A Potential IRI based Phishing Obfuscation Strategy and Counter Measures

Yingjie Fu, Xiaotie Deng, Liu Wenyin

Dept of Computer Science, City University of Hong Kong, 83 Tat Chee Ave.,
Hong Kong SAR., China
{anthony@cs., csdeng@, csliuwy@}cityu.edu.hk

Abstract. We anticipate a potential phishing strategy by obfuscation of Web links using Internationalized Resource Identifier (IRI). In the IRI scheme, the glyphs of many characters look very similar while their Unicodes are different. Hence, certain different IRIs may show high similarities. Therefore, it is quite difficult for normal Web users to distinguish them. The potential phishing attacks based on this strategy are very likely to happen in the near future with the boosting utilization of IRI. We invent a detection approach to this phishing strategy. We construct a Unicode character similarity list based on their visual similarity and semantic similarity. We use Nondeterministic Finite Automaton (NFA) model to identify the potential IRI based phishing patterns. We build the phishing IRI pattern generation system, by which, NFA could be further represented with regular expression (RE) to adapt it to anti-phishing systems. A framework is also proposed to build such anti-phishing systems.

1. Introduction

Phishing webpages are webpages forged to mimic the webpages of certain real companies offering Internet transactions in order to spoof end users to leak their private information. Phishers usually use visually and semantically similar URIs and similar webpages to spoof people. Unwary Internet users who are induced to access phishing webpages could be deceived to expose their bank accounts, passwords, credit card numbers, or other important information to the phishers.

Phishing webpages have recently been found at faster and faster paces. Higher attention has been paid by industries, researchers, and new media, as well as legislative bodies most recently. The Anti-Phishing Working Group [2] reported that the number of phishing attacks is increasing by 50% for each month and up to 5% of the phishing email receivers may respond to them. While the public has been gradually educated and alerted to such scams, phishers are resorting to more and more sophisticated tricks to avoid detections.

In this paper, we report a potential phishing attack that depends on the utilization and popularization of Internationalized Resource Identifier (IRI) [6], as could be a quite severe problem. We also propose a solution to this problem.

1.1 IRI and its Vulnerability to Phishing

Rapid evolution of the Internet requires advanced security developments to make it safe. It is not an over claim if we say it is unimaginable if the Internet is not available. However, the Internet is still not mature, and its development is an endless process. In the very beginning, people use IP address directly to access a webpage or other Internet resources. Later, uniform resource identifiers (URI) [3], which are ASCII based, are used to locate or access a webpage or

other Internet resources. With the popularization of the Internet, people speaking languages other than English are demanding to use a unified representation of internet resource identifiers to locate the information resources, while URI has its inborn deficiency for this purpose. It uses ASCII and can represent a set of very limited number of readable characters. IRI is such a standard proposed as a complement of URI. An IRI is a sequence of characters chosen from the Unicode [18], which could be used instead of URIs to identify resources. However, the utilization of IRI could bring in severe, potential phishing attacks, since the Universal Character Set (UCS)[18] covers almost all characters in the word for information exchanges, in which a lot of visually similar characters and semantically similar characters co-exist. Figure 1 shows some of the similar characters to "3" and "bank".

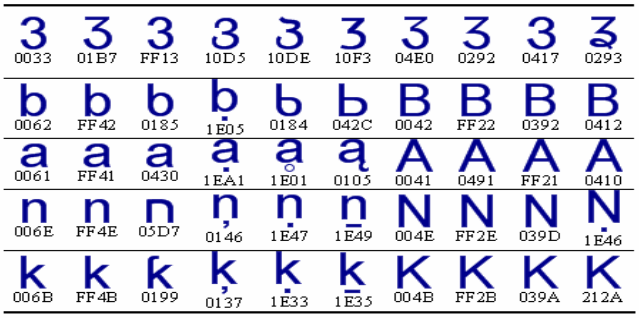

Figure 1. Similar characters of "3" and "bank" in Arial Unicode MS Font (adapted from UC-SimList [8], and the codes for each character are in hexadecimal form)

It is very easy for phishers to spoof users by replacing characters in a target IRI with similar ones. Although a phishing IRI looks very similar or exactly the same with a target one, they are definitely different in coding level, and people could be victims of this kind of scams without knowing what is happening. Hence, this kind of phishing attacks is very likely to happen in the near future.

1.2 Detection of IRI based Phishing Attacks

We provide a solution to this kind of phishing attacks. We first generate the matrix of similarities between each pair of characters according to their similarity of glyphs. We use the matrix to generate the visual similarity list of UCS (UC-SimList$_v$). We investigate the semantic relationships of characters in UCS and generate semantic similarity list of UCS (UC-SimList$_s$). We utilize the two lists to generate similarity list of UCS (UC-SimList). The UC-SimList can be used to easily find similar characters to a given one.

We use NFA as the tool to construct potential phishing IRI patterns. We construct NFA with the keywords of IRIs that we need to protect. If an IRI is suspected (cf. the overall anti-phishing strategy in [12] and [13]), we use the NFA to check its acceptance. In case it is accepted and is not a protected IRI, it is a phishing IRI; otherwise, we take it as a normal one. We convert NFA to regular expression (RE) to represent the potential phishing IRI patterns. As NFA is equivalent to RE, the NFA is converted to RE for ease of embedding this IRI based phishing obfuscation detection method to systems need to perform anti-phishing tasks. We build up the phishing pattern generator, which could be used to generate regular expressions of phishing patterns. And we also propose a framework to build such anti-phishing systems.

1.3 Organization of the Paper

The rest of this paper is organized as follows. Section 2 introduces related works. Section 3 addresses Unicode character similarity and the method to generate UC-SimList. In Section 4, we present the NFA based IRI obfuscation pattern modeling approach. In Section 5 we discuss the approach of regular expression generation from NFA based IRI obfuscation patterns. In Section 6, we present the Obfuscation Pattern Generator for Anti-Phishing. Finally, we conclude our work and discuss future work in Section 7.

2. Related Works

A uniform resource identifier (URI), as defined in [3], is a sequence of characters from a subset of the repertoire of ASCII [1]. These character scripts are Latin language oriented and used for representing natural language words, abbreviation, or other semantics, such that URI provides a smooth way for Latin language understanding people in the information world to identify resources in the Internet. Currently, most of the resource identifiers in the Internet are URIs.

As most of non-Latin language scripts have a direct mapping from their characters to A~Z through pronunciation, such as Chinese Pinyin and Japanese Romaji, there are no big problems for URI representation of non-Latin languages. However, it could bring a lot of URI conflictions and semantic ambiguity problems. With the development of information technology, people are using localized operating systems, applications, and internet services. It is shown in a practical way that people are eager to use information systems with their native scripts. There are more and more protocols and formats that utilize wider range characters. IRI is a complementary of URI which is a sequence of characters from a subset of UCS [18]. UCS use 16 bits to represent a character, so that it can represent up to 65,536 characters and control symbols at the most, and covers almost all major language characters in the world for information interchanges. IRI can bring much convenience for non-English speakers. It provides a more natural way for people who do not know English but Chinese to input 中国银行.公司 (“中国银行” pronounces “Zhong Guo Yin Hang” and stands for “Bank of China”; “公司” pronounces “Gong Si” and for “Company”) rather than www.bank-of-china.com. For people who do not know English but Japanese, they may prefer to input 東京三菱銀行.会社 (“東京三菱銀行” pronounces “Tou Kyou Mitsu Bishi Gin Ko”, and stands for “Bank of Tokyo-Mitsubishi”; “会社” pronounces “Kai Shya” and for “Company”) rather than www.bank-of-tokyo-mitsubisi.com. Although Unicode has advantages of character representation, there is no true type font which could cover all characters in latest UCS (Unicode Standard 4.3 [18]) so far. The largest font that we can find by now is Arial Unicode Font MS [15]. It is a complete Unicode font of Unicode Standard 2.1 [18]. It contains all the characters in Arial plus full fonts for Japanese, Chinese, Korean, Arabic, and Hebrew, plus all of different symbol characters and character ranges [14].

The phishing problem has emerged for several years. The most straightforward way for a phisher to spoof people is to make the appearance of webpage links and webpages similar to the real ones.

The link based phishing obfuscation can be carried out in four basic ways:

a. Adding suffix to domain name of URL. E.g., revise www.citibank.com to www.citibank.com.us.ebanking;
b. Using actual link different from visible link. E.g., the HTML line: www.citibank.com;
c. Using bug in real webpage to redirect to other webpages. E.g., the bug of eBay website: http://cgi.ebay.com/ws/eBayISAPI.dll?MfcISAPICommand=RedirectToDomain&DomainUrl=*PHISHINGLINK* can direct you to any specified *PHISHINGLINK*;
d. And replacing similar characters in the real link. E.g., replace “I”s (uppercase “i”) with “l”

(lowercase of "L") or "1" (Arabic number one), such as WWW.CITIBANK.COM to WWW.C1TlBANK.COM.

The webpage based obfuscation can be carried out in three basic ways:

a. Using the downloaded webpage from real website to make the phishing webpage appear and react exactly the same with the real one;
b. Using script or add-in to web browser to cover the address bar to spoof users to believe they have entered the correct website;
c. And using visual based content (E.g., image, flash, video, etc.) rather than HTML to avoid HTML based phishing detection.

Various anti-phishing methods have been proposed. Strong authentication of webpages [17] is widely used in security demanded websites. Commercial legislation actions against internet frauds are done in different countries. Previous research works provide various duplicate document detection approaches, which focus on plain text documents and use pure text features in similarity measure [4] [9] [16]. However, the most effective strategy for phishing detection is probably an active approach based on visual comparison of DOM [19] generated from HTML, such as the one proposed by Liu et al. in [12][13], which uses the region based approach to visual similarity of webpages. Another visual based phishing detections technique is proposed by Fu et al. in [7], which is a phishing detection approach based on image level visual similarity assessment using the EMD algorithm. People also proposed anti-phishing methodologies from the cryptography perspective [5] [10].

We utilize an RE generator to generate REs automatically from keyword-level NFA. Different RE engines may have different mechanisms to perform pattern matching. Some of them use NFA which can do back tracking during pattern matching, such as those in Perl, Python, Emacs, and Tcl, and others use DFA to achieve better time efficiency, such as those in Awk, Egrep, and Lex. DFA engines run in linear time because they do not need to do backtracking. However, they cannot catch patterns with back-references or sub-expressions. Although we do not use back-reference or sub-expression in our approach, we use NFA to achieve better extensibility for future use. A string is accepted by an NFA if there is some sequence of possible moves which will put the automaton in a final state at the end of the string. A string is rejected only if there is no possible sequence of moves by which a final state can be chosen at every state [11].

3. Unicode Character Similarity

There are many similar characters in Unicode Character Set (UCS) [18]. We mean similarity in two aspects: visual similarity and semantic similarity. The similarity list of UCS (UC-SimList) can be generated from visual similarity list of UCS (UC-SimList$_v$) and semantic similarity list of UCS (UC-SimList$_s$), i.e., UC-SimList(C)=UC-SimList$_v$(UC-SimList$_s$(C)), where C is a character, UC-SimList$_s$(C) denotes the semantically similar character set of C, UC-SimList$_v$($\cdot$) denotes the visually similar character set of character set $\cdot$, and UC-SimList(C) denotes the similar character set of character C.

Visual similarity of two characters is measured by human vision, denoted as $vs \in [0,1]$. There are many methods to evaluate visual similarity of character glyphs. As our dataset is quite large, consisting of 38,621 readable characters (Unicode Arial MS Font Ver.1.01 [15]), we use the simple, fast, but effective pixel-overlapping method. The visual similarity calculated with this method is empirically general for different fonts since characters have the same style in the same font, such that, most similar character-pairs are still similar when we change to use another font to calculate the similarity of the pairs of characters.

We use Unicode Arial MS Font Ver.1.01 as our character image generation font because, among all fonts we could find, this font covers the largest number of characters. It covers most of

the characters in the latest Unicode standard Ver. 4.3 [18] and all characters of Unicode Ver. 2.1 [18]. We calculate the similarity of each pair of characters using formula (1):

$$vs(c_1, c_2) = \frac{|OverlapPix(c_1, c_2)|}{p|Pix(c_1)| + (1-p)|Pix(c_2)|} \quad \textbf{(1)}$$

, where $|OverlapPix(c_1, c_2)|$ is the number of overlapping pixels of the bitmaps of character c_1 and c_2 , $|Pix(c)|$ is the number of pixels of character c , and $p \in [0,1]$ is the adjuster for tuning the similarity computation validity. Experiment shows that the definition of p in formula (2) performs the best.

$$p = \begin{cases} 1 & if\ |Pix(c_1)| \geq |Pix(c_2)| \\ 0 & otherwise \end{cases} \quad \textbf{(2)}$$

Experiments show that when we set the size of Arial Unicode Font to 30 points, it will perform well to evaluate the similarities of character-pairs without losing accuracy. It takes about 1 minute to compute the similarities of one character to all the others and their ranks, and about 4 weeks to generate the entire UC-SimList$_v$ on an ordinary PC (P4 2.4G CPU and 512M RAM). We select a threshold to define whether a pair of characters is similar. Experiments show that 0.8 is a good choice. UC-SimList$_v$ in different threshold-versions is now available at [8]. Figure 2 shows examples of the ranking result of the first 10 similar characters to "M". The characters with bold hexadecimal Unicodes have $vs \geq 0.8$.

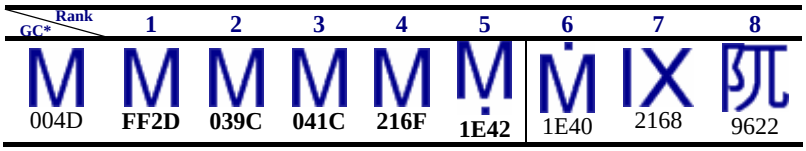

Figure 2. Examples of Ranked Similar Characters to "M" (GC* denotes the given character).

Semantic similarity of two characters is the measurement of character similarity in term of meaning. It is quite common that one character has more corresponding representations. English has Uppercase and Lowercase of the letters and uppercase "BANK" corresponds to lowercase "bank"; similarly, Chinese characters can be written as simplified and traditional, such as, Simplified Chinese "银行" (Bank) corresponds to Traditional Chinese "銀行" (Bank); and Japanese Katakana "ギンコウ" (Bank) corresponds to Japanese Hiragana "ぎんこう" (Bank). In certain cases, two forms of representations may be the same. E.g., we could see the Simplified Chinese "行" is the same with Traditional Chinese "行". In certain cases, one character may correspond to more than one semantically similar character. E.g., Japanese "木" (pronounces "Ki", and stands for "wood"), also can be represented as "キ" and "き".

In terms of creating UC-SimList$_s$, we need to find all semantically similar characters for each of the readable 38,621 characters.

The construction of UC-SimList$_s$ is very important. However, it is complicated. We have constructed the UC-SimList$_s$ of English (Uppercase and Lowercase), Chinese (Simplified and Traditional), and Japanese (Katakana and Hiragana) characters and it is available at [8]. We also leave other language parts as an open engineering problem and expect people could add and refine UC-SimList$_s$ as follow-up work.

Finally, we merge UC-SimList$_v$ and UC-SimList$_s$ to generate the UC-SimList. We define the semantically similar pair of characters has similarity 1. E.g. we consider $ss("a","A") = 1$ where $ss(c_1, c_2)$ is the semantic similarity of character c_1 and c_2 . Since the pair of characters will not be considered as similar when

their similarity is less than a threshold, we use different thresholds to generate UC-SimLists. We denote a certain UC-SimList with "UC-SimList" following the threshold. E.g. The UC-SimList generated with threshold 0.8 is denoted at UC-SimList0.8. We used 0.95, 0.9, 0.85, and 0.8 as thresholds to generate UC-SimLists and they are also available at [8]. So far, we could check UC-SimList to find all similar characters for a given one, and if there is no similarity in the UC-SimList for a pair of characters, their similarity is considered to be 0.

4. Modeling the IRI based Phishing Patterns with NFA

We use NFA to model the potential IRI based phishing obfuscation patterns. In general, the modeling process is to convert the IRI list which we need to protect (e.g., login page IRIs of banks) into an NFA, which can be utilized to perform phishing IRI detection. It can be carried out with the following procedure: (1) we manually construct an NFA the keyword level semantically for each IRI; (2) the NFAs constructed from all IRIs are merged into one NFA; (3) each nonempty transition in the NFA will be replaced by an NFA with parallel-connected similar characters of the exact letter in the original transition; (4) each empty transition in the NFA will be replaced by an NFA with parallel-connected symbols that are valid in IRI. This final NFA is the one used to perform phishing IRI recognitions.

Suppose we have a list of IRIs L_{IRI} which need to be protected, we construct the keyword-level pattern NFA NFA_{kw} from L_{IRI} , i.e. $L_{IRI} \rightarrow NFA_{kw}$ .We find the keywords for each IRI, expand the keywords semantically, connect them properly with empty transitions to form an isolated NFA, and then merge the isolated NFAs into one NFA NFA_{kw} in a parallel way. We consider three most potential types of semantic expansion of a keyword, which have high potential to be used for IRI based phishing obfuscation.

a. Pronunciation-based replacement:

A phisher may replace some part of the keywords of real IRI to other string without changing the pronunciation. As shown in Figure 3, a phisher can change Chinese word to its Pinyin, Japanese to corresponding Romaji, and English to other common used form. All these cases are pronunciation based.

Language	Original	Replacement
English	BANKFORYOU	BANK4U
Chinese	你的银行	NiDeYinHang
Japanese	あなたの銀行	ANaTaNoGinKou

Figure 3. IRI based phishing obfuscation using the same pronunciations

b. Abbreviation-based replacement:

A phisher may use the abbreviations of the keywords of real IRI to other string for obfuscation. As shown in Figure 4, a phisher may change the keywords to their abbreviations, or reversely by replacing abbreviations to full names.

Language	Original	Replacement1	Replacement2
English	CitiBank	CitiB	CB
Chinese	中国银行	中银	中行
Japanese	東京銀行	東銀	日本東京銀行

Figure 4. IRI based phishing obfuscation using abbreviation

c. Translation-based replacement:

A phisher may translate some keywords in a real IRI to other languages for obfuscation. As shown in Figure 5, there are many mutations of each part of an IRI through translations-based

replacement. There may be other semantics-based keywords replacement methods, which could be proposed by people in further study.

Language	Original	Replacement1	Replacement2	Replacement3
English	City Bank	City銀行	Cityバンク	シティ銀行
Chinese	中国银行	ChinaYinHang	中国Bank	ChinaBank
Japanese	東京銀行	TokyoGinkou	東京Bank	TokyoBank

Figure 5. IRI based phishing obfuscation using translation

Suppose we have an IRI $i \in L_{IRI}$. We convert i into sequences of separated words using the above three most potential types of semantic expansion of keyword to construct NFA_{kw}. We denote the keyword-level pattern NFA as $NFA_{kw} = \bigcup_{i \in L_{IRI}} NFA_{kw_i}$, where $NFA_{kw_i} = \bigcup_{1 \le m \le N_i} NFA_{\varsigma_m}$ is the keyword-level pattern NFA of i, N_i is the number of possible sequential combinations of the expanded keywords of i, ς_m denotes one such combination, where $1 \le m \le N_i$, and "$\bigcup$" denotes parallel connection relationship of NFAs. We construct NFA_{kw} manually because the number of IRIs which need to be protected is not big and we can construct NFA_{kw_i} for each $i \in L_{IRI}$ carefully. Figure 6 and Figure 7 shows the examples of the constructed NFA_{kw_i} when $i =$"*www.citybank.com*" with two different construction methods. We could see that, $N_i = 6 \times 5 = 30$ in Figure 6. However, if we know that some of the combinations will not happen, it can be reduced to a smaller one as shown in Figure 7, where we could see N_i is reduced to $3 \times 2 + 1 \times 1 + 2 \times 1 = 9$. We denote the initial state as q_0, final state as ◎ and empty string as λ. E.g, $\lambda + citi + \lambda + bank + \lambda$ = *citibank* is an acceptable string of NFAs in both Figure 6 and Figure 7, while $\lambda + citi + \lambda +$ 银行 $+ \lambda$ = *citi*银行 is not an acceptable string of NFA in Figure 7 but can be acceptable in Figure 6. In a practical system, we use XML to represent NFA_{kw}, as is discussed in Section 6.

The NFA_{kw} is generated by merging all NFA_{kw_i}, $i \in L_{IRI}$. The merging process can be accomplished by merging all initial state into a unique q_0, and final states into a unique ◎.

We construct the character-level pattern NFA NFA_c from NFA_{kw}, i.e. $NFA_{kw} \rightarrow NFA_c$. This process can be accomplished in the following two steps:

a. Replace nonempty characters in NFA_{kw} with parallel-connected corresponding similar characters that can be found from UC-SimList by empty strings, and we denote it as NFA_c', i.e., $NFA_{kw} \rightarrow NFA_c'$. We follow the example in Figure 7 to get the similar characters of each character of {c,i,t,y,b,a,n,k,花,旗,城,市,銀,行,シ,テ,ィ,バ,ン,ク} from UC-SimList0.8. Figure 8 shows the similar character sets of these characters, each element is represented by a character followed by its hexadecimal code.

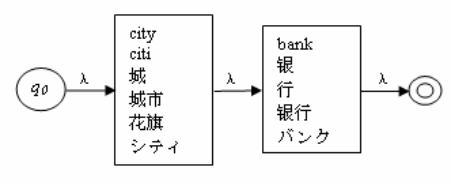

Figure 6. Complete combination of NFA_{kw_i} when $i = "www.citibank.com"$

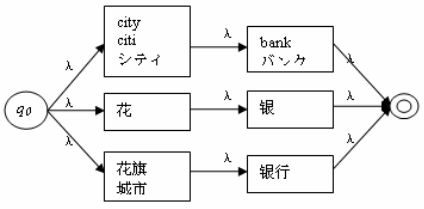

Figure 7. Reduced combination of NFA_{kw_i} when $i = "www.citibank.com"$

Querying Character	Form1			Form2			
c:0063	c:0063	c :03F2	c :217D	C:0043	C :0421	C :216D	C :FF23
	c :0441	c :FF43	c:0063	Ć :0404	Ç :0480	Ç :04AA	Ç :10BA
	ɕ :0454	ç :04AB	ç:00E7	Ç:00C7	O :039F	O :041E	O:004F
	ↄ :0254	ɕ :0255		O:FF2F			
银:94F6	银:94F6			銀:9280			
シ:30B7	し:3057	レ:30EC		シ:30B7	ジ:30B8		
…	…			…			

Figure 8. Similar character sets of each character in {c,i,t,y,b,a,n,k,花,旗,城,市,银,行,シ,テ,イ,バ,ン,ク}("…" denotes the omitted content)

We denote the NFA formed by similar character set of character $char$ as $S(char)$. Figure 9 shows the examples of $S(シ)$, $S(c)$, $S(银)$. The similar characters in each set are connected in a parallel way. We replace each character $char$ in NFA_{kw} with $S(char)$ to generate $NFA_c^{'}$, as shown in Figure 10. And there could be more IRIs other than $i = "www.citybank.com"$ in L_{IRI}. They are represented in a simplified way at the upper and lower position of NFA_{kw_i}, where we denote the NFA generated from other IRIs with $NFA_{kw_\bullet}$.

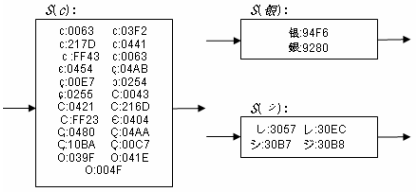

Figure 9. The NFA representation of $S(c)$, $S(银)$, and $S(シ)$

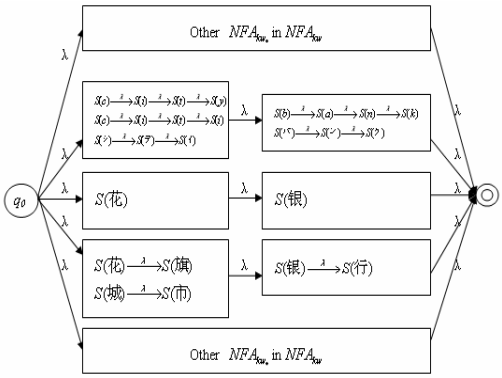

Figure 10. $NFA_c^{'}$ generated from NFA_{kw}

b. Replace each empty transition ($\xrightarrow{\lambda}$) in $NFA_c^{'}$ with a transition with all delimiter-like characters in a parallel way and we call it delimiter transition (DT). The delimiter-like character set is denoted as Σ, which includes the empty string λ and all characters look like the 22 valid symbols defined in RFC3986 [3] and RFC3987 [6], as shown in Figure 11, i.e. $\Sigma = \{\lambda, S(-), S(.)...\}$. We denote the generated NFA as NFA_c , i.e., $NFA_c^{'} \rightarrow NFA_c$, as shown in Figure 12. More concatenated DTs can be used to replace the empty string in $NFA_c^{'}$, and each DT means one appearance of delimiter appearance in the context. We use one DT in our approach for simplicity. So far, we have the key-character pattern NFA of L_{IRI} , i.e. NFA_c , which is the final NFA that can be used to detect phishing IRI.

"-"	"."	"_"	"~"	":"	"/"
"?"	"#"	"["	"]"	"@"	"!"
"$"	"&"	"'"	"("	")"	"*"
"+"	","	";"	"="		

Figure 11. The complete valid symbol list of RFC3986 and RFC3987

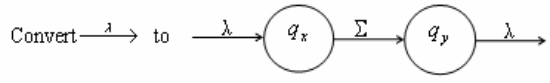

Figure 12. Empty transition replacement (q_x and q_y denotes two states)

5. Regular Expression Generation

Regular expression (RE) is equivalent to NFA. We convert NFA_c to RE to ease the adaptation approach to anti-phishing systems. As NFA_c is generated from L_{IRI} with specified rules, the structure of NFA_c is not complicated. We can apply the following procedure to convert NFA_c to an RE. We use "|" to denote union, "" (empty string) for concatenation, "{m,n}" for repetition number is between m and n . In our practical approach (secession 6), we configure m to be 0 and n to be 1 for simplicity. Figure 13 shows the RE generation procedure in our approach.

a. Get the sub-NFA generated from $i \in L_{IRI}$ in NFA_c which has not been processed, and denote it as NFA_{c_i} ;

b. Get the $S(char)$ (similar character set of character $char$), concatenate each character in $S(char)$ with "|", and bracket the generated string;

c. If not all $S(char)$ has been processed then goto b; otherwise, continue;

d. Concatenate strings in parallel relationships with "|", and bracket the generated string;

e. If not all NFA_{c_i} has been processed then goto a, otherwise, continue;

f. Replace all delimiter transitions (DTs) in NFA_c to delimiter-like character set repetition $\Sigma\{m,n\}$;

g. Concatenate strings in parallel relationships with "|";

h. Return the generated string.

Figure 13. RE generation procedure

```
(
(((Γ(c)Σ{m,n}Γ(i)Σ{m,n}Γ(t)Σ{m,n}Γ(y))|(Γ(c)Σ{m,n}Γ(i)Σ{m,n}Γ(t)Σ{m,n}Γ(i))|(Γ(シ)Σ{m,n}Γ(テ)Σ{m,n}Γ(イ)))
Σ{m,n}((Γ(b)Σ{m,n}Γ(a)Σ{m,n}Γ(n)Σ{m,n}Γ(k))|(Γ(バ)Σ{m,n}Γ(ン)Σ{m,n}Γ(ク))))
|
(((Γ(花)))
Σ{m,n}((Γ(银))))
|
(((Γ(花)Σ{m,n}Γ(旗))|(Γ(城)Σ{m,n}Γ(市)))
Σ{m,n}((Γ(银)Σ{m,n}Γ(行))))
)
```

Figure 14. RE of NFA_{c_i} when $i = "www.citybank.com"$

```
((u0063)|(u4410)|(uFF43)|(u03F2)|(u217D)|(u0454)|(u04AB)|(u00E7)|(u0255)|(u0481)|(u0188)
|(u0043)|(u216D)|(u0421)|(uFF23)|(u0404)|(u0480)|(u04AA)|(u10BA)|(u00C7)|(u03DA)|(u0187))
```

Figure 15. $\Gamma(char)$ (RE of $S(char)$, when $char = "c"$)

Figure 14 shows the RE generated from NFA_{c_i}, where $i = "www.citibank.com"$. We represent the RE string generated in step c as $\Gamma(char)$, where $char$ is the same character in $S(char)$, and Figure 15 shows an example of $\Gamma(char)$ when $char = "c"$ where we use "u"+Unicode to represent a character. The RE of NFA_c is the concatenation of all REs of NFA_{c_i} ($i \in L_{IRI}$) with "|", as shown in step h of the RE generation procedure.

6. Our Anti-Phishing Framework and the Phishing IRI Pattern Generator

We propose an IRI based phishing obfuscation detection framework. We can use the framework to find out IRIs that could be thought of as susceptible by human. The framework is composed of three modules: RE generation module, IRI parsing module and phishing IRI detection module, as shown in Figure 16. We represent modules with dash-dot-doted block, data with dashed block, data processor with solid block, and data flow with solid arrow-line.

In the RE generation module, we process L_{IRI} manually to generate NFA_{kw}. We built up a tool, RE Generator for Anti-Phishing (REGAP Ver.0.21 [8]), to convert NFA_{kw} directly to RE without intervening the generation of NFA_c. We use XML to represent NFA_{kw} in REGAP as input. Figure 17 shows an NFA_{kw} definition with only one IRI, $"www.citibank.com"$, whose structure was shown in Figure 6. Figure 18 demonstrates the generated RE, where delimiter-like character set Σ is defined in Figure 19. The REs generated by REGAP are standard, such that it can be easily adapted to Perl, Java, C#, Python, PCRE, PHP, VI, JavaScript, Shell tools, etc.

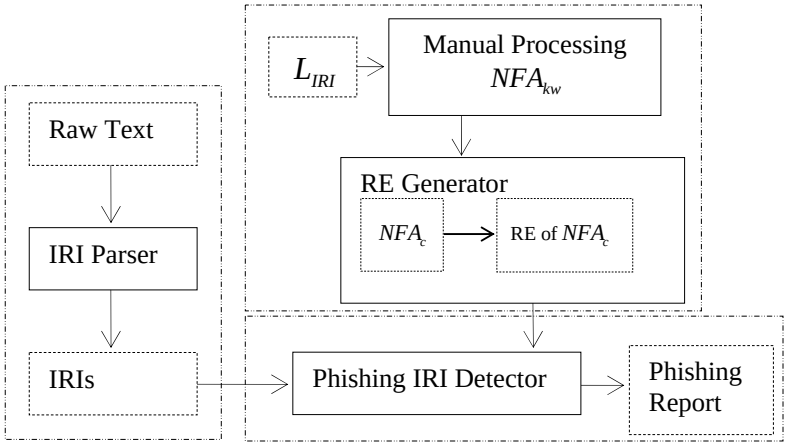

Figure 16. Framework for IRI based Phishing Obfuscation Detection System

```
1.  <IRILIST>
2.  <IRI>
3.  <IRINAME>www.citybank.com</IRINAME>
4.  <WORD>
5.  <WORDNAME>city</WORDNAME>
6.  <SW>city</SW>
7.  <SW>citi</SW>
8.  <SW>城</SW>
9.  <SW>城市</SW>
10. <SW>花旗</SW>
11. <SW>シティ</SW>
12. </WORD>
13. <WORD>
14. <WORDNAME>bank</WORDNAME>
15. <SW>bank</SW>
16. <SW>银</SW>
17. <SW>行</SW>
18. <SW>银行</SW>
19. <SW>バンク</SW>
20. </WORD>
21. </IRI>
22. </IRILIST>
```

Figure 17. An NFA_{kw} definition example of REGAP

```
"((((c |c |c | c |c|є |ç |ç|ɔ |ɕ )Σ (i | i |i|i |¡|i |i |i |i |I|I |I |I| I |I |I |i |I |I |I |I|í| ! |!|! )Σ (t| t
|t |ŧ |t |f |t |t | f |f)Σ (y | y |y|y |y )))|((c |c |c | c |c|є |ç |ç|ɔ |ɕ )Σ (i | i |i|i |¡|i |i |i |i |I|I |I |I| I
|I |I |i | I |I |I |I|í| ! |!|! )Σ (t| t |t |ŧ |t |f |t |t | f |f)Σ (i | i |i|i |¡|i |i |i |i |I|I |I |I| I |I |I |i | I
|I |I |I|í| ! |!|! ))|((城|珹|城))|((城|珹|城)Σ (市))|((花)Σ (旗))|((シ|ジ)Σ (テ|テ|〒)Σ (ィ)))[.]*(((b| b |b |ḅ |ɓ
|ḇ |þ|ɓ |ƀ |ḥ | h |h |ḥ |ẖ |h|ṕ )Σ (a |a| a |ạ |ą |ą |ɐ |ɐ |ə|ə |ə |ɐ |ɐ )Σ ( n |n|ṇ |n |ṇ |ɳ |ɳ |ṉ |ɳ |
ɳ |ŋ |н |н )Σ ( k |k|ķ |ḳ |ķ |ḳ ))|((银))|((行|行))|((银)Σ (行|行))|((ハ|ハ)Σ (ン)Σ (ク))))"
```

Figure 18. RE generated by REGAP from the NFA_{kw} definition in Figure 17.

```
Σ =(¯ |- |-|_ | - | - |-|- |- )|(. |·|.|· | . |ˉ |·|· |· )|(¯ | ¯| _ |_|¯|⁻|–  |¯ )|(~| ~ |− )|(: |:| : |:|: )|(/| / )|( ?
|?)|(#| # )|([| [ |[ |l |l |ll | |Ĺ |Í |Í)|( ] |]|ĵ |ĵ |j |j | j |j|Ĩ )|(@|@)|( ! |! |!| I |II |II |I |I |I | I |I |I|¡| i
|i|i |i |i |ı |ı |i |i |i )|($| $ )|( & |&)|( ' |')|((| ( )|( ) |))|(*| * |⋆ )|(+| + )|(,| , |’ |, )|( ; |;|; )|(=| = )|(\"| " )
```

Figure 19. Delimiter-like character set Σ

In the IRI parsing module, the raw texts which could contain any phishing IRI, such as those from Email body, Instance Message, BBS, Chatting room, or files, are processed by an IRI Parser to collect all IRIs appearing in the raw text. In the phishing IRI detection module, the Phishing IRI Detector utilizes the RE generated from the first module and the IRIs generated from the second module to perform a pattern matching process. If an IRI can be accepted by the

RE and is not an exact IRI in the protected IRI set, it is considered as a potential phishing IRI. Finally, the potential phishing IRIs are reported.

7. Conclusion and Future Work

In this paper, we address a potential IRI based phishing obfuscation problem and propose an effective solution. We built up the UC-SimList, which can be used to find similar characters for a specified one easily. We propose an effective detection approach to this problem using NFA, and address its construction detail. We built up a potential phishing IRI pattern generator REGAP Ver. 0.21, which can facilitate the generation of regular expression from the keyword-level NFA. With the utilization and popularization of IRI, this kind of phishing attacks are very likely to appear and our method is ready for them.

Currently, we only use the simple pixel-overlapping method to measure visual similarly or characters. More accurate methods, like OCR, could definitely be employed.

We have investigated only three popular ways to replace keywords in IRIs. However, there may be more semantics-based keywords replacement methods which could be used for IRI-based phishing attacks and they can be investigated in further work.

As discussed in Section 3, we can only included English (Uppercase and Lowercase), Chinese (Simplified and Traditional), and Japanese (Hiragana and Katakana) characters the current UC-$SimList_s$. There is more work for other languages need to be accomplished in future work. We construct the keyword-level NFA manually at this stage. Although the atomization of this process may involve more complicated semantic analysis, it still could be possible to find a simple method for it. The anti-phishing framework we proposed in Section 6 could be easily adapted to various systems which can process the content of Email, Instance Message, BBS, Chatting room, or files, etc., to perform phishing IRI detection.

Acknowledgement

We thank Dr. Felix Sasaki from University of Bielefeld for his useful discussion and suggestions.

References

1. ANSI, *American Standard Code for Information Interchange*, www.ansi.org
2. Anti-Phishing Working Group, http://www.antiphishing.org.
3. Berners-Lee T., Fielding R., Masinter L., *RFC 3986: Uniform Resource Identifier (URI): Generic Syntax*, The Internet Society (2005), Jan. 2005.
4. Chowdhury A., Frieder O., Grossman D., and McCabe. M. *Collection statistics for fast duplicate document detection*. ACM Transactions on Information Systems (TOIS) 20(2), pages 171–191, 2002.
5. Dhamija R., Tygar J. D., *The Battle Against Phishing: Dynamic Security Skins*, Symposium On Usable Privacy and Security (SOUPS) 2005.
6. Duerst M.,Suignard M., *RFC 3987: Internationalized Resource Identifiers (IRIs)*, The Internet Society (2005), Jan. 2005.
7. Fu Y., Liu W., Deng X., *EMD based Visual Similarity for Detection of Phishing Webpages*, Research Output of Dept. of CS, City University of Hong Kong, http://www.cs.cityu.edu.hk/~anthony/publication/emdphishing.pdf
8. Fu Y., http://www.cs.cityu.edu.hk/~anthony/AntiPhishing/IRI, Jun. 2005

9. Hoad T.C. and Zobel J. *Methods for identifying versioned and plagiarized documents*, Journal of the American Society for Information Science and Technology, 54(3), pages 203–215, 2003.
10. Jakobsson M., *Modeling and Preventing Phishing Attacks*, Phishing Panel of Financial Cryptography, 2005
11. Linz P., *An Introduction to Formal Languages and Automata Third Edition*, Jones and Bartlett Publishers, Inc.,2001
12. Liu W., Huang G., Liu X., Zhang M., Deng X., *Detection of Phishing Webpages based on Visual Similarity*, Poster, WWW2005, pp.1060-1061.
13. Liu W., Huang G., Liu X., Zhang M., Deng X., *Phishing Webpage Detection*, to appear in Proc. ICDAR 2005
14. Microsoft, *Description of the Arial Unicode MS font in Word 2002*, http://support.microsoft.com/kb/q287247/
15. Microsoft, *Arial Unicode MS, Version 1.01*, 2000
16. Nanno T., Saito S., and Okumura M., *Structuring Webpages based on repetition of elements*, In Proceedings of the 7th International Conference on Document Analysis and Recognition (ICDAR2003), 2003.
17. Netscape, http://wp.netscape.com/eng/ssl3/
18. The Unicode Consortium, http://www.unicode.org/
19. Wood L., *Document Object Model (DOM) Level 1 Specification*. http://www.w3.org/TR/REC-DOM-Level-1.