Fuzzing E-mail Filters with Generative Grammars and N-Gram Analysis

Sean Palka
George Mason University

Damon McCoy
International Computer Science Institute

Abstract

Phishing attacks remain a common attack vector in today's IT threat landscape, and one of the primary means of preventing phishing attacks is e-mail filtering. Most e-mail filtering is done according to a either a signature-based approach or using Bayesian models, so when specific signatures are detected the e-mail is either quarantined or moved to a Junk mailbox. Much like anti-virus, though, a signature-based approach is inadequate when it comes to detecting zero-day phishing e-mails, and can often be bypassed with slight variations in the e-mail contents. In this paper, we demonstrate an approach to evaluating the effectiveness of e-mail filters using a fuzzing strategy. We present a system that utilizes generative grammars to create large sets of unique phishing e-mails, which can then be used for fuzzing input against e-mail filters. Rather than creating random text, our approach maintains a high degree of semantic quality in generated e-mails. We demonstrate how our system is able to adapt to existing filters and identify contents that are not detected, and show how this approach can be used to ensure the delivery of e-mails without the need to white-list.

1 INTRODUCTION

Fuzzing is a technique commonly employed when testing boundary conditions and constraints of a system or protocol. The general idea is to provide a wide range of variations on input parameter values to determine whether various conditions and state transitions are handled appropriately. By testing boundary conditions, fuzzing is often able to identify weaknesses or vulnerabilities that can be exploited.

Fuzzing is a brute force attack, in that the fuzzing tests provide a massive amount of variations in hopes of finding a few that might cause unexpected conditions. However, there is no guarantee that fuzzing will discover any exploitable vulnerabilities. While fuzzing is sometimes employed by attackers in the wild, for example when targeting web-based applications and services, the nature of the fuzzing is very loud. This type of activity is easily noticed, often raises alarms, and puts network defenders in a heightened state of awareness. As such, fuzzing is usually performed offline whenever possible, where it can be automated by the attackers to avoid detection.

There are several types of fuzzing tools available, depending on the type of system or protocol being tested [16, 43, 48, 51]. Pattern- or list-based fuzzers are not very intelligent in that they will try a series of predetermined inputs and variations on those inputs, and report the outcome. More intelligent fuzzers, on the other hand, will vary the tests according to previous responses, and are able to more efficiently find error conditions that might be exploitable. While fuzzers have been developed for testing most types of systems and protocols, fuzzing techniques are far more difficult to apply to social engineering attacks like phishing.

In this paper, we describe our approach to fuzzing e-mail contents for phishing attacks in order to identify gaps in e-mail filters. We utilized PhishGen, a previously released open-source software solution for dynamically generating phishing e-mail contents using grammars, to create the e-mails. We then applied a technique called n-gram analysis, in which sentences are broken smaller fragments for analysis, to determine which sentence structures resulted in filtering. Our analysis used detected feedback to adjust production rules in the grammar, so that PhishGen focused on successful rules rather than rules that may have produced content that was filtered. Instead of generating random text though, as might be expected from a fuzzing utility, the system is able to

maintain semantic validity in the e-mails that are generated.

2 BACKGROUND

Many research approaches to phishing attack detection focus on either detecting the incoming message at the mail server, or detecting the e-mail inside the mail client once downloaded locally by a user. Research has focused on analyzing the incoming e-mail for suspicious elements, such as obfuscated links or suspicious phrases, or using feature selection algorithms to try and discriminate between phishing e-mails and normal e-mails [4,6,9,32]. Once identified, a phishing e-mail can then be removed from the users work stream, and the risk of them clicking on malicious links or attachments is completely negated.

For the most part, technical controls are limited in their attempt at preventing phishing in live networks for three reasons. First, most theoretical models for detecting or preventing phishing have limitations that make them unsuitable for deployment in live networks [2]. They may be too slow, or have a high false positive rate that makes automated filtering unreliable. Second, users that are susceptible to phishing will often disregard or even bypass warnings, and in some cases actually disable detection tools [39, 50]. Finally, as a social engineering attack, phishing is constantly evolving to bypass technical controls and target the user. Once a technical control is made available to the public, the attackers will often find loopholes or vulnerabilities that will allow their attacks to bypass the control [20, 24]. So, while technical solutions such as browser add-ins and e-mail filtering provide some level of protection, they cannot at present provide a fully effective solution to the phishing problem.

2.1 ETHICAL CONSIDERATIONS

PhishGen is available by request to researchers interested in developing phishing prevention techniques and detection algorithms. The intended purpose is to provide automated content generation for use in training algorithms, but as we demonstrate in this paper it also has potential offensive capabilities. There is always some concern that such tools give additional capabilities to the bad guys. In the wild, phishers and spammers already have numerous tools to execute complex attacks [33, 35, 38]. They also use underground forums to share and disseminate new tools [34,49]. Additionally, it is already well established that existing signature-based filters can be bypassed by tweaking key features, and attackers already have means of testing their content to bypass filters [27].

When used as a fuzzing tool, PhishGen tends be very noisy. In order to adapt to filters, multiple rounds of e-mails are typically required, and as with most fuzzing tools a fairly large number of test cases are required. An attacker would probably not be interested in these multiple rounds of follow-on attacks once they have received a successful response during an actual attack. So as a tool for the bad guys, PhishGen would probably be considered fairly cumbersome. For network defenders, on the other hand, PhishGen can be used to simulate multiple attacks without involving users or risking actual data breaches, and as such could be a valuable assessment tool.

3 RELATED WORKS

Due to the regularity of phishing attacks, it has become a heavily researched area in academia. The majority of available research focuses on detection methods, preventative approaches, or analysis of why users fall for phishing attacks. A wide range of existing research focuses on analysis of the incoming e-mail for suspicious elements [3, 5, 15, 19, 44], or on detecting fraudulent web sites [17, 25, 30, 42]. Finally, some researchers have developed technical components for browsers that prevent access or warn the user when something suspicious is detected [11, 18, 37].

Some attempts have also been made to model the overall attack process of phishing [10, 12, 21, 22, 26], and several studies have also been performed on the effectiveness of various technical controls [14, 23, 24, 47, 50]. Several studies have been performed on why users fall for phishing attacks [12, 13, 40, 45, 46], and the effectiveness of various training options as well [7, 8, 28, 29, 41].

A fairly common practice for network defense is to setup signature-based countermeasures, similar to how many anti-virus solutions work. For example, by providing signatures for specific keywords and phrases like 'Viagra' or links that say 'click here', some spam and phishing attacks can be quickly identified. Additionally, references to known-malicious websites, domains, or IP addresses can be used in a blacklisting approach, or alternatively site reputation can be used as a filtering method [31]. While PhishGen does contain functionality for URL obfuscation, our fuzzing technique is focused primarily on evading content-based filtering by modifying grammatical elements. We assume that an attacker can test IP addresses and domain names/URL against common filtering techniques to mitigate the chance that they are blacklisted, however it is much more difficult to identify grammatical elements of an e-mail that might

result in filtering.

As mentioned in previous sections, some research has already been performed in the area of filter bypass using synonym replacement [27]. In this type of approach, filters that are triggered by specific terms are bypassed using synonyms, for example by substituting the phrase 'authentication tokens' for 'passwords'. This results in content that may bypass filters, but will often become illegible or confusing as more obscure synonyms are required. In our previous research, we showed how obvious or grammatically irrelevant content results in significantly lower click-through rates than well-crafted and convincing content [36]. In our study, no users clicked on links in an e-mail that contained gibberish text, while In order to maintain effective content, which differentiates our approach from previous research, we incorporate a feedback function that allows PhishGen to learn effective techniques without sacrificing semantic content. So instead of substituting synonyms for particular words or phrases, PhishGen will utilize entirely different grammatical structures or alter the entire logic of the e-mail based on successful feedback.

4 SYSTEM DESIGN

In this section we present the key design components within PhishGen that enable it to be used as a fuzzer for e-mail filters. These include a semantic validity engine, n-gram analysis. and an adaptive rule weighting component. While a normal protocol fuzzer could easily test a wide variety of input strings sent to a mail server, we are interested primarily in those e-mail messages that are both semantically valid and could be used in an effective attack. These messages must be able to bypass existing filters, but also maintain semantic integrity so that a user might be tricked into responding. Semantic integrity is an important concept, because phishing attacks often rely on coercion through effective messaging. Our semantic validity engine ensures that the messages are coherent, while the adaptive rule weighting allows for variations in generated content [36]. Meanwhile our n-gram analysis technique allows smarter decisions when applying production rules to the e-mail generation process.

4.1 Semantic Validity Engine

Generating random character strings or blocks of arbitrary text that would not get flagged by anti-phishing detection capabilities is fairly simple, however such an approach provides very little benefit as it does not reflect the type of actual payloads that would be used in phishing attacks. As mentioned earlier, these types of e-mails have been shown to result in decreased click-through rates as well, and so would not be considered useful examples. To provide realistic content as part of the fuzzing process, PhishGen utilizes a modified context-free grammar (CFG) that is is able to create multiple variations of an e-mail according to various production rules. By utilizing this rule-based text generation approach, the tool is able to generate semantically valid text that performs as well as manually generated e-mails in live exercises [36]. So while the tool is identifying gaps in detection capabilities by showing how certain e-mails make it past detection measures, it is also generating content that reflects realistic attack vectors. The same generated content could also be used as part of a live phishing exercise or training initiatives.

In addition to generating useful content, PhishGen is also able to provide a method for identifying potential updates to existing countermeasures. Simply providing an example of an e-mail that makes it past filters is insufficient, a useful fuzzing tool must be able to identify specific features within a generated e-mail that should be included in existing signature databases. PhishGen is able to mark specific production rules as suspicious, and include markers within the generated e-mail text. During the template generation phase, suspicious elements are bookended with unique HTML tags that can be used by administrators to pull useful signatures out of successful e-mails, while not affecting the overall look-and-feel of the e-mail within a browser or e-mail client. For example, the following block of HTML would not affect how an e-mail looks:

```
<a href='http://www.13.org'> CLICK
 HERE <susp13414>PLEEZ</susp13414>
                </a>
```

However, after it has been confirmed as successfully evading filters, the specific signature within the HTML can be identified by the `<susp13414>` tag, and administrators can add the signature to a countermeasure signature databases.

4.2 N-Gram Analysis

PhishGen maintains a list of potentially bad production rule combinations that are determined through n-gram analysis. These rule combinations are used to overrule decisions within the CFG when it makes choices between two rules with the same weight, or when there is sufficient evidence that a rule is resulting in the e-mail being filtered or ignored. When a decision is made to choose one production rule over another, PhishGen first looks

$1 \rightarrow 3 \rightarrow 5 \rightarrow 7$ (n=4)			
1-grams	2-grams	3-grams	4-grams
1	$1 \rightarrow 3$	$1 \rightarrow 3 \rightarrow 5$	$1 \rightarrow 3 \rightarrow 5 \rightarrow 7$
3	$3 \rightarrow 5$	$3 \rightarrow 5 \rightarrow 7$	
5	$5 \rightarrow 7$		
7			

Table 1: N-grams for $1 \rightarrow 3 \rightarrow 5 \rightarrow 7$

at the rule weight, and then looks at the list of potentially bad rule combinations to see if there is an optimal choice. This list of bad production rule combinations is accomplished by doing an n-gram analysis of rule combinations based on positive and negative feedback from the exercise environment.

To generate an e-mail, PhishGen expands production rules until there are no more available production rules for expansion. For each generated e-mail, the software keeps track of the rules chosen and the order they were chosen in, resulting in a chain of production rules that act as a signature for that e-mail. So, one e-mail might be generated by applying production rule 1, followed by production rule 3, then 5 and 7 in that order. For ease of reading, we can represent this chain of production rules as $1 \rightarrow 3 \rightarrow 5 \rightarrow 7$.

Table 1 illustrates the results of breaking out $1 \rightarrow 3 \rightarrow 5 \rightarrow 7$ for n-gram analysis. First, a list must be created of all sequences of production rules from length 1 to n, where n is the number of production rules in the original chain. For $1 \rightarrow 3 \rightarrow 5 \rightarrow 7$, n=4 because the chain is 4 production rules long, so the software would look at sequences of production rules that are length 1, 2, 3 and 4. A different signature of $1 \rightarrow 3 \rightarrow 5 \rightarrow 7 \rightarrow 8$ would have n=5 and PhishGen would look at 1-grams, 2-grams, 3-grams, 4-grams, as well as 5-grams. The longer the chain of production rules, the larger the set of sub-sequences that are analyzed. Once the list is generated, PhishGen assumes that each of these subsequences of rules will cause an e-mail to be flagged, and so they are classified as "presumed bad". If PhishGen detects a successful response from the exercise environment that can be attributed to that e-mail, any subsequence used for that e-mail can then be marked as "known good" because it was not quarantined.

Figure 1 illustrates an example of discriminating "known good" and "potentially bad" sequences of production rules using n-gram analysis, and how this granular approach can find specific sub-sequences that might result in an e-mail being detected. After generating the e-mails, PhishGen marks all n-grams from the associated production rule chains as "potentially bad". However, when a response comes back for the e-mail that was generated by the signature $3 \rightarrow 5 \rightarrow 6 \rightarrow 8$, PhishGen knows the associated n-grams with that signature did not cause the e-mail to be quarantined. For example, it knows that choosing the sequence of production rules $3 \rightarrow 5 \rightarrow 6$ is safe, and so it removes that 3-gram from the "potentially bad" list. It is important to note, from Figure 1, that the sequence $3 \rightarrow 5 \rightarrow 7$ is still marked as "potentially bad", even though $3 \rightarrow 5$ is not. This is because that sequence of production rules was associated with a different e-mail that did not result in a response from the exercise environment. It could be that production rule 7 is a rule that causes an e-mail to become filtered, perhaps by adding a word to the e-mail that triggers countermeasures.

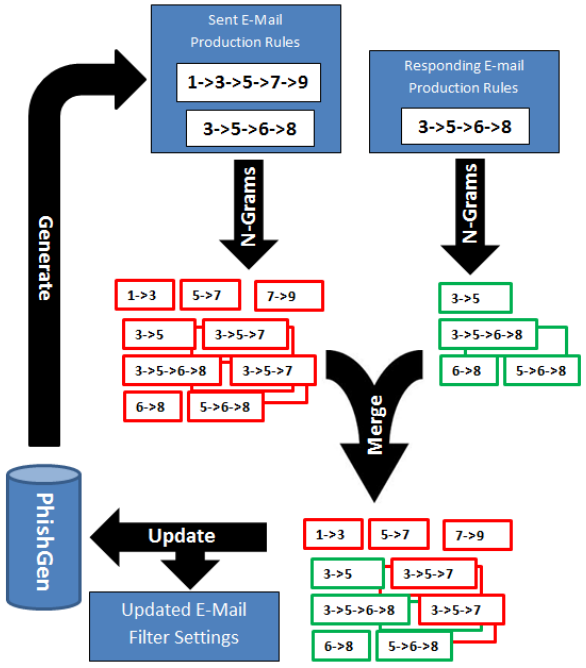

Figure 1: N-gram analysis process used to update list of "presumed bad" and "known good" production rules.

In general, PhishGen assumes the worst and favors rule combinations that are either known to be good, or at least are not included in the bad combination list. New sequences of production rules are not prevented from being chosen, but once a rule is used to create an e-mail it is considered to be ineffective until a response is received from the exercise environment. As such, PhishGen will continue to utilize new rules as well as recombining previously validated sequences of production rules.

4.3 Adaptive Rule Weighting

One of the fundamental problems with CFGs is that they can create an infinite set of strings if the rules are not properly designed. By definition, a CFG cannot determine a specific number of times a rule has been applied, and so there must be some external control in place to either validate rules in the grammar, or else place a limitation on the number of times a rule can be applied. To address this issue, PhishGen utilizes a probabilistic context-free grammar (PCFG) and reduces the likelihood of using a rule each time it is applied. Additionally, prior to generating e-mail templates, PhishGen provides a method for evaluating a given set of grammar rules to ensure that all of the rules are valid, and that every rule has a terminating branch.

In order to prevent repetition of rules, and generally force a more diverse set of generated content, PhishGen alters the associated weight of a rule every time it is used. So during each iteration, a given rule is unlikely to be chosen twice until all other available rules have also been selected. This feature also has a unique benefit, in that PhishGen can also adjust rule weights based on external stimulus. For example, while rule weights are decreased during selection, they are also increased when a successful response is detected from the exercise domain. Each e-mail generated by PhishGen is logged, as well as the production rules used to generate it, so when a response is received the weights of all associated production rules can be adjusted.

PhishGen decreases the weight of a production rule every time a rule is used, and increases the weight when a rule is part of an e-mail that results in a successful response from the testing environment. In this way the system gradually phases out less successful rules in favor of rules that have been validated by user responses. Similarly, if a rule results in an e-mail being quarantined or filtered, that rule will eventually be phased out because users will never receive the e-mail, and thus cannot respond. Just because a rule is successful once, however, does not mean it will always be used. Each time the rule is used the weight is decreased, and so if users gradually acclimate to detecting specific language, the rule will eventually be phased out in favor of other rules that generate content that have not had as much exposure.

4.4 Fuzzing Capabilities

While the semantic validity engine is used to create grammatically and semantically relevant content, the adaptive rule weighting and n-gram analysis components are what allows the tool to be used as an intelligent fuzzer. The n-gram analysis focuses on sequences of production rules, and informs the decision making process by avoiding blocks of text that might be resulting in detection. The goal of this analysis is to avoid potentially bad combinations of production rules. The adaptive weighting, on the other hand, focuses on selection strategies for individual production rules, and ensures that rules are not being overused based on performance.

5 EXPERIMENTAL DESIGN

To test the fuzzing capability of PhishGen, we developed a series of simulations that would measure the ability to bypasses existing countermeasures. In this section we describe how the content was created for these simulations, and the environments that were tested against. An archive containing the production rule template used by PhishGen to create all simulation e-mails in this study, as well as copies of the individual e-mails for each round of simulations, is available for download at :

`http://www.diskread.com/fuzzing.zip`

This archive can be used to validate that our approach was not simply replaying successful e-mails, but that new contents were being generated for each round of testing. The archive, as well as a copy of the PhishGen software, can also be requested by sending an e-mail to spalka@gmu.edu .

5.1 Content Generation

E-mail content for each simulation was generated by PhishGen using a set of production rules that had been previously validated in other research [36]. Specifically, we utilized a template that contained a series of production rules to generate an e-mail asking users to click links in order to update a password or user profile. Figure 2 illustrates an example e-mail generated from this template. This e-mail template was chosen because it contained some rules that would generate clearly suspicious content that would presumably trigger existing detection capabilities, but it had also been demonstrated as very effective in previous live exercises. In all, the production rules template contained 129 individual production rules, and took less than an hour to develop.

In many cases, templates that we developed for testing went completely undetected in the various environments that were being tested, and so were excluded from the results reported in this paper. The focus on this paper is to demonstrate the learning capability that supports fuzzing, which is best illustrated when e-mails are ini-

tially detected. However we have provided one such template in the archive for demonstration purposes, so that researchers could test against other environments not included in this analysis.

For each simulation, PhishGen was first used to create 100 random e-mails, and these e-mails were then sent through to the specific environment being evaluated. All e-mails for all rounds of testing were generated from the same set of production rules for consistency. Information about the emails that were not filtered was then fed back into PhishGen so that it could adjust the weights on production rules and update the n-gram analysis tables for the next round of testing. Each round of simulations utilized 4 sets of 100 e-mails, with a total of 3,200 generated e-mails being sent overall.

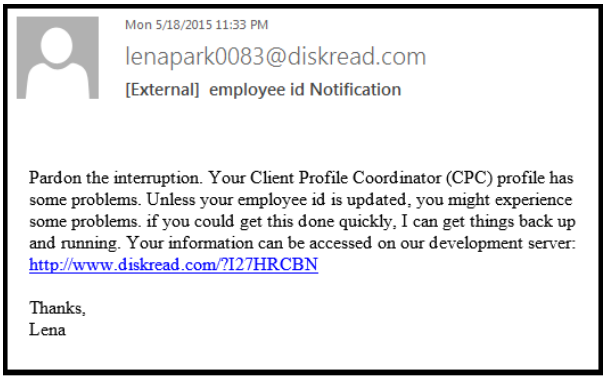

Figure 2: Example of an e-mail generated by PhishGen

5.2 Tested Environments

The first environment that we used for testing our fuzzing approach was a corporate environment. Specifically, we were allowed to run simultations within the production e-mail environment for a large commercial firm, given that our simulations were not targeting users. This environment supported business operations for over 20,000 users, and was configured with multiple phishing and spam detection measures. This environment primarily utilized a signature-based approach to detection, but also incorporated commercial solutions like Cisco IronPort spam filtering. There were likely additional detection measures in place that we were unaware of, however this illustrates how our fuzzing approach can still be used in black-box environments. For our e-mail client, we used the same configuration used by the company user population, which was a Microsoft Outlook 2013 installation with McAfee E-mail Security installed as an add-in. We did modify the base configuration to have the Junk E-mail Options set to high, so that we could provide as restrictive an environment as possible. This environment was representative of the types of e-mail environments we have seen at other companies.

The second environment utilized SpamAssassin with Bayesian-style probabilistic classification, as an alternative to a signature-based approach. SpamAssassin was trained on 795,092 spam and phishing e-mails sourced from a spam codex maintained by Untroubled Software (`http://untroubled.org/spam/`) and a scam e-mails database maintained by Scamdex (`http://www.scamdex.com/`).

Initially, we were interested in testing our approach against online e-mail systems like Gmail. However, after testing several types of e-mails and production rules, we determined that these environments were not useful for demonstrating our fuzzing approach because they failed to identify any of our generated e-mails as suspicious. For example, the example e-mails that were detected by SpamAssassin and the corporate e-mail environment were not filtered when sent to a Gmail account. We believe the most important reason for this is the lack of exposure. Gmail looks for viruses and malware signatures, patterns within e-mails, and most importantly it learns from user-reported e-mails [1]. For example, Gmail will provide warnings for messages with similarity to known suspicious messages, detected spoofed sender addresses, known phishing attacks, and even empty message contents. However, Gmail relies heavily on reports of phishing and spam by users, and our e-mails were never sent in the wild. Our e-mails were not sent with malicious attachments or virus signatures, but surprisingly they did not meet any thresholds that would classify the e-mails as suspicious either. While this does not mean Gmail cannot detect any e-mail generated by PhishGen, it does indicate the difficulty in identifying ”zero-day” e-mails in general.

6 RESULTS

For our testing we had PhishGen working off of perfect information from the exercise environment. Any e-mail that made it to the Inbox without modification was considered ”successful” and provided as feedback to the software. In a live environment, not every e-mail would result in a human response, and so the software would have less information per round of e-mails on which to base weight adjustments. Also, PhishGen would assume that a non-response was due to filtering, rather than through failure to coerce a human response. This is a unique capability of PhishGen, because in a live envi-

Environment	E-mail Round			
	1	2	3	4
Corporate	19%	12%	0%	0%
Corporate	14%	0%	0%	0%
Corporate	18%	1%	0%	0%
Corporate	15%	1%	1%	0%
SpamAssassin	15%	1%	0%	0%
SpamAssassin	9%	1%	0%	0%
SpamAssassin	10%	0%	0%	0%
SpamAssassin	20%	1%	0%	0%

Table 2: Detection rates for multiple simulations across two test environments.

ronment PhishGen will ultimately favor production rules that trigger human responses as well. For the purposes of our research to show how PhishGen can be used as an intelligent fuzzer, though, we removed the human variable from the simulations.

Table 2 shows the percentage of detected e-mails for both environments across multiple simulations. In all, we ran four sets of simulations in each environment, with each simulation utilizing 4 rounds of 100 e-mails. In both environments, PhishGen was able to get 100% of e-mails through the filters within 3 rounds of e-mails being sent. This indicates that the n-gram analysis approach was able to identify production rules that were resulting in e-mails being filtered, which then informed the subsequent rounds of e-mail generation.

Overall, there was very little difference in detection capabilities between the SpamAssassin and Corporate test environments. Both appeared to catch around 15% of the e-mails in the first round, with a significant drop in detection for the second round. In one simulation, the Corporate environment was able to catch more than 10% of the second round e-mails, but then the detection rate dropped to zero in the third round of e-mails. While this only appeared once, it does show that the approach can sometimes misinterpret the feedback from the exercise environment, or employ previously unselected production rules that result in filtering.

7 CONCLUSIONS

The results of our simulations show that PhishGen was successful in evading filters over the course of several simulations, regardless of the type of countermeasures that were deployed. This was primarily due to the n-gram analysis and adaptive rule weighting schemes that were utilized, which allowed the system to learn which production rules were effective.

From our results, we can see that both environments appeared to be filtering based on some threshold values that were not met by the majority of our e-mails, and those thresholds were primarily based on suspicious keywords. For SpamAssassin, this is likely due to the fact that we did not include a lot of keywords that occur regularly in phishing or spam e-mails, such as 'Viagara' or 'click here'. In the event those keywords were used, and detected, our system avoided them in future generations. In the Corporate environment, there were far more variables involved, and there may have been some security measures that were not disclosed to us, so it was a true black-box test of the environment. However, we can still see from our results that the same issue appears to affect this system. Our content did not meet whatever thresholds were set, at any point between the mail server and the mail client, and so the e-mails made it through.

For the most part, we feel that this is primarily due to the same issue that we encountered with Gmail, in that zero-day e-mails are hard to detect because filters cannot predict what contents will be suspicious in the future. Once an e-mail is reported, it becomes far easier to reduce impact on a user population if it is seen again. The same is true if particular words, phrases, suspicious domains, or other known-bad content is used. Otherwise, though, an e-mail will simply blend in with normal traffic. Our approach was successful because we are crafting semantically meaningful content using words that are, for the most part, benign. We break down the e-mail into smaller pieces, and eventually remove pieces that are potentially getting flagged. Current detection measures cannot evaluate the intent of an e-mail, and they contain no previously identified suspicious content there is no reason to mark zero-day attacks as malicious.

With the ability to generate dynamic sets of e-mails that identify gaps in filter coverage, PhishGen can be used for a variety of purposes from penetration testing to email filter tuning. For example, the approach can be used as part of live phishing exercises to identify e-mails that are not quarantined or flagged as suspicious, which can provide an additional level of realism by avoiding white-listing requirements.

8 ACKNOWLEDGMENTS

We would like to thank George Mason University and the Volgenau School of Engineering for their support and constructive criticism during the development of PhishGen. Additionally. we would also like to acknowledge the great feedback provided by the anonymous reviewers of this paper, especially Dr. Collin Mulliner for his contributions shepherding this paper. This work was

supported by National Science Foundation grant NSF-1237076.

References

[1] Spam and suspicious emails - gmail help.

[2] Ammar Almomani, B. Gupta, Samer Atawneh, A. Meulenberg, and Eman Almomani. A survey of phishing email filtering techniques.

[3] Ammar Almomani, B. B. Gupta, Tat-chee Wan, Altyeb Altaher, and Selvakumar Manickam. Phishing dynamic evolving neural fuzzy framework for online detection zero-day phishing email. 2013.

[4] Andr Bergholz, Jan De Beer, Sebastian Glahn, Marie-Francine Moens, Gerhard Paa, and Siehyun Strobel. New filtering approaches for phishing email. 18(1):7–35.

[5] Andr Bergholz, Jan De Beer, Sebastian Glahn, Marie-Francine Moens, Gerhard Paa, and Siehyun Strobel. New filtering approaches for phishing email. 18(1):7–35, 2010.

[6] Andre Bergholz, Jeong Ho Chang, Gerhard Paa, Frank Reichartz, and Siehyun Strobel. Improved phishing detection using model-based features. In *CEAS*.

[7] Daniel Bliton, Aimee Norwood, and Sean Palka. Unannounced phishing exercises and targeted training: Results and lessons learned. In *The Interservice/Industry Training, Simulation & Education Conference (I/ITSEC)*, volume 2011, 2011.

[8] Brian M. Bowen, Ramaswamy Devarajan, and Salvatore Stolfo. Measuring the human factor of cyber security. In *Technologies for Homeland Security (HST), 2011 IEEE International Conference on*, pages 230–235, 2011.

[9] Madhusudhanan Chandrasekaran, Krishnan Narayanan, and Shambhu Upadhyaya. Phishing email detection based on structural properties. In *NYS Cyber Security Conference*, pages 1–7.

[10] Richard Dazeley, John L. Yearwood, Byeong H. Kang, and Andrei V. Kelarev. Consensus clustering and supervised classification for profiling phishing emails in internet commerce security. In *Knowledge Management and Acquisition for Smart Systems and Services*, pages 235–246. Springer, 2010.

[11] Rachna Dhamija and J. Doug Tygar. The battle against phishing: Dynamic security skins. In *Proceedings of the 2005 symposium on Usable privacy and security*, pages 77–88, 2005.

[12] Rachna Dhamija, J. Doug Tygar, and Marti Hearst. Why phishing works. In *Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 581–590, 2006.

[13] Julie S. Downs, Mandy B. Holbrook, and Lorrie Faith Cranor. Decision strategies and susceptibility to phishing. In *Proceedings of the second symposium on Usable privacy and security*, pages 79–90, 2006.

[14] Serge Egelman, Lorrie Faith Cranor, and Jason Hong. You've been warned: an empirical study of the effectiveness of web browser phishing warnings. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1065–1074, 2008.

[15] Ian Fette, Norman Sadeh, and Anthony Tomasic. Learning to detect phishing emails. In *Proceedings of the 16th international conference on World Wide Web*, pages 649–656, 2007.

[16] Patrice Godefroid, Adam Kiezun, and Michael Y. Levin. Grammar-based whitebox fuzzing. In *ACM Sigplan Notices*, volume 43, pages 206–215. ACM.

[17] Mingxing He, Shi-Jinn Horng, Pingzhi Fan, Muhammad Khurram Khan, Ray-Shine Run, Jui-Lin Lai, Rong-Jian Chen, and Adi Sutanto. An efficient phishing webpage detector. 38(10):12018–12027.

[18] Chun-Ying Huang, Shang-Pin Ma, and Kuan-Ta Chen. Using one-time passwords to prevent password phishing attacks. 34(4):1292–1301.

[19] Huajun Huang, Liang Qian, and Yaojun Wang. A SVM-based technique to detect phishing URLs. 11(7):921–925.

[20] Danesh Irani, Steve Webb, Jonathon Giffin, and Calton Pu. Evolutionary study of phishing. In *eCrime Researchers Summit, 2008*, pages 1–10.

[21] Collin Jackson, Daniel R. Simon, Desney S. Tan, and Adam Barth. An evaluation of extended validation and picture-in-picture phishing attacks. In *Financial Cryptography and Data Security*, pages 281–293. Springer.

[22] Markus Jakobsson. Modeling and preventing phishing attacks. In *Financial Cryptography*, volume 5, 2005.

[23] Markus Jakobsson and Steven Myers. *Phishing and countermeasures: understanding the increasing problem of electronic identity theft*. Wiley. com.

[24] Markus Jakobsson and Steven Myers. *Phishing and Countermeasures: Understanding the Increasing Problem of Electronic Identity Theft*. Wesley, 2006.

[25] Hansi Jiang, Dongsong Zhang, and Zhijun Yan. A classification model for detection of chinese phishing e-business websites.

[26] HarishBabu Kalidasu, B. Prasanna Kumar, and KV Ajay Kumar. Battle against for phishing based on captcha security. 1(5):497–519.

[27] Christoph Karlberger, Gnther Bayler, Christopher Kruegel, and Engin Kirda. Exploiting redundancy in natural language to penetrate bayesian spam filters. In *First USENIX Workshop on Offensive Technologies (WOOT07), Boston, MA*.

[28] Ponnurangam Kumaraguru, Justin Cranshaw, Alessandro Acquisti, Lorrie Faith Cranor, Jason Hong, Mary Ann Blair, and Theodore Pham. School of phish: A real-word evaluation of anti-phishing training (CMU-CyLab-09-002). 2009.

[29] Ponnurangam Kumaraguru, Yong Rhee, Alessandro Acquisti, Lorrie Faith Cranor, Jason Hong, and Elizabeth Nunge. Protecting people from phishing: the design and evaluation of an embedded training email system. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 905–914, 2007.

[30] Gang Liu, Bite Qiu, and Liu Wenyin. Automatic detection of phishing target from phishing webpage. In *Pattern Recognition (ICPR), 2010 20th International Conference on*, pages 4153–4156, 2010.

[31] Justin Ma, Lawrence K. Saul, Stefan Savage, and Geoffrey M. Voelker. Beyond blacklists: learning to detect malicious web sites from suspicious URLs. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1245–1254. ACM.

[32] Sebastan Maldonado and Gaston LHuillier. SVM-Based feature selection and classification for email filtering. In *Pattern Recognition-Applications and Methods*, pages 135–148. Springer.

[33] David Maynor. *Metasploit Toolkit for Penetration Testing, Exploit Development, and Vulnerability Research*. Elsevier.

[34] Marti Motoyama, Damon McCoy, Kirill Levchenko, Stefan Savage, and Geoffrey M. Voelker. An analysis of underground forums. In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*, pages 71–80. ACM.

[35] Gunter Ollmann. The evolution of commercial malware development kits and colour-by-numbers custom malware. 2008(9):4–7.

[36] Sean Palka and Damon McCoy. Dynamic Phishing Content Using Generative Grammars. In *Tenth IEEE/ACM International Workshop on Automation of Software Test*, 2015.

[37] Bryan Parno, Cynthia Kuo, and Adrian Perrig. *Phoolproof phishing prevention*. Springer.

[38] Nikola Pavkovic and Luka Perkov. Social engineering ToolkitA systematic approach to social engineering. In *MIPRO, 2011 Proceedings of the 34th International Convention*, pages 1485–1489. IEEE.

[39] Troy Ronda, Stefan Saroiu, and Alec Wolman. Itrustpage: a user-assisted anti-phishing tool. In *ACM SIGOPS Operating Systems Review*, volume 42, pages 261–272. ACM.

[40] Steve Sheng, Mandy Holbrook, Ponnurangam Kumaraguru, Lorrie Faith Cranor, and Julie Downs. Who falls for phish?: a demographic analysis of phishing susceptibility and effectiveness of interventions. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 373–382, 2010.

[41] Steve Sheng, Bryant Magnien, Ponnurangam Kumaraguru, Alessandro Acquisti, Lorrie Faith Cranor, Jason Hong, and Elizabeth Nunge. Anti-phishing phil: the design and evaluation of a game that teaches people not to fall for phish. In *Proceedings of the 3rd symposium on Usable privacy and security*, pages 88–99, 2007.

[42] Pravin Soni, Shamal Firake, and B. B. Meshram. A phishing analysis of web based systems. In *Proceedings of the 2011 International Conference on Communication, Computing & Security*, pages 527–530, 2011.

[43] Michael Sutton, Adam Greene, and Pedram Amini. *Fuzzing: brute force vulnerability discovery*. Pearson Education.

[44] F. Toolan and J. Carthy. *Feature selection for Spam and Phishing detection. eCrime Researchers Summit (eCrime), 2010*. IEEE.

[45] Arun Vishwanath, Tejaswini Herath, Rui Chen, Jingguo Wang, and H. Raghav Rao. Why do people get phished? testing individual differences in phishing vulnerability within an integrated, information processing model. 51(3):576–586.

[46] Ryan T. Wright and Kent Marett. The influence of experiential and dispositional factors in phishing: An empirical investigation of the deceived. 27(1):273–303.

[47] Min Wu, Robert C. Miller, and Simson L. Garfinkel. Do security toolbars actually prevent phishing attacks? In *Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 601–610, 2006.

[48] Zhi-yong Wu, Hong-chuan Wang, Le-chang Sun, Zu-lie PAN, and Jing-ju LIU. Survey of fuzzing. 27(3):829–832.

[49] Michael Yip. An investigation into chinese cybercrime and the underground economy in comparison with the west.

[50] Yue Zhang, Serge Egelman, Lorrie Cranor, and Jason Hong. Phinding phish: Evaluating anti-phishing tools. 2006.

[51] W. U. Zhi-yonga, X. I. A. Jian-juna, S. U. N. Le-changa, and ZHANG Minb. Survey of multi-dimensional fuzzing technology [j]. 8:004.