Detection of Social Engineering Attacks Through Natural Language Processing of Conversation Dialogs

Abstract

As computer security approaches improve, social engineering attacks have become more prevalent because they exploit human vulnerabilities which are hard to automatically protect. Phishing emails are a common form of social engineering attack, but the most effective attacks involve dialog between the attacker and the target. An attack dialog can be transmitted by several vectors including email, texting, chat application, phone, and in-person communication. A robust approach to detecting a social engineering attack must be broadly applicable to a range of different attack vectors.

We present an approach to detecting a social engineering attack which applies natural language processing techniques to identify suspicious comments made by an attacker. Social engineering attacks involve either *questions* which request private information, or *commands* which request the listener to perform tasks which the speaker is not authorized to perform. Our approach uses natural language processing techniques to detect questions and commands, and extract their likely topics. Each extracted topic is compared against a topic blacklist to determine if the question or command is malicious. Our approach is generally applicable to many attack vectors since it relies only on the dialog text. We have applied our approach to analyze the transcripts of several attack dialogs and we have achieved high detection accuracy and low false positive rates in our experiments.

1. INTRODUCTION

A critical threat to information security is *social engineering*, the act of influencing a person to take an action that may or may not be in their best interest. For the purpose of this study, we focus on the acts that would not be in the targets best interests, such as using psychological manipulation of people in order to gain access to a system for which the attacker is not authorized [10, 18, 11]. Cyberattackers target the weakest part of a security system, and people are often more vulnerable than a hardened computer system. All manner of system defenses can often be circumvented if a user reveals a password or some other critical information. Social engineering is a modern form of the confidence scam which grifters have always performed. Phishing emails, which fraudulently request private information, are a common version of the attack, but social engineering comes in many forms designed to exploit psychological weaknesses of the target. The use of modern communication technologies, including cellular phones and the internet, have greatly increased the reach of an attacker, and the effectiveness of the attack.

Social engineering attacks involve communication between the attacker and the victim in order to either elicit some information, or persuade the victim to perform a critical action. Information gathered might include explicitly secure information such as a credit card number, or seemingly innocuous information which can support a larger attack, such as the name of a coworker. An attacker might also convince the victim to perform tasks which would support an attack, such as going to a website. Numerous experimental studies over the years have demonstrated the susceptibility of people to social engineering attacks [9, 13, 29, 19, 3]. The effectiveness of social engineering has encouraged attackers to use it more frequently, relying on social engineering as a component of larger attacks. A study by Verizon of security breaches in 2013 has shown that 29% of all security breaches involve social engineering to extract information for use primarily for phishing, bribery, and extortion [28]. These attacks were executed primarily via email but also in-person, via phone, SMS, websites, and other documents. The frequency and effectiveness of social engineering makes it a serious security issue which must be addressed.

In order to manipulate a victim, the attacker needs to gain the trust of the victim. Trust is gained over the course of a dialog during which the attacker applies a variety of psychological techniques. Phishing emails and websites are a class of social engineering attack which are simple, attempting to establish trust in a single communication to a victim. Phishing techniques are simple to deploy but they are not as effective as *dialog-based* attacks because gaining trust often requires a two-way communication with a victim.

Previous work in the automatic detection of social engineering attacks is limited to the detection of phishing emails and websites, such as the phishing website filters built into Microsoft Internet Explorer and Mozilla Firefox. However, existing approaches are dependent on the properties unique to emails and websites, including the website URL and email header information. These approaches are limited to emails/websites and do not attempt to detect the more subtle class of social engineering attacks which are purely dialog-based. Other previous work has focused on the training of individuals about social engineering attacks in order to make them more aware and resistant in the future [8, 23]. User training can provide resistance to a wider range of attack types, but its effectiveness is inconsistent, depending on the abilities of individuals which can vary a great deal. An automated approach for social engineering detection is needed which can be applied to a broad range of attack types, requiring minimal effort from the individual user.

The attacker seeks to gain the trust of the victim through diverse psychological techniques which themselves can be very difficult to detect [10, 18]. However, in order to achieve his goal, the attacker must perform one of the following detectable dialog acts:

- **Ask a question** related to some secure data
- **Issue a command** to perform a secure operation

We present an approach to the detection of social engineering attacks by examining all text transmitted from the attacker to the victim and checking the *appropriateness* of the statement topic. A statement is considered to be inappropriate if it either requests secure information or requests the performance of a secure operation. Our approach uses natural language processing (NLP) techniques to detect questions and commands, and to extract their likely topics. Each sentence is parsed and the resulting parse tree is searched for patterns which are indicative of questions and commands. Once a question or command is detected, its potential topics are identified by locating verb/noun pairs which relate verbs with their direct objects. Each topic is evaluated by comparing it to the contents of a *topic blacklist* which describes all forbidden topics of conversation with the victim. We assume that the topic blacklist will be derived manually either based on a basic understanding common security requirements, or an existing security policy document associated with a system.

1.1 Detection Example

To concretize our discussion on detecting social engineering attacks, we next motivate our approach with an example of attack detection. For this example we assume that social engineering is used to attack the information technology infrastructure of a corporate entity. We assume that there is a security policy document which has been manually processed to define an appropriate topic blacklist. Figure 1 shows a statement found in the policy document which is used to define a blacklist entry.

TBL 1: Networking equipment must not be manipulated.

Figure 1: Security policy statement

The statement in Figure 1 is manually analyzed to generate the blacklist entry shown in Table 1.

	Action	Resource
Entry 1	manipulate	networking equipment

Table 1: Topic blacklist entry

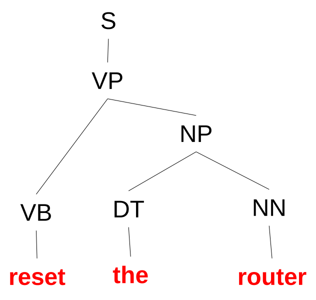

Figure 2: Parse tree of an inappropriate statement

For this example we assume that the social engineering attack is launched via texting, social media chat, or email, so that the dialog is already provided in text form. At some point during the dialog, the attacker makes the statement, “Reset the router”, whose parse tree is shown in Figure 2. The statement is recognized as a command by examining its parse tree and detecting a pattern which is common in direct commands. The topic of the command is extracted as *(reset, router)*, which are the verb and its direct object. The verb, “reset”, is a type of manipulation, so it matches the action “manipulate” in Entry 1. Also, the direct object of the predicate, “router”, is a type of networking equipment, so it matches the resource in Entry 1. Since the statement’s topic matches a topic blacklist entry, access will be denied and a warning message is transmitted to the victim.

1.2 Need for Semantic Analysis

Our approach detects social engineering attacks in dialogs by performing a semantic analysis of each sentence in order to extract some aspects of the meaning of the sentence. A straightforward alternative to our approach would be to search sentences for the occurrence of sets of words which are considered to be suspicious. The type of semantic analysis which we propose provides superior results to a more superficial word occurrence approach because our approach infers relationships between words in a sentence. As an example we compare the following two sentences found in a dialog, “Reset the router”, and, “If you reset your PC, the router will fail”. The first sentence is an inappropriate command which should be flagged as a social engineering attack, while the second sentence is not. A superficial word occurrence detection approach might detect the first sentence as an attack based on the use of “reset” and “router” together. However, the second sentence contains the same words, albeit used in a different way, so the second sentence would become a false positive. By performing semantic analysis our approach distinguishes between these two sentences, noting that in the case of the second sentence, the verb “reset” and the noun “router” are not directly related. Using semantic information results in improved precision and accuracy compared to a more superficial approach.

1.3 Role of Authentication

The appropriateness of a statement with respect to security depends on the identity of the speaker and his relationship to the listener. If a trusted party asks for sensitive information then it may not represent a social engineering attack. In order to consider the speaker's identity during attack detection we would need to implement some form of authentication. Authentication is outside the scope of this paper, so we assume that the speaker is untrusted. However, the ideas presented in this paper can easily be extended to consider speaker identity if an authentication system is in use.

1.4 Paper Outline

The remainder of the document is organized as follows. The expected usage scenarios of our system are presented in Section 2. Section 3 summarizes previous related research efforts. An outline of the steps of our approach is presented in Section 4. A detailed descriptions of the main steps of our approach are presented in Sections 5 and 6. The nature of the topic blacklist is described in Section 7. Section 8 presents the detection algorithm we use to identify inappropriate topics in text. Experimental results are presented in Section 9, and Section 10 describes our conclusions.

2. USAGE SCENARIOS

Figure 3 depicts two scenarios in which the social engineering detection system would be used. The system may interact with either an email client or a texting/chat application, as shown in Figure 3a, receiving textual dialog information and responding with textual alerts when a social engineering attack is detected. Another scenario shown in Figure 3b, involves the use of a speech recognition tool to detect social engineering in an in-person conversation or in a phone conversation. In this scenario we assume that a device with audio input capabilities, such as a cellular phone or a Google Glass [27], is being used to capture the conversation. An existing speech recognition tool, such as the Web Speech API [25], would be used to generate text which would then be passed to the social engineering detection system. In the case of attack detection, the system would generate an audio alert for a device such as a cellular phone, or a video alert for a Google Glass.

3. RELATED WORK

3.1 Spam Detection

Spam, most commonly in email form, describes unsolicited messages which may be used to convince the user to perform some action, such as purchasing a product. SpamAssassin [26] evaluates emails using a large set of rules which score the email based on the existence of character patterns in the header or body of the email. The scores are accumulated and emails with score higher than a fixed threshold are identified as spam. Some rules are URL-based, such as searching a blacklist database to find each URL in the message. Some rules check for common spam text strings inside the body of the email. Many other spam filters implement various techniques for combining the results of text matching rules [1, 21, 17].

3.2 Phishing Detection

A number of approaches have been presented to identify phishing websites and phishing emails. Phishing website identification approaches observe the features of the website and apply a set of rules which distinguish anomalous website properties. Identifying features used include the existence of misleading URLs, the existence of specific images, client-side search history, and password requests [6]. Detection rules consider values of individual features and correlations between feature values, such as the inclusion of a company logo at a website whose URL is not related to the company. Attackers can duplicate many of the features of a good website, so researchers have explored the use of W3C DOM objects which are more difficult to duplicate [20]. In [30] each website is characterized by the term frequency/inverse document frequency (TD-IDF) value of the text found within the page. The words with the top TD-IDF values in a page are supplied to Google's search engine to find the page, assuming that a phishing website would not be produced as a Google search result.

Several techniques have been proposed to detect phishing emails by extracting features of the email header and body. Commonly used features include the use of IP-based URLs, URLs linked to new domains, HREF values which do not match the displayed link, and HTML emails which allows URL names to be masked [7]. Researchers in [5] focus on the detection of HTML phishing emails by automatically posting fake data to the associated websites and verifying the correctness of the responses. Machine learning techniques have been used to distinguish phishing emails based on term document frequencies of key terms in the email, and their similarity to known phishing emails [22].

3.3 Social Engineering Detection/Prevention

Existing techniques for detection and prevention of the broader class of social engineering attacks depend on training potential victims to be resistant to manipulation. Training regimens have been proposed in previous work [23, 8] which educate users on the techniques used in previous attacks, and the importance of various pieces of information. A multi-level training approach has been presented which matches the degree of training to the responsibility level of the user [8]. Training techniques depend on the userâĂŹs awareness of his/her mental state and thought processes, referred to as *metacognition* [8]. A social engineering detection approach presented in previous work also depends on the user's metacognition in order to answer a sequence of security-related questions before providing data to an external agent [4]. In general, approaches which rely on the the cognitive abilities of the user will be unreliable since the mental state of users vary greatly, and can often be manipulated by a clever attacker.

4. SYSTEM OUTLINE

Figure 4 shows an outline of the detection process for each sentence. The first step is to parse the sentence using a context-free grammar of English, to generate a parse tree. The parse tree is analyzed during the **Question/Command Detection** step to identify patterns in the parse tree which are indicative of questions and commands. If a sentence is identified as a question or command then the subtree of its parse tree which contains the question or command is analyzed by **Topic Extraction** to find potential verb/noun *topic pairs* which describe the topic of the question or command. The topic pairs are compared to the topics in the

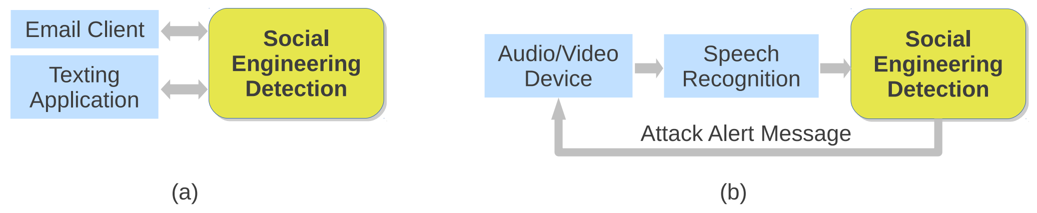

Figure 3: Usage scenarios, (a) interaction with email client or texting application, (b) interaction with audio/video device

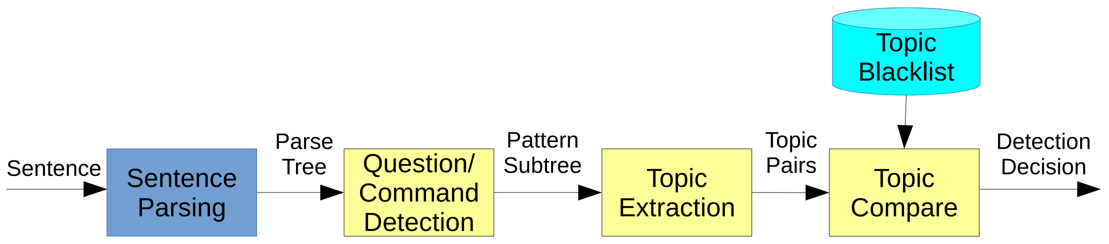

Figure 4: Structure of the Detection System

topic blacklist to determine whether or not the topic of the question or command is considered to be malicious.

We use the Stanford Parser [14] to parse each sentence and generate a parse tree. The Stanford Parser is a probabilistic context-free parser which is a well used open source project in the natural language processing community. The symbols used in the parser's grammar are from the Penn Treebank Tagset [16] which includes all accepted parts of speech. The following sections describe the Question/Command Detection and Topic Extraction steps in detail, as well as the construction of the topic blacklist.

5. QUESTION/COMMAND DETECTION

The goal of this step is to examine the parse tree of a sentence to determine whether or not the sentence contains a question or a command. If the sentence is found to contain a question or a command, the subtree of the parse tree which expresses the question or command is identified and passed to the next step. Each sentence contains one or more clauses and each clause can express a number of different speech acts [2, 24]. Clauses can be categorized based on the speech act which they are used to perform, but we are interested in two types of clauses, 1) *imperative clauses* which are used to issue commands, and 2) *interrogative clauses* which are used to ask questions. Both imperative and interrogative clauses are associated with several syntactic forms which can be recognized as patterns in the parse tree of a sentence. We detect questions and commands by searching the parse tree of each sentence to find the patterns associated with imperative and interrogative clauses. We use the Tregex tool [15] to match patterns in parse trees. In the following sections we present the parse tree patterns for both imperative and interrogative clauses.

5.1 Command Detection

Clauses which express commands are called *imperative* clauses. A **direct imperative** clause does not contain a subject, as in the sentences, "Go home" and, "Stop right there". Even without the subject it is clear that speaker is referring to the listener. A direct imperative clause is recognized as a clause with a verb but no subject. The subject of a sentence cannot be recognized directly by a traditional English parser since the term *subject* refers to a semantic role rather than a part of speech. For the purpose of command detection, we identify the subject of a clause as a noun phrase before the main verb in the clause. This is usually the case because the subject appears before the verb in most English sentences. Since noun phrases and verbs are parts of speech which are identified by the Stanford parser, we identify a direct imperative clause as one which does not contain a noun phrase before the verb. For example, the sentence "reset the router", whose parse tree is shown in Figure 2, is an imperative clause containing a verb, *reset*, with no noun phrase before it. Only noun phrases contained in the main clause are considered and subordinate phrases are ignored. A subordinate clause contains a noun and a verb, but does not express a complete thought. The noun phrase in a subordinate clause is not considered to be the subject of the main clause. For example, the sentence, "when the car starts, go home", contains the imperative clause "go home". The noun phrase "the car" precedes the main verb "go" but the noun phrase is contained in the subordinate clause "when the car starts", so it is ignored.

In many social settings, direct commands can seem rude and might alarm a victim to the fact that a social engineering attack is in process. For this reason, a speaker will often use a **soft imperative** which is a suggestion rather than a direct command. Soft imperatives include a second-person pronoun and a modal verb such as "you could" or "you should". An example of a sentence expressing a soft imperative is, "you should shut down the router", whose parse tree is shown in Figure 5. We detect a soft imperative clause by identifying a verb with a preceding modal verb and the pronoun "you" before that. In the case shown in Figure 5, the verb (VB) "shut" is preceded by the modal (MD) "should" and the pronoun (PRP) "you".

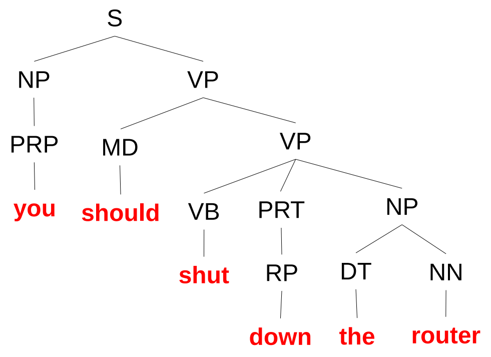

Figure 5: Parse tree of a soft imperative

5.2 Question Detection

Interrogative clauses are broadly classified as either *closed* (yes/no) questions which request a yes or no answer, or *open* questions which request a more elaborate answer. Yes/no questions are characterized by *subject/auxiliary inversion*, placing the auxiliary verb before the subject in the sentence. Auxiliary verbs are helper verbs such as "will", "may", or "can", which add meaning (tense, aspect, etc.) to the clause. In a statement, the auxiliary verb is placed after the subject as in, "I can eat". However, the sentence becomes a question when the subject and auxiliary positions are reversed as in, "Can I eat". The Stanford Parser automatically detects subject/auxiliary inversion, and the Penn Treebank tagset includes several tags which are used to demark subject/auxiliary inversion. The **SQ** tag indicates an inverted yes/no question, and the **SINV** tag indicates inversion of the subject and auxiliary which is associated with closed questions. An example is seen in the parse tree shown in Figure 6, for the question, "Can I eat". We detect a yes/no question by identifying either the **SQ** or **SINV** tags in the parse tree of a sentence.

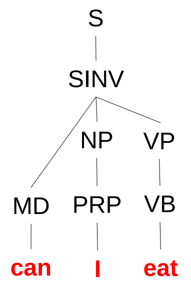

Figure 6: Parse tree of a yes/no question

Most open questions are *wh-questions* which use one of the wh-pronouns: who, whom, what, where, when, why, which, whose and how. The Stanford Parser automatically detects the use of wh-pronouns as part of different types of phrases, and the Penn Treebank tagset includes the following tags which are used to demark wh-phrases: **WHADJP** - wh adjective phrase, **WHAVP** - wh adverb phrase, **WHNP** - wh noun phrase, and **WHPP** - wh prepositional phrase. We detect a wh-question clause as one with a wh-pronoun at the beginning of the clause.

6. TOPIC EXTRACTION

We capture the topic of a clause as a pair, the main verb of the clause and its direct object. Given a parse tree which represents the clause, the goal of this step is to identify all likely topic pairs. Identification of the main verb is straightforward based on the tags generated by the parser. To find the direct object we assume that the direct object is a noun or noun phrase located after the verb in the clause. This assumption is usually correct because the *Subject-Verb-Direct Object* sentence form is very common in practice. There may be several nouns after the verb, so our approach produces one potential topic pair involving the verb and each noun after the verb. For example, in the sentence, "He ate pizza in the cafeteria", the topic pairs identified would be *(ate, pizza)* and *(ate, cafeteria)*. This approach can potentially lead to false positives because invalid topics are generated, such as *(ate, cafeteria)* in the previous example. This approach can also lead to false negatives in the case that the direct object is not located after the verb. However, topic extraction did not result in either false positives or false negatives in our results, leading us to believe that this approach is sufficiently accurate for this purpose.

In order to fully specify sentence topics in the presence of pronouns, it is necessary to perform *pronomial anaphora resolution*, determining the antecedent of a pronoun. Each pronoun must be replaced with the noun or noun phrase which it corresponds to in the context of the dialog. For example, suppose the dialog contains the following two sentences in sequence, "The pizza is good. Eat it". In order to determine the topic of the second sentence we must first resolve the pronoun "it" and replace it with its antecedent "pizza". This is a well studied problem and acceptable algorithms have already been presented in previous work, such as the Hobbs Algorithm [12]. We have found that in common dialog, a heuristic can be applied which replaces each pronoun with the noun phrase in the text which most recently precedes the pronoun. In the case of the example, "The pizza is good. Eat it", the most recent noun preceding the pronoun is "pizza", so we assume that "pizza" is the antecedent. This heuristic is simple, but it is sufficiently accurate for this purpose.

7. TOPIC BLACKLISTS

The topic blacklist is the list of statement topics whose discussion would indicate a violation of security protocol. Each topic either describes sensitive data or a sensitive operation. Each topic is composed of two elements, an *action* and a *resource*. The *action* describes an operation which the subject may wish to perform. The *resource* is the resource inside the system to which access is restricted.

We assume that the topic blacklist will be derived manually either based on a basic understanding common security requirements, or from an existing security policy document associated with a system. The blacklist would be created to match the security requirements of the system which it is being used to protect. A blacklist used to protect a regular individual would contain generic topics such as {"send", "money"} or {"tell", "social security number"}. A topic blacklist used by a corporate entity would contain topics related to the business of that entity.

8. DETECTION ALGORITHM

The top-level algorithm for scanning the dialog to identify attacks is shown in Figure 7. The outer loop scans through each sentence s in the text spoken by the attacker, $TEXT$. Each sentence is parsed at line 2. Since the English language can have ambiguous syntax, it is often possible to generate multiple valid parse trees for each sentence. The function *ParseNbest(s, 10)* on line 2 generates the 10 most likely parse trees for each sentence and returns the list of parse trees, *pt*. On line 3, the Question/Command Detection algorithm described in Section 5 is used to evaluate all parse trees in the list *pt*. If 30% or more of the parse trees are found to contain either a question or a command, the subtree describing the question or command is returned. If the subtree is non-empty, indicating that a question or command was found, then the potential topics are extracted at line 5, using the approach described in Section 6. The inner loop, starting at line 6, iterates through each entry t in the topic blacklist, TBL. Line 7 invokes the $MatchTopic$ function to determine if the topic of entry t matches any of the likely sentence topics in the list *tp*. If the topic is found then the attack is detected and a warning message is produced at line 8.

The algorithm for the $MatchTopic$ function is shown in Figure 8. The loop starting at line 1 iterates through all of the likely topics in the list of sentence topics, *tp*. The *Compare* function is used on line 2 to compare the action and resource of the sentence topic and the TBL entry. If both the action and resource are found to match then the function returns TRUE on line 3. The *Compare* function takes two words as arguments and generates an equivalence set for each argument which contains the stem, synonyms, hypernyms of the word. The stem of a word is its base form with a fixed tense and quantity (singular vs. plural). Synonyms are words with the same meaning and hypernyms are words whose meanings represent more general terms. The *Compare* function returns TRUE if there is any overlap between the equivalence sets of the two words.

9. RESULTS

We have evaluated our system by applying it to three social engineering dialogs which we refer to as Dialogs A, B, and C. Each of these dialogs is the transcript of a phone conversation between a professional social engineer acting as an attacker, and a victim working at a company with an internal computer system. The social engineers goal in each dialog is to extract information about the local computer system and to convince the victim to download and execute code provided by the attacker. In order to achieve these goals, the pentester used social engineering techniques described in previous work [11]. During the audits, the victims are unaware that the calls are part of an audit being performed, so their responses and other reactions are genuine, reflecting a true attack scenario. We have received all necessary approvals to evaluate anonymized versions of the dialog transcripts.

```
1. foreach $s \in TEXT$
2.     pt = ParseNbest(s, 10)
3.     st = DetectQuestionCommand(pt)
4.     if ($st \neq \emptyset$)
5.         tp = ExtractTopics(st)
6.         foreach $t \in TBL$
7.             if MatchTopic($tp$, $t$)
8.                 ATTACK_DETECTED
```

Figure 7: Social Engineering Detection Algorithm

```
1. foreach $lt \in tp$
2.     if (Compare($lt.action$, $t.action$) AND
           Compare($lt.resource$, $t.resource$))
3.         return TRUE
4. return FALSE
```

Figure 8: MatchTopic Algorithm

Table 2 shows information about each dialog. The second column is the number of sentences spoken by the attacker in the dialog. We do not include the number of sentences spoken by the victim because our approach does not evaluate those sentences. The third column is the number of sentences in each dialog which contain malicious questions or commands which we have identified manually.

Dialog	Sentences	Violations
A	24	3
B	30	6
C	20	1

Table 2: Social Engineering Attack Dialogs

Table 3 shows the topic blacklist which we use. The blacklist was manually generated based on our understanding of common generic social engineering attack goals. Synonyms for the terms in the blacklist are not shown.

Action	**Resource**
download	file
open	Internet Explorer
update	records
give	social
give	address
give	cert

Table 3: Topic Blacklist

9.1 Results Summary

As shown in Table 2, there are a total of 10 malicious sentences out of a total of 74 sentences. Our tool detected 6 of the 11 malicious sentences, with no false positives. We compute the *precision* and *recall* metrics according to their standard definitions shown in equations 1 and 2. In these equations, *tp* is the number of true positives, fp is the number of false positives, and fn is the number of false negatives.

$$precision = \frac{tp}{tp + fp} \quad (1)$$

$$recall = \frac{tp}{tp + fn} \quad (2)$$

Our results contain 6 true positives, 0 false positives, and 4 false negatives. The precision of our approach is 100% and the recall is 60%.

9.2 Performance Results

All experiments were run on an Intel i5 processor, 3.4 GHz. The performance results of each step are shown in the following list.

Sentence Parsing 5.357 seconds

Question/Command Detection 0.101 seconds

Topic Extraction 0.004 seconds

Topic Comparison <0.001 seconds

These measurements are the CPU time required to process all 74 sentences. Sentence Parsing and Question/Command Detection are performed 10 times for each sentence and the CPU time required is reflected in these performance results.

9.3 True and False Detection Examples

An example of a true positive detection is the sentence, "If you could email me the cert, that would be great". The sentence is malicious because it contains the soft command, *"you could email me the cert"*. This is detected as a soft command and its topic is identified as *(email, cert)*, where "email" is the action and "cert" is the resource.

The four sentences which are malicious but were not detected (false negatives) went undetected because recognizing their malicious nature would require an understanding of the *linguistic context*, the meaning of the sentences preceding the malicious sentence in the dialog. An example of a false negative based on lack of context are the sentences, "Click run", and "Yeah, click OK". The malicious nature of these sentences depends on understanding that "run" and "OK" refer to something unknown and potentially dangerous. In the dialog containing these sentences, the victim is first directed to go to a URL provided by the attacker. Our current approach evaluates sentences almost completely in isolation, without considering information gained from prior sentences. The only exception to this in our work is our resolution of pronomial anaphora which does consider noun phrases in the recent context. More general consideration of linguistic context is an important problem for future work.

10. CONCLUSIONS

We present an approach to detect social engineering attacks by verifying discussion topics against a topic blacklist. The approach is robust enough to effectively analyze the language of real attacks. The use of natural language processing techniques to detect sentence type and extract topics is novel. The definition of the topic blacklist is manual but we do not believe that it is onerous for a person with an understanding of the security requirements of the system being protected. The performance of our tool is good enough to provide attack warnings in real time during a conversation to prevent the victim from violating security protocol.

11. REFERENCES

[1] Keno Albrecht, Nicolas Burri, and Roger Wattenhofer. Spamato - an extendable spam filter system. In *Conference on Email and Anti-Spam (CEAS)*, 2005.

[2] L. Austin, J. *How to do things with words.* Oxford University Press, 1962.

[3] Taimur Bakhshi, Maria Papadaki, and Steven Furnell. A practical assessment of social engineering vulnerabilities. In *Human Aspects of Information Security and Assurance (HAISA)*, 2008.

[4] Monique Bezuidenhout, Francois Mouton, and Hein S. Venter. Social engineering attack detection model: Seadm. In *Information Security for South Africa (ISSA)*, 2010.

[5] M. Chandrasekaran, Ramkumar Chinchani, and S. Upadhyaya. Phoney: mimicking user response to detect phishing attacks. In *International Symposium on a World of Wireless, Mobile and Multimedia Networks, 2006 (WoWMoM)*, 2006.

[6] Neil Chou, Robert Ledesma, Yuka Teraguchi, Dan Boneh, and John C. Mitchell. Client-side defense against web-based identity theft. In *Network and Distributed Systems Security Symposium (NDSS)*, 2004.

[7] Ian Fette, Norman Sadeh, and Anthony Tomasic. Learning to detect phishing emails. In *Proceedings of the 16th International Conference on World Wide Web*, 2007.

[8] David Gragg. A multi-level defense against social engineering. Technical report, SANS Institute, December 2002.

[9] Tony Greening. Ask and ye shall receive: A study in social engineering. *SIGSAC Rev.*, 14(2), April 1996.

[10] C. Hadnagy. *Social Engineering: The Art of Human Hacking.* Wiley, 2010.

[11] C. Hadnagy, P. F. Kelly, and Ekman P. *Unmasking the Social Engineer: The Human Element of Security.* Wiley, 2014.

[12] J Hobbs. Readings in natural language processing. chapter Resolving Pronoun References. 1986.

[13] A. Karakasilitiosis, S. M. Furnell, and M. Papadaki. Assessing end-user awareness of social engineering and phishing. In *Australian Information Warfare and Security Conference*, 2006.

[14] Dan Klein and Christopher D. Manning. Accurate unlexicalized parsing. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics - Volume 1*, 2003.

[15] R. Levy and G. Andrew. Tregex and tsurgeon: tools for querying and manipulating tree data structures. In *International Conference on Language Resources and Evaluation*, 2006.

[16] Mitchell P. Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. Building a large annotated corpus of english: The penn treebank. *Comput. Linguist.*, 19(2), June 1993.

[17] Jun ming Xu, Giorgio Fumera, Fabio Roli, and Zhi hua Zhou. Training SpamAssassin with active semi-supervised learning. In *In Proceedings of the 6th Conference on Email and Anti-Spam (CEAS'09*, 2009.

[18] K.D. Mitnick and W.L. Simon. *The Art of Intrusion: The Real Stories Behind the Exploits of Hackers,*

Intruders and Deceivers. Wiley, 2009.

[19] Gregory L. Orgill, Gordon W. Romney, Michael G. Bailey, and Paul M. Orgill. The urgency for effective user privacy-education to counter social engineering attacks on secure computer systems. In *Proceedings of the 5th Conference on Information Technology Education*, 2004.

[20] Ying Pan and Xuhua Ding. Anomaly based web phishing page detection. In *Computer Security Applications Conference, 2006. ACSAC '06. 22nd Annual*, 2006.

[21] Calton Pu, Senior Member, Steve Webb, Oleg Kolesnikov, Wenke Lee, and Richard Lipton. Towards the integration of diverse spam filtering techniques. In *Proceedings of the IEEE International Conference on Granular Computing (GrC06), Atlanta, GA*, pages 17–20, 2006.

[22] Venkatesh Ramanathan and Harry Wechsler. phishgillnet-phishing detection methodology using probabilistic latent semantic analysis, adaboost, and co-training. *EURASIP Journal on Information Security*, 2012.

[23] J.W. Scheeres. *Establishing the Human Firewall: Reducing an Individual's Vulnerability to Social Engineering Attacks*. Biblioscholar, 2012.

[24] J. Searle. *Speech acts: an essay in the phiolosophy of language*. Cambridge University Press, 1969.

[25] Glen Shires and Hans Wennborg. Web speech api specification. Technical report, Speech API Community Group, 2012.

[26] SpamAssassin Open Source Spam Filter. http://spamassassin.apache.org.

[27] Thad Starner. Project glass: An extension of the self. *IEEE Pervasive Computing*, 12(2), April 2013.

[28] *2013 Data Breach Investigations Report*. Verizon, 2013.

[29] Michael Workman. A test of interventions for security threats from social engineering. *Inf. Manag. Comput. Security*, 16(5), 2008.

[30] Yue Zhang, Jason I. Hong, and Lorrie F. Cranor. Cantina: A content-based approach to detecting phishing web sites. In *Proceedings of the 16th International Conference on World Wide Web*, 2007.