2009 10th International Symposium on Pervasive Systems, Algorithms, and Networks

Automatically Generating Classifier for Phishing Email Prediction

Liping Ma, Rosemary Torney, Paul Watters, Simon Brown
Internet Commercial Security Laboratory (ICSL)
Centre for Informatics and Applied Optimization
Graduate School of Information Technology and Mathematical Sciences
University of Ballarat, Australia
Email: (l.ma, r.hay, p.watters, s.brown)@ballarat.edu.au

Abstract

Phishing is a form of online identity theft that employs both social engineering and technical subterfuge to steal consumers' personal identity data and financial account credentials. Phishing email prediction has drawn a lot of attention from many researchers. According to current anti-phishing research, a classifier generated by decision tree produces the most accurate predictions. However, there appears not to be any open source available to transfer such a decision to an implementable classifier. The work presented in this paper builds a decision tree parser which automatically translates a decision tree into an implementable program language so that the decision is useful in real world applications. Experiment results show that the parser performs as well as the original decision.

1. Introduction

Phishing is defined by the Anti-Phishing Working Group (APWG) as a form of online identity theft that employs both social engineering and technical subterfuge to steal consumers' personal identity data and financial account credentials ([1]). Social engineering is the process of deceiving people into giving away access or confidential information. The phisher, using social engineering techniques, attempts to quickly establish their authenticity and authority to request the data, a valid reason for the request and a sense of urgency to comply. The email often then directs the victim to a bogus site that is almost or completely indistinguishable from the genuine authority, reinforcing the inclination for compliance, and where the request for sensitive information is made. The cost of phishing scams is not limited to the money that is skimmed by phishers. Businesses and financial institutions are spending large amounts of time and money on anti-phishing activities, and online businesses are suffering as up to a quarter of customers shy away from internet purchases altogether to protect themselves ([2]).

Most servers have firewalls and filters to try to limit the exposure of their customers to spam and phishing emails and websites. This first line of defence against phishers uses a decision created from some form of machine learning and uses a routine based on this decision to classify emails and URLs as phishing or safe. To counteract this, phishers are constantly changing their sites, URLs and styles. The average time that a phishing site is live can be as little as 5 days ([2]). Clients can use a server to host a website for 5 days before there is any financial cost, and more importantly any traceable details given. The firewalls and filters can only deal with what they know, such as the phishing flags that have been encountered in the past. When new methods are encountered, the machine learning decision that the firewall/filter is based on is required to be updated.

Currently, the updating of the filter or firewall is done manually. Once the machine learning method produces the decision, it has to be manually translated into implementable code. Although there are some semi-automated methods, a comprehensive search of the internet failed to reference any automated methods to convert a decision into implementable code.

Phishers are rapidly evolving new methods and sites, constantly improving their bait. The time it takes to develop new filters using manual or semi-automated methods could be considered counter productive. By the time the new filter is rolled out, the phishers have moved on and are using a new set of tools, which are not covered by the new filter. A system to automatically translate a decision into implementable code would negate the need to recode filters and firewalls for every new decision. This would reduce the time taken to implement the new decision, reduce the workload required and reduce the expense of new filters/firewalls.

In this paper we describe an automated decision translation system that takes the decision tree produced by the C4.5 ([3]) decision tree learning method and produces implementable code using Python. The resultant program mimics the tree and achieves the same accuracy results.

The rest of the paper is structured as follows: Section 2 gives the background of anti-phishing and text classification. Section 3 provides the details of phishing email prediction. Section 4 illustrates how the parser of creating classifier automatically is implemented. Section 5 shows the experimental results by comparing the accuracy of decision tree and the classifier and finally Section 6 concludes the work and direction for the future.

978-0-7695-3908-9/09 $26.00 © 2009 IEEE
DOI 10.1109/I-SPAN.2009.153

2. Background

There are currently several products available that use text classification to try to limit the potential damage caused by phishing. AntiPhish [4] is a browser extension which is used to protect inexperienced users against spoofed web site-based phishing attacks. AntiPhish is a plug-in tool which keeps track of users' sensitive information and prevents this information from being passed to a web site that is considered untrustworthy or unsafe. A text classification algorithm is responsible for identifying whether a web site is a phishing site based on addresses used in a form. It compares a legitimate URL and IP address with URL the page actually locates. AntiPhish focuses more on tracking sensitive information provided by a user. [5] identified a website as a suspect phishing site when the visual similarity value is above a pre-defined threshold.

Another widely-deployed technique is based on using a blacklist of phishing domains to force the browser to refuse to visit, such as PwdHash [6], [7] and SpoofGuard [7], [8] by Stanford University. However, it is currently unclear how effective such blacklisting approaches are in mitigating phishing attacks in reality since phishing sites are very short lived and some features of other types might be ignored.

Various methodologies have recently been developed for document classification and representation to assist in anti-phishing [9], [4], [10], [11], [7], [5], [12], [13], [14] using different machine learning approaches. [9] developed the system PILFER using a support vector machine (SVM), [10] employed a Markov model, while [11] and and [12], [13], [14] used the decision tree ([15], [16]) as their classifier for preferring the robustness that C4.5 provides.

Text classification[17] is technique that automatically categorize text documents into pre-defined classes/types based on their contents. As documents cannot be directly interpreted by a classifier, a feature selection procedure is required that coverts a document into a compact representation suitable for the learning algorithm and the classification task. Deciding which features are relative or descriptive has always been a central problem in machine learning techniques. For example, [18] defined a "relevant feature" as one that is neither irrelevant nor redundant to the target concept, an "irrelevant feature" does not affect the target concept in any way, and a "redundant" feature does not add anything new to the target concept.

Reference [12] (detail refers to 3) proposed a new method by developing a technique to build a robust detector using a short feature vector space selecting a limited set of "good features" without compromising the classification accuracy. First, features are collected by observation; then the features are evaluated and a best feature set was selected according to the score gained by each feature. Finally, a few machine learning methods, especially the decision tree machine learning algorithm uses the optimized feature vectors to classify the documents. In classifying phishing emails, text classification is achieved via machine learning on a very small set of hybrid features.

If Google is used to search for "C4.5, 904,000 results about C4.5 are listed, which shows that C4.5 is favoured by many researchers. However, there is does not appear to be any open source product available that is able to implement the decision tree generated by C4.5. The work presented in this paper is a system that automatically transfers the decision tree into an implementable program so that the decision is applied to real world applications.

3. Phishing Email Prediction

Paper [12] presents a system of creating a robust classifier for phishing prediction. Considering phishing emails are largely similar in style, we believe that not only the content is important, but also the structural and special features of phishing emails. The quality of features used in phishing detection determines greatly the effectiveness of a classifier. We implement the classifier using a few machine learning methods and evaluate the accuracy across the different methods and sets of features. This algorithm reinforces the classifier by eliminating "noisy" features so that only good features are selected. We developed a method to identify the classifier by analysing possible features which indicate the relationship among learning methods and accuracies. From the analysis, we were able to confidently identify a feature set and recommend a proper classifier.

3.1. Features Defined in Emails

A phishing email usually contains multimedia information, including images and text, where the text information may contain plain text, HTML, URLs, scripts, styles and etc. However, the information cannot be recognized by a classifier directly, rather it needs to be characterized according to the needs of the system.

According to the analysis, phishing emails contain different types of features that have been defined manually based on observation. Three types of features are defined as:

- **Content features** which are domain-specific keywords that help to identify particular semantic contexts within the document. These contexts are used to assist in identifying if specific information exists, such as terms in a blacklist.
- **Orthographic features** are style characteristics that are used to convey the role of words or sentences, such as HTML features, size of document, the existence of url, forms, scripts or images, etc.
- **Derived features** are developed from the existing content or orthographic features. For example, whether in an email, the visible link is same as the hidden link; whether the content is readable (i.e. whether the colour

contrast between background and font are enough for human's vision), etc.

In our implementation, we experimented with seven features belong to the above three types:

1) links: the total number of links in an email.
2) nonv_links: total number of invisible links. This feature is calculated by an algorithm developed according to vision standards provided by W3C. In particular, if the colour deference between the background and font of link in an email is less than 500, the link is considered to be an invisible link.
3) nonmatching_urls: a binary value to show whether the visible url is as the same as the hidden url.
4) forms: a binary value to show the existence of any forms in an email.
5) scripts: the existence or type of the scripts in an email. The value is 0 if there is no script in the email. The value will be from 1 to 6 for different script types, namely text/execmascripts, text/javascripts, application/ecmascripts, application/javascripts, text/vbscripts and other scripts.
6) body_BL_words: the total appearance of the words in the blacklist in the body of an email. The blacklist includes sensitive terms, such as account, update, confirm, verify, secur, notif, log, click, inconvenien, bank, urgent, alert, etc.
7) subject_BL_words: total appearance of the words in the blacklist in the "subject:" line in an email.

3.2. Email Presentation

After the features have been defined, we developed a set of methods to extract all seven possible useful features from each email. Let $D = \{d_1, d_2, \ldots, d_{|D|}\}$ denote all the documents and $V = \{v_1, v_2, \ldots, v_{|V|}\}$ be the feature vector space. Where $|D|$ and $|V|$ are the number of documents and size of the feature vectors respectively. Let a_{ij} be the value of jth feature of ith document. Therefore, the presentation of each document is $A_i = (a_{i1}, a_{i2}, \ldots, a_{i|V|})$, and each document is $A = \{a_{ij}\}$ where $i = 1, 2, \ldots, |V|; j = 1, 2, \ldots, |D|$.

The values of all features are numerical however in vastly different ranges. For example, the body_BL_words could number in the thousands byte the number of nonv_links may be under five. To treat all the original features as equally important, the value of each feature needs to be normalized before the classification process. Feature values are normalized using the quotient of the actual value over the maximum value of that feature so that numerical values are limited to the range $[0, 1]$.

3.3. Detecting Phishing Emails

The architecture of our classification system consists of four components: Feature Generator, Learner, Inductor and Classifier. Figure 1 illustrates the system architecture.

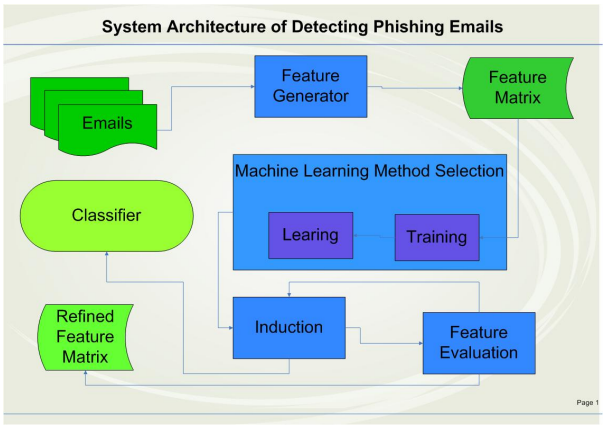

Figure 1. System architecture

- The Feature Generator uses the content, orthographic and derived knowledge to produce a set of feature vectors, one per document. All the documents will be represented as a matrix.
- Giving a *Feature Matrix*, the Learner employs a machine learning algorithm (such as system C4.5 [3]) to train and test the classifiers.
- According to the information gain generated from the Learner, the Inductor runs the machine learning algorithm with less features. Again, the classifier is trained and tested and new information gain is generated. This step is implemented recursively until the best feature set is identified.
- The Classifier applies the decision tree learning algorithm to classify the emails, but this time using the selected feature vectors which have a smaller vector space and a better discriminator.

4. Decision Parser Generation

The format of the decision tree generated by C4.5 is rigid which makes the automatic transform possible by creating a decision tree parser. The parser consists of four major components, namely:

1) $getVars(inName)$ takes a name of a file as input and returns a list of variables used in the decision tree.
2) $procFile(inName, varList, varNames)$ finds the valid part of a decision tree, and call $procLine(line, varList, varNames)$ routine that parses individual line of the selected part of the decision tree.

3) $procLine(line, varList, varNames)$ is a routine that parses a line and transfers it into Python Statement. It works together with its calling function $procFile(inName, varList, varNames)$.
4) $writeProg(outName, varList, varNames)$ writes the $main$ function, walks through the results produced be $procFile(inName, varList, varNames)$, and writes each statement to the output file,and appends the last statement to implement the main function created.

The complete algorithm for parsing decision tree and creating an implementable decision automatically is given as follows:

Input: Document contains all the decisions
Step 1 Extracting variables from the file produced by C4.5 along with the tree. This step identifies the variable names and classes.
Step 2 Pre-processing the decision tree set. The parser firstly remove information in the document which is irrelevant to the actual tree, including headings, comments and evaluation figures. Then the parser identifies a subtree and stores the position of the subtrees in the main tree it belongs to. At the point, every individual tree including subtrees are identified.
Step 3 Creating the classifier. The parser passes through each tree iteratively. According to analysis or decision tree and the features of Python, a set of transferring rules are identifies. Certain notations and format in the decision tree is transferred into corresponding Python statements which follow the syntax of Python. For example, a node is translated as "if" statement, a "|" is translated to an indent, and "+" and "-" are translated into return the decision.
Output: An implementable classifier.

5. Experiment

We have implemented the system described in both Sections 3 and 4 predicting phishing emails and generating an implementable classifier automatically by a parser. The aim of our experiment is to provide evidence that the parser works as well as the original decision tree.

The data used in our experiment are the live emails received by WestPac and their customer in 2007. We have used a total of 659,673 emails consisting of both phishing emails and legitimate emails, and those emails were semi-automatically classified. 613,048 emails are legitimate and 46,525 of the emails are phishing emails, which equates to 7% of the emails being phishing emails.

5.1. Performance of the Classifier

We have implemented the system described in Section 3.3. The data has been implemented using five learning algorithms namely decision tree, random forest, multi-layer perceptron (MLP), naive bayers and support vector machine (SVM). Performance is measured in accuracy which is a percentage of correct predictions over all the emails.

For each experiment, the data was partitioned into two disjoint sets (some documents formed the training set Tr contains both type emails, while the rest formed the testing set Te). The classifier was trained using Tr and then all of the documents in Te were classified using this classifier and the accuracy was measured. We used Cross-Validation for the learning process. Te and Tr are randomly-generated combinations. The size of each training set is approximately 593,616, and each testing set is approximately 65,957.

We ran C4.5 over the generated ten training and testing sets. The experimental results show that all the classifiers perform reasonably well without any feature selection done by machine, especially when the feature vector space is very small. The accuracy of the training is all above 99.2% and most of the testing is also above 99.5% ([12]).

Feature collection gives us a set of possible instances. However, not every feature is effective as a discriminator. Therefore, we need to select a relevant subset from the initial feature set upon which to focus our attention, while ignoring the rest. Under our approach, induction is used for feature selection. To discover the importance of each feature, the information gain (IG) of each feature is calculated to rank the importance one. The larger the information gain is, the more useful a feature will be. By observation, the "subject_BL_words" is the feature with best quality, while the "forms" gives the least discrimination and possibly brings noise to the classifier. The classifier is trained using the smaller vector space features. As a result, a classifier can be built by using a certain number of instances without affecting the overall performance. Our solution can train a classifier much faster without reducing its effectiveness because of the very low dimensionality.

Experiments were conducted with five machine learning methods to find the one which performd the best. We have implemented the classifier using decision tree, random forest, multi-layer perceptron, naive bayers and support vector machine (SVM). The result showed that decision tree generated the highest accuracy which builds a good classifier.

5.2. Performance of The Decision Parser

We implemented the parser using Python and wrote the classifier in Python as well, which is writing Python programs with Python. Python was selected for several reasons. Python is an interpreted language, and therefore there is no need to compile the resulting program which reduces one step in the process. Unlike many other languages, Python uses indents to indicate blocks of code which means there is no need to overtly close a block of code since merely using different number of tabs (indents) changes the block. Python is also loosely typed, therefore it is not necessary

to declare variables of a specific type, and use them only to hold data of that type. Once the classifier is created we may translate the Python programs to other programming language too use available software.

Paper [12] shows the efficiency of the classifiers defined by the decision tree learning method. The experiment carried in this paper was based on the classifiers created from our previous work. The input is a set of decision trees and the output is a Python program which includes a number of decision blocks corresponding to the decision tree set.

We have implemented the decision generated by our parser and compared with the results from the original training and testing.

	Decision Tree		Classifier	
Fold	Training	Testing	Training	Testing
1	99.2%	99.2%	99.8%	99.8%
2	99.2%	99.2%	99.8%	99.8%
3	99.2%	99.2%	99.8%	99.8%
4	99.2%	99.2%	99.8%	99.8%
5	99.2%	99.2%	99.8%	99.8%
6	99.3%	99.3%	99.6%	99.6%
7	99.3%	99.3%	99.6%	99.6%
8	99.3%	99.3%	99.3%	99.3%
9	99.3%	99.3%	99.1%	99.1%
10	99.6%	99.6%	96.1%	96.1%

Table 1. Accuracy rate (%) of decision and classifier in both training and testing of the 10-fold cross validation.

The experimental results that the classifier generated by the parser perfectly reflects the decision created in the learning process.

6. Conclusion

C4.5 is a robust decision learning method and is favoured by many researchers. However, there is does not appear to be any open source product available that is able to implement the decision tree generated by C4.5. Rebuilding a classifier is time and labour consuming. The work presented in this paper aims to automate the process from a decision to an implementable program. We have developed a system that automatically transfers the decision tree into an implementable program so that the decision is applied to real world applications. This work would reduce the time taken to implement the new decision, the workload and the expense of building new classifier.

In this paper, we used phishing emails as our running examples. Because of the rigid format of decision tree, this work may be employed in many other applications. Experiment results show that the parser performs as well as the original decision.

Acknowledgment

The authors would like to thank Wespac Bank, IBM and Victoria State Government who funded this project.

References

[1] http://www.antiphishing.org/, "Anti-phishing working group."

[2] S. A. and H. W., *Nature and Distribution of Phishing*. Pearson Prentice Hall, 2009.

[3] J. R. Quinlan, "C4.5: Programs for machine learning," 1993.

[4] E. Kirda and C. Kruegel, "Protecting users against phishing attacks," *The Computer Journal*, 2005.

[5] L. Wenyin, G. Huang, L. Xiaoyue, Z. Min, and X. Deng, "Detection of phishing webpages based on visual similarity."

[6] D. Boneh, "Spoofguard," http://crypto.stanford.edu/SpoofGuard/, Tech. Rep.

[7] B. Ross, C. Jackson, N. Miyake, D. Boneh, and J. Mitchell, "A browser plug-in solution to the unique password problem," http://crypto.stanford.edu/PwdHash/, 2005.

[8] ——, "Stronger password authentication using browser extensions," in *14th Usenix Security Symposium*, Baltimore, MD, USA, 2005.

[9] I. Fette, N. Sadeh, and A. Tomasic, "Learning to detect phishing emails," in *Proceedings of the 16th International World Wide Web Conference (WWW 2007)*, May 2007.

[10] C. Kruegel, G. Vigna, and W. Robertson, "A multi-model approach to the detection of web-based attacks," July 2005.

[11] C. Ludl, S. McAllister, E. Kirda, and C. Kruegel, "On the effectiveness of techniques to detect phishing sites," in *Proceedings of Detection of Intrusions and Malware and Vulnerability Assessment (DIMVA) 2007*, Lucerne, Switzerland, July 2007.

[12] L. Ma, B. Ofoghi, P. Watters, and S. Brown, "Detecting phishing emails using hybrid features," in *Proceedings of the Cybercrime and Trustworthy Computing Workshop (CTC-2009)*, Brisbane, Australia, 2009.

[13] L. Ma, J. Shepherd, and A. Nguyen, "Document classification vis structure synopsis," in *ADC2003, Fourteen Australasian Database Conference*. Adelaide, Australian: Australian Computer Society Inc, Feburary 2003, pp. 59–66.

[14] L. Ma, J. Shepherd, Y. Zhang, and A. Nguyen, "Enhancing text classification using synopses extraction," in *WISE2003, 4th International Conference on Web Information Systems Engineering*. Roma, Italy: IEEE Press, December 2003, p. .

[15] J. R. Quinlan, "Induction of decision trees," *Machine Learning*, vol. 1, pp. 81–106, 1986.

[16] ——, "Improved use of continuous attributes in c4.5," *Artificial Intelligence Research*, vol. 4, pp. 77–90, 1996.

[17] T. M. Mitchell, "Machine learning," 1997.

[18] M. Dash and H. Liu, "Feature selection for classification," 1997.