

Fighting Unicode-Obfuscated Spam

Changwei Liu
Indiana University
changwei@gmail.com

Sid Stamm
Indiana University
sstamm@indiana.edu

ABSTRACT

In the last few years, obfuscation has been used more and more by spammers to make spam emails bypass filters. The standard method is to use images that look like text, since typical spam filters are unable to parse such messages; this is what is used in so-called “rock phishing”. To fight image-based spam, many spam filters use heuristic rules in which emails containing images are flagged, and since not many legit emails are composed mainly of a big image, this aids in detecting image-based spam. The spammers are thus interested in circumventing these methods. Unicode transliteration is a convenient tool for spammers, since it allows a spammer to create a large number of homomorphic clones of the same looking message; since Unicode contains many characters that are unique but appear very similar, spammers can translate a message’s characters at random to hide black-listed words in an effort to bypass filters. In order to defend against these unicode-obfuscated spam emails, we developed a prototype tool that can be used with SpamAssassin to block spam obfuscated in this way by mapping polymorphic messages to a common, more homogeneous representation. This representation can then be filtered using traditional methods. We demonstrate the ease with which Unicode polymorphism can be used to circumvent spam filters such as SpamAssassin, and then describe a de-obfuscation technique that can be used to catch messages that have been obfuscated in this fashion.

Keywords

Spam emails, Unicode characters, obfuscated emails, de-obfuscated emails, SpamAssassin

1. INTRODUCTION

Since 1998, large quantities of spam email have been sent to recipients’ email accounts. Currently, spam email is estimated to make up 75-80% of the overall volume of email sent [25], which is not only annoying and time-consuming

for users, but also adds to network congestion. Additionally, some spam emails pose a serious threat, in the form of phishing. While most spam is just unsolicited bulk e-mail, phishing spam can be an attempt by attackers to get people to reveal private information (such as bank account information or PIN numbers) by providing deceptive links in those emails. Both types of emails involve some sort of deception centered around obfuscation, or hiding the truth of the message from filters or a quick visual look at the message.

In response to these deceptive problems, people have developed many methods to fight against spam email. At first, blacklists and whitelists were used to block or accept email [3]. Later, private channels between senders and receivers were created as a means to classify good email and spam email [13, 10]. This was done because spammers can change IP addresses very easily, circumventing blacklists and whitelists, and possibly causing users to lose good email [12]. At about the same time, spam filters (tools that identify spam by detecting keywords and phrases in an email message’s headers and body text) were put into practice [7]. An effective, mature and currently widely-deployed open source spam filter called SpamAssassin [26] can catch most of the common spam email by being correctly “trained”, a process involving feeding the filter good and bad email, letting it learn from that data.

However, spammers are very agile. In response to the above methods of blocking spam, they have developed ingenious techniques to circumvent anti-spam tools. With the wide-spread application of excellent spam filters such as SpamAssassin and CRM114, spammers have developed “image-based” spam emails (like the Rock Phishing discussed above), word obfuscation, and other methods to get around the filters [20]. One particularly difficult obfuscation mechanism is HTML redrawing; spammers can slice a message up into columns, and send the columns in the email physically appearing to filters as rows. Using some HTML tricks, the email can display the slices vertically, instead of horizontally as they appear to the filters, essentially rendering text top-to-bottom first instead of left-to-right. Spam filtering software sees the columns in order, more-or-less garbage, but when rendered on the screen, people can read across the columns and decipher the spam.

Among all these circumvention techniques, word obfuscation is a very prevalent method used by spammers. In general, they use following techniques to hide spam words from filters: misplaced spaces, purposeful misspelling, embedded special characters, Unicode letter transliteration and HTML redrawing. When these techniques are combined,

spammers can produce a vast number of ways to hide keywords. For example, there are more than six hundred quintillion (600,000,000,000,000,000,000,000) ways available to spell the word Viagra [2]! Dealing with misplaced spaces, purposeful misspelling and embedded special characters has been mentioned in other papers such as [1]. We will focus on Unicode letter substitutions (also called partial-word transliterations). Although Unicode letter substitution is just a small part of obfuscation, as there are many visually or semantically similar characters existing in UCS (Universal Character Set), the spammers can use similar but unequal characters to replace corresponding original English characters, thus producing spam emails that are practically undetectable to most peoples' eyes. This requires that the community adopt new tactics in order to maintain the upper hand against spammers and phishers.

In this paper, we describe a script to emulate the process of replacing English characters at random with Unicode characters. We then show the effect of processing those generated obfuscated messages through SpamAssassin, which assigns scores based on the "spamminess" of a message. The result shows that obfuscated email can bypass SpamAssassin quite easily. We also show how to de-obfuscate the messages so they will be assigned the more accurate, much higher scores. (This is not merely a matter of converting the previous obfuscation, as sometimes, additional language-based heuristics must be used to resolve situations in which there is more than one possible and sensible preimage, e.g., involving characters such as "l" and "1".) As a result, we can integrate the de-obfuscating script into SpamAssassin to fight against obfuscated email messages.

The rest of this paper is organized as follows. Section 2 is related work and background, section 3 describes our specific experimental process, results, analysis, and shortcomings to this technique. We finish up with a summary of our findings and future work.

2. RELATED WORK AND BACKGROUND

Spam emails have created onerous work on email providers as well as email users all over the world. According to the International Messaging Anti-Abuse Working Group, spam accounted for about 80% of all email-traffic on Internet during the first three months of 2006. If providers and users don't take effective anti-spam measures, email might become so inconvenient people will stop using it. Since 1998, considerable progress has been made to stop spam emails. Basically, there are two kinds of anti-spam methods: one prevents spam emails before they reach users' email boxes, and the other removes them from users' inboxes.

2.1 Anti-Spam Techniques

The simplest methods of preventing spam emails before they reach users' email boxes are black list and white list. A black list is a database of IP addresses that send spam emails [12]. In practice, this method is effective, but spammers can easily switch their IP addresses by switching to another computer, using a bot network, or using a different mail relay. By contrast, a white list is a list of "known" senders or IP addresses, and is used to prevent anyone *not* in the list from sending mails to a user's mailbox. Unfortunately, this causes users to miss good emails sometimes, resulting in a loss of unexpected-but-important emails [3].

Better solutions have been developed [13, 10]. R.J. Hall suggests using electronic mail channel identifiers [13]: a sender gives every receiver a specific channel by giving him a unique email address that differs in channel ID but has the same user name (for example, Jim could ask Alice to send him mail at `jim+1@server.com`, and ask Bob to use `jim+2@server.com`). Incoming emails will be divided to different classes according to receivers' email address channels. Those emails that come from an unknown email address will be put in a public channel (like `jim@server.com`) and subject to more scrutiny. A similar method is suggested where a sender and receiver communicate with a unique email address that is composed of a core address and an extension [10]. The extension can be obtained by a handshake between a receiver and a sender, which may cause the receiver to pay a computational or monetary cost. Of course, this monetary cost can be used to "punish" spammers as well.

Both of the above anti-spam techniques can be eavesdropped by attackers on the communication lines. This can be defended against with message authentication codes [16]. Unfortunately, it also adds to more computational cost to sending mail.

2.2 Commonly Used Anti-Spam Tools

The three methods mentioned above can effectively prevent spam emails with complicated "passwords" and computational or monetary cost, but there are some people who simply will not make the effort to complete the complicated process, or do not wish to pay the cost. This shadows the convenience of email and suppresses its usefulness. Because of this, many users resort to email filters to prevent spam emails. Currently, the most popular spam filter is content-based spam filter [19]. Another simple, standard spam filter technique is called Filtering by Duplicate Detection (FDD) [14]. The idea is that casual spammers probably send a couple, very similar emails to the same recipient. This similarity can be used by email software agent to identify spam and delete these emails received more than once. However, Hall points that spam countermeasures based on duplicate detection schemes are foiled if spammers randomly assign users' email addresses to different subsets and send somewhat different versions to each subset [14]. Duplicates then won't be received, but a few very similar ones might. Moreover, spammers can easily increase their effectiveness in combatting FDD by increasing the number of subsets.

Keyword-based filtering and latent semantic indexing are some more examples of content-based filtering techniques. They block spam emails by finding bad words in message headers or body. This is an easy but efficient method; the drawback is that these techniques rely on manually constructed (probably non-exhaustive) keyword lists, and thus perhaps many errors [4].

By continually training a naive Bayesian filter with spam emails and non-spam (ham) emails can get a robust, adaptive and highly accurate system. Particular words have particular, different probabilities of occurring in spam email versus in legitimate email. When being trained, the Bayesian filter will adjust the probabilities that a word appears in spam emails or legitimate emails based on input from the user. Later, the filter is used (out of training) to identify spam based on the results of training.

Both CRM114 [23] and SpamAssassin [26] combine several technologies with Bayesian techniques to fight spam. The creators of Spamassassin even claim that it “typically differentiates successfully between spam and non-spam in between 95% and 99% of cases” [27]. This measurement is based on the SpamAssassin corpus [28], but shows the confidence its creators have over its accuracy.

Spammers can use many different ingenious techniques, including obfuscation, to circumvent these filters. Just as mentioned in the introduction, spammers can make use of Unicode to obfuscate emails while still maintaining “good looking” messages.

2.3 Partial-Word Unicode Transliteration

At a time before Unicode was deployed, there were too many computer encoding systems to satisfy different requirements of people who speak different languages. To some extent, those different encoding systems resulted in conflict. For example, two encodings can use the same code to represent two different characters or use different codes for the same character, which makes it extremely important to carefully translate the characters when moving from one encoding scheme to another. Amidst all the confusion, Unicode was designed to represent almost all standard characters in the world. However, people did not know that it also brings more problems—there are too many similar characters in Unicode, thus making it hard to visually differentiate between characters that may be technically different. Spammers can thus exploit this visual similarity to replace original characters in emails, generating messages without any blacklisted keywords that can bypass spam filters including CRM114 and SpamAssassin. For example, a cyrillic *v* (U+FF56) can be used to replace the latin *v* (U+0076) and the cyrillic *i* (U+0456) to replace the latin *i* (U+0069) in “Viagra” to get a technically different, yet visually similar word. In particular, phishers may use similar characters to replace IRI/IDN in URLs to hide from blacklists, or look visually legit [6]. The IRI/IDN attack and the spam obfuscation are called “Unicode attack”, and are used to send users “perfect looking” phishing emails. These messages include the legit-looking address, in an attempt to persuade users to give out their private information [11, 15, 17].

To fight this “Unicode attack”, a UC-Simlist was structured [6]. In this list, every character is followed with some visually or semantically similar characters. The listed characters are paired with a similarity value from 0.8 to 1, with 1 being *visually or semantically identical*. Though this UC-Simlist just includes English, Chinese and Japanese and a complete version is still needed, it was enough for our tests based on English. Our goal was to replace original English characters with Unicode characters in spam emails, resulting in obfuscated spam emails; we then aimed to convert non-English characters back to form words that might signal spam filters. Our technique will be described in depth later.

Many anti-phishing tools are available to help users protect themselves from phishing attacks. In defense against homograph phishing attacks, the IRI/IDN SecuChecker and browser address/status bars can be used to warn users, sometimes with different colors [5]. Some phishing websites are very well spoofed and hidden, easily fooling users, and several recent studies (see, e.g., [8, 21, 22]) indicate that typical users often ignore warnings, so the best anti-spam method

probably involves simply blocking spam emails, including phishing emails.

This paper is not directly about phishing prevention, so anti-phishing techniques will not be discussed here, though similar techniques to ours may be employed to more-or-less “homogenize” any string (URL or other) into one language or character set.

3. OUR EXPERIMENT

The final goal of our scheme is to block obfuscated spam emails by converting obfuscated characters to original English characters, but, before that, we first emulate attackers by creating obfuscation spam emails and phishing spam emails with similar characters in Unicode.

3.1 Simulating the Threat

Our experiment was conducted in two phases: first we wrote a script to produce obfuscated spam emails from raw spammy samples, replacing original English characters at random with Unicode characters. Next, we wrote a script to de-obfuscate these obfuscated emails by detecting and changing those non-English characters to English characters.

After training SpamAssassin, we used it to process the original (not-yet-obfuscated) spam emails, obfuscated emails and de-obfuscated emails to measure the scores SpamAssassin assigned to each. We then analyzed those scores to see whether or not the obfuscated spam emails can bypass spam filters, but the de-obfuscated ones wouldn’t. In short, the result of the experiment was promising, showing that all obfuscated emails can circumvent the spam filter, but our de-obfuscation script helps SpamAssassin detect the obfuscated emails by converting them to de-obfuscated forms. However, this requires a trade-off of a little computational time since the script has to spend a few moments searching for and removing possible obfuscation.

To improve the speed of the de-obfuscation script, we focus only on reducing to ASCII characters; we just searched for Unicode characters used for obfuscation in the simlists of ASII charaters and then replaced them with corresponding ASCII characters. (In fact, there is no need to find any other characters since our experiment is based on spam emails in English.) The computational time is approximately 5-7 seconds on a 2GHz system for a normal-size email, but the time will be longer for longer emails because the script will spend more time searching and replacing non-English Unicode characters with original English characters. Additionally, our code can be optimized with techniques discussed in Section 4.2 to run more quickly.

We also performed the same experiment on spam phishing emails. Because spam phishing emails are generally well written and “perfect-looking”, not like spam emails peddling erectile disfunction medication or “hot sex” that normally don’t need to look like some authority like a bank. We adjusted our scripts to produce nearly “perfect-looking” spam phishing emails (only substituting English characters for exact-match unicode substitutes). However, the result is the same as mentioned above: the obfuscated emails still pass through the filter, but can be de-obfuscated using our technique.

3.2 Experiment Tools.

UC-Simlist. We used the UC-Simlist to detect “Unicode attacks”. According to visual similarity and semantic similarity, Fu, et al [5, 6] constructed a UC-Simlist, where a character is paired with a set of visually and semantically similar Unicode characters. When replacing a reference character in the original message with a similar target character, we used targets whose similarity to the reference character is equal to 1 to produce decent-looking obfuscated spam emails (this made the change as inobvious as possible). In addition, using only English messages allowed us to use a smaller portion of the UC-Simlist, reducing the amount of computational time required.

SpamAssassin. Using the de-obfuscation technique, any good spam filter like SpamAssassin or CRM114 should produce similar results. We chose to use SpamAssassin as the spam filter since it is free software and widely used. It is also easy to configure and add new rules. Furthermore, it is written in Perl, making it easy to integrate our scripts (also written in Perl). SpamAssassin uses a diverse range of tests to identify spam emails including not only header tests, body phrase tests, white/blacklist, collaborative spam identification databases, DNS Blacklists (“RBLs”), characters sets and locales, but also Bayesian filtering [26] that can be trained by each individual user with spam emails and non-spam (ham) emails. If the Bayesian filter is well-trained, it will catch up to “99.5% of spam with less than 0.03% false positives” [10].

In our experiment, we used SpamAssassin’s public corpus [28], Jose Nazario’s phishing corpus [18] and spam or phishing emails collected at Indiana University/Purdue University in Indianapolis (IUPUI) to train SpamAssassin. We randomly selected 3000 spam emails as well as more than 3000 ham (legit) emails from the corpora to train SpamAssassin. As it would not affect the analysis for the final results, those spam email and ham email headers were removed and only the content was used for our tests.

Linux and Perl. Our obfuscating and deobfuscating processing was written in Perl 5.6 with the SpamAssassin perl modules installed from the CPAN repository. This experiment was conducted on Fedora Core 6, with Linux kernel version 2.6.20.

3.3 Training SpamAssassin

First, we trained SpamAssassin using the method mentioned in Section 3.2. SpamAssassin has a threshold value, which is used to decide whether an email is spam or not. All emails whose scores are above this threshold are spam, while those that have lower scores than the threshold are not classified as spam. In general, the default threshold is 5, but users can set it according to their own needs. Anyhow, the threshold’s value will not greatly affect our desired outcome. In our experiment, the default value of 5 was used.

After training SpamAssassin, we selected some typical spam emails from the collection of spam emails in the IUPUI corpus for obfuscating and de-obfuscating. There are more than 1,000 text spam emails available for our experiments, and their scores through SpamAssassin range from 5.4 to

more than 20. After analysis, we found that those spam emails that contained URI listed in the Spamhaus Blocklist (SBL) or the list from ws.surbl.org blocklist or obvious spam keywords (like porn-related words), got higher scores than ordinary spam emails, and could likely benefit most from Unicode-related obfuscation. We chose to use more than 100 spam emails like these whose scores were larger than 7.9. To obfuscate those emails whose scores are below 7.9 is not be very useful, since their lower scores show that they are not obvious spam emails.

3.4 Obfuscating Spam

Next, we obfuscated the chosen set of spam emails. Our obfuscation script randomly selected some original characters in the input messages and replaced them with characters chosen at random from the UC-Simlist entry for the original character. For example, in the sentence “Finally the real thing”, our script might choose characters “i r h”. Each one of these letters is looked up in the UC-Simlist and one of the respective similar characters whose similarity is 1 is picked at random to replace it. Each English character might have many similar characters; in fact, in the UC-Simlist, the character U+006C (“l”) has more than 16 similar characters (see Table 1).

1:217C:l	1:FF29: I	1:0406:l	1:FF4C: l
1:2160:l	1:006C:l	1:10C0:l	1:0399:l
1:0049:l	0.93103:05C0:l	0.93103:05C0:l	0.89843:0196:l
0.8879:1F77:l	0.87931:1F30:l	0.87931:1F31:l	0.87878:0140:l

Table 1: Similar characters of 006C “l” in UC-Simlist. Format: <similarity>:<hex code>:<character>. Here, we use *similarity* to mean the visual or semantic correspondence, as defined by Fu et al. The similarity “1” means that the Unicode character is visually or semantically identical to the “spoofed” character.

Our obfuscation script randomly chose a similar character among these similar characters whose similarity is 1 to replace the original character. This randomness gives a different obfuscated spam email every time the obfuscation script is run with the same spam email (though the result looks the same, and the inputs are all the same). This would get a different score for every different attempt at obfuscating a given spam email. In this case, we generated 10 obfuscated emails for each input message and computed the average score to determine an expected value (Table 2).

3.5 De-Obfuscating the Spam

Once we figured out the expected score of each obfuscated email, we had to see what the expected de-obfuscation score will be. We wrote a second script that converts the non-English Unicode characters in its input to a (hopefully same-as-original) English character. Because we chose only English inputs, the script could convert the non-English characters to its original (pre-obfuscated) character nearly every time.

The de-obfuscating script addresses every character with value greater than 255 (the barrier of dividing ASCII En-

glish characters and other Unicode characters). It also addresses all characters that are numeric and between two letters, such as the number one in “pen1s”. For each of these chosen characters, a reverse-lookup is performed on the UC-Simlist (the exact inverse of what was performed to chose an obfuscated replacement during the obfuscation-stage). The de-obfuscation reverse lookup is a bit more complex since the script must choose the right original English character; for example, “1” is not only similar to “l”, but also similar to “i”. To avoid using a dictionary to find the “best” result, we can maximize accuracy by attempting the reverse-lookups in order of English-letter frequency (for example, checking if something matches “O” before checking to see if it matches “Q”). This de-obfuscation script could be further refined for more accuracy sacrificing a bit of speed; an English language dictionary could be employed to make sure that the de-obfuscation generated something resembling an English word. We did not employ this, as it turns out even choosing any replacement that has a perfect (1) similarity will recover most of the original characters well enough.

Since we used the average of ten runs to determine the expected score of the obfuscated messages, we de-obfuscated each of those ten messages and took the average score of the de-obfuscated messages to determine how effective the de-obfuscation was expected to be.

4. RESULTS

Using the obfuscation and de-obfuscation techniques mentioned above, we produced some obfuscated spam emails (based on some selected from the IUPUI spam corpus) and then de-obfuscated them. Each message’s “spammyness” was measured by SpamAssassin, and the average score for the ten-runs of obfuscated and de-obfuscated messages were calculated.

Table 2 and Figure 1 shows the experimental results for a size-ten subset of the original emails we tested. The contents of the spam messages can be found in the appendix A.

No.	Raw Score	Obfuscated	De-obfuscated
1	16.2	3.15	16.2
2	21.7	5.6	21.7
3	17.2	4.38	17.2
4	10.6	1.9	10.6
5	7.9	5.85	7.9
6	21.7	5.0	21.7
7	19.2	5.4	19.2
8	12.1	2.55	12.1
9	13.1	5.94	13.1
10	10.6	5.35	10.6

Table 2: Scores of 10 spam emails—the emails are shown in appendix A. The table also shows the average scores, as produced by ten independent obfuscations and associated de-obfuscations. The scores were calculated by SpamAssassin.

From Figure 1, we can see that obfuscated emails got lower scores than original emails. In fact, in these spam emails, there are many keywords, phrases or URLs tagged as spam token in databases of SpamAssassin that has been trained, but in those spam emails’ obfuscated emails, most of those keywords, phrases or URLs were changed, which are *not* in SpamAssassin’s keyword list, so they aren’t scored as

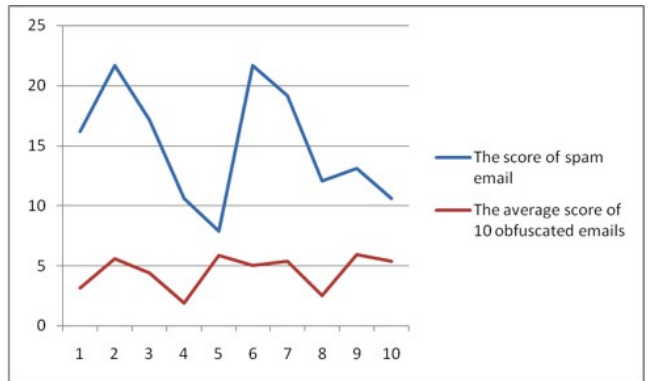


Figure 1: Score comparisons between spam messages’ raw score and obfuscated score for 10 emails found in appendix A.

highly. Thus, obfuscated emails’ scores will probably always be lower than spam emails.

Also, we can see that some spam emails got high scores, while some of them got lower scores. For example, message 1’s score is much higher than message 5. This is because message 1 is a common spam email including not only bad keywords, but also a blacklisted URL. After obfuscation, some bad keywords as well as the URL in SpamAssassin’s blacklist were changed (making them new to SpamAssassin). As a result, obfuscated message scores are on average much lower than the original score. In contrast with message 1, message 5 does not have a blacklisted URL, so its score is lower (7.9). Also, its body is short, which can’t be changed very much, and ends up still above the threshold of 5 (classified as spam).

In fact, not all spam emails’ scores are fixed. It depends on SpamAssassin’s training for its naive Bayesian learning (mentioned in 3.2). However, this does not affect our results, since those keywords and URLs partly changed by Unicode characters in spam emails will be new to specific trained SpamAssassin (no matter how it is trained).

In Figure 2, some parts of the first spam message and one of its obfuscated versions are displayed. (Screenshots of the full-text can be found in appendix B). In the obfuscated email, we can see that the link at the bottom and some words like “pen1s” and “enlargment” were partly changed into non-English Unicode characters. (Notice in this original spam message, some key words had already been obfuscated by spammers in order to bypass spam filter. Examples of this are “enhacment” and “pen1s”—note the mis-spelling and the use of digits. We note that these two are still blocked by SpamAssassin). It was the obfuscation’s change of the URL and these keywords that decreased the email’s spam score.

As mentioned above, we used characters whose similarity is specified as 1 for obfuscation characters to get highest-quality obfuscation. As a result, our de-obfuscated script can quite closely convert those obfuscated messages back to their original form. When close to the original form, their corresponding scores are very close to (if not exactly the same as) the original messages. However, it is almost impossible to convert all obfuscated characters to the “real” English characters without some sort of context inference. For example, the word of “pen1s” in the spam email of figure 2, we can invert “pen1s” to “penis” by judging if the number

...
A top team of British scientists and medical doctors have worked to develop the state-of-the-art Pen1s Enlargment Patch delivery system which automatically increases pen1s size up to 3-4 full inches.
...
Here's the link to check out!
<http://www.all-love-pillzz.net/pt/?46&wnwug>

...
A top team of British scientists and medical doctors have worked to develop the state-of-the-art Pen1s Enlargment Patch delivery system which automatically increases pen1s size up to 3-4 full inches.
...
Here's the link to check out!
<http://www.all-love-pillzz.net/pt/?46&wnwug>

Figure 2: A spam email before and after obfuscation. The screenshots of the full texts can be found in appendix B. In this figure, we can see that, in original spam email, the spammer has changed the form of the keyword “penis” as “pen1s” and “penls” and misspelled “enlargement” as “enlargment”, trying to bypass spam filter. However, after being trained, SpamAssassin can block this spam email. In the obfuscated email, some characters in many keywords, even including “pen1s” and “penls” in original spam email, were replaced with other similar Unicode characters in UC-Simlist by our obfuscation script. Therefore, those keywords that have been changed are new to SpamAssassin, which can help the obfuscated spam email to bypass SpamAssassin.

1’s left and right characters are letters instead of numbers, but it is hard to catch “l” and invert it to “i” since both “l” and “i” are English characters. This is since we did not make the script to look up in a dictionary whether a word is real or not. (This *can* be done, but will increase the processing time.) As a result, not all de-obfuscation will be perfect, and in fact some might recover words more fully than present in the original source.

We did this experiment with those more than 100 spam emails, analyzing their obfuscated and de-obfuscated versions. The results showed that obfuscated emails’ average score is usually much lower than original spam emails for typical spam emails, which means that well obfuscated emails can surely bypass spam filter. However, after being de-obfuscated by our script, they can more easily be caught.

4.1 The Phishing Email Twist

In comparison with spam emails, phishing emails are well written and look more “authentic”. Before doing experiments with phishing emails, we adjusted the Simlist used for finding non-English replacements, making the obfuscated emails look “perfect”. For example, in most fonts, the Cyrillic “i” and “v” look exactly the same as the ASCII equivalents, but they are different to the filter. This subset of the Simlist produced “perfect” looking results, though shrunk the number of obfuscation variations for each input message. We assert that it is hard for a human reader to discern which characters are not English characters in the obfuscated version in the sample (Figure 3).

Since phishing emails are well written, they often bypass a filter more easily. Of course, we can adjust SpamAssassin

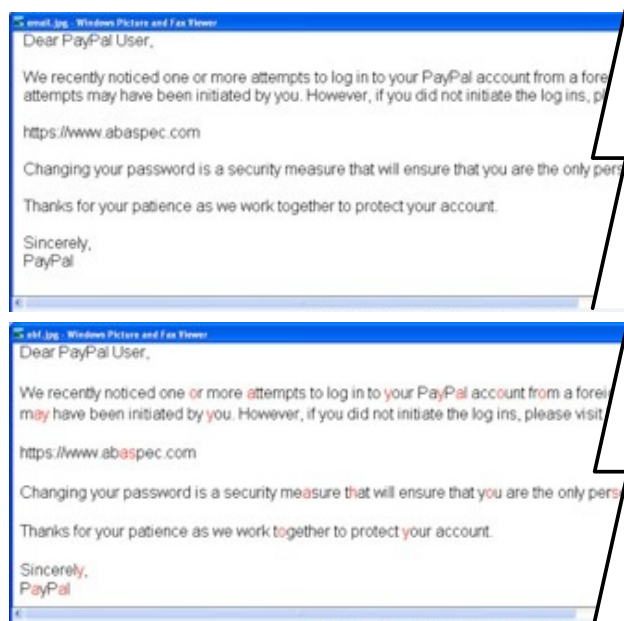


Figure 3: A phishing email before and after “perfect” obfuscation. In order to make the obfuscated email look perfect, we just chose Unicode characters that take up the same size space as English characters and look very similar to their corresponding English characters to replace original English characters. Moreover, we adjusted the font to make all characters, including original English characters and Unicode characters, more similar. We use red to identify the obfuscated characters in the after figure.

by training them, but this probably adds to more probability of a false positive, or filtering out a real bank message. However, since phishers are in the business to make money, the phishing emails always include some telephone numbers, links to some websites or postal addresses. For those in which there are some keywords or URLs in SpamAssassin’s databases, we will get a similar result as the spam email tests.

However, in our experiment for dozens of phishing spam emails obtained from various sites on the Internet, we found that most of them do not get high scores at all through SpamAssassin in the first place. One possible reason is that the phone numbers and website links are not listed in SpamAssassin’s blacklists. Another reason might be that phishers may put authentic websites in those emails, but redirect users to malicious websites using techniques like pharming.

4.2 Shortcomings and Improvements

Our de-obfuscation method presented works well, but is not perfect. It’s tough to measure its accuracy since each person receives a different set of spam emails, and the spammers regularly change techniques. The few seconds per message overhead is not desirable — a person does not wish to wait five seconds for each email they receive. The performance of our scripts should be optimized to perform better and faster.

4.2.1 Tweaks and Improvements

There are tweaks and changes that can make our de-obfuscation faster or more accurate. For example, two UC-Simlist indexes can be used: one that maps English/ASCII characters to similar-looking Unicode characters, and another that maps each (non-English) Unicode character back to English/ASCII characters that resemble it. These two structures can be placed into hash tables to enable very fast lookups for obfuscation and de-obfuscation, respectively. Using these hash tables to (de-)obfuscate characters would make the processing time for each email negligible.

To improve accuracy, we could use dictionary lookups to verify that the de-obfuscation of a word worked, or help guide the de-obfuscation choices when there are many de-obfuscations of a word (such as in the case where the script must choose between an upper-case “I” or a lower-case “l”). Additionally, more advanced word-stemming [1] or edit-distance (from banned keywords) techniques can be used on words in the message to more accurately find “spammy” content.

5. CONCLUSION

Nowadays, spammers are more intense and agile than ever before and they employ many tactics while trying to get their messages past spam filters. We have described a few methods they are using, including one that involves obfuscation by similar characters in Unicode: replacing English characters with non-English equivalents. We described a de-obfuscation that quite well reverses this obfuscation process; this script can be integrated into an existing spam filter to improve its efficacy against spam that is obfuscated in this fashion. However, spammers probably use more obfuscation, such as html techniques and such on, to bypass filters, so this is not at all a complete solution for spam filtering, but this is surely a step in the right direction.

6. ACKNOWLEDGEMENTS

We would like to thank Markus Jakobsson for his guidance, Arvind Ashok for help with the experiments, and Rachna Dhamija and Jon Praed for their helpful feedback on a previous version of the paper.

7. REFERENCES

- [1] S. Ahmed, F. Mithun, “Word Stemming to Enhance Spam Filtering,” in the Conference on Email and Anti-Spam (CEAS’04) 2004.
<http://www.ceas.cc/papers-2004/167>.
- [2] R. Cockerham, “There are 600,426,974,379,824,381,952 ways to spell Viagra.”
<http://cockeyed.com/lessons/viagra/viagra.html>. Retrieved on 25 July 2007.
- [3] D. Cook, J. Hartnett, K. Manderson, J. Scanlan, “Catching Spam Before it Arrives: Domain Specific Dynamic Blacklists,”
<http://crpit.com/confpapers/CRPITV54Cook.pdf>.
- [4] L. F. Cranor, B. A. LaMacchia, “Spam!” Communications of the ACM, August 1998.
- [5] A. Y. Fu, W. Zhang, X. Deng, W. Liu, “Safeguard against unicode attacks: generation and Application of UC-simlist,” in the 15th International World Wide Web Conference (WWW’06), May 2006.

- [6] A. Y. Fu, X. Deng, W. Liu, G. Little, “The Methodology and an Application to Fight Against Unicode Attacks,” in Proceedings of the Second Symposium on Usable Privacy and Security (SOUPS’06) July 2006. ACM Press.
- [7] F. D. Garcia, J.H. Hoepman, J. V. Nieuwenhuizen, “Spam Filter Analysis,” arXiv report, February 2004. Available at http://arxiv.org/PS_cache/cs/pdf/0402/0402046v1.pdf
- [8] S. L. Garfinkel and R. C. Miller, “Johnny 2: a user test of key continuity management with S/MIME and Outlook Express,” Proceedings of the 2005 Symposium on Usable Privacy and Security, 2005, pp. 13 – 24
- [9] P. Graham, “Better Bayesian Filtering,” Spam Conference, January 2003. Available at <http://www.paulgraham.com/better.html>.
- [10] E. Gabber, M. Jakobsson, Y. Matias, A. Mayer, “Curbing Junk E-mail via Secure Classification,” Financial Cryptography, 1998.
- [11] E. Gabrilovich, A. Gontmakher, “The Homograph Attack,” Communications of the ACM, February 2002.
- [12] J. Goodman, G. V. Cormack, D. Heckerman, “Spam and the Ongoing Battle for the Inbox,” Communications of the ACM, February 2007.
- [13] R. J. Hall, “Channels: Avoiding Unwanted Electronic Mail,” Communications of the ACM, Volume 41 Issue 3, 1998.
- [14] R. J. Hall, “A Countermeasure to Duplicate-detecting Anti-spam Techniques,” Available at <http://citeseer.ist.psu.edu/279802.html>, accessed 25 July 2007.
- [15] M. Jakobsson, “Modeling and Preventing Phishing Attacks,” Phishing Panel in Financial Cryptography 2005. Available at www.informatics.indiana.edu/markus/papers/phishing_jakobsson.pdf
- [16] M. Jakobsson, J. Linn, J. Algesheimer, “How to Protect Against a Militant Spammer,” <http://www.informatics.indiana.edu/markus/papers/spam.pdf>, accessed 1 July 2007.
- [17] M. Jakobsson and S. A. Myers (Eds.), *Phishing and Countermeasures: Understanding the Increasing Problem of Electronic Identity Theft*. ISBN 0-471-78245-9, Hardcover, 739 pages, December 2006.
- [18] J. Nazario, “Phishing Corpus,” http://monkey.org/~jose/blog/viewpage.php?page=phishing_corpus. Accessed 22 May 2007.
- [19] U. Shardanand, P. Maes, “Social Information Filtering: Algorithms for Automating ‘Word of Mouth’,” Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. May 1995.
- [20] B. Thorson, “How Spammers Bypass E-mail Security,” EE Times, 25 July 2007.
<http://www.eetimes.com/showArticle.jhtml?articleID=23900564>
- [21] A. Tsow and M. Jakobsson, “Deceit and Deception: A Large User Study of Phishing,” Technical Report TR649, Indiana University, August 2007. <http://www.cs.indiana.edu/pub/techreports/TR649.pdf>
- [22] S. Srikwan, M. Jakobsson, “Using Cartoons to Teach

Internet Security.” DIMACS Technical Report
2007-11, July 2007.

[http://www.informatics.indiana.edu/markus/
documents/security-education.pdf](http://www.informatics.indiana.edu/markus/documents/security-education.pdf)

- [23] CRM114. <http://crm114.sourceforge.net>,
Accessed 22 May 2007.
- [24] Anti-Phishing Group of City University of Hong
Kong, <http://antiphishing.cs.cityu.edu.hk>.
- [25] Messaging Anti-Abuse Working Group, Email
Metrics Program: “The Network Operator’s
Perspective, Report #4—3rd and 4th Quarters
2006,” Available at [http://www.maawg.org/about/
MAAWGMetric_2006_3_4_report.pdf](http://www.maawg.org/about/MAAWGMetric_2006_3_4_report.pdf)
- [26] SpamAssassin.
<http://wiki.apache.org/spamassassin>, Accessed
22 May 2007.
- [27] SpamAssassin Readme file.
[http://www.cpan.org/modules/by-module/Mail/
Mail-SpamAssassin-2.64.readme](http://www.cpan.org/modules/by-module/Mail/Mail-SpamAssassin-2.64.readme) Accessed 22 May
2007.
- [28] SpamAssassin public Corpus,
<http://spamassassin.apache.org/publiccorpus>,
Accessed 25 May 2006.

APPENDIX

A. 10 SAMPLE SPAM MESSAGES BEFORE OBFUSCATION



Figure 4: This email scored 16.2 by SpamAssassin. As mentioned in the analysis in Section 4 the body of this email is obviously a spam email. SpamAssassin recognizes the message as spam due to the black-listed URLs. Both the content and link cause SpamAssassin to assign the message a high score. Obfuscating lowers the score because the key words, phrases, and links are disguised.

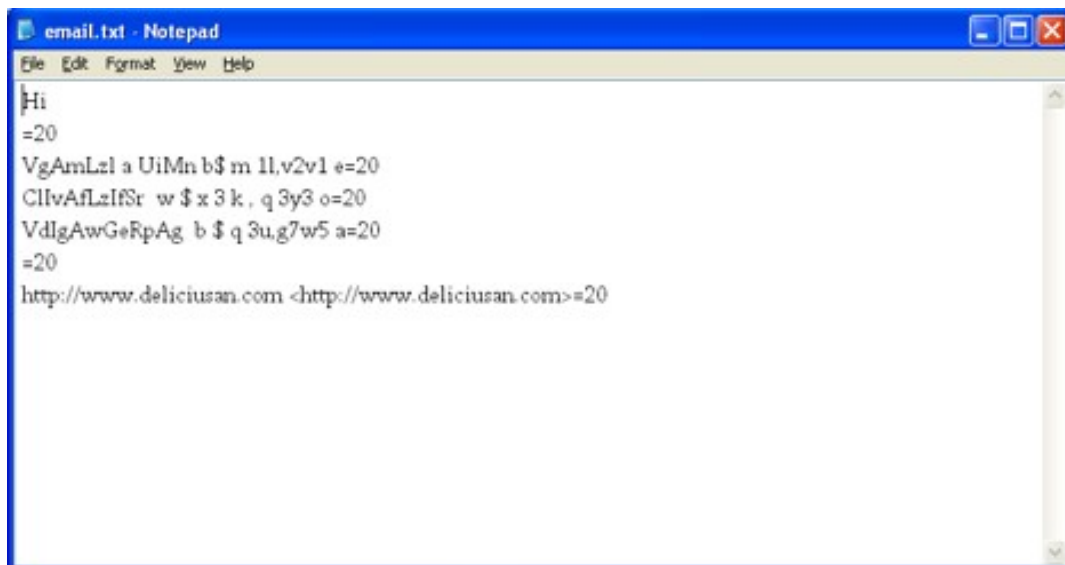


Figure 5: This email scored 21.7 by SpamAssassin. Although the text of the body of this email is confusing to users (In fact, spammers can use a Java script to remove those small letters among capital letters before receivers read this kind of emails), it is tagged as spam by SpamAssassin trained by spam emails and ham emails. Especially, the link is in SpamAssassin's several blocklists, which make it have a high score.



Figure 6: This email scored 17.2 by SpamAssassin. Although the text of the body of this email is confusing to users (In fact, spammers can use a Java script to remove those small letters among capital letters before receivers read this kind of emails), it is tagged as spam by SpamAssassin. Especially, the link is in SpamAssassin's blocklists, which make it have a higher score.

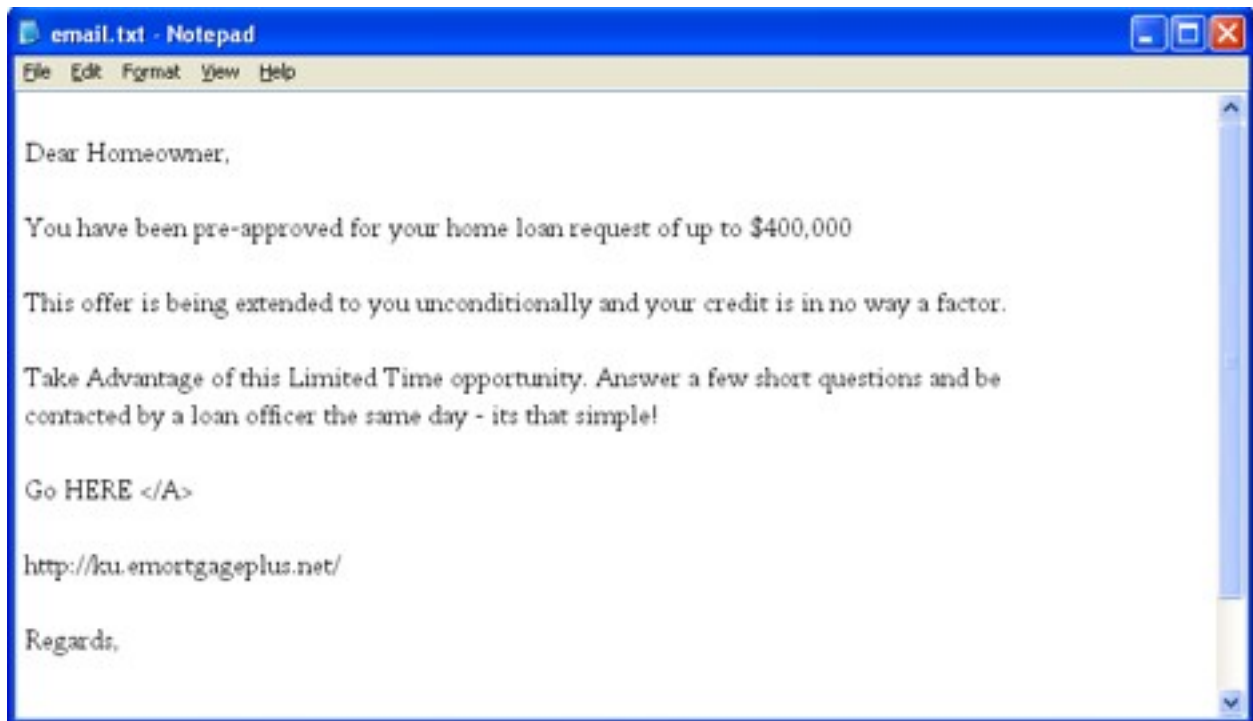


Figure 7: The email scored 10.6 by SpamAssassin. Both the content and link contribute to the score.

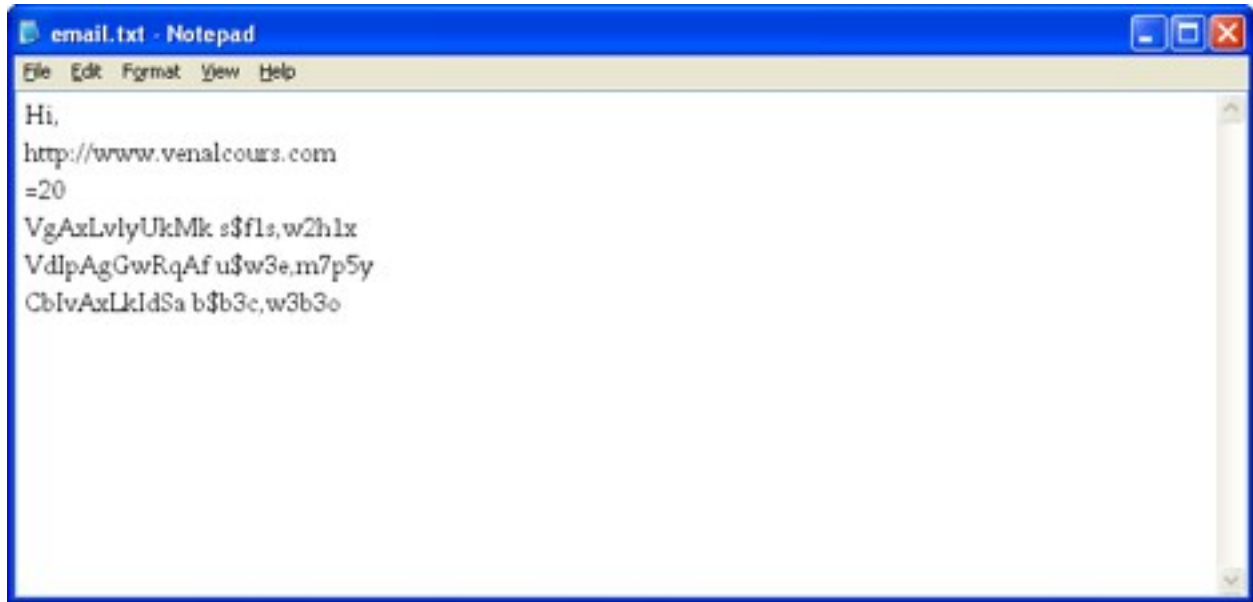


Figure 8: The email scored 7.9 by SpamAssassin. The body is judged to be spam, but the link is not in the black list, so the score of this email is not as high as that of other emails with blocked links.



Figure 9: The email scored 21.7 by SpamAssassin. Although the text of the body of this email is confusing to users, it is tagged as spam by SpamAssassin, largely due to the link. We note that spammers can use CSS to remove the small letters before the email is displayed.



Figure 10: The email scored 19.2. The body of this email is not pure text, but it is still considered spam by SpamAssassin. In addition, the link contributes to the high score.

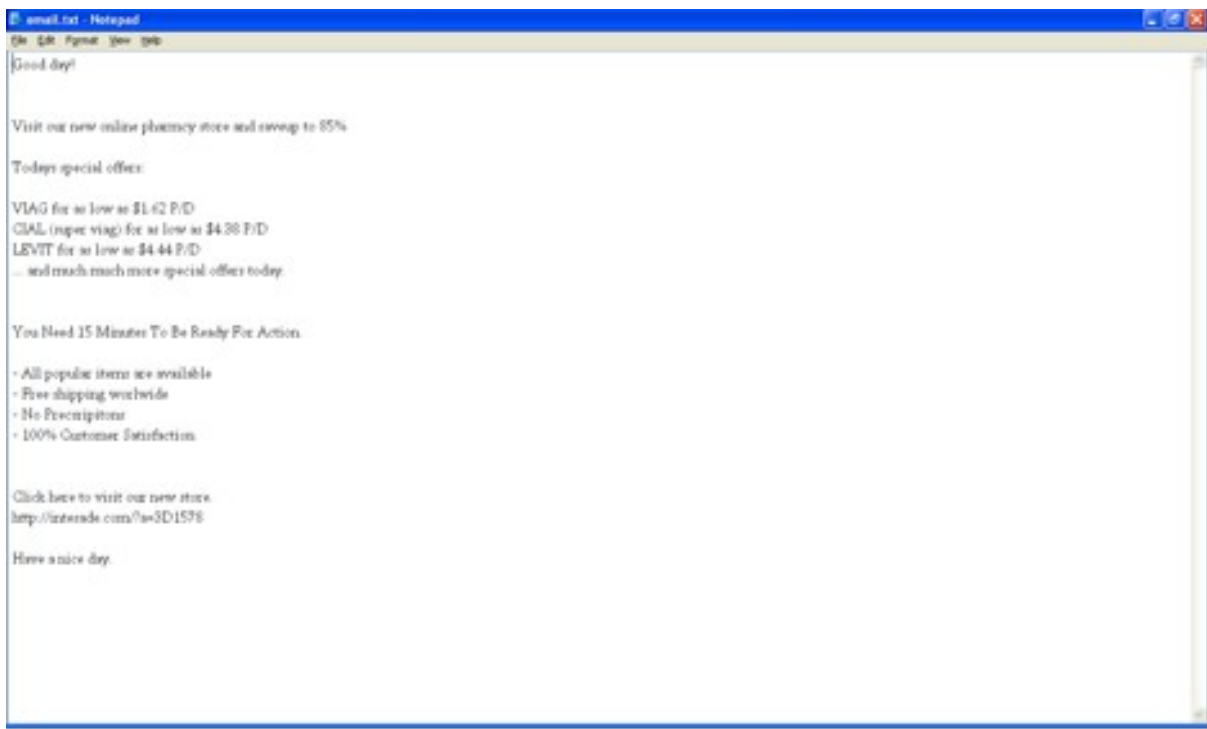


Figure 11: The email scored 12.1. The content of this email is judged as spam by SpamAssassin. The link in SpamAssassin's blocklist also contributes to this email's higher score gotten from SpamAssassin.



Figure 12: The email scored 13.1. The small letters can be removed using CSS.



Figure 13: The email scored 10.6. SpamAssassin considers this spam and blocks the link. Both the content and link cause a higher score.

B. FULL-SIZE SCREENSHOTS



Figure 14: The original spam message before obfuscation. We can see that the spammer has changed the form of the keyword "penis" as "pen1s" and "penls" and misspelled "enlargement" as "enlargment", trying to bypass spam filter. However, after being trained, SpamAssassin can block this.

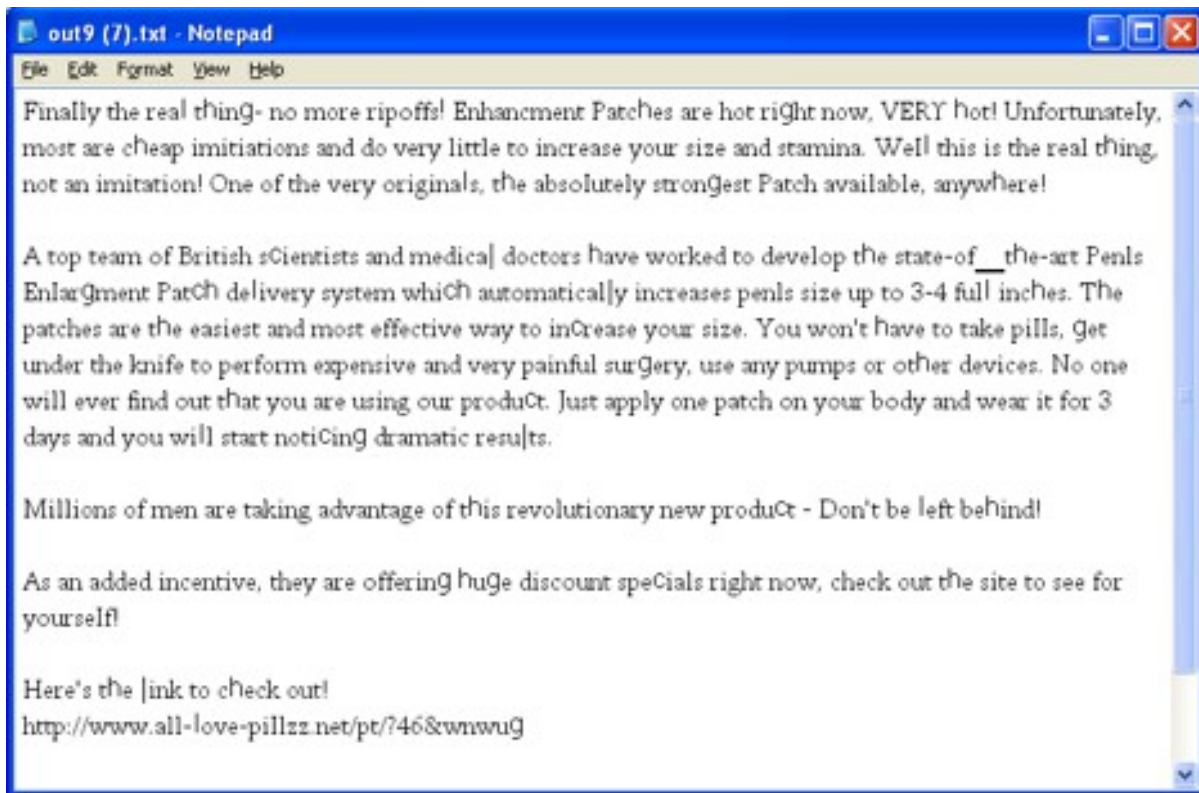


Figure 15: The spam message after obfuscation. In this obfuscated email, some characters were replaced with other similar Unicode characters in UC-Simlist by our obfuscation script. This helps the obfuscated spam email bypass SpamAssassin.