be present in a URL to be analysed by the classifier and would also result in a 0 value for the normalised data. For this reason, for each obfuscation resistant lexical field, a second input is created that has the value 0 to indicate that a feature was not present, or a 1 to represent that a feature was present and is simply the same value as the shortest encountered. If values are encountered that are smaller than this smallest value they are adjusted to the value 0, while larger values will be adjusted to the value 1 so as to not over compensate for those inputs. For this reason, training data needs to be fairly large to keep the likelihood of this scenario occurring low.

B. Feed forward mechanism

The feed forward mechanism is the process by which a URL is classified. Each of the inputs are represented as either binary values, or as floating point values from 0 to 1. Each of these values are then multiplied by the associated synapse wieght and then combined using a sum of all these inputs. This is represented by equation 1 and is known as a Linear Combination.

$$\text{Linear Combination} = \sum_{i=1}^{n} x_i w_i \tag{1}$$

The result is then compared to the bias or threshold for the neuron in the hidden layer. The output of the network is then given by this comparison.

$$f(x) = \left\{ \begin{array}{ll} 1 & \text{if } x \cdot w + b > 0 \\ 0 & \text{if } x \cdot w + b \leq 0 \end{array} \right\} \tag{2}$$

Equation 2 shows this process where f(x) represents the output of the classifier. In normal operation, this output is considered the classification and in our implementation 0 represents a benign URL, while an output of 1 represents a malicious URL.

C. Training

Since the network is initialised using random values, it does not represent any knowledge when it is created. Neural networks are a model that allow themselves to be trained using training data where the correct classification for each input is known before the network performs a classification. The basic method is to use as many examples of both positive and negative classification inputs as possible as training data. Each example is then classified by the network, compared to the expected result and a correction is given back to the neural network which then makes small adjustments to the synapses and thresholds. This is done repeatedly until the classifier achieves a satisfactory level of accuracy, which may be determined by a measure of error.

There are several methods of training the perceptron. The method implemented is the Online Perceptron Learning algorithm. Firstly, a URL from the training data is classified using the newly initialized perceptron. The first step in training is to calculate the error, given in equation 3.

$$\gamma = Y_d - Y \tag{3}$$

The error (γ) is calculated by subtracting the output of the classifier Y from the desired output Y_d defined by the source of the original URL (malicious or benign).

Once the error has been calculated, a weight adjustment is then calculated for each of the synapses using equation 4.

$$\Delta w = \alpha \times x_i \times \gamma \tag{4}$$

In equation 4 Δw represents the weight adjustment. This value is added to the current value of the synapses given in equation 5.

$$w_{t+1} = w_t + \Delta w \tag{5}$$

Once the adjustments have been made, the process is repeated with the next URL in the training set. This process will be repeated until a threshold is reached, usually an acceptable level of accuracy. Le and Ma [6], [3] achieved an accuracy level of 94% while using this method of training. In our own test, the same accuracy was achieved in 30 iterations over the training data set and confirmed with a separate data set of testing data.

D. Improvements

The first method of training a perceptron that tried to mitigate the problems associated with the Online Perceptron trianing method is known as Confidence Weighted (CW) training. As already mentioned, CW maintains a confidence in each feature through the use of a covariance matrix Σ and a mean value of all the input weights μ. In this notation, Σ_i represents the confidence in feature i from the covariance matrix, while μ_i represents the mean value for the input i from within the μ vector. The actual classification of a URL is performed in the same way as previously mentioned, that is, by multiplying input values by their respective input weights and through a step activation function. When training is performed, the method shown in equation 6 is used.

$$(\mu_{t+1}, \Sigma_{t+1}) = \arg\min_{\mu\Sigma} D_{KL}(\mathcal{N}(\mu, \Sigma) \| \mathcal{N}(\mu_t, \Sigma_t)), \quad (6)$$

$$\text{s.t.} \Pr_{w\sim\mathcal{N}(\mu,\Sigma)}[y_t(w \cdot x_t] \geq \eta \quad (7)$$

In [6] the authors achieved an accuracy of 96% when training the classifier to identify phishing websites.

The second approach uses a method called Adaptive Regularization of Weights (AROW) and builds on the CW method of training. It has a higher tolerance for noisy input data than CW and may also train on correct classifications owing to the ability to update confidence in input features and is represented by equation 8.

$$(\mu_{t+1}, \Sigma_{t+1}) = \arg\min_{\mu\Sigma} D_{KL}(\mathcal{N}(\mu, \Sigma) \| \mathcal{N}(\mu_t, \Sigma_t)) \quad (8)$$

$$+\lambda_1 l_{h^2}(y_t, \mu \cdot x_t) + \lambda x_t^T \Sigma x_t, \quad (9)$$

$$\text{s.t.} \Pr_{w\sim\mathcal{N}(\mu,\Sigma)}[y_t(w \cdot x_t] \geq \eta \quad (10)$$

When this method of updating the input weights of the classifier, Le [6] shows that an overall accuracy of 97% was achieved, making the classifier as accuract as a fully featured classification (a classification that uses lexical features of the network as well as addition information gathered by look ups such as WHOIS information).

E. Future Improvements

A method of increasing the output accuracy of the network is to use Cross Validation, which uses a second, independent, data set to validate training progress without impacting the structure of the weighting of the neural network. This is done because a network will start to optimize itself to the data set on which is trained rather than the real world optimal solution [10]. A second technique that is to be implemented and tested is how the use of an second intermediate hidden layer of varying sizes may impact the accuracy of the network, by allowing the network to handle non linearly separable dependencies between features.

VII. Normandy

The Normandy framework comprises of several individual parts that automates the process of the providing up-to-date networks for any application of the Perceptron Model as a solution for malicious URL detection. It specifically targets browser extensions and other plugins that have the ability to do online retrieval of new networks on a regular basis. The advantage of using these neural networks is that only a simple descriptor of the neural network, as well as a list of words, be required for a full update, resulting in very small updates. Also, the nature of neural networks means that these updates need to be executed infrequently, and a weekly update schedule has been decided upon to facilitate rapid response to any trend shifts in malicious URL creation. The framework is written in Python but is implemented primarily for use on a Linux server. A general overview of Normandy is displayed in figure 5.

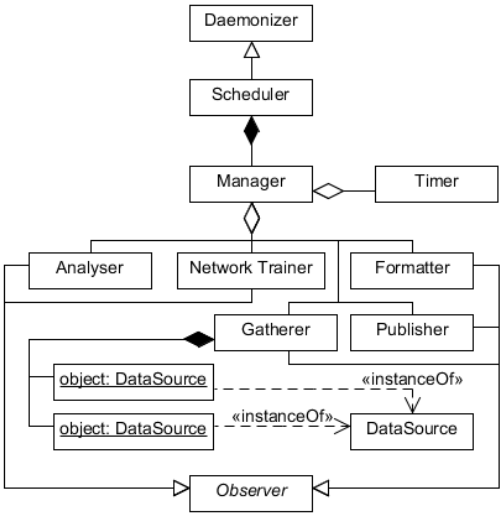

Figure 5. Overview of Normandy

A. Scheduler

Scheduler is the top level component within the framework. It has been deamonized to allow it to run as a background process on a server, accepting system signals to force operations such as exit. *Scheduler's* primary purpose is to organise and run other components of the framework in the correct order and at specific times. It has been built using the Observer design pattern to allow for fairly complex execution orders. This allows scheduler to execute other framework components, in the form of observers, at specific times or after specific events. An example of this is that once the *gatherer* and *formatter* have executed, both the *analyser* and *Network trainer* may be executed concurrently. It also has the advantage of making further complex scheduled events simple to implement in the

form of observers by simply inheriting properties and methods from the Observer class, stored in a specific folder and adding the observer to a configuration file.

B. Gatherer

The *Gatherer* observer is set to run twice daily (twelve hourly) and its primary purpose is to gather training data in the form of malicious and benign URLs. At the time of writing, the primary source of malicious data is gathered from Phishtank[7]. This is implemented through the use of a class that acts as a definition of a standard interface for data source retrieves. This class is called DataSource and has a single method called fetchData() which must be implemented by all child classes. These classes must exist in a folder and are enabled and disabled by use of a configuration file. Every new data source has its own implementation of a DataSource due to the different APIs required by each source, as well as the different formats in which data is returned. Each DataSource object will do preliminary formatting that converts the data into raw URLs and timestamps. This data is then stored in the *preformatted* table in a MySQL database.

C. Formatter

While each DataSource has functionality to format the data it retrieves, this is not the final formatting that the URL will receive. The *Formatter* observer runs after the *Gatherer* completes. It uses the data stored in the *preformatted* table and serves as a final formatting step, converting the formatting to one which can be used by the *Analyser* and the *Network Trainer* observers. This approach to formatting also allows the final format of the URLs to be changed without having to refactor each DataSource implementation.

D. Analyser

The *Analyser* observer is of secondary importance in terms of the Normandy framework. Its purpose it to resolve domain names to IP addresses. This is done so as to perform geolocation on these IP addresses in several different forms, each of which require an IP address to operate so as to avoid repeated lookups of the same domain. This geolocation is a subclass of the *AnalyserTask* class, the purpose of which is to allow for interesting metrics to be implemented easily and run on new data every time it is fetched. Like the *DataSource* classes, these *AnalyserTasks* are loaded into an array when configured to do so from a configuration file. The framework is simply sent a SIGHUP signal which causes it to reload its configuration files. This allows a developer to build functionality into the framework without having to restart it.

E. Network Trainer

Once the Formatter has completed its tasks, the Network Trainer will initialize itself by fetching training data from the database. The data has been formatted into the structure which the Network Trainer may use to begin training. Its responsibility is to use the new data to create new neural networks which will incorporate features which may not have been observed in the wild already. Once an acceptable level of accuracy has been achieved, the network is serialized into JSON format and stored in the database. The *Network Trainer* has been written to be as generic as possible, using subclassable classes that allow developers to implement new methods of training or activating the network without modification of *Normandy* itself.

F. Publisher

The final Observer, at the time of writing, is the publisher. It has the job of taking the JSON neural network descriptor and the bag of words and compressing them into a zip format. It then pushes these to the database of a webservice which may be used by client side implementations. These clients simply query the webservice and download a new version of the neural network once a week. They may then decompress the zip file and replace their current neural network description and bag-of-words used to break up URLs that they encounter. This is a useful approach as it is platform and implementation independent as it only contains a list of words and a list of input weights.

VIII. Conclusion

The authors of Ma et al [3] and Le et al [6] have shown how lexical elements alone can be used to train classifiers to an accuracy that makes them a viable solution to contemporary methods such as blacklists. In our own tests we have validated this by creating an Online Perceptron that is trained using the same lexical features and similar training data. In addition to this we have designed and implemented a framework which automates the process of data gathering and network training, providing a means of keeping clients up to date with current trends in URL obfuscation methods.

Future research will focus around using statistical analysis in the form of another perceptron model to determine if a domain name has been algorithmically generated. This will be combined with classifiers designed to detect both phishing and malicious URLs and a top level classifier to determine the weighting of each of these inputs. Additionally this will provide a graduated estimation of the intent of a URL based on all three classifiers. Also, statistical analysis will be performed on the training data to try to determine if there are other possible heuristics that could be used to try to increase the accuracy of the classifier.

References

[1] G. Aaron and R. Rasmussen. (2012, April) Global phishing survey: Trends and domain name use in 2h2012. Anti Phishing Work Group. http://www.antiphishing.org. [Online]. Available: http://www.antiphishing.org/reports/APWG_GlobalPhishingSurvey_2H2011.pdf; Last accessed: 06/05/2012

[2] (2012, April) Phishing activity trends report 1st half 2011. Anti Phishing Work Group. [Online]. Available: http://www.antiphishing.org/reports/apwg_trends_report_h1_2011.pdf ; Last accessed: 06/05/2012

[3] J. Ma, L. K. Saul, S. Savage, and G. M. Voelker, "Beyond blacklists: Learning to detect malicious web sites from suspicious urls," in *Proceedings of theSIGKDD Conference. Paris,France*, 2009.

[4] S. Egan and D. B. Irwin, “An evaluation of lightweight classification methods for identifying malicious urls,” in *Internet Security South Africa*, 2011.
[5] S. Egan and B. Irwin, “High speed classification of malicious urls,” 2011.
[6] A. Le, A. Markopoulou, and M. Faloutsos, “Phishdef: Url names say it all,” September 2010.
[7] Phishtank. [Online]. Available: http://www.phishtank.com/; Last accessed: 27/04/2011
[8] Maxmind geoip api. [Online]. Available: http://www.maxmind.com/app/api; Last accessed: 11/05/2011
[9] Open directory. [Online]. Available: http://www.dmoz.org/; Last accessed: 11/05/2011
[10] R. Kohavi, “A study of cross-validation and bootstrap for accuracy estimation and model selection.” Morgan Kaufmann, 1995, pp. 1137–1143.