WEB IDENTITY SECURITY: ADVANCED PHISHING ATTACKS AND COUNTER MEASURES

ANTHONY YINGJIE FU

DOCTOR OF PHILOSOPHY

CITY UNIVERDITY OF HONG KONG

SEPTEMBER 2006

CITY UNIVERSITY OF HONG KONG
香港城市大學

Web Identity Security: Advanced Phishing Attacks and Counter Measures
網絡偽裝及身份安全鑑證

Submitted to
Department of Computer Science
電腦科學系
in Parcial Fulfiillment of the Requirments
for the Degree of Doctor of Philosophy
哲學博士學位

BY

Anthony Yingjie Fu
傅英杰

September 2006
二零零六年九月

Acknowledgement

Here I give special thanks to Professor Xiaotie Deng, my advisor and supervisor. He is a wonderful mentor and an excellent scientist. His foresight in computer science guided me to an exciting research area. His patient instruction and nice hints encouraged me to think in a more profound and pervasive way. I also give special thanks to Dr. Wenyin Liu, my advisor and co-supervisor. He is a very motivated scientist and helpful advisor. He has an insightful understanding of engineering research. I started to love engineering research due to his impact. I also thank Professor Robert C. Miller, my supervisor at MIT. He is one of the most creative people I've ever met and worked with. His challenging questions always push me to squeeze solutions from my head. I learnt that hard problems really need hard thinking to solve. All their instructions and help finally guided my research works to this PhD dissertation.

I gratefully thank my qualifying members, Professor Xiaohua Jia, and Dr. Chung Keung Poon, for their helpful advice and suggestions on my research reports' evaluation in the past years.

I would like to thank Yin Liu of Microsoft, Guanglin Hwang of Oracle for time spent with me on research problems. I should thank Greg Little and Min Wu of MIT for so many great and nice discussions. Their ideas are wonderful to make interesting topics. I should also thank Simson L. Garfinkel of Harvard University and Richard Conlan of Northeastern University for their active discussions and suggestions at the CHI-SEC reading group meetings.

Thank you, Hao Zhao, Xiao Wu, Xiaowen Liu, Tommy Yang, Wan Zhang, Jianxi Fan, and Xiaojian Tian of CityU. Thank you, Derek Rayside, Philip Guo, Robert Seater,

Tahina Ramananadro of MIT. You are all very nice and helpful. The daily life as a graduate student is always difficult, full of hard work, and sometimes boresome. Your help makes it easy. I owe you guys.

I owe the most to my Dad Jintao Fu, Mom Lianfen Dong, and Daphne. Your love and support are always my great motivation to research works.

I'm sorry to those who are not listed here. So many people have helped me during my PhD study. Here, I would like to thank all of you who have helped on my research works of this thesis!

Abstract

Phishing is an emerging type of social engineering crime on the Web. Most phishers initiate attacks by sending emails to potential victims. These emails lure users to access fake websites, and induce them to expose sensitive and/or private information.

The rapid development and evolution of phishing techniques pose a big challenge in Web identity security for computer science researchers in both academia and industry. Advanced counter measures are required in urgency.

All phishing attacks spoof users from the visual level and semantic level, i.e., they make the appearances of web pages look similar to the real ones and make the web links and web page contents semantically related to the real ones. All such scams happen through human computer interaction. In this dissertation, we address a series of advanced counter measures against the most prominent phishing scams. We also propose a tool to provide web page originality verification.

The backbone of this dissertation consists of four parts: *Visual Assessment Approach*, *Semantic Assessment Approach*, *Human Computer Interaction Enforcement*, and *Website Originality Verification*. We develop a system with algorithmic solutions for handling the phishing problem with different tools. In addition, we present experimental results showing the effectiveness of our work. Finally, we discuss future research topics and directions.

In the Visual Assessment Approach, we detect phishing attacks using visual features of web pages. Study in [48] demonstrated that visual assessment based detection is successful, but it fails in advanced phishing attacks, as shown in Chapter 2 (Section 2.2).

The reason is that visual assessment at the code level (HTML) is not actually reflecting the classification curve of human eye assessment. In this dissertation, we evaluate visual features that are completely from the computer screen. We use Earth Mover's Distance (EMD), a linear programming model, to assess the visual features' similarity. We create a system, *SiteWatcher*, to monitor email servers and client side network traffic. We parse out the web links in suspected emails and network traffic, and retrieve corresponding web pages from the Web in HTML. We further convert the retrieved web pages into images and use EMD to calculate their similarity to the protected web pages. Our experiments show that this method is superior to all other visual assessment methods.

The Semantic Assessment Approach focuses on the text similarity assessment. Text based obfuscation is mainly seen in Unicode attacks. We propose a general methodology to detect Unicode attacks. We evaluate the similarity of Unicode strings from char-char similarity (visual similarity and semantic similarity), word-word similarity (semantic similarity), and word string similarity (semantic similarity). We build up a Unicode attack detection system and evaluate its effectiveness and performance. To our knowledge, this is the first Unicode attack detection system.

The Human Computer Interaction Enforcement Approach focuses on developing new human computer interaction model to guide users to avoid making mistakes. As a matter of fact, whatever phishing attacks happen on visual level or semantic level, all spoofing scams are carried out through human computer interaction (HCI). However, we demonstrate these applications' GUIs are not guaranteed to be secure. Our study shows that no place on computer screens is guaranteed to be secure, as shown in Chapter 2 (Section 2.4.1). The secure HCI is more difficult than ever expected, because anti-

phishing applications have graphical user interfaces (GUI) and these GUI can be faked. We call it Screenjacking, which means phishing attacks applications, including the anti-phishing applications. To solve this problem, we propose five different UI methods: *Spring Loaded Security Key*, *Application Trace*, *Genuine Skin*, *Merged Approach*, and *User Challenge*, to improve GUIs' security. We also propose two password mechanisms: *Semi-Random Password*, and *Context Sensitive Password*, to improve public HCI security and password security. Semi-Random Password can provide better security for key logging based attacks. Context Sensitive Password is an exciting tool that protects users from both phishing and Screenjacking attacks.

Website Originality Verification aims to give strong evidence based definitions to phisher and victim. We design and build a system, *DistAca* (short for "Distinguisher for Academia"), to verifying the originality of web pages (or any other data). We can make secure evidence to prove a websites' originality with DistAca.

The above four approaches compose a systematic anti-phishing solution. This solution aims at the advanced counter measures against Web identity fraud. The experiment shows that these approaches are effective to protect users from phishing attack.

Table of Contents

List of Figures

Chapter 1

Introduction

Phishing is a kind of amalgamation of Web technology and social engineering. The most popular phishing scams are carried out using phishing web pages. Phishing web pages are forged to mimic certain legitimate companies' web pages. Phishing websites usually trick users into leaking their sensitive information and private data by counterfeiting trustworthy web identities. Unwary users may easily be deceived by such scams. Victims of phishing web pages may expose their bank accounts, passwords, credit card numbers, or other important information to malicious people.

Phishing is a relatively new internet crime in comparison to viruses and hacking, but it is becoming increasingly common. One report of Anti-Phishing Working Group (APWG) in 2004 [6] showed that phishing web page number was increasing 50% for each month and usually 5% of the phishing email receivers would responded to the scams. There were 15,050 phishing cases reported to APWG in June, 2005 [6]. Phishing became a severe Web security and privacy problem. It has caused huge negative impacts to various areas of our society already. The problem has drawn a lot of attention from both academic and industrial researchers. Phishing is threatening people's confidence to use the Web to conduct online finance related activities, personal information management, and online information exchange. It is in the public interest for Internet service providers to implement anti-phishing measures to protect their users.

The anti-phishing approaches can be structured into three levels: visual assessment (graphic level), semantic assessment (text level), and human computer interaction (HCI) enforcement. Both graphic level phishing attacks and text level attacks happen through the interaction between human and computer, such that the improvement of HCI mechanisms directly impacts the security and usability of Web applications. Another very important and basic problem is protecting web pages by providing strong evidence to prove their originality, such that we can define phisher and victim.

Visual assessment addresses the approach to protecting web pages from being mimicked on the graphic level. To spoof Internet users, phishing web pages usually have high visual similarities to the targeted real ones. Some of these web pages even look identical to the real ones. Phishers can simply download targeted web pages from the real websites and put them onto the phishing websites. Some users will be spoofed if they are unwary to visually familiar web pages. These users may usually make a simple judgment by thinking "I saw this web page many times, so it must be the right one." Hence it is helpful to detect phishing web pages by analyzing the similarity of the HTML structure of web pages. Previous works have explored this idea and it works, i.e., Liu et al [48]. However, many security sensitive websites are becoming aware of these phishing methods and using web pages that cannot be crawled, e.g. www.hsbc.com uses dynamic web pages, and www.ppshk.com uses Java Applet. Hence, phishers have to recreate phishing web pages with other techniques. As a result, the recreated web pages may be very different from the real ones on coding level. The fact is that phishers do not have to construct visually similar web pages using similar code level representations. What they really need is to construct web pages that look and behave like the real ones. Phishers may also

be sly enough to create homographic web pages to avoid a code level filters' detection. Therefore, we cannot expect to detect phishing web pages simply by analyzing their HTML. In this dissertation, we will bring the web page similarity assessment approach completely to the screen appearance. We propose to convert HTML web pages to images, and use Earth Mover's Distance to evaluate the images' similarity to the protected ones. We build a system, SiteWatcher, based on this method. SiteWatcher parses out web links from suspected network traffic. It retrieves web pages of these web links and converts them to images. It calculates the similarity of these images to the protected ones to make classification.

Semantic assessment addresses the hazardous phishing attacks on the text level with high potential. Phishers usually use visually and semantically similar web addresses to deceive users. While the public has been gradually educated, and has become more alert to the simple scams above, phishers are resorting to more and more sophisticated tricks to avoid detection. One widely used method is to cover the address bar with Java Script or ActiveX components. They have also tried to add prefixes or suffixes in web links to mimic real web links, e.g. they can use suffix spoofing like www.citibank.com.info.biz to mimic www.citibank.com. They can even make a web link look exactly identical, e.g. the two fake web links in Figure 52 to www.ebay.com. This is a kind of Unicode attack. Unicode attacks are caused by the coexistence of visually and semantically similar characters in the Unicode repertoire. The critical technology to solve Unicode attacks is to construct a Unicode Character Similarity List (UC-SimList), which can be used to find the visual and semantic similarity of given characters. Unicode attacks can also be used in spam attacks and web identity attacks. We identify these problems and address the

most up-to-date research results in this dissertation. A higher level of Unicode based spoofing is on natural language, so semantic similarity assessment of Unicode strings turns out to be important. We contribute one word-word semantic similarity list as well, which can be used for making further text semantic similarity assessments. We present the solutions to all of these text related attacks as semantic assessment approaches.

Both visual level attacks and semantic level attacks use the HCI on the client side to make attacks happen. The mechanism of HCI dominates the security and usability of Web applications. As a matter of fact, many researchers on anti-phishing highlighted phishing as the failure of HCI mechanisms, and made HCI-SEC (SEC for security) an active research area. Researchers have tried many different ways to improve and redesign user interfaces. However we demonstrate that "*Nowhere on the computer screen can be considered secure*". Therefore, the lock-like icons indicating the usage of SSL, anti-phishing toolbars, and even the web browser can be faked. All such attacks can be carried out through web browsers. We call this kind of phishing attack *Screenjacking*, which means phishing to applications (rather than web pages). Ironically, these applications mainly refer to anti-phishing applications, e.g., all anti-phishing toolbars, as well as PWDHash [61], Dynamic Secure Skin [20], and Web Wallet [73]. We address several anti-Screenjacking proposals: Spring-Loaded Button, Application Trace, Genuine Skin, Merged Solution, and Interactive Image Filter. These designs may help solve the Screenjacking problems well. We also propose two password mechanisms to improve user security and privacy protection. Semi-Random Password can protect users' passwords by allowing them to insert random characters. It can make secure authentication without providing the complete password, which is a securer mechanism

against key logger based attacks. This could also be the only password mechanism that is secure from key logger attacks to a certain extent. We prove Semi-Random Password is secure mathematically, and prove it is secure from time attack through user experiments. Context Sensitive Password protects users' passwords by requesting that users revise a personalized context string. It provides a mechanism to guide users' recognition of a context string into the critical path of their workflow. It is such an exciting password mechanism because it can defend against phishing and Screenjacking attacks simultaneously.

We also address one basic problem of anti-phishing by answering "*Who is mimicking whom* (*or how to verify a web identity possess the originality of one web page rather than others*)*?*" One tricky dilemma for all anti-phishing methods is that we have not provided a way to prove the originality of web pages. Anti-phishing turns out to be meaningless if we cannot identify phisher and victim. For example, if a phisher created a web page that looks very similar to www.citibank.com and claims CitiBank is mimicking his web page, how does CitiBank prove that its web page is the original? As more and more web sites are created, we cannot expect to define the originality of all web pages empirically. The same problems also occur in article plagiarism. Hence, web sites' owners need a convenient tool to provide solid evidence which can prove that a set of web pages are created by them at a certain time. We argue that a time stamping system is not secure unless the client application is open source. However, no such secure system exists. We design and implement such a system, DistAca, which can help solve such problems.

The rest of this dissertation is organized as follows. We talk about phishing attacks and their potential impacts in Chapter 2. We review related research works and address the

general framework for our counter measures in Chapter 3. We discuss the visual assessment approach in Chapter 4. We address the semantic assessment approach in Chapter 5. We discuss human computer interaction enforcement in Chapter 6. We introduce the design and usage of DistAca system to distinguish originality for web identity in Chapter 7. Finally we conclude all these research works in Chapter 8.

Chapter 2

Phishing Attacks and Their Potential Impacts

No evidence shows when and where the first phishing case took place, but phishing, a web based crime, was first reported as a web problem in 2001 by Consumer Sentinel [14], the complaint database developed and maintained by the FTC (Federal Trade Commission) of the USA. 55,727 cases of Internet related fraud complaints were reported that year. From then on, the related complaints increased dramatically, doubling for each subsequent year. Most of the fraud complaints have been recognized as phishing attacks. Phishing attacks have also been found growing in an accelerative manner. One report of the Anti-Phishing Working Group in 2004 [6] reported that the number of phishing attacks increases 50% each month and 5% of the phishing email receivers respond to them. The phishing problem has drawn a lot of attention in both academic and industrial research areas.

Phishing techniques were very simple in the early stage. Phishers just download web pages from the legitimate websites or create websites that look like the real ones. Evolving with the anti-phishing techniques, users are progressively aware of and alert to such scams. Many users learned to check the SSL icon and domain name in the address bar. However, phishers are always trying to use more sophisticated activities to circumvent detection and user suspicion. Phishers are always trying their best to make the appearance of web links and the content of their web pages look similar to the real ones.

Various more complicated and hard-to-detect phishing techniques are used by phishers. Hence anti-phishing turns out to be a very difficult seesaw battle.

2.1 Phishing Attack Classification

The most popular phishing strategies can be classified as *web page obfuscation* and/or *web link obfuscation*.

Web page obfuscation can be carried out in two basic ways, (a) Use the downloaded web pages from real websites or create similar web pages to make them appear and react similar to the real ones; (b) Use graphical components rather than HTML to avoid HTML based phishing detection, e.g. the graphical components could be Java Script, ActiveX, Macromedia Flash, Java Applet, and Image. They can be used to create visually identical web pages as well.

Web link obfuscation can be carried out in five basic ways, (a) Add prefix or suffix to domain name to generate faked URL, such as www.citibank.com.info123.com ; (b) Use actual links which are different from the visible ones, e.g. CitiBank ; (c) Utilize website bugs to redirect the link to the phishing web pages, e.g. http://www.google.com/url?sa=U&start=4&q=http://www.mit.edu/~ayf can be redirected to a webpage other than Google; (d) Use cousin domain names (e.g., to replace certain characters in the target URL with similar characters) [27], e.g. use www.1CBC.com (“1” is number one) to mimic www.ICBC.com (“I” is upper case “i”); (e) Use Java Script or ActiveX components to mimic or cover the address bars to make users believe they have entered the correct websites.

2.2 Web Page Attack

Most phishers would create web pages that are visually similar (or identical) to the targeted ones to increase the disguising level. They can simply download the web pages from the real websites or manually recreate them. We found most of the early phishing web pages are very similar to the real web pages by investigating their appearances and HTML code. Hence it is possible to detect phishing web pages by evaluating the visual similarity of web pages. Liu et al proposed the HTML based visual similarity assessment in [48] to address this problem. They have a system running in an internal server of City University of Hong Kong [5]. However, this system cannot detect phishing web pages that are created with different source codes, because Flash, Movie, ActiveX, Java Applet and various types of visual components can be embedded into the web pages instead of HTML. Therefore, a real web page can correspond to countless fake web pages with different coding. Appendix A demonstrates that visually identical web pages can be composed by totally different coding. Homographic web page is one of our major motivations of investigating into the phishing detection method at the graphical level.

2.3 Unicode Attack and IRI/IDN Attack

The globalization of information processing systems promoted the usage of Unicode, which allows users speaking different languages to represent character based information in their own scripts. Unicode is convenient for users. However it also brings potential security risks, because there are many visually or semantically similar characters coexist in the UCS (Universal Character Set). UCS has a large set of characters. It covers the characters and symbols used in of most languages' scripts in the world. We use a

character-character similarity algorithm described in Section 5.2.2 to find similar characters in UCS. Figure 1 shows a few characters similar to “a”, “b”, and “c” respectively in UCS, where the hexadecimal number under each character is the character code. Obviously, there are at least two other characters look exactly the same with character “a”, one with “b” and four with “c”. There will be more similar characters if we count in semantically similar characters, e.g., “a” and “A”.

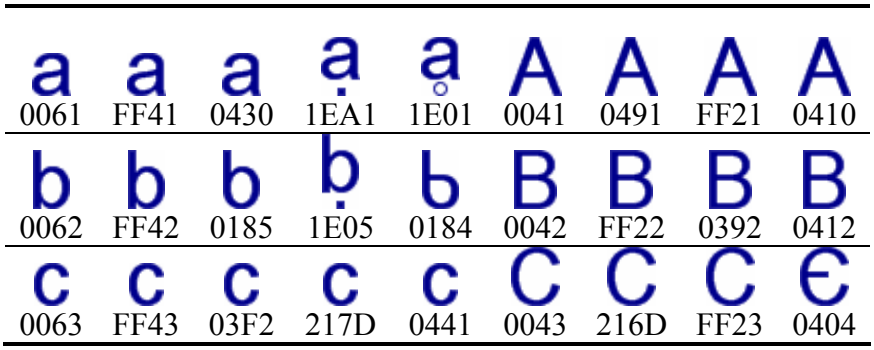


Figure 1. Similar characters to “a”, “b”, and “c” (in Arial Unicode MS Font)

Users do not usually look into the code of Unicode strings to verify their validity. Hence, this phenomenon opens a door for malicious people to spoof users by replacing some characters with other visually or semantically similar ones from the UCS. We call it *Unicode Attack*.

We classify Unicode attacks into three categories: (1) *Spam attacks*, many machine learning techniques used in anti-spam filters take an email as a sequence of characters, and use email contents to generate spam patterns. However, if spammers replace characters with similar ones into emails’ content, we think they may bypass such filters. Phishing attacks are mostly initiated from spam emails. Hence anti-spam has a priority in anti-phishing problems. (2) *Phishing attacks*, malicious people can use similar characters as replacements in IRI/IDN [21] to create visually similar domain names. Ordinary users

may not look into the code under the domain name strings to verify their validity. User studies in [19] show that almost all users judge the validity of a website by the domain name, which makes this attack particularly hazardous. (3) *Web identity attack*, there is a requirement for information systems to use Unicode strings to represent user names (or accounts). Users also identify each other through the appearance of their user names (accounts). This allows malicious people to imitate other ones by registering visually identical or similar user names (account) with Unicode attack.

Unicode attack based phishing attack is the most potentially hazardous problem. Current internet resources are identified by ASCII based Uniform Resource Identifiers (URIs) [8]. However, URIs could be cumbersome for users speaking languages other than English would like to use more familiar character scripts to identify their web resources. Most non-Latin language scripts have a mapping to Latin characters, such as Chinese Pinyin and Japanese Romaji. Users are using these methods to represent URIs in their languages, such that different companies or organizations may want the same domain name while they want it for different semantic meanings. Such problem is called URI confliction and semantic ambiguity. Nevertheless, with globalization of information technology, users are using localized operating systems, applications, etc. Users are eager to use these systems with their native scripts, including the activity of locating universal resources. IRI/IDN is proposed as a complementary to URI. It is a sequence of characters from a subset of UCS. This permits most non-Latin scripts to be freely represented in IRI/IDN. This allows Chinese users who are unfamiliar with English to input "花旗银行.公司" ("花旗银行" is pronounced "Hua Qi Yin Hang" and stands for "Citibank"; "公司" is

pronounced “Gong Si” and stands for “Company”) rather than “citibank.com”. Japanese users may enter “シテイバンク.会社” (“シテイバンク” is pronounced “Shi Tei Ban Ku”, and stands for “Citibank”; “会社” is pronounced “Kai Shya” and stands for “Company”) to access the web page of CitiBank as well. It makes the Internet more user-friendly, but we need to address the potential threats in terms of Unicode attacks. Similarly, the problem may exist as a system allows Unicode string based web identities, e.g., someone might be able to register an account and pretend to be Bill Gates (such as billg@microsoft.com for MSN Messenger) and send a message to his CEO, “Hi, Steve, I finally decided to open the source code of Vista and donate Google a billion dollars. Please do it asap!”

Unicode provides a huge number of possible mutations for strings. For example, a simple Unicode string, “citibank” could have 24(c) * 58(i) * 21(t) * 58(i) * 24(b) * 22(a) * 21(n) * 14(k) -1=263,189,025,791 potential mutations. It may not be surprising that we have not found any registration system (including domain name registrars, chatting rooms, BBS services, etc.) is attempting to detect Unicode attacks. As a matter of fact, we have easily registered domain names which are visually similar to several prominent websites. For instance, we registered “www.中国银行.com” (“中国银行” is in Chinese and stands for “bank of China”, where “国” is a very similar character to “国”), “www.中国銀行.com” (“中国銀行” is in Japanese, for “bank of China”), and these two domain names are similar to “www.中国银行.com”, which is the real one. We also registered “www.和

記黃埔.com" ("和記黃埔" is Japanese for "Hutchison", a company in Hong Kong) to mimic "www.和记黄埔.com". We were able to link these domain names to our Anti-Phishing Group website [8], so if the domain name registrars cannot monitor the visually and / or semantically similar domain names' registration, then phishers may disguise phishing websites much better. Hence, the problem makes the detection of Unicode attacks an important direction for researchers in computer security, privacy, and HCI-SEC to look into.

Unicode attack is not just simpal similar-character replacement. It can be more complicated. For instance, spammers can add noisy symbols to spam content, as well as replace words with semantically similar words in order to throw off spam filters. In general, the problem of detecting Unicode attacks may require layers beneath and above the character similarly level, including preprocessing to denoise the data in order to know which characters to compare, and higher level language processing to detect similarities at a word or semantic level.

2.4 Screenjacking Attack

Following the proposals on various anti-phishing toolbars, Ross et al proposed *PWDHash* [61], Dhamija et al proposed *Dynamic Security Skin* [20], and Wu et al proposed *Web Wallet* [73] for improvement of user interface design. Behind the discussion on these anti-phishing applications, there is another interesting problem to address, "*How to make an anti-phishing tool is trustable to users, because faked anti-phishing tool could be the most harmful attack to these tools [73]?*" In this dissertation, we introduce a concept and

phenomenon, *Screenjacking*, which means phishing attack to client side applications. The act of counter measures against *Screenjacking* is referred to as *Anti-Screenjacking*.

2.4.1 No Part of the Screen can be Assumed as Secure

The major method that we are using to access the Web is web browser. Phishers are always trying their best to make the faked web pages look and behave as similar as possible to the real ones. On the other hand, the web browsers are adding more and more features (e.g. Java Script, ActiveX, Java Applet, Macromedia Flash), plug-ins (e.g. Adobe Acrobat, Google Toolbar, MSN Toolbar, OICQ), and other similar components to empower web browser features. These components and applications are making web browsers too powerful. For instance, ActiveX components and Java Scripts have been used by phishers frequently to hide real address bars in web browsers and other attacks. Hence, under no circumstances we can assume a part on the computer screen is secured (not faked) any more. We demonstrated that ActiveX can take control of the complete screen in [1].

2.4.2 The Applications with GUI are in Danger

Most software products are using graphical user interfaces (GUI) nowadays, including most of the security related applications. Anti-phishing toolbars are using GUIs to do configurations, system maintenance, and phishing alerts' indications. Other new web security technologies other than anti-phishing toolbars, such as *PWDHash* [61], *Dynamic Security Skin* [20], and *Web Wallet* [73] are also using GUIs to finish similar processes. The problem turns out that these applications are also under potential attacks from malicious people. They can simply fake GUIs or cover the legitimate ones. Therefore, we

need to design a counter measure against this kind of attacks. We demonstrate that the three anti-phishing tools, PWDHash, Dynamic Security Skin, and Web Wallet, are potentially to be faked in web pages in Appendix B.

2.5 Web Page Originality

There are cases that we want to convince somebody that we had done something earlier than others. In the Web, originality juggle is a dilemma situation that a malicious guy creates a web page, e.g. looks similar to www.citibank.com, and claim CitiBank is copying the web page from him. Generally, in our research works, there are also such problems we can ask, "*How to prove a certain piece of paper or articles to be original? Can we provide an effective counter measure against the article plagiarism problems?*" There are cases that, a reviewer of a paper rejects the refereed paper and write another one with a similar idea. Hence, we are expecting a method to prove that we have written a web page or paper without disclosing the content (of web pages or papers). We can build a system to solve such kind of problems easily, effectively, conveniently. Suppose we have a web page or an article finished, we can use a special system to provide evidence that you have ever done it at a certain time by simply pressing a button. It is an interesting problem that worth to be addressed in the dissertation.

Web page originality is important, because it is the fundamental and strong way to solve the conflict between phisher and victim. When conflict happens, there should be a government department (in Hong Kong, it is the court). The part that can provide strong evidence surely can have advantage at making arguments.

Chapter 3

Literature Review and General Framework

3.1 Literature Review

3.1.1 On Web Page Security

Phishing web page detection can use the same methods used for plagiarize document detection, and this document similarity evaluation techniques can be used for the anti-phishing approaches. Typical research efforts on document duplication detection include techniques such as syntactic analysis [10], collection statistics [12], displaying structure [11] [57] [77], visual based understanding [34], and vector space model [65]. Hoad et al have surveyed various methods on plagiarized document detection [38]. Natural language is quite complicated and these methods are not able to detect phishing attacks effectively today [48], although we still consider it to be a promising direction. Liu et al proposed the visual similarity assessment method for web pages in [48], where the concept of DOM [71] based visual approach to phishing detection was first introduced. This method first decomposes the web pages into salient (visually distinguishable) block regions, and then evaluates web pages' similarity in three metrics: block level similarity, layout similarity, and overall style similarity. All these metrics are based on matching assessment of the salient block regions. Our visual assessment approach in this dissertation follows that overall strategy but uses a different way to calculate visual similarity of web pages.

3.1.2 On Character Security

As human use symbols to record and represent languages in information processing and to write HTML and other scripts, symbolic representations are the most widely used interface (interactive media) between human and computer. There is a brief history of using illusive "0/1"s to represent information dating back to the early computer age. To improve the efficiency of human-computer interaction, people later invented ASCII [4] to encode textual information. This made it much easier for users to operate computers. However, ASCII only included only Latin characters and commonly used symbols. People who use other languages need to install additional character sets to satisfy their requirements, such as GB2312 and HZ for simplified Chinese, BIG5 for traditional Chinese, and Shift-JIS for Japanese, etc. With the development of symbol technology and the requirement of information exchange, people expected a unified character system to represent all of these characters. Hence, Unicode was invented. Today, the most popular version of Unicode (ver. 2.1 [67]) uses 16 bits and can represent up to 2^{16}=65,536 characters (the latest version, ver 4.3, uses 32 bits to represent a character) and can represent almost all standard characters/symbols in the world.

Unicode is widely used all over the world. We use it in emails, web pages, resource identifiers (IRI/IDN), and user name (account) registration systems. However, there are many visually and semantically similar characters in the UCS and it is quite easy to use them to generate numerous similar/fake Unicode strings from a given one, called *Unicode attacks*. Some similar/fake Unicode strings are visually identical to the original one. Similar attacks called *homograph attack* were also reported in [31]. They demonstrated the possibility of creating a domain name www.microsoft.com by replacing

"c" and "o" with identical Cyrillic characters. The potential for abusing Unicode increases as it becomes a trend in modern information processing systems. A survey of similar characters in UCS and the problem of IRI/IDN based phishing were introduced in [27]. However no IRI/IDN oriented anti-phishing technique has ever been proposed. As IRI/IDN based phishing attacks could be a critically severe problem for the Internet in the near future, it could disastrous to delay solving this problem until the usage of IRI/IDN becomes popular and phishers start using similar characters in UCS to register forged IRI/IDNs. We address the counter measures in this dissertation.

3.1.3 On Human Computer Interaction Security

Research on anti-phishing by improving user interface and usability turns out to be quite interesting. *PWDHash* [61] is system proposed by Ross et al in Stanford. Researchers tried to change the user behavior of inputting password. Passwords are always inputted by adding prefix "@@" or using "F2" key as the "secure key". "@@" and "F2" key invoke the password filling process that the browser plug-in catches the password indicator and replace the password with a hashed one. They use domain name as hashing salt, so the phishing website cannot provide the correct hashing salt, such that the phishing websites will not get the correct password. This method depends on users' intervention and cannot effectively stop users from exposing their information when they forget using "@@" and "F2" key. *Dynamic Security Skin* [20] was proposed by Dhamija et al in UC Berkeley. It requires that users recognize a visual indicator in the protected web page. However, users normally do not bother to look at such indicator before inputting their passwords because Dynamic Security Skin is not in the critical path of user' workflow. Wu et al [74], an MIT anti-phishing group, demonstrated that visual

indicators are not working well through user study. They proposed *Web Wallet* [73], by which all sensitive information can be effectively controlled. However, these systems may fail against Screenjacking attacks as introduced in Section 2.4.1 that all these methods are not able to avoid the attacks targeting at the anti-phishing tools themselves. We address several possible solutions in this dissertation.

3.1.4 On Originality Verification

In phishing attacks, people are empirically classifying one website as faked or real. Therefore, we demand a method to give solid proof to support such classification. After all, a phisher could claim that a legitimate website is phishing him! A similar matter happens with academic research and patents. However, the measures employed in those cases are not ideal for protecting the web pages, as they cost a lot of money, time, and human effort. People can only say they have done some original work by publishing papers or making library archived technical reports. One alternative is to archive them in ePrint [16]. However, libraries use librarian to process the technical reports and ePrint use reviewers to read the articles before putting them into the database. Since such systems are depending on human, the security baseline is to trust these people. In this dissertation, we introduce an open-source system, DistAca, to solve this problem without relying on trusted human mediators. DistAca addresses a similar idea to Time Stamp Protocol (TSP, RFC 3161[2]). The real system in the industry is E-Time Stamp [23], which uses TSP. However, E-Time Stamp does not open the source code of the client side, which makes the client application behaves like an electronic version of a librarian.

3.1.5 Other

There are many popular anti-phishing techniques and tools in the industry area. The most popular anti-phishing methods include authentication, which includes email authentication and web page authentication (e.g., [57]), email filtering and web page filtering, attack analyzing and tracing, immune-system-like phishing report and detection, and Law enforcement from government. Many of them are deployed as web browser toolbars, email client agent toolbars, and distributed server systems. APWG provides a solution directory [6] which contains most of the popular anti-phishing companies in the world. Another commonly used web security method is [57] SSL. However, it only provides secure communication channels between clients and servers. It is true that attacks cannot get useful information through eavesdropping, but they are not necessarily to use SSL since many users do not care about SSL. People have also proposed methodologies for evaluating the risk of phishing attacks, e.g. [40].

3.2 Counter Measure Framework

Phishing activities happen at user side computers. What phishers are trying to do is to spoof users and induce them to make mistakes. We assume the screen is the only interface that users can get information from, and keyboard (and a mouse) is the only interface that users can use to input information. Hence, the spoofing scams used by phishers are to deceive users to misrecognizing its fraudulence. They can also make the interactive processes similar to the real ones, in order to spoof users. Users accept information from screen and perform recognition on both graphic and text levels. Hence, we classify counter measures from these two levels. These two classes of counter

measures are based on web browsers, but some advanced phishing techniques can break out the range of web browsers. We demonstrate that no place on the screen can be assumed to be secure. We try to solve such a problem by enforcing HCI mechanisms. Finally, we will discuss web page originality strong verification and describe our implementation of DistAca to follow up the proposed idea.

3.2.1 Visual Assessment

Visual assessment of phishing detection is carried out on the graphic level of a computer screen. It is not secure to depend on evaluating web page similarity on code level, as shown in Appendix A. Hence, we propose an effective approach on the graphical level. We use Earth Mover's Distance (EMD) to measure web page visual similarity. EMD describes the method to transport different weighted products from a set of stores to another set with minimum total cost. We use this model to represent the minimum effort to transform one image into another. We convert the involved web pages into images with lower resolution, then use color and coordinate features to represent the image signatures. We calculate the signature distances (EMDs) of these images to protected ones by calculating an EMD threshold vector through supervised training. A large scale experiment with 10,281 suspected web pages was carried out to demonstrate high classification precision, phishing recall, and applicable time performance for online enterprise solution. We compare this method with two other visual assessment methods to manifest its advantages. We also built up a prototype system which is running online, which has caught many real phishing cases.

3.2.2 Semantic Assessment

In semantic assessment approaches, we would like to detect phishing attacks on the text level. Our discussion will focus on Unicode string based attacks. Unicode is a dominant character representation method for information processing. However, it presents a very dangerous usability and security problem for web based applications. The problem is that many characters in UCS are visually or semantically similar to each other. It provides a way for malicious people to carry out spam attacks, phishing attacks, and web identity attacks. In this dissertation, we address the potential attacks, and propose a methodology to fight against them. To evaluate the feasibility of our methodology, we construct Unicode Character Similarity List (UC-SimList), and then implement a visual and semantic based edit distance (VSED), as well as a visual and semantic based Knuth-Morris-Pratt algorithm (VSKMP), to detect Unicode attacks. We develop a Unicode attack detection tool, IDN-SecuChecker, to detect phishing web links and fake user name (account) attacks. We also introduce a system that can be used to automate the generation of regular expressions to detect phishing IRI/IDNs. The foundation of semantic assessment includes the construction of UC-SimList and word-word semantic similarity (WWSS) List. We propose three methods to construct the UC-SimList, which includes pixel overlapping [27], Kernel Density Estimation [15], and quadratic spline similarity. We propose one method, Information-Based Measure [60], to construct the WWSS. Each of these methods forms a section in this dissertation.

3.2.3 Human Computer Interaction Enforcement

Many studies of anti-phishing rely on phishing detection applications under a basic assumption that "*the web browsers and anti-phishing applications are real*". This

assumption is not true in advanced phishing attacks. We demonstrate that nowhere on the screen can be assumed as secure. Users play an important factor in anti-phishing. As Screenjacking is a big potential threat of HCI based anti-phishing applications, we address several possible anti-Screenjacking solutions. We use *Web Wallet* [73] as the target application to create the referential UI, which can be the control study to evaluate the effectiveness of proposed anti-Screenjacking methods. We use 5 methods to achieve secure user interface: (1) Spring-Loaded Button and disabling other applications. (2) Application Trace, which uses hash visualization and specified screen area to illustrate the genuineness of an application. (3) Genuine Skin, which uses specified images as the hallmark of the products of a company, and uses a visual detection engine in the background to detect illegitimate imitation. (4) Interactive Image Filter, which uses personalized images and image-filters to challenge the application that users are interacting with. (5) Merged Method, which uses an integrated version of the first three methods as one approach. We also propose two security-enhanced password mechanisms: Semi-Random Password and Context Sensitive Password, to enforce the UI security of passwords.

3.2.4 Distinguish Originality

The originality of a website is very important, and there in no secure way to prove the originality of it. We design and implement such a system, DistAca. We call the processes of time-stamping a file into the DistAca *distinguish*. Any type of document can be distinguished by DistAca, e.g. text, audio, video, and image. The process of distinguishing one file is as simple clicking a button. In this dissertation, we address the design, implementation, and demonstration of DistAca.

Chapter 4

Visual Assessment Approach

In this chapter, we propose an effective approach to detecting phishing web pages. We employ the Earth Mover's Distance (EMD) [37] to calculate the visual similarity of web pages. The most important reason that internet users could become phishing victims is that phishing web pages usually have high visual similarity with real web pages, such as visually similar block layouts, dominant colors, images, and fonts etc. We follow the anti-phishing strategy in [48] to obtain suspected web pages. We parse URLs in suspected emails, obtain web pages of these URLs, and convert them into normalized images. We represent signatures of these images with features composed of the dominant color category and its corresponding centroid coordinate to calculate the visual similarity to protected web pages. The linear programming algorithm [36] is used to calculate EMDs. EMD represent the dissimilarity of two given signatures. An anti-phishing system can protect many web pages simultaneously. A threshold is calculated for each protected web page using supervised training. If the EMD based visual similarity of a web page exceeds the threshold of a protected web page, we classify the web page as phishing.

Experiments on a large scale have been carried out. Our experiments showed high classification precision, high phishing recall, and satisfactory time performance. We also made a performance comparison to the method proposed by Liu et al in [48] and the HTML based EMD method. Our experimental results showed that the proposed method is superior to the other two.

4.1 EMD based Visual Assessment

4.1.1 The Earth Mover's Distance

EMD [37] is a method to evaluate the distance (dissimilarity) between two given signatures. A signature is a set of features and their corresponding weights. The method comes from a well known transportation problem. Suppose we have *m* producers, each with a weight representing the amount of product it has. We denote producer set *P* as:

$$P = \{(p_1, w_{p_1}), (p_2, w_{p_2}), ..., (p_m, w_{p_m})\} \quad 1)$$

Suppose we also have *n* customers, each with a weight indicating the amount of product he needs. We denote the consumer set C as:

$$C = \{(c_1, w_{c_1}), (c_2, w_{c_2}), ..., (c_n, w_{c_n})\} \quad 2)$$

Producers want to transport their products to consumers. Suppose the distances of each pair of producer and consumer are given, and they are represented into a distance matrix *D*, which is defined before calculating EMD. It is represented as:

$$D=[d_{ij}], \text{ where } 1 \le i \le m \text{ and } 1 \le j \le n \quad 3)$$

Producers all produce the same product and consumers all consume the same product. The transportation fee is proportional to both distance and product weight. The task is to find a flow matrix *F*, which contains factors indicating the amount of product to be moved from one producer to one consumer.

$$F=[f_{ij}], \text{ where } 1 \le i \le m \text{ and } 1 \le j \le n \quad 4)$$

The transported product amount from *P* to *C* should be as much as possible and the total transportation fee should be minimized. The total cost of transportation fee can be represented as:

$$\sum_{i=1}^{m}\sum_{j=1}^{n} f_{ij} \cdot d_{ij} \qquad 5)$$

The calculation of *F* is subject to the following constraints:

$$s.t.\begin{cases} f_{ij} \geq 0 \quad where\ 1 \leq i \leq m, 1 \leq j \leq n \\ \sum_{j=1}^{n} f_{ij} \leq w_{pi} \quad where\ 1 \leq i \leq m \\ \sum_{i=1}^{m} f_{ij} \leq w_{cj} \quad where\ 1 \leq j \leq n \\ \sum_{i=1}^{m}\sum_{j=1}^{n} f_{ij} = Min(\sum_{i=1}^{m} w_{pi}, \sum_{j=1}^{n} w_{cj}) \end{cases} \qquad 6)$$

This is a linear programming problem. We solve it to get *F*, and then calculate EMD. The EMD can be represented as:

$$EMD(P,C,D) = \frac{\sum_{i=1}^{m}\sum_{j=1}^{n}(f_{ij} \cdot d_{ij})}{\sum_{i=1}^{m}\sum_{j=1}^{n} f_{ij}} \qquad 7)$$

It has been shown in practice that EMD has advantages in representing problems involving multi-featured signatures. EMD allows for partial matches in a very natural way, and is especially fit for cognitive distance evaluation [46] and vision problems [33][63][64].

4.1.2 Web page Preprocessing and Signature Generation

We retrieve the suspected web pages and protected web pages from the web and generate their signatures. The preprocessing of web pages is a three step process: (a) Obtain the image of the web page from the given URL; (b) Perform normalization; and (c)

Represent the web page image as a web page visual signature (consists of color and coordinate features).

The process of displaying a web page in a web browser on the screen from HTML and accessory files (including pictures, Flash movies, ActiveX plug-ins, Java Applets, etc.) is the web page rendering. We use the GDI (graphic device interface) API provided by Microsoft Internet Explorer to get web page images and save them as jpeg files at original size. The images of the original sizes are transformed into images with normalized sizes (e.g. 100*100). The Lanczos algorithm [41] is used to calculate the resized image because it has very strong anti-aliasing properties in Fourier domain, and it is also easy to compute in spatial domains. Lanczos algorithm can also generate sharp images, and intuitively, sharp images can provide clearer signature for images. Figure 2 shows examples of original web pages and resized images (to 100*100 and 10*10, respectively). www.bbb.org is an example of a square-like image, www .banktechnews.com is an example of a longer image, and www.bankofcyprus.com is an example of a wider image. Each of them is normalized into fixed-size square images. We use the normalized images to present the signature of each web page. As there is no method can calculate the optimal size of the normalized image, we empirically choose a roughly optimal size based on the experiments in Section 4.2.

URL & Original Size	Original	100*100	10*10
www.bbb.org 800*796			

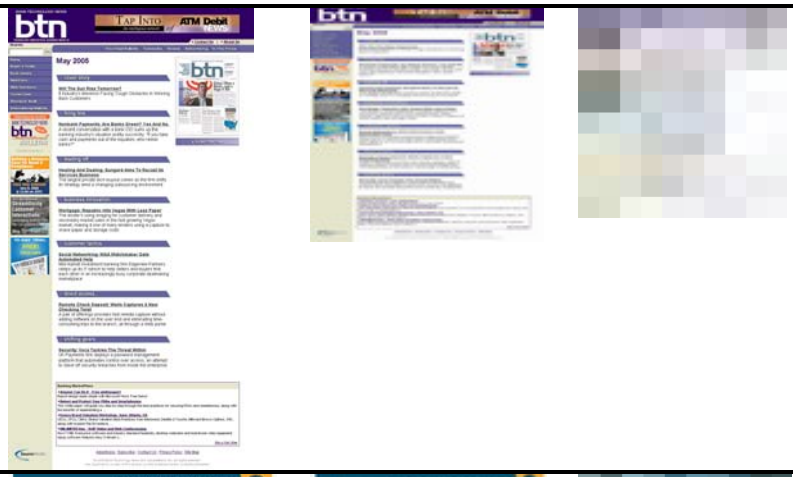
www.banktechnews.com 800*1519	
www.bankofcyprus.com 800*600	

Figure 2. Examples of web page preprocessing results

A signature of an image is a feature vector which can effectively represent the image. The signature of an image used in our approach comprises features and their corresponding weights. A feature comprises a color and the centroid of its position distribution in the image. The color of each pixel in the resized images is represented using the ARGB (alpha, red, green, and blue) scheme with 4 bytes (32 bits). A color can be represented with a 4-tuple <*A*, *R*, *G*, *B*>. However, this is a huge color space, which includes 2^{32}=4,294,967,296 colors. In practice, we use a degraded color space to represent the signature of an image. We define the Color Degrading Factor (CDF) to be the scale of each color component making a change. Thus, we have $(2^8/\text{CDF})^4$ colors in our degraded color space. A degraded color can be represented as <*A-(A mod CDF), B-(B mod CDF), C-(C mod CDF), D-(D mod CDF)*>. For example, when CDF=32, we have 4096 colors in the degraded color space. The centroid of each degraded color is calculated

using $C_{dc} = \sum_{i=0}^{N_{dc}} \frac{c_{dc,i}}{N_{dc}}$, C_{dc} is the centroid of degraded color *dc*, $c_{dc,i}$ is the coordinates of the i^{th} pixel that has degraded color *dc*, and N_{dc} is the total number of pixels that have degraded color *dc*, i.e., the frequency of *dc*. A feature F_{dc}, which has degraded color *dc* can be represented with *dc* and C_{dc}, F_{dc}=<*dc*, C_{dc}>. The weight corresponding to this feature is the color's frequency N_{dc}. A complete signature *S* is represented as $S = << F_{dc_1}, N_{dc_1} >, < F_{dc_2}, N_{dc_2} >, ..., < F_{dc_N}, N_{dc_N} >>$, where *N* is the total number of degraded colors. The feature-weight tuples in *S* are ranked in the descending order of their weights, i.e., $N_{dc_i} \geq N_{dc_{i+1}}$ for $1 \leq i \leq N-1$. In our approach, we do not use all of the features. We choose the first N_s most frequent colors in *S* to be the signature, where N_s is less or equal to *N*, and we denote it as S_s. When *N* is less than N_s, *S* is chosen to be exactly S_s.

4.1.3 Computing Visual Similarity from EMD

We use EMD to calculate the similarity of two web pages based on their signatures as follows. The distance matrix $D=[d_{ij}]$ ($1 \leq i \leq m, 1 \leq j \leq n$) is defined in advance in a straightforward way: we first calculate the normalized Euclidian distance of the degraded ARGB colors, and then calculate the normalized Euclidian distance of centroids. The two distances are added up with weights *p* and *q* respectively to form the feature distance, where *p*+*q*=1.

Suppose we have feature $\varphi_i = < dc_i, C_{dc_i} >$, where $dc_i = < dA_i, dR_i, dG_i, dB_i >$, feature $\varphi_j = < dc_j, C_{dc_j} >$, where $dc_j = < dA_j, dR_j, dG_j, dB_j >$, the maximum color distance $MD_{color} = \| < MaxA - 0, MaxR - 0, MaxG - 0, MaxB - 0 > \|$, where *MaxA*, *MaxR*,

MaxG, and *MaxB* are the maximum numbers of the four components of ARGB respectively in the specified color space, and the maximum centroid distance $MD_{centroid} = \sqrt{w^2 + h^2}$, where *w* and *h* are the width and height of the resized images respectively. The normalized color distance ND_{color} is defined as

$$ND_{color}(dc_i, dc_j) = \frac{\sqrt{(dc_i - dc_j) \times (dc_i - dc_j)^T}}{MD_{color}} \quad 8)$$

The normalized centroid distance $ND_{centroid}$ is defined as

$$ND_{centroid}(C_{dc_i}, C_{dc_j}) = \frac{\sqrt{(C_{dc_i} - C_{dc_i}) \times (C_{dc_i} - C_{dc_i})^T}}{MD_{centroid}} \quad 9)$$

The normalized feature distance between φ_i and φ_j is defined as

$$ND_{feature}(\varphi_i, \varphi_j) = p \cdot ND_{color}(dc_i, dc_j) + q \cdot ND_{centroid}(C_{dc_i}, C_{dc_j}) \quad 10)$$

So far, $\boldsymbol{D}=[d_{ij}]$, where $d_{ij} = ND_{feature}(\varphi_i, \varphi_j)$ can be calculated before performing *EMD* calculation.

Suppose we have signature $S_{s,a}$, and signature $S_{s,b}$, where $S_{s,a}$ has *m* features and $S_{s,b}$ has *n* features. The flow matrix $\boldsymbol{F}_{ab}=[f_{ij}]$ $(1 \le i \le m, 1 \ge j \le n)$ can be calculated through linear programming, and the EMD between $S_{s,a}$ and $S_{s,b}$ can be calculated as:

$$EMD(S_{s,a}, S_{s,b}, D) = \frac{\sum_{i=1}^{m}\sum_{j=1}^{n} f_{ij} \cdot d_{ij}}{\sum_{i=1}^{m}\sum_{j=1}^{n} f_{ij}} \quad 11)$$

In our experiments in Section 4.2 we observe that the linear programming within 100 features for each signature can normally be calculated in 100 iterations when the "Epsilon" for indicating optimization is set as 1e-6, which is sufficiently precise to

calculate the EMD of two signatures. Since $d_{ij} = ND_{feature}(\varphi_i, \varphi_j) \in [0,1]$, $EMD(S_{s,a}, S_{s,b}, D) \in [0,1]$. If $EMD(S_{s,a}, S_{s,b}, D) = 0$, the two images are identical, and if $EMD(S_{s,a}, S_{s,b}, D) = 1$, the two images are completely different. We define EMD based visual similarity of two images as:

$$VS(S_{s,a}, S_{s,b}) = 1 - [EMD(S_{s,a}, S_{s,b}, D)]^{\alpha} \quad 12)$$

where $\alpha \in (0,+\infty)$ is the amplifier of visual similarity. We use α to make visual similarity to be better distributed between $(0,1)$ rather than too dense at either side without affecting the ranking relationship of the visual similarity values of web pages.

4.1.4 Classification

We use a special threshold for each given protected web page to classify a web page as a phishing web page or a normal one. Suppose we have a set of protected web pages; We have to calculate the threshold vector for them first. When a suspected web page comes up, we calculate its visual similarity to all of the protected ones and determine if it is phishing any of them. The threshold vector is represented as:

$$T = < T_1, T_2, ..., T_{N_{protected}} >,$$

where $T_i (1 \le i \le N_{protected})$ denotes the threshold of the i^{th} protected web page. $T_i (1 \le i \le N_{protected})$ is defined as:

$$T_i = \underset{t \in VSS_i}{argmin}(MissClassification(t)) - \delta \quad 13)$$

where VSS_i is the historic visual similarity set of (i.e., contains the similarity values of the historically tested web pages to) the i^{th} protected web page, and $MissClassification(t)$ is the number of mis-classified web pages in case we use t as the threshold. There are two

types of mis-classifications: (a) False alarm, the visual similarity is larger than or equal to *t* but in fact the web page is not a phishing web page (false positive); (b) Missing, the visual similarity is less than *t* but in fact the web page is a phishing one (false negative). *t* should be selected to be as small as possible without increasing the mis-classified number. We use dynamic programming to calculate *t*. Each phishing detection record in VSS_i correlates to two accessory parameters, the false alarm number (*fa*) and false negative (*fn*). They denote, when we use the EMD of a training record as threshold, how many false alarms and false negatives it will yield. We simply choose *t* to minimize *fa*+*fn*. If more than one *t* achieves the same minimal *fa*+*fn*, we choose the smaller one. When a new training record arrives, we need to set new *fa* and *fn* for this record, adjust all other accessory parameters in VSS_i , and choose the EMD of the record that can minimize *fa*+*fn* again. Obviously, the new *t* can easily be found, since it is near the original one. δ is a slack constant, and we use it to decrease missing cases by increasing false alarms because false negative is much more harmful than false alarms in the anti-phishing systems. δ can be chosen empirically from 0 to 0.1 based on our observation from a large scale experiments and more detail is discussed in Section 4.2.

When a suspected web page comes up, we calculate the visual similarity vector, which can be represented as

$$VS = < vs_1, vs_2, ..., vs_{N_{protected}} >,$$

where $vs_i (1 \le i \le N_{protected})$ denotes the visual similarity of this web page to the i^{th} protected web page. We calculate the classification result using the following equation.

$$IsPhishing(VS) = \begin{cases} 1 & if\ \ max(VS - T) \ge 0 \\ 0 & if\ \ max(VS - T) < 0 \end{cases} \qquad 14)$$

where $max(VS - T)$ denotes the maximum factor in $VS - T$. *IsPhishing*(*VS*)=1 indicates that the web page is a phishing web page, while *IsPhishing*(*VS*)=0 indicates that the web page is normal.

4.2 Experiments and Performance Evaluation

We carried out a large scale experiment to evaluate the performance of our EMD based phishing detection approach. We used 10,272 homepage URLs retrieved from Google with 26 keyword queries: bank, biology, car, Chinese, company, computer, English, entertainment, government, health, Hong Kong, house, Linux, money, movie, network, phishing, regional, research, science, spam, sport, television, university, web, and Windows. We use each word to collect up to 1,000 homepage URLs. Duplicated URLs are removed to keep each URL unique. In addition to these web pages, we have 9 phishing web pages collected from real phishing attack cases. The 10,281 (10,272+9) web pages form the Suspected Web Page Set in our experiments. The Protected Web Page Set contains 8 real web pages attacked by the 9 phishing web pages. (These datasets are available in our website [30])

We empirically set the parameters w=h=100, α=0.5, $|S_s|$=20, p=q=0.5, and CDF=32 in our experiments. This configuration provides satisfactory recognition of phishing web pages and acceptable computation time at the same time, as we can see from the experiments (see Section 4.2.3 for details).

We first demonstrate, in Section 4.2.1, that the performance of our method using dynamically supervised training starting with a zero threshold vector (the similarity thresholds for all protected web page are 0). Then we demonstrate, in Section 4.2.2, the

performance of our method with a trained threshold vector (or when the trained threshold vector is given). Finally, we discuss the parameter tuning process in Section 4.2.3.

4.2.1 Experiment Result with Dynamically Supervised Training

Assume our anti-phishing system starts without any pre-training, i.e. all elements in the threshold vector are set to 0 initially. It is trained while it is running. When a suspected web page comes up, the threshold vector is dynamically adjusted using the threshold calculation method addressed in Section 4.1.4. The suspected web pages in the Suspected Web Page Set are randomly ordered and sent to the system one by one. We record the classification results in each step. Table 1 shows the statistical results of the experiment after we tested all 10,281 suspected web pages. Most of the phishing web pages (7 out of 9) are detected when $\delta = 0$, and almost all phishing web pages (8 out of 9) can be detected when $\delta = 0.005$. The last phishing web page is difficult to detect because it is not visually similar to the real one at all, as shown in Figure 3 (which also shows that the most similar web page of Real ICBC Asia under this measure among the 10,281 suspected web pages is www.frlp.utn.edu.ar). Since our phishing detection method is based on visual similarity, it cannot detect those phishing web pages that look different from the real one. However, from our experiences and investigation, almost all phishers have tried their best to make the appearance of their phishing web pages visually similar to the real ones, presumably because they do not want to decrease their potential victims' response rate.

To demonstrate the advantage of our method, we compare it with two additional visual methods for phishing detection: (a) HTML/DOM based EMD and (b) Region Matching (the method addressed in [48]).

Method (a) segments the given web page into blocks. Each block is represented with a DOM based signature (size, position, foreground color and background color etc.). The web page is represented with a DOM based signature, too, by combining the block level signatures, which is then used to calculate the EMD based similarity. Table 2 shows the statistical results of this experiment.

Method (b) measures visual similarity of web pages in terms of similar key regions/blocks, similar page layout, and similar styles (e.g., font family, size, decoration, and even spacing). In this method, a web page is finally reported as a phishing suspect for human confirmation if the visual similarity is higher than its corresponding preset threshold. In [48], the threshold is fixed for all protected web. We carried out the same scale of experiment, using the 10,281 suspected web pages, and show its performance statistics by varying the threshold, as shown in Table 3.

In these three experiments, the Phishing ICBC Asia web page is always the most difficult one to be detected. Considering only the other 8 reasonable phishing web pages, which are visually similar to their real ones, our proposed method's performance is superior to both methods (a) and (b). While our method can recognize these 8 visually similar phishing web pages at $\delta = 0.005$ by giving 65 (73-8) false alarms, method (a) can do this in the best case at $\delta = 0.03$ but results in 849 (857-8) false alarms, and method (b) can do this at threshold=0.76 but produces 697 (705-8) false alarms. In addition, recall that the parameter that needs to be adjusted in our method and method (a) is δ, however the parameter that needs to be adjusted in method (b) is the threshold. Hence we use different parameters to evaluate the performances of the three methods.

Table 1. Phishing detection performance of our proposed method (using the image based EMD)

δ	Total Report Number	Correct Number
0	42	7
0.005	73	8
0.01	134	8
0.015	229	8
0.02	403	9
0.025	693	9

Table 2. Phishing detection performance of method (a) (using the HTML/DOM based EMD)

δ	Total Report Number	Correct Number
0	61	7
0.005	99	7
0.01	153	7
0.015	257	7
0.02	393	7
0.025	572	7
0.03	857	8
0.035	1244	9
0.04	1925	9

Table 3. Phishing detection performance of method (b) (using the method in [48])

Threshold	Total Report Number	Correct Number
0.985	2	2
0.97	2	2
0.955	2	2
0.94	3	3
0.925	3	3
0.91	3	3
0.895	3	3
0.88	3	3
0.865	3	3
0.85	7	3
0.835	13	3
0.82	30	4
0.805	71	5
0.79	170	6
0.775	376	6
0.76	705	8
0.745	1230	8
0.73	1974	8
0.715	3029	9
0.7	4617	9

4.2.2 Experiment Result with Trained Threshold Vector

In this experiment, we train the threshold vector using a training dataset. From the 10,272 randomly collected web pages, we select 1,000 web pages, and combine them with the 9 phishing web pages together to form the training dataset (containing 1009 web pages). We use the training dataset to calculate the thresholds for the 8 protected web pages. The training result is listed in Table 4. We use the other 9,272 (10,272-1,000) web pages, and combine them with the 9 phishing web pages to form the Suspected Web page Set. Table 5 shows the classification precision, phishing recalls, and false alarms of this Suspected Web page Set using these thresholds. There is one missing case in this experiment because the real and phishing web page of ICBC(Asia) are not visually similar to each other. Hence, the reasonable upper bound of classification precision and phishing recall are 99.87% and 88.88% respectively.

In practical applications we are expecting to achieve better recall even though the precision and false alarm rates could be sacrificed a little (we treat the missing problem as more severe than the false alarm problem in anti-phishing). We can look forward to report more phishing web pages to achieve better recall rate. We use a slack constant δ to detect more potential phishing web pages. Table 6 shows the classification precision, phishing recall, and false alarm values when we set $\delta = 0.005$. Figure 4 shows the web page of "Real-Bank of Oklahoma" and one of its phishing web pages detected.

Figure 5 shows four false alarm examples for "Real-Bank of Oklahoma" when $\delta = 0.005$. The first three false alarm web pages look indeed very similar to the web page of "Real-Bank of Oklahoma". Hence, it is reasonable and worthwhile to classify them as phishing web pages for further examination. The fourth one does not look similar to human but it

is also classified as a phishing under $\delta = 0.005$. However, we consider it as a necessary sacrifice to reduce false negative possibilities.

Table 4. Thresholds for Protected Web page Set Trained with the 1000+9=1009 Training Web pages

Protected Web page	Threshold
real-Bank of Oklahoma - Online	0.8469
real-eBay1	0.9434
real-eBay2	0.9493
real-ICBC(Asia)	0.7385
real-Key Bank	0.9323
real-US Bank	0.9573
real-Washington Mutual	0.8541
real-Wells Fargo Sign On	0.9255

Table 5. Classification Precision, Phishing Recall, and False Alarm List (evaluated with 10272-1000+9=9281 suspected web pages, $\delta = 0$)

Protected Web page	Classification Precision	Phishing Recall	False Alarm
real-Bank of Oklahoma	9280/9281	1/1	1
real-eBay1	9280/9281	2/2	1
real-eBay2	9280/9281	1/1	1
real-ICBC(Asia)	9276/9281	0/1	5
real-Key Bank	9280/9281	1/1	1
real-US Bank	9280/9281	1/1	1
real-Washington Mutual	9280/9281	1/1	1
real-Wells Fargo	9280/9281	1/1	1
Overall	99.87%	88.88%	12

Table 6. Classification Precision, Phishing Recall, and False Alarm List (evaluated with 10272-1000+9=9281 suspected web pages, $\delta = 0.005$)

Protected Web page	Classification Precision	Phishing Recall	False Alarm
real-Bank of Oklahoma	9277/9281	1/1	4
real-eBay1	9278/9281	2/2	3
real-eBay2	9278/9281	1/1	2
real-ICBC(Asia)	9275/9281	0/1	5
real-KeyBank	9279/9281	1/1	2
real-US Bank	9275/9281	1/1	6
real-Washington Mutual	9279/9281	1/1	2
real-Wells Fargo	9277/9281	1/1	4
Overall	99.70%	88.88%	28

Figure 3. Visual Comparison of Real and Phishing ICBC (Asia) Web pages

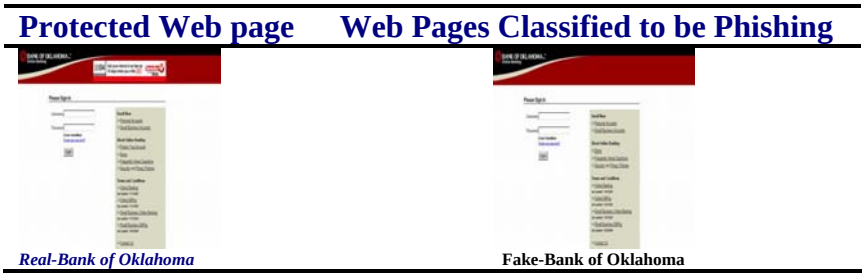


Figure 4. Correct Detection Result for the web page of Real-Bank of Oklahoma

Figure 5. Examples of wrong classification for Real-Bank of Oklahoma

4.2.3 Impact of the Parameters and Parameter Tuning

The results in 4.2.1 and 4.2.2 are based on the best parameters we obtained from empirical evaluation. These experiments also show that the system performance is quite good using these parameters. We found that the global optimal parameters are quite hard to find through experiment because it needs a huge amount of calculation, and we are also not expecting to improve the performance a lot through calculating the global optimal parameters. We evaluated whether our empirical parameter settings can achieve roughly local optimal performance. We did experiments to tune the parameters (w, h,$|S_s|$,

p, *q*, and CDF) to see if we could find better parameter settings, which also assessed the value of these parameter settings.

The tuning approach comprises 4 parts: *w*, *h*; $|S_s|$; *p*, *q*; and CDF. As α is only an amplifier and does not affect web page similarity ranking and classification, we do not need to tune α. We simply set α=0.5. Each tuning is based on the start point of the default parameter setting: *w*=*h*=100, $|S_s|$=20, *p*=*q*=0.5, and CDF=32. When we tune one part, the other parts are fixed at those values. We use human vision to determine whether two web pages are similar when building the *ground truth* dataset. We first find all similar web pages for each protected web page manually. The similar web pages we manually identified for each protected web page form a ground truth group dataset for benchmarking. There are a total of 255 web pages determined in these 8 groups. Table 7 shows the number of ground-truth similar web pages for each protected web page. We take $N_{sample} \in \{5,10,...50\}$ as the sample number for each protected web page. From the 10,281 suspected web pages, the N_{sample} most similar web pages, evaluated by our method, to each protected web page are automatically checked for correctness according to our ground truth datasets. If a web page in the N_{sample} collected web pages is in the corresponding ground truth group, we count it as a correctly detected similar web page. We count the total number of correctly detected web pages for all of the 8 protected web pages and use it as the vertical axes of Figure 6, Figure 7, Figure 8, and Figure 9. In these figures, each line represents a different N_{sample} value and the horizontal axes show the variations of parameters we want to tune.

Table 7. Ground Truth Sample Number for each Protected Web page

Protected Web page	Sample Number
real-Bank of Oklahoma - Online	85
real-eBay1	8
real-eBay2	19
real-ICBC(Asia)	30
real-Key Bank	14
real-US Bank	47
real-Washington Mutual	42
real-Wells Fargo Sign On	10
Total	255

4.2.3.1 Tuning *w* and *h*

We have 4 configuration options ($w = h = 10,\ 10\sqrt{10},\ 100,\ and\ 100\sqrt{100}$) to tune *w* and *h*. As shown in Figure 6, the ranking performance of our approach reaches a peak at $w \times h = 100 \times 100$. Although the CUP time is proportional to $w \times h$, experiment shows that one pair of visual similarity can be computed in less than 0.1 seconds and $w \times h = 100 \times 100$ is a reasonable choice. Hence it is reasonable to determine $w = h = 100$. However, it does not mean $w \times h = 100 \times 100$ is optimal. $w = h = 100$ only indicates the roughly local optimal configuration of *w* and *h*. The same also applies to Figure 7, Figure 8, and Figure 9.

4.2.3.2 Tuning $|S_s|$

We use 6 configuration options ($|S_s|$=5, 10, 15, 20, 25, and 30) to tune $|S_s|$. As shown in Figure 7, the ranking performance of our approach reaches a peak at $|S_s|$=20. Although it does not lose much performance when $|S_s|$ is larger than 20, the time performance is worse since when we use a bigger color number to compute visual similarity, more CPU

time is consumed. $|S_s|$ should be taken as low as possible if the classification performance can be guaranteed. Thus, $|S_s|$=20 is a reasonable choice.

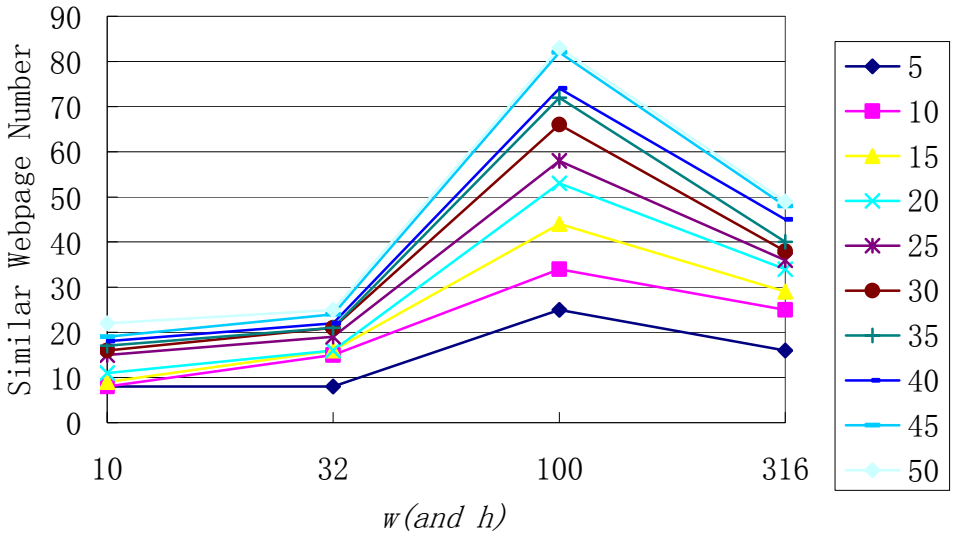


Figure 6. Performance Variation with *w* and *h*.

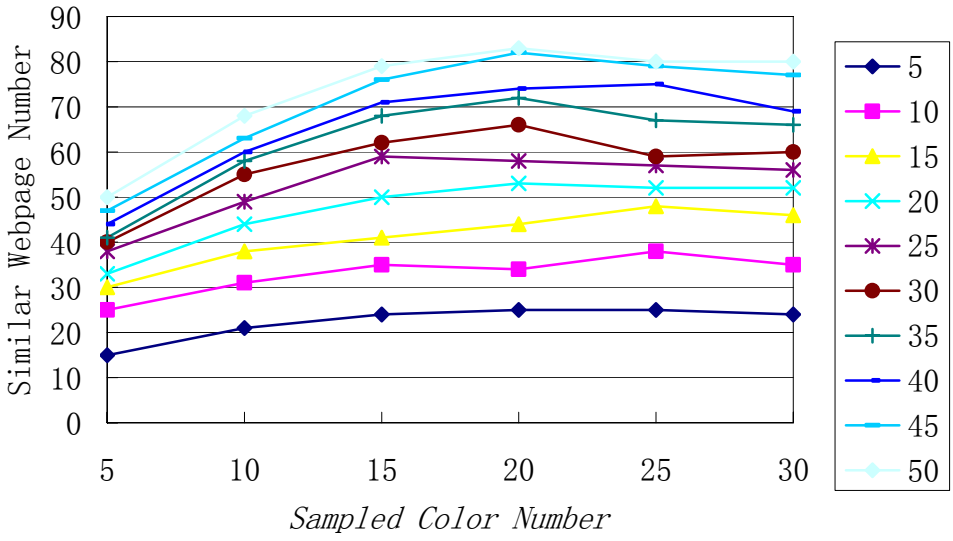


Figure 7. Performance Variation with $|S_s|$.

4.2.3.3 Tuning *p* and *q*

We use 11 configuration options ($p:q=0:1,0.1:0.9,0.2:0.8,...,0.9:0.1,1:0$) to tune *p* and *q*. As shown in Figure 8, the ranking performance of our approach does not vary much with *p* and *q*. We think the reason is that color and color distribution in an image have similar impact on human eye perception. When *p* ranges from 0.3 to 0.7, the ranking performance is relatively high. It is reasonable if we use *p*=*q*=0.5. As we can see, the proportion of *p* and *q* does not affect the performance much. Hence, we also suggest simply using *p=1 so that* the calculation of $ND_{centroid}$ can be avoided and the computing time can be saved.

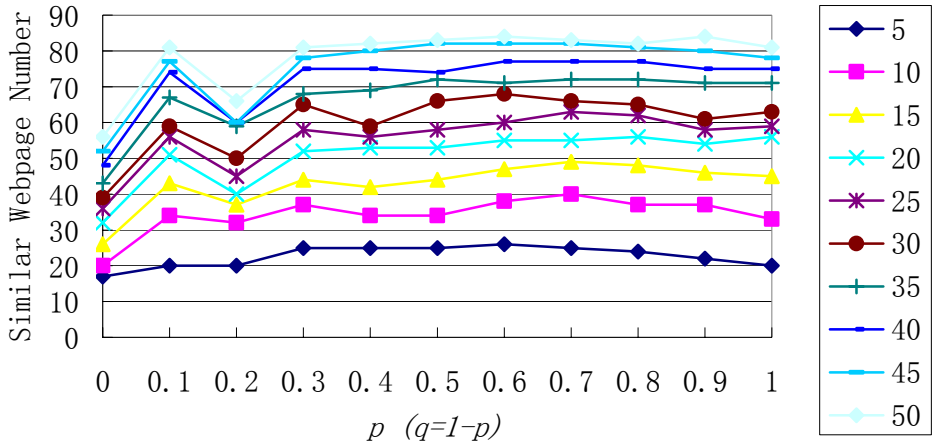


Figure 8. Performance Variation with *p* and *q*.

4.2.3.4 Tuning CDF

We use 8 configuration options (CDF=8, 16, 24, 32, 40, 48, 56, and 64) to tune CDF. As shown in Figure 9, we can set CDF=32 to obtain the best performance.

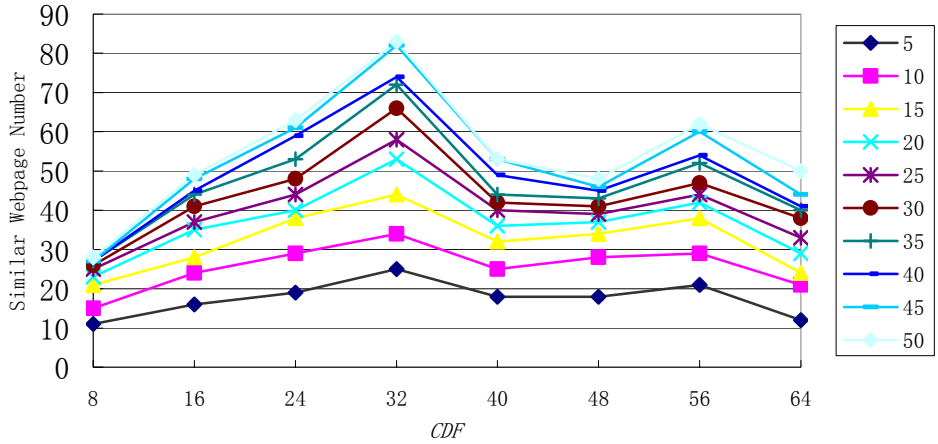


Figure 9. Performance Variation with CDF

Finally, we decide to use the following parameter configuration: w=h=100, α =0.5, $|S_s|$=20, p=q=0.5, and CDF=32. In the aspect of time efficiency, each EMD based visual similarity of two web page can be calculated in 0.02 second using an ordinary PC (with a single Intel Xeon 3.0G CPU, 4 Hyper Threads, and 512M RAM) with this parameter configuration, which is sufficiently fast for online phishing detection.

4.3 The Anti-Phishing System

We built an anti-phishing system using the proposed approach. The system can automatically detect potential phishing web pages by comparing their similarities to the protected web pages. To our knowledge, almost all phishings start by sending phishing-emails to internet users. Spoofing contents are written in this kind of emails to make users eager to access their website and fill in personal information. We built the anti-phishing engine into the Anti-Phishing Proxy which filters all traffics going through the email server. We also built and deployed an Anti-Phishing Server, which acts like anti-virus database servers. The Anti-Phishing Proxy keeps the phishing definitions updated from

the Anti-Phishing Database Server. Figure 10 shows the architecture of our system. The Anti-Phishing Database Server is the center for registration of legitimate websites which want protection, and maintains a phishing link list. It also acts logically as a part of the Anti-Phishing Proxy to preprocess protected web pages, such that this design makes good system scalability.

The registered legitimate web pages are preprocessed in advance, and their signatures are saved in the database. The owner of the protected web page can specify a set of sensitive keywords (e.g., “eBay” for www.ebay.com) during registration to the Anti-Phishing Database Server, which are used for checking the emails. Anti-Phishing Proxy synchronizes the database from the Anti-Phishing Database Server. If an email contains any of these sensitive words, the URLs in this email are parsed out. URLs that are in the protected list are deleted and the URLs that can be found from the phishing link list are reported immediately. The rest URLs are suspected ones, and are compared to the protected web page associated with the sensitive word(s) using the proposed approach. If the visual similarity is larger than the threshold associated with the protected web page, the suspected web page is classified as phishing and an alert is reported. The system administrator on behalf of the Anti-Phishing Database Server can read the alert and make a final decision on whether the report is correct or not. The feedback from the administrator can help make more accurate classification later. By doing so, a lot of human efforts can be saved for phishing monitoring and detection. Many phishing cases have been caught by our system and listed in [5].

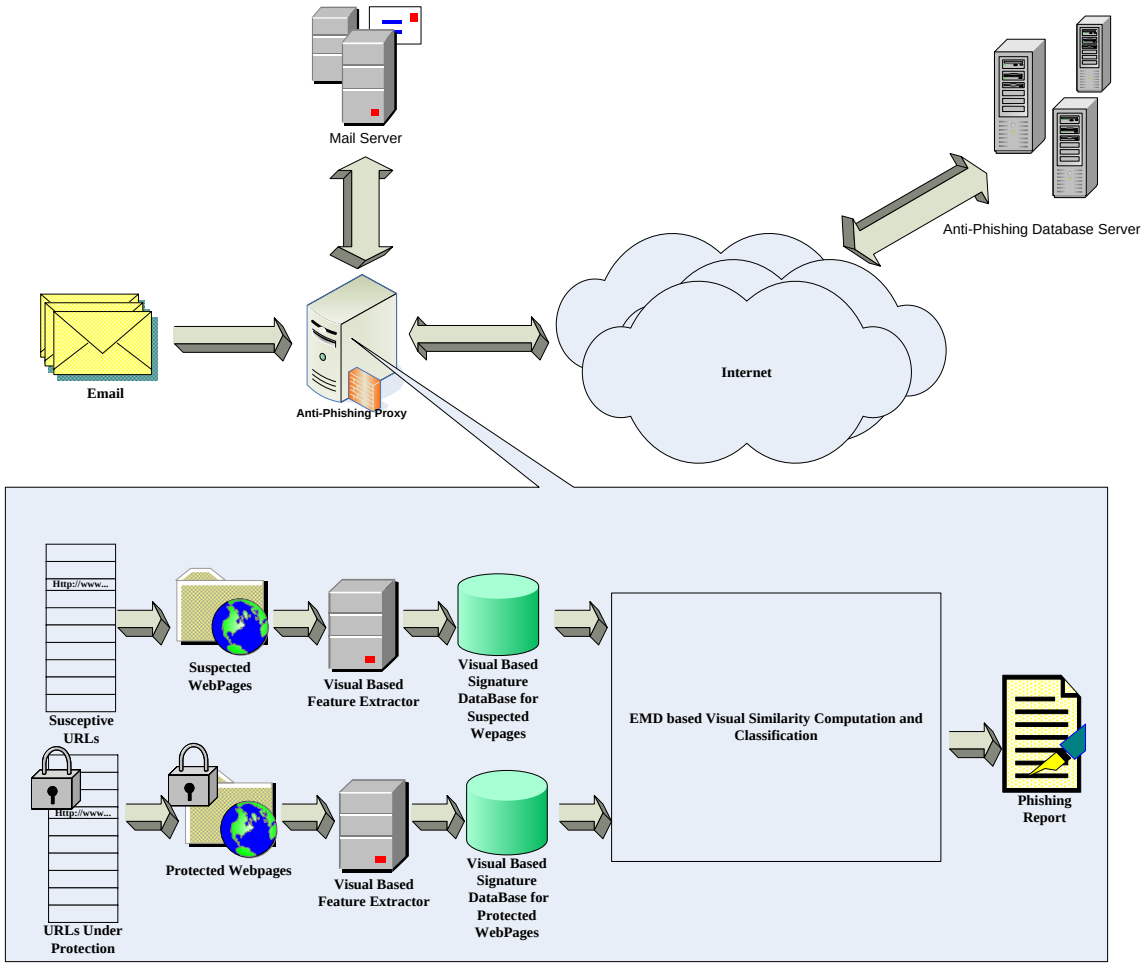


Figure 10. Architecture of the Anti-Phishing System

We also have a client side system, SiteWatcher. It also uses the similar technology and detail is available at [5]. SiteWatcher will be commercialized through the cooperation of Anti-Phishing Group of CityU and Reasonable Software [59].

4.4 Conclusion and Future Work

Phishing has become a severe problem of internet security. We propose a phishing web page detection method using the EMD based visual similarity assessment. This approach works at the pixel level of web pages rather than at the text level, and can thus detect

phishing web pages only if they are “visually similar” to the protected ones without considering the similarity of the source codes. Our experiments also show that our method can achieve satisfying classification precision and phishing recall, and the time efficiency of computation is acceptable for online use. An anti-phishing system is built up on a mail server, which could be a prototype for industrial application.

However, our method is a visual method and assumes the phishing web pages are visually similar to their attacking targets. The method could not detect those which are not visually similar. In order to address this problem, we will consider using textual features together with the visual features in future work. There are also other potentially promising works to do for further exploration. The approach proposed here is an important component in the anti-phishing system. However other techniques could also be explored to further improve the detection accuracy, e.g., OCR (like the research works of Ashook in [7]), web page semantic structure analysis, and natural language level semantic analysis.

Although most phishers will make the phishing web page looks as similar as possible to the real one. Our proposed system assesses the complete web page rather than assesses a part of it. If a phisher makes a partially similar web page to the real one, our system may fail. Our system can be further improved in this aspect.

The use of the method proposed in this approach may not just be limited to the server sides. We are also working on developing a client side application, SiteWatcher Client, which can be installed by individual users for phishing detections (a trial version can be download from [5]). SiteWatcher Client works in a similar way to anti-virus applications. It can periodically update the phishing database from SiteWatcher Server. It will also

report new discovered phishing links to SiteWatcher Server. SiteWatcher will consider adding reported phishing links into the phishing database. SiteWatcher Client can monitor the TIC/IP packets a PC received and issue alerts when phishing URLs are detected.

Chapter 5

Semantic Assessment Approach

The semantic assessment approach for anti-phishing focuses on text level attacks. We propose a methodology for Unicode attack detection. This methodology provides effective guidance for building up various phishing counter measures. We propose three methods to calculate char-char similarity list and one method to calculate word-word similarity list. We use string similarity algorithms to evaluate the similarity of given Unicode strings. Following the methodology, we carry out experiments under a paradigm implementation.

We built a Unicode Character Similarity List (UC-SimList), by which we can easily retrieve the visual and semantic similarity of any given pair of characters in UCS. We also built two effective algorithms and implemented into an anti-phishing tool, IRI/IDN SecuChecker, to detect similar/fake IRI/IDN. We demonstrate several possible usages to domain name registrars, user name (account) registrars, and web browser based phishing detection add-ins. As a part of the solution for detecting word level phishing attacks, we construct WWSS List (word-word semantic similarity list). We also propose a non-deterministic automaton based model to detect phishing IRI/IDNs and built a tool, REGAP, to automate the regular expression generation for phishing attack detection.

5.1 Unicode Attack

With the population of the Internet, people from various countries, regions, and cultures join into the information exchange evolution. One biggest cornerstone of making these characters and symbols display properly is the utilization of Unicode. UCS (Universal Character Set) is a character repertoire (with character codes) of Unicode. The most popular version of Unicode uses 16 bits to represent on character code. However, we demonstrated in [27] that a lot of visually similar characters coexist in UCS. Figure 11 shows the samples of similar characters from UCS. “b” (U+0062), “a” (U+0061), “n” (U+006E), and “k” (U+006B) are the characters used in English, as shown in the first column of Figure 11, while the characters in the rest columns are visually similar characters.

b	b	ḅ	ḇ	ƀ
0062	FF42	1E05	1E07	00FE
a	a	a	ạ	ậ
0061	0430	FF41	1EA1	1E01
n	n	ṇ	n	ṇ
006E	FF4E	0146	043F	1E47
k	k	ƙ	ḳ	ḵ
006B	FF4B	0199	1E33	1E35

Figure 11. Similar characters to “b”, “a”, “n”, and “k” (in Arial Unicode MS Font). Code under each character is the character code in hexadecimal form.

The coexistence of similar characters in UCS could enable a series of Web security problems, such as Spamming, Phishing, and Web Identity Faking.

5.1.1 Spam Attack

Spammers can create numerous spam emails while keeping the same visual of original email as shown in Figure 12.

A	n		a	p	p	l	e		a		d	a	y	,	k	e	e	p
41	6E	20	61	70	70	6C	435	20	61	20	64	61	79	2C	6B	65	65	70
s		t	h	e		d	o	c	t	o	r		a	w	a	y	!	
73	20	74	68	435	20	64	6F	63	74	6F	72	20	61	77	61	79	21	

(1) Sample 1

A	n		a	p	p	l	e		a		d	a	y	,	k	e	e	p
410	FF4E	20	430	70	70	49	435	20	FF41	20	64	430	FF59	2C	FF4B	435	FF45	70
s		t	h	e		d	o	c	t	o	r		a	w	a	y	!	
455	20	FF54	68	64	20	217E	6F	63	FF54	FF4F	72	20	61	77	61	443	1C3	

(2) Sample 2

Figure 12. Sample of Unicode attack based spam. Sample 1 is the original message, while Sample 2 is the mutated (visually identical) one. Code under each character is the character code in hexadecimal.

Most phishing attacks attract users' attention by sending phishing emails. Phishing emails are also spams, so anti-spam techniques can help block phishing attacks. However, most of the current anti-spam techniques use machine learning techniques as their solutions. However, the usage of Unicode attack could simply go around such anti-spam filters' detection, because a simple Unicode string can generate a huge set of that are visually or semantically similar versions. As we demonstrated in Section 2.3, "citibank" has 24(c) * 58(i) * 21(t) * 58(i) * 24(b) * 22(a) * 21(n) * 14(k) -1=263,189,025,791 potential mutations. Hence there is a severe threaten of spamming attacks that may be caused by Unicode attacks.

5.1.2 Phishing Attack

Phishers can replace visually similar characters into the Internationalized Resource Identifiers (IRI) [21] to mimic the real ones, as shown in Figure 13.

w	w	w	.	e	b	a	y	.	c	o	m
77	77	77	002E	65	62	61	79	002E	63	006F	006D
w	w	w	·	e	b	a	y	.	c	o	m
77	FF57	FF57	2027	FF45	62	FF41	443	002E	441	03BF	006D
w	w	w	·	e	b	a	y	·	c	o	m
FF57	FF57	77	FF65	FF45	FF42	61	79	00B7	63	03BF	217F

Figure 13. Samples of Phishing. The first weblink is the original/real one and the rest two weblinks are mutated/faked ones. Code under each character is the character code in hexadecimal form.

Another possible attack is that the original website/web page text context could be replaced by similar characters, such that no Anti-Phishing systems (i.e., SiteWatcher [5] [29] [48] [75]) can catch it, because these anti-phishing systems must detect at least one sensitive word (e.g. HSBC) in the emails or web pages before suspecting them, otherwise the computational resource may cause too much overhead.

5.1.3 Web Identity Attack

It is usual that the Web based systems (e.g. Website, Email, Chatting Room, BBS, Instant Message, Blog, Wiki) use text string to represent the user name. There is not many problems if simply use ASCII [5] characters. However, if Unicode strings are used to represent user accounts, the system will be fragile to Unicode attack, especially when user account is the only way for users to identify each other. Therefore, malicious people can easily gain their potencial victims' trust.

5.1.4 Homograph Attack and Unicode Attack

We could see there are many types of *Unicode attacks* and all of them are utilizing the fact of the similar characters' coexistence in the UCS. The possibility of using similar characters to generate fake domain name using national alphabets is first reported by Gabrilovich et al in [31], which is named *Homograph Attack*. This concept is general. It

covers all specific attacks of replacing similar characters in a text string. *Unicode Attack* is the largest subset of homograph attack. It focuses on the survey of UCS which is the most populated character set for internationalization of character sets.

The basic idea of carrying out *Unicode attack* is to generate the mutations of the original Unicode string (such as spam content, domain name, and user account) by replacing similar characters. The basic idea of safeguarding the (Web) systems from *Unicode attack* is to evaluate the similarity of the suspected string(s) to the original one(s). A generic methodology for the counter measures against *Unicode attack* is reported in Section 5.2 and [26], where the construction of UC-SimList and WWSS List are considered as the most critical parts of the Unicode string similarity evaluations.

5.2 Methodology of Unicode Attack Detection

We propose a general methodology to assess the similarity of Unicode strings. We organize the methodology from several levels view of string: (1) *char-char similarity*, which includes visual similarity and semantic similarity between two characters; (2) *word-word similarity*, which only contains semantic similarity; and (3) *string similarity*, it is the highest level, and the evaluation is based on the previous two levels.

5.2.1 Preprocessing

Spammers usually add noisy characters into the original Unicode strings to avoid detections, so we need string preprocessing to reduce or eliminate them. UCS contains many symbol characters (e.g., '|' and '\' in string "y0U|HaVE/A |FrEe \G|fT ++"). Such noisy characters make it more difficult to detect similar/fake Unicode strings. Hence, the preprocessing should be able to replace the symbol characters with empty strings, or

meaning for characters (depending on the string similarity evaluation requirements). The symbol character list can be constructed by referencing the specification of Unicode [67] manually. Unicode string preprocessing should be quite useful for phishing IRI/IDN detection, especially for detecting spam emails, erotic content, and dirty words, because malicious people are always attempting to avoid the detections for any information filters. However, preprocessing for general Unicode text strings is not simple. First of all, we do not have a complete list of symbol characters. The UCS is a big, complicated and growing list. Also, we cannot conclude that all symbols are noise; for instance, "|" can be used by malicious people to replace "I" in the word "GIfT" (changing it to "G|fT"). Therefore, there is a use for future work in Unicode preprocessing alone.

5.2.2 Character Level Similarity

The basic trick of a Unicode attack is to replace some characters with similar ones. We address character similarity in two dimensions: visual similarity and semantic similarity.

5.2.2.1 Visual Similarity List (UC-SimList_v)

A visual similarity list can be constructed automatically by comparing the glyphs in UCS. If necessary, we can optimize it manually. However, the ideal way is to have an algorithm to construct the list in a completely automatic way without additional manual work, since UCS is a huge database and the manual work could be overwhelming.

5.2.2.2 Semantic Similarity List (UC-SimList_s)

A semantic similarity list can only be constructed manually by referencing the specification of the Unicode repertoire because we cannot find an algorithm or a tool with

the necessary knowledge to do it. We still do not have a complete semantic similarity list, and we consider it as a future work.

5.2.2.3 Char-Char Similarity List (UC-SimList)

The overall char-char similarity list can be constructed by combining the visual and semantic similarity lists. We consider multiplication as a good combination method, which means if the visual similarity of "ạ" (1EA1) and "a" (0061) is 0.9 and the semantic similarity of "a" (0061) and "A" (0081) is 1, we can calculate the overall similarity between "ạ" (1EA1) and "A" (0041) as 0.9×1=0.9. We also use this method in Section 5.3 to calculate the character level similarity. Other combination methods may also be possible.

The char-char similarity list is a good resource for assessing the similarity of Unicode stings, recovering an original string from its noisy versions, and maybe other relevant tasks. For instance, we can do noise reduction to the example string in Section 5.2.1 and retrieve the intended message: "you have a free gift". We can then use the denoised string to perform word level similarity assessments with any candidate strings as addressed below in Section 5.2.3.

5.2.3 Word Level Similarity

Unicode attacks can be carried out by replacing words with other semantically similar ones (e.g. synonyms). The following four types of semantic substitutions are most likely to occur in the near future:

5.2.3.1 Phonetic Substitution:

Phishers may change some words of the string but still keep the original pronunciation, e.g., “BankForYou” can be changed to “Bank4U”, “中国银行” to “ZhongGuoYinHang”, “日本銀行” to “にほんぎんこう” or “NiHonGinKou”.

5.2.3.2 Acronym Substitution:

A malicious person may use the acronyms of the keywords of the original Unicode strings to carry out attacks, e.g., “BankOfChina” to “BOC”, “中国银行” (Bank of China) to “中银”, and “とうきょうだいがく” (the University of Tokyo) to “とうだい”, etc.

5.2.3.3 Language based Substitution:

A malicious person may translate some words in the original Unicode string to another language to carry out attacks. For example, “BankOfChina” to “中国银行” or “中国バンク”

5.2.3.4 Synonym Substitution:

The words in a Unicode string could be replaced with their synonyms, e.g., “this is spam” to “this is junk mail”, or “Hi, buddy” to “Hello, friend”, etc.

These four types of word level obfuscations could be used together in many ways to make a single string even more complicated and difficult to detect, while still allowing humans to understand them.

The solution to detect word level obfuscations is to establish a word-word semantic similarity list to assess the similarity of strings. However, the list could be very complicated and large. We have constructed one based on word-word semantic similarity assessment algorithms. However it is for English only. It turns out that we need to use excellent word-word similarity algorithms and a lot of human knowledge. It is also a growing list since new words are invented quickly. The list data structure should be well constructed because it is quite large.

5.2.4 String Similarity Algorithms

We propose the methodology of using char-char similarity, as addressed in Section 5.2.2, and word-word similarity, as addressed in Section 5.2.3, to calculate the similarity of Unicode strings. We are expecting use algorithms that can be applied to Unicode string similarity assessment. These algorithms should be easy to apply UC-SimList and WWSS List as character level and word level similarity assessments.

Intuitively many standard string similarity evaluation algorithms from information retrieval (IR), natural language processing (NLP), and bioinformatics string processing can be applied, such as Edit Distance [45], KMP[43], Needleman-Wunch Distance [56], and n-gram etc. Many of them are based on character level similarity. Hence, we can apply UC-SimList directly to them. There are also many semantic similarity evaluations for strings in computational linguistics research area. We can apply WWSS List to such algorithms. One other straight forward usage of WWSS List is to consider words as characters, and use character level similarity algorithms to calculate string similarity. Time complexity should be well considered when we choose specific algorithms to calculate the string similarity. As a paradigm implementation to our methodology, we

implement an application based on implementing VSED (Visual and Semantic based Edit Distance) and VSKMP (Visual and Semantic based KMP Algorithm) on character level as an example in Section 5.5. An approach of using NFA (Nondeterministic Finite Automata) is also introduced in Section 5.7, where both the character level and word level similarity are considered as well.

5.3 UC-SimList Generation

Since online calculation of character similarity is expensive, a lookup table (char-char similarity list) can be pre-constructed offline, which consists of a list of similar characters for each individual character in Unicode. We refer to the lookup table as UC-SimList. The visual similarity is calculated through calculating the similarity of each pair of characters.

5.3.1 General Approach

UC-SimList is a list, from which we can easily find the similarity of a given pair of characters. The construction of UC-SimList requires two sub-lists, UC-SimList_v and UC-SimList_s.

5.3.1.1 UC-SimList_v (Visual Simialrity)

UC-SimList_v is a list, from which we can easily find the visual similarity of a given pair of characters, such as the visual similarity of "a" (U+0061) and "а" (U+0430) is 100% and we denote it as Sim_v("a(U+0061)", "а(U+0430)")=100%. The construction of UC-SimList_v can be considered as a pattern similarity evaluation problem and we construct

it in an automatic way. In Section 5.3.2, 5.3.3, and 5.3.4, we propose three methods to calculate the visual similarity for characters in UCS.

5.3.1.2 UC-SimList_s: Semantic Similarity

UC-SimList_s is also a list, from which we can easily find whether a given pair of characters is semantically equivalent, such as the semantic similarity of "a" (U+0061) and "A" (U+0041) is 100%, and we denote it as Sim_s("a", "A")=100%. The construction of UC-SimList_s may need much more human invervention. However, the alphabetic lists of different languages could help solve most of the problems, such as English (Upper/Lower Cases), Chinese (Simplified/Traditional Characters), and Japanese (Katakana/Hiragana). Indubitably, much more languages and language usage customs should be considered and included into the UC-SimList_s.

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. For example, 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. For example, Japanese "木" (pronounces "Ki", and stands for "wood"), also can be represented as "キ" and "き".

The construction of UC-SimList_s is very important. However, it is complicated. We have constructed the UC-SimList_s of three languages, English (Uppercase and Lowercase), Chinese (Simplified and Traditional), and Japanese (Katakana and Hiragana). The UC-SimList_s is also available in [28]. We also leave other language parts as an open engineering problem and expect people could add and more languages to UC-SimList_s as follow-up work.

5.3.1.3 UC-SimList

UC-SimList is generated by considering the semantically similar characters for each character in UCS, finding all visually similar characters of each semantically similar character, and ranking all these characters with their visual similarities for each character. That is, UC-SimList(*c*)=UC-SimList_v(UC-SimList_s(*c*)), where *c* is a given character, UC-SimList_s(*c*) denotes the semantically similar character set of *c*, UC-SimList_v(・) denotes the visually similar character set of character set ・, and UC-SimList(*c*) denotes the similar character set of character *c*. For example, to construct UC-SimList, we first need to find the similar characters in UC-SimList_s for a given character, e.g. we find two characters, "a" and "A", are semantically similar to "a" (including "a" itself). We use the semantically similar characters as a source and find all of the visually similar characters of the source from UC-SimList_v, e.g. we find "а" (U+0430), "ａ"(U+FF41), "Α"(U+0491), "Ａ"(U+FF21), "А"(U+0410) are 100% similar to either "a"(U+0061) or

“A”(U+0041) in UC-SimList_v. (We consider each character is semantically similar to itself and calculate the similarity of a given pair of characters by multiplying visual similarity and semantic similarity).

In many cases, we can find corresponding replacement to one character, such as “a” to “A” (English in lower-case and upper case), “银” to “銀” (Chinese in simplified-form and traditional-form), “あ” to “ア” (Japanese in hirakana and katakana). The investigation on constructing UC-SimList_s is heavily dependent on language usage and character representation on the semantic level of each language character set in UCS, such as “一” and “壹” (stands for “one” in Chinese), and there is not a known automatic way to construct it. Therefore, UC-SimList_s has to be constructed manually.

We denote UC-SimList T as the Unicode Character Similarity List that only contains the characters with similarities larger than T (the similarity threshold), e.g., UC-SimList0.8 is a subset of UC-SimList that only contains the characters with similarities larger than 0.8. We also define the notation UC-SimList_v T in a similar way.

We use Unicode Arial MS Font Ver.1.01 as our character image generation font since 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 [67] and all characters of Unicode Ver. 2.1 [67].

5.3.2 Overlapping Based Assessment

In this method, the visual similarity of character c_1 and c_2, $vs(c_1,c_2)$, can be calculated using Eq. 15

$$vs(c_1, c_2) = \frac{|OverlapPix(c_1, c_2)|}{p|Pix(c_1)| + (1-p)|Pix(c_2)|} \quad 15)$$

where $|OverlapPix(c_1,c_2)|$ is the number of overlapping pixels of the bitmaps of c_1 and c_2, $|Pix(c)|$ is the number of pixels of character c, and $p \in [0,1]$ is the factor for tuning the similarity computation validity. Our experiment shows that p performs the best when it is defined as Eq. 16

$$p = \begin{cases} 1 & if\ |Pix(c_1)| \geq |Pix(c_2)| \\ 0 & otherwise \end{cases} \quad 16)$$

Semantic similarity of two characters is the measurement of character similarity in term of meaning. It is common that one character has more corresponding representations in the same or different languages. In our approach, we define the semantic similarity of two characters as a binary value, i.e. either 1 or 0. For example, *ss*("a", "A")=1, and *ss*("a", "b")=0, where $ss(c_1,c_2)$ is the semantic similarity of character c_1 and c_2. We calculate the similarity of each pair of characters using their pixel overlapping percentage. Our experiments show that when we set the size of Arial Unicode Font to 30 points, it performs 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 (Pentium IV 2.4G single CPU and 512M RAM). We select a threshold to define whether a pair of characters is similar. Our experiments show that 0.8 is a good choice. UC-SimList_v in different threshold-versions is now available at [28]. Most visually similar characters can be found out. However it could not guarantee all visually similar characters of each characters rank the highest. 005A ("Z") is more visually similar to

017E (“Ž”) than to 01A9 (“Σ”), 03A3 (“Σ”), 0032 (“2”), and FF12 (“２”). However, 017E (“Ž”) does not rank high. Hence, we may need a more sophisticated method to some such problem.

We use different thresholds to generate UC-SimLists. We used 0.95, 0.9, 0.85, and 0.8 as thresholds to generate UC-SimLists and they are also available at [28]. 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.

5.3.3 Kernel Density Estimation (KDE) Based Assessment

5.3.3.1 Kernel Density Estimation

A character is represented and measured in 2D kernel densities around the sample points on its contour. Therefore, characters’ similarity can be intuitively measured by the similarity of their 2D densities. Kullback-Leibler (KL) divergence [15] is a useful dissimilarity measure of two densities. Let the density of one character be U(x), that of the other be V(x), the dissimilarity between the two characters, Dis(U,V), can be defined as $Dis(U,V)=\frac{1}{2}(KL(U(x),V(x))+KL(V(x),U(x)))$, where U and V can be estimated by Gaussian functions. Zhang et al have demonstrated that the effectiveness of KDE on symbol recognition in [78].

5.3.3.2 UC-SimList Generation

We choose Arial Unicode MS 1.01 as the font in our experiments and the UC-SimList_v construction. Arial Unicode MS 1.01 is a true type font (TTF). Each character is

represented with one or more contour(s); each contour comprises quadratic spline(s) (QS) and/or straight line(s) (SL); and each QS/SL is represented with critical points. We retrieve the font information (the critical points of each character) from Arial Unicode MS 1.01 and convert them to sets of points.

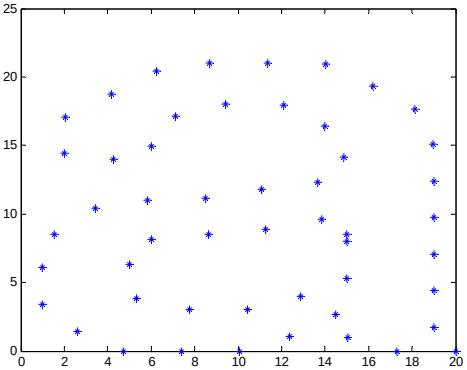
We denote N to be the number of points in the converted point sets. If N is larger, the quality of the representation will be the better, as shown in Figure 15. Our experiment shows that, when N=50, it is good enough to represent the visible characters in the range of U+0000 to U+00FF (ie. ASCII). Our experiment also shows that the process of calculating the KDEs for one character to the other ones (2^16-1=65535 characters) takes about 1 hour when N=100, such that we need $\frac{1}{2^{16}-1}C_{2^{16}}^2$ hours (about 3.74 years) to finish the calculation (We used a P4 2.4GHz PC with 1GB memory). Hence, it is unrealistic to calculate a complete UC-SimList_v. As a matter of fact, it will take much longer to calculate if we concern about the information loss and use N=200. Our experiments shows that N=200 will be good enough to represent all characters in the UCS. Figure 15 shows one example of representing Chinese character "银" using 200 points. Hence, we only calculate the characters in the range of U+0000 to U+00FF (ASCII). In fact these characters are the most frequently used characters and most easily to be targeted to carry out *Unicode attacks*.

The utilization of KDE brings us the accuracy improvement comparing with the pixel-overlapping based assessment [27] [26], where U+94F6: 银 and U+9512: 钖 are considered to be similar. However, U+9512:钖 ranked eleventh using that method as

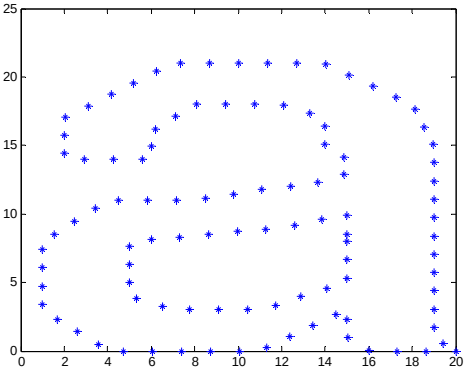
shown in Figure 15 because the two characters' glyphs have some offset to each other, such that the common area of the two characters are reduced. In comparison with the former method, the KDE based assessment performs much better and rank U+9512:银 to be the second similar character to U+94F6:银. UC-SimList_s, UC-SimList_v, and UC-SimList are available at [5] for free download for researchers.

Similarity Rank	Pixel-overlapping based assessment		KDE based assessment	
	Similarity	Char Code and Char	Similarity 1-KDE	Char Code and Char
1	0.765574	U+953F:长	0.94583	U+9503:锃
2	0.75	U+9530:锰	0.9448	U+9512:银
3	0.746032	U+9502:锂	0.93425	U+94BD:钽
4	0.741408	U+94FB:锻	0.93394	U+953F:长
5	0.740973	U+9491:钑	0.92788	U+94A1:钡
6	0.733114	U+9510:锐	0.92713	U+9502:锂
7	0.729299	U+88C9:裉	0.9271	U+953D:镽
8	0.728171	U+94C0:铀	0.92619	U+94BF:钿
9	0.727422	U+94C7:铇	0.91884	U+953C:镼
10	0.726524	U+9526:锦	0.91836	U+9522:锢
11	0.724194	U+9512:银	0.91583	U+94B7:钷
12	0.72293	U+9525:锥	0.91101	U+6068:恨

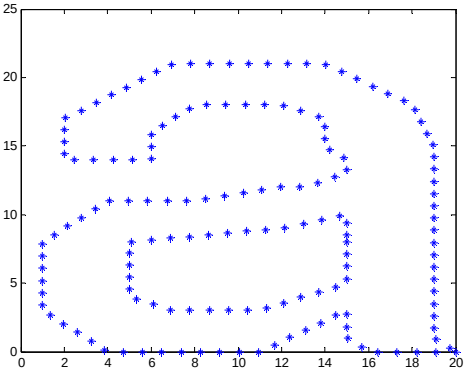
Figure 14. The comparison of using pixel-overlapping based assessment and KDE based assessment using sample target character, U+94F6:银 (The four digits between "U+" and ":" are the character numbers in hexadecimal form and the character is following ":").



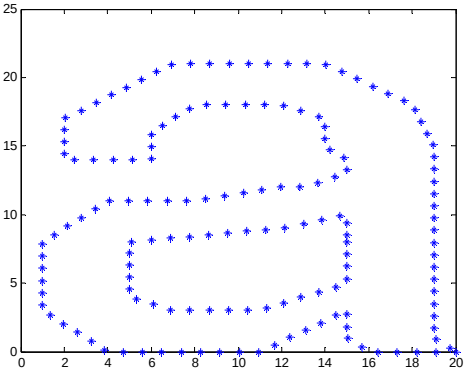
N=50



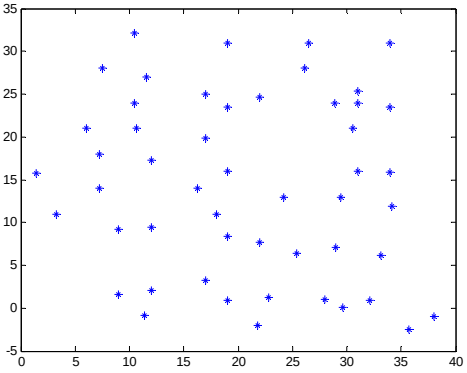
N=100



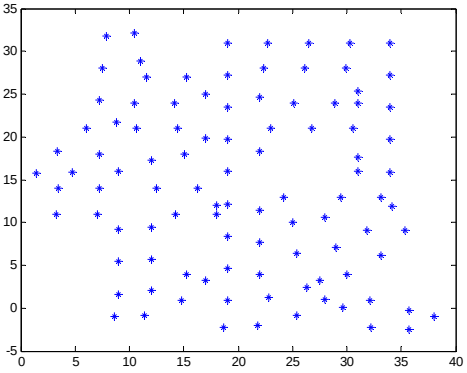
N=150



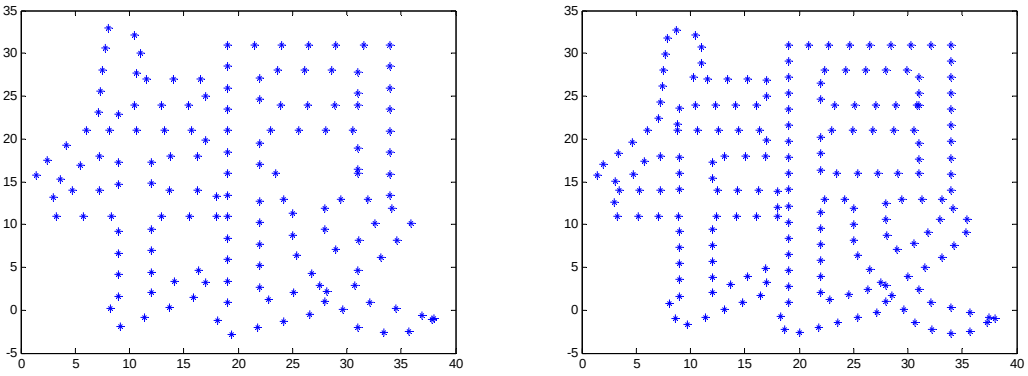
N=200



N=50



N=100



N=150 N=200

Figure 15. The converted point sets of "a" and "银" with N=50, 100, 150, and 200. Intuitively, when N=100 it is good enough to represent "a", while when N=200, the representation of "银" could be satisfied. (Unit of X-axis and Y-axis: Font-Point)

5.3.4 Quadratic Spline Based Assessment

5.3.4.1 Motivation

KDE has a good performance of calculating character similarities. However it is slow. The problem comes from that we need to calculate for each pair of sampled points in KDE. However, the true type fonts, such as MS Arial Unicode, are represented with quadratic splines in the font databases. We propose a way to calculate the similarity of splines in a continuous mathematical way, such that we can calculate the similarity of characters on the fly.

5.3.4.2 Model Design

All characters are represented with quadratic splines. These splines are in different square planes. We consider the similarity of two points, X and Y, as shown in Figure 16 (a) and (b). Suppose the appearance of X and Y satisfy 2 dimensional normal distribution, we can calculate the probability of X and Y being identical. The normal distribution probability

functions are like hoods on the top of X and Y. We draw a line AB through X and Y, and make a plane perpendicular to plane in Figure 16 (c). The section curves look like Figure 17. Therefore, the common volume of the two hoods, as shown in the shadow area, is the probability of X and Y being identical.

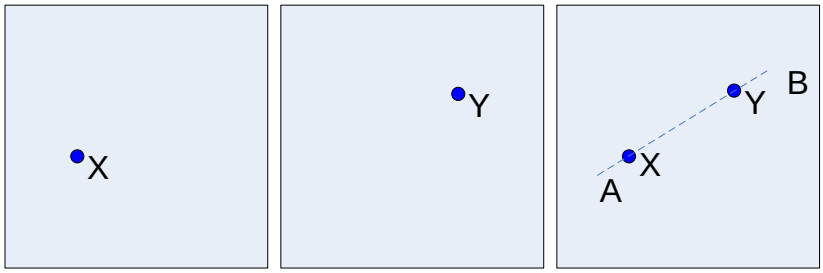


Figure 16. Sample Points (X and Y) under Evaluation

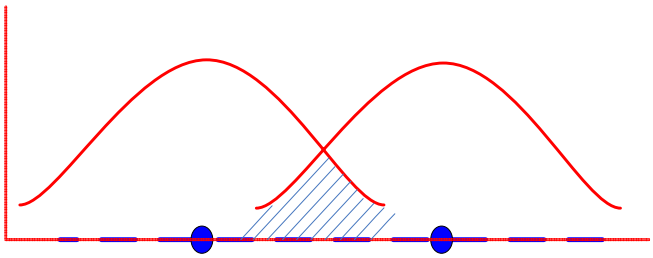


Figure 17. Probability of being Identical between Sampled Points (X and Y)

The PDF (Probability Distribution Function) of two-dimensional normal distribution in canonical form is shown in Eq. 17. σ_x and σ_y are standard deviations, C is the covariance matrix, ρ is the covariance of distribution on x-axis and y-axis, X is the variants, and $\overline{X}$ is the expectation of X.

$$p(X) = \frac{1}{2\pi\sqrt{|C|}} \exp\left(-\frac{1}{2}(X-\overline{X})^T C^{-1}(X-\overline{X})\right), \text{ where } C = \begin{bmatrix} \sigma_x^2 & \rho\sigma_x\sigma_y \\ \rho\sigma_x\sigma_y & \sigma_y^2 \end{bmatrix} \quad 17)$$

We assume $\rho = 0$, $\sigma_x = \sigma_y = \sigma$, and $\overline{X} = (0, 0)$, $p(X)$ can be represented as Eq. 18.

$$p(X) = \frac{1}{2\pi\sigma^2}\exp\left(-\frac{X^T X}{2\sigma^2}\right) \quad 18)$$

We can see the shape of the PDF surface is radically symmetric. The half volume of the intersection in Figure 17 can be represented in Eq. 19 and Eq. 20.

$$\int_{-\infty}^{\infty}\int_{\frac{d}{2}}^{\infty}\frac{1}{(2\pi\sigma^2)\, e^{\frac{x^2+y^2}{2\sigma^2}}}\,dx\,dy \quad = \quad \frac{1}{2}\,\mathrm{Erfc}\left[\frac{d}{2\sqrt{2}\,\sqrt{\sigma^2}}\right] \quad 19)$$

where d is the length of line segment XY.

$$\mathrm{Erfc}(z) = 1 - \mathrm{Erf}(z), \text{ where } \mathrm{Erf}(z) = \frac{2}{\sqrt{\pi}}\int_0^z e^{-t^2}\,dt \quad 20)$$

where Erf(z) is called error function and Erfc(z) is the complementary of Erf(z).

According to the discussion in [50], Erf(z) cannot be expressed in terms of finite additions, subtractions, multiplications, and root extractions. Hence, it must be either computed numerically or otherwise approximated. Sergei Winitzki [66] has given a very good and handy approximation of Erf(z) as shown in Eq. 21.

$$\mathrm{Erf}(x) \approx \left(1 - e^{\left(-x^2\frac{\frac{4}{\pi}+ax^2}{1+ax^2}\right)}\right)^{\frac{1}{2}} \quad \text{, where} \quad a = \frac{8}{3\pi}\,\frac{\pi-3}{4-3\pi} \approx 0.14 \quad 21)$$

Such that we can get the approximated Erfc(z) as shown in Eq. 22.

$$\mathrm{Erfc}(x) \approx \mathrm{ApproxErfc}(x) = 1 - \sqrt{1 - e^{-\frac{x^2\left(\frac{4}{\pi}+0.14\,x^2\right)}{1+0.14\,x^2}}} \quad 22)$$

We want to evaluate the effectiveness of this approximation, and Figure 18 shows the approximation effect. The lower curve is ApproxErfc[x] and the upper one is Erf[x], where $X \in [0, 3]$

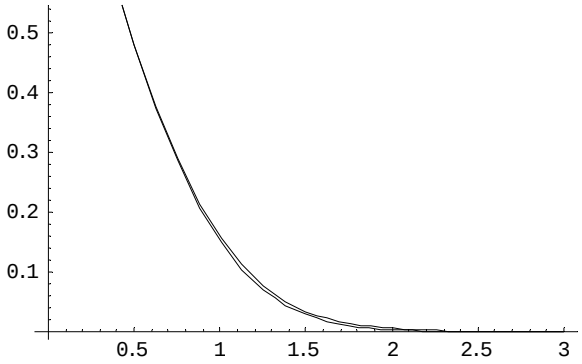


Figure 18, Approximation result of ApproxErfc[z].

We use parameter equations to represent quadratic splines. Suppose the two splines under comparison are QSpline1[x1(t1), y1 (t1)] and QSpline2[x2(t2), y2 (t2)], as shown in Eq. 23, where a1(xa1, ya1), b1(xb1, yb1), c1(xc1, yc1) are three critical points used to define QSpline1, and a2(xa2, ya2), b2(xb2, yb2), c2(xc2, yc2) are three critical points used to define QSpline2.

$$\begin{aligned} x1(t1) &= (xa1 - 2\,xb1 + xc1)\,t1^2 + (2\,xb1 - 2\,xa1)\,t1 + xa1 \\ y1(t1) &= (ya1 - 2\,yb1 + yc1)\,t1^2 + (2\,yb1 - 2\,ya1)\,t1 + ya1 \\ x2(t2) &= (xa2 - 2\,xb2 + xc2)\,t2^2 + (2\,xb2 - 2\,xa2)\,t2 + xa2 \\ y2(t2) &= (ya2 - 2\,yb2 + yc2)\,t2^2 + (2\,yb2 - 2\,ya2)\,t2 + ya2 \end{aligned} \qquad 23)$$

We use parameter equations to express double curve integration as shown in Eq. 24, which is the function to compute the similarity of the two splines. This integration can be calculated by numerical integration.

$$\int_{QSpline2}\int_{QSpline1} \frac{1}{2}\,\text{ApproxErfc}\left[\frac{||\,l_1 - l_2\,||}{2\sqrt{2}\,\sqrt{\sigma^2}}\right] dl_1\,dl_2$$

$$= \int_0^1\int_0^1 \frac{1}{2}\,\text{ApproxErfc}\left[\frac{1}{2\sqrt{2}\,\sqrt{\sigma^2}}\left(\sqrt{((x1-x2)^2 + (y1-y2)^2)}\right)\right]$$
$$((xa1 - 2\,xb1 + xc1)\,2\,t1 + (2\,xb1 - 2\,xa1) + (ya1 - 2\,yb1 + yc1)\,2\,t1 + (2\,yb1 - 2\,ya1))$$
$$((xa2 - 2\,xb2 + xc2)\,2\,t2 + (2\,xb2 - 2\,xa2) + (ya2 - 2\,yb2 + yc2)\,2\,t2 + (2\,yb2 - 2\,ya2))$$
$$dt2\,dt1 \qquad 24)$$

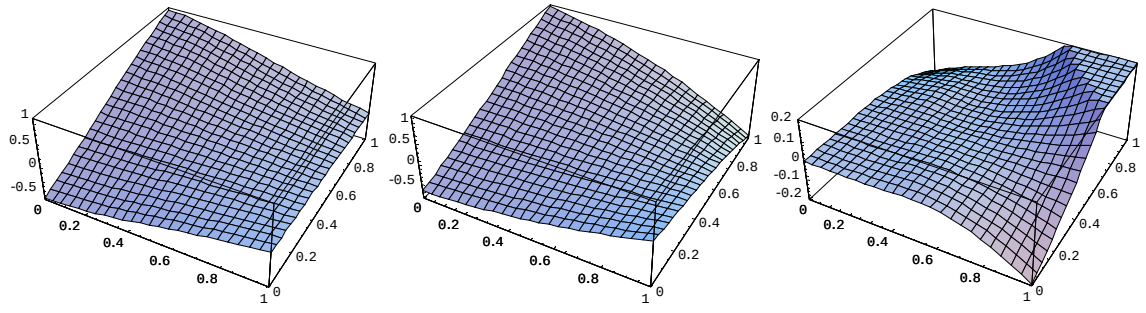
5.3.4.3 Polynomial Approximation Fails

We have tried to make one step of approximation using polynomial expressions but failed. In this section, we demonstrate polynomial approximation fails to fit Eq. 24.

Suppose xa1=0, xa2=0.5, xb1=0.7, xb2=0.2, xc1=0.9, xc2=0.3, ya1=0.4, ya2=0.5, yb1=0.8, yb2=0.43, yc1=0.5, yc2=0.9, and σ =1, so the third order polynomial approximation can be generated, as shown in Eq. 25.

$$\frac{1}{2}\,\mathrm{Erfc}\left[\frac{1}{2\sqrt{2}\,\sqrt{\sigma^2}}\left(\sqrt{(x1-x2)^2+(y1-y2)^2}\right)\right]$$
$$((xa1-2\,xb1+xc1)\,2\,t1+(2\,xb1-2\,xa1)+(ya1-2\,yb1+yc1)\,2\,t1+(2\,yb1-2\,ya1))$$
$$((xa2-2\,xb2+xc2)\,2\,t2+(2\,xb2-2\,xa2)+(ya2-2\,yb2+yc2)\,2\,t2+(2\,yb2-2\,ya2))$$
$$\approx \qquad (25)$$
$$-0.6501914998096506`+0.228503872832102`\,t1+0.7517609471605787`\,t1^2-$$
$$0.07300001263430961`\,t1^3+1.4583012414988377`\,t2-0.39880128375967594`\,t1\,t2-$$
$$1.698022918180373`\,t1^2\,t2+0.6382405091859811`\,t2^2-0.7106046491219699`\,t1\,t2^2-$$
$$0.3414335077448442`\,t2^3$$

Although the shapes of original equation (Figure 19, a) and simulated one (Figure 19, b) look alike, the error is big (Figure 19, c). We also used the fourth order polynomial approximation, and the result does not improve much to the third order approximation in Figure 19.



(a) (b) (c)

Figure 19. Approximation result using third order polynomials.

5.4 Word-Word Semantic Similarity (WWSS) List Generation

We anticipate a list to look up the semantic similarities of words. In this section, we discuss word-word semantic similarity list, which is a large table that can easily find out the semantic similarity of given pair of words. We use Word Frequency of Written and Spoken English [44] and Information based Measure of semantic similarity [60] to calculate our WWSS List.

5.4.1 Word Frequency of Written and Spoken English

Word Frequency of Written and Spoken English (WFWSE) is generated by Leech et al in [44]. It is derived from the British National Corpus (BNC) [9]. BNC is a 100 million word collection of written and spoken language English corpus. It includes about 90% in written part and 10% in spoken part. WFWSE provides lists for the whole BNC. WFWSE simply is the best word frequency statistical work we can find.

5.4.2 Information based Measure

WordNet [72] is a good tool for carrying out semantic similarity evaluations. The traditional semantic similarity evaluation uses edge counting approach. Philip Resnik proposed information based measure for word semantic similarity evaluation in 1999 [60]. In this method, a measure of word-word semantic similarity using their shared information is proposed. Resnik's experiments evaluation against the human judgment shows it is better than edge counting. We consider his method as a feasible machine for what we are trying to do.

In this method, we first calculate the frequency of a given word *c*, as shown in Eq. 26.

$$freq(c) = \sum_{n \in words(c)} count(n) \quad 26)$$

where *words*(*c*) is the set of words subsumed by concept *c*. Concept probabilities can be computed simply as shown in Eq. 27.

$$p(c) = \frac{freq(c)}{N} \quad 27)$$

where *N* is the total number of words under evaluation. Therefore, Eq. 27 represents the normalized frequency. We calculate the similarity of two words, c_1, c_2 with Eq. 28.

$$sim(c_1, c_2) = \max_{c \in S(c_1, c_2)} [-\log p(c)] \quad 28)$$

where $S(c_1, c_2)$ is the common set of hyper names of the both c_1 and c_2.

5.4.3 WWSS List Generation (Implementation)

WordNet 2.0 is a free tool of cognitive laboratory of Princeton University. Crowe provided a C# port [49] to WordNet. We use this port to link WordNet to perform operations. However, it is still not possible to generate the complete WWSS List because we still do not have an effective way to can calculate the semantic similarity of words with different POSes (Position of Sentences). Hence WWSS List generation is still an on-going study. In this section, we demonstrate the generation of WWSS of the most frequently used 3030 nouns in WFWSE and all of their hypernyms that have relationships with them, so we have a total of 8587 nouns counted in.

We calculate the word frequency. The AS-IS relationship in WordNet is in the structure of a tree. Word frequency can be calculated by counting the total offspring number of a given word.

In this demo implementation, the N in Eq. 27 is 3030. Hence, the concept usage probability can be calculated.

The common information of a pair of given nouns can be calculated using Eq. 28. It is obvious that the WWSS List is a sparse matrix. Many pairs of words do not have any relationship at all, so their common information values are 0. We do not save such values. We use hash table to improve the searching speed. A key of hash table is a 32 bit integer the higher 16 bits represent the first word and the lower 16 bits represent the second word. The final similarity list is 339 MB. Figure 20 shows several random samples from WWSS List and the samples' semantic similarities are empirically satisfying human understanding.

Word1	Word2	Semantic Similarity
Left	Right	9.94
Coin	Money	8.56
Coin	Paper	4.91
Money	Paper	4.91
Doctor	Professor	8.28
Teacher	Professor	8.89
Coin	Student	0.0

Figure 20. The demo samples from the generated WWSS List

The reason we want the WWSS List is to save time. We did experiments to test the performance of online calculation of WWSS and searching it from the WWSS List (which actually is a hash table). The hashing search can save a lot of time. However, the generation process is quite time consuming. It took about 15 days on a PC with Pentium IV 2.4G and 1G RAM. However we only need to generate it once.

5.5 A Demo Implementation Study on the Methodology

We carry out a case study by applying the methodology. We organize this section in three subsections. In Section 5.3, we introduced our method of generating a Unicode similarity list (UC-SimList). The word-word similarity is an ongoing study and we would rather omit this part since it does not affect the discussion here. In Section 5.5.1, we present a possible string similarity algorithm. It is implemented as the algorithm for use in our experiments. In Section 5.5.2 we introduce the vision and semantics based edit distance. Section 5.5.3 introduces similar/faked Unicode string generation. In Section 5.5.4, we address the problem of associating relatively long similar/fake Unicode strings with the genuine ones. We use the strings to imitate spam attacks. In Section 5.5.5, we address the problem of detecting phishing IRI/IDNs in a set of protected ones. It is a special and critical issue, so it should be considered specifically. In Section 5.5.4 and 5.5.5, we discuss aspects of the experimental data generation, classification effectiveness and time performance evaluation.

5.5.1 Unicode String Similarity Algorithm for Experiments

We use edit distance (a.k.a. Levenshtein Distance) [45] to calculate the dissimilarity of the pair of Unicode strings under evaluation. Edit distance represents the minimum editing operations needed to transform one string into another, where the only operations are insertion, deletion, and substitution. We define the cost (cost function) of insertion and deletion to be 1 and the cost (cost function) of substitution to be 1 minus the similarity in UC-SimList0.8. We use a standard dynamic programming algorithm to calculate edit distance. Its time complexity is $\Theta(mn)$, where m and n are the respective

lengths of the two Unicode strings. Experiments in Section 5.5.4 and 5.5.5 show that this is sufficient for online similar/fake Unicode string detection. The edit distances are normalized by dividing by *Max*(*m*,*n*), such that we can define a threshold to classify whether a given suspected Unicode string is too similar to a string in a set of protected Unicode strings.

5.5.2 Vision and Semantics based Edit Distance

We use edit distance (a.k.a, Levenshtein distance [45]) to represent the dissimilarity of Unicode strings. Edit distance represents the minimum editing operations required to transform one string into another by insertion, deletion, and substitution. We use a dynamic programming to calculate edit distance for better efficiency and performance.

Suppose we have two strings $w = c_{w,1}c_{w,2}...c_{w,m}$ and $v = c_{v,1}c_{v,2}...c_{v,n}$. We calculate an $(m+1)\times(n+1)$ list $M = [d_{i,j}]$, as shown in Figure 21, where $d_{i,j}$ is the edit distance of $w_i = c_{w,1}c_{w,2}...c_{w,i}$ and $v_j = c_{v,1}c_{v,2}...c_{v,j}$, $0 \le i \le m$, $0 \le j \le n$, and λ denotes the empty string. Each $d_{i,j}$ is calculated as shown in line 12 of Figure 22, where σ is the cost of insertion and deletion, $\gamma(c_{w,i}, c_{v,j})$ for the cost of substitution of $c_{w,i}$ to $c_{v,j}$ or reversely, and initial value of $d_{0,0}$ is zero. $d_{m,n}$ is the edit distance of w and v. The time efficiency is $\Theta(mn)$, where m and n are lengths of the pair of Unicode strings under calculation. Experiments in Section 5 show it is good enough for online similar/fake Unicode string detection. In the edit distance calculation, we redefine the cost function of character substitution as the distance of the pair of characters under comparison, which is addressed in Section 3, and set the insertion and deletion cost to be 1 respectively. The edit distances are normalized by dividing the larger number of m and n, i.e. $Max(m,n)$, such that we can define a

threshold to classify a given suspected Unicode string (e.g., a spam email, fake web page or phishing IRI/IDN with replaced similar characters) is a similar/fake one or not against a series of protected Unicode strings. Figure 22 shows the pseudo for calculating the normalized edit distance of a pair of Unicode strings, w and v.

$\mathbf{w}$ \ $\mathbf{v}$	λ	$c_{v,1}$	$c_{v,2}$	…	$c_{v,n}$
λ	d_{00}=0	1	2	…	n
$c_{w,1}$	1	$d_{1,1}$	$d_{1,2}$	…	$d_{1,n}$
$c_{w,2}$	2	$d_{2,1}$	$d_{2,2}$	…	$d_{2,n}$
...	…	…	…	…	…
$c_{w,m}$	m	$d_{m,1}$	$d_{m,2}$	…	$d_{m,n}$

Figure 21. Dynamic programming list for calculating edit distance

NED($w = c_{w,1}c_{w,2}...c_{w,m}, v = c_{v,1}c_{v,2}...c_{v,n}$)

1. {
2. if(m ==0&& n ==0) return 1;
3. if(n == 0) return $m / Max(m,n)$;
4. if(m == 0) return $n / Max(m,n)$;
5. σ =1; $d_{0,0}$ =0;
6. for(int $i = 1; i <= m; i++$) $d_{i,0} = d_{i-1,0} + \sigma$;
7. for(int $j = 1; j <= n; j++$) $d_{0,j} = d_{0,j-1} + \sigma$;
8. for(int $i = 1; i <= m; i++$)
9. {
10. for(int $j = 1; j <= n; j++$)
11. {
12. $$d_{i,j} = Max\begin{cases} d_{i,j-1} + \sigma \\ d_{i-1,j} + \sigma \\ d_{i-1,j-1} + \gamma(c_{w,i}, c_{v,j}) \end{cases}$$
13. }
14. }
15. return $d_{m,n} / Max(m,n)$;
16. }

Figure 22. Algorithm for calculating the Normalized Edit Distance (NED)

5.5.3 Similar/Faked Unicode String Generation

We use the UC-SimList_v and UC-SimList to generate similar Unicode strings from a given one. The generation process is quite simple: we only need to replace one or more of character(s) in the given string to their similar characters. However the combination could be large for one given Unicode string and we only list several samples of them. We demonstrate the generated similar strings in Appendix C.

It is obvious that the string similarity is losing when we a use similarity list with lower threshold number, however, the variety will increase since more similar characters can be used as substitution candidates. Thus, it is quite easy for phishers/hackers to use a similar character table like UC-SimList_v and UC-SimList to mimic the real websites and domain names, and it is also possible for spammers to use UC-SimList_v or UC-SimList to duplicate various spam emails to avoid the detection of spam email filters.

5.5.4 Experiments with Normal Text Strings

We begin our experiments with three Unicode strings extracted from three web pages, namely the English, Chinese and Japanese versions of CitiBank, as shown in Figure 23. We denote these strings as US_E, US_C, and US_J, respectively. We then generate similar/fake strings based on each of strings by substituting some characters with visually or semantically similar ones from UC-SimList_v or UC-SimList. We generate 5 sets of similar/fake ones using each of UC-SimList T and UC-SimList_v T for each original string, where $T \in \{0.8, 0.85, 0.9, 0.95, \text{and } 1\}$. Each of the original Unicode strings corresponds to 10 sets of similar/fake ones and each set contains at most 100 similar/fake Unicode strings. We denote the 10 sets of similar/fake Unicode strings of US_E as $SUS_E(X)$, the 10 sets of US_C as $SUS_C(X)$, and the 10 sets of US_J as $SUS_J(X)$, where

X∈{0.8, 0.85, 0.9, 0.95, 1, V0.8, V0.85, V0.9, V0.95, and V1}. SUS_E(0.8) is the similar/fake Unicode string set generated using UC-SimList0.8, and SUS_E(V0.8) is the similar/fake Unicode string set generated using UC-SimList_v0.8, and so on. We use 83,198 Unicode strings in TREC-5 Confusion Track (part original-01) [42] as noise data and denote this set as *RUS*. We combine SUS_E(X) with *RUS* to form the set of suspected Unicode strings for US_E and calculate the precision and recall value of our similar/fake Unicode string detection algorithm for US_E with varying thresholds (from 0 to 1, with a step of 0.01). A partial result is shown [5].

We also perform the same experiments on US_C and US_J, as shown in Figure 25 and Figure 26 respectively, and the complete results are also available in [5]. We calculate the precision value using Eq. 29 and the recall value using Eq. 30 respectively. *TF* denotes the total ground truth number of fake ones (phishing) in the corresponding test set.

$$precision = \frac{CN}{TN} \times 100\% \quad (29)$$

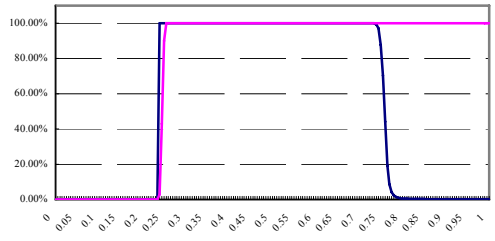
$$recall = \frac{CN}{TF} \times 100\% \quad (30)$$

It is obvious that in a certain threshold range, we can achieve very good precision and recall values (both are approximately 100%) at the same time, i.e., 0.25~0.75 for US_E, 0.05~0.95 for US_C, and 0.05~0.97 for US_J. The wider the range is, the easier the classification of similar/fake Unicode strings is. Intuitively, similar/fake Unicode strings of Chinese and Japanese are easier to be detected than those of English, and the reason is that English has far more similar characters in UCS than Chinese and Japanese do. We can observe two phenomena from the precision and recall figures. First, the threshold range is wider when the similar/fake Unicode strings are generated using UC-SimList T or UC-SimList_v T with higher T (similarity threshold) values. Second, the threshold

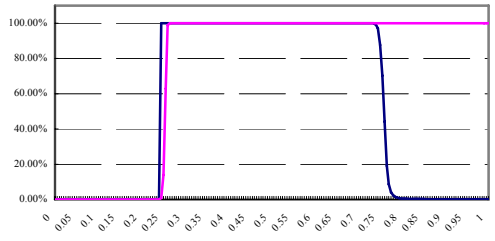
range is wider when similar/fake Unicode strings are generated using UC-SimList_v T rather than UC-SimList T with the same T. However the second phenomenon is not very obvious in our experiments when we used UC-SimList0.8 as the cost function in Section 5.5.1. These phenomena also adapt to the experiment in Section 5.5.5. It takes 0.15 seconds to calculate the similarity of a Unicode string to US_E, 0.035 second to US_C, and 0.045 second to US_J on average (using a PC with P4 2.4G CPU and 512M RAM). The proportion of computation time, 0.15:0.035:0.045=1:0.23:0.3 is approximately equivalent to the character number proportion of the three original Unicode strings, 313:79:102=1:0.25:0.33, which verifies that the computational complexity grows linearly with the target string length as addressed in Section 5.5.1. All of the experimental data sets are available at [5].

From	Original Unicode strings (attacked targets)
www.citibank.com	Every Internet user should know about spoof (a.k.a. phishing or hoax) e-mails that appear to be from a well-known company but can put you at risk. Although they can be difficult to spot, they generally ask you to click a link back to a spoof web site and provide, update or confirm sensitive personal information. (total: 313 characters)
www.citibank.com.cn	最近，电子邮件用户成为全球网络黑客的攻击目标。花旗银行相信让所有网上银行用户了解邮件欺诈是至关重要。因此，我们为您提供了一系列建议以防止您的金融信息受到攻击。(total: 79 characters)
www.citibank.co.jp	最近、シティバンクを装って送られる偽の電子メールが増えております。一般的にこのような電子メールにあるリンクをクリックすると、暗証番号や口座番号など個人の機密情報の入力を促す偽のウェブページがあらわれます。(total: 102 characters)

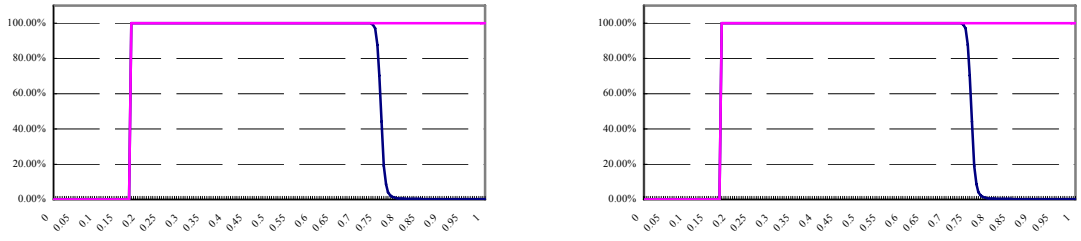
Figure 23. Original Unicode strings for English, Chinese and Japanese from the web pages of CitiBank



X=0.8

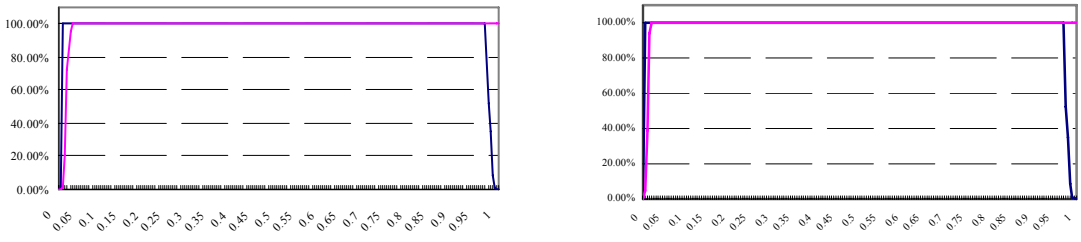


X=V0.8

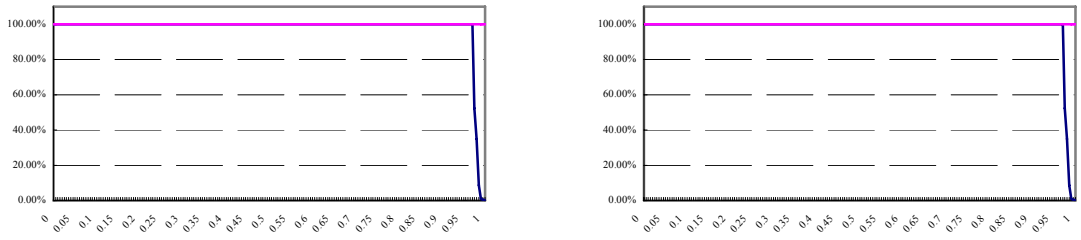


X=1 X=V1

Figure 24. Precision and recall evaluation of detecting similar/fake Unicode strings to US_E (the purple curve is recall, the blue one is precision, the x-axis denotes the varying threshold and the y-axis denotes the precision/recall percentage value)

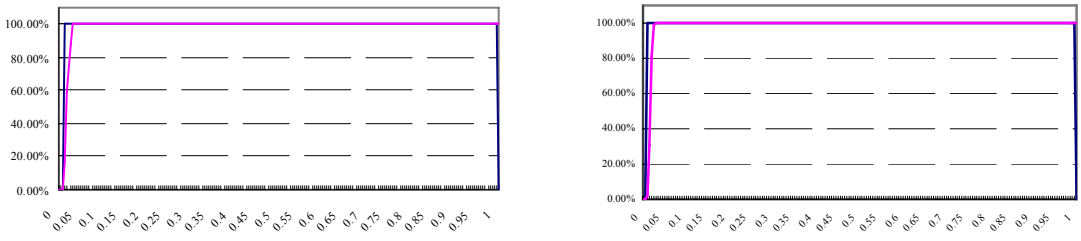


X=0.8 X=V0.8

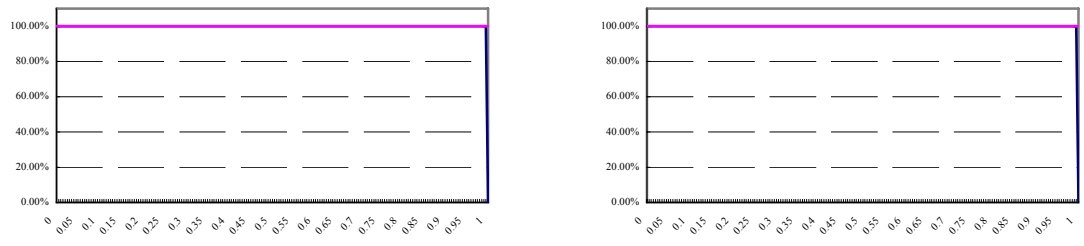


X=1 X=V1

Figure 25. Precision and recall evaluation of detecting similar/fake Unicode strings to US_C (the legend is the same as in Figure 24)



X=0.8 X=V0.8



X=1 X=V1

Figure 26. Precision and recall evaluation of detecting similar/fake Unicode strings to US_J (the legend is the same as in Figure 24)

5.5.5 Experiments with IRI/IDNs

Although IRI/IDN could be the complement or the replacement of URI in the near future, IRI/IDN is not used widely at present, and the number of real phishing URIs or IRI/IDNs is small. Hence, we also need to generate phishing IRI/IDNs for our experiments. Suppose we have 10 IRI/IDNs under protection as listed in Figure 27. We will denote these as US_{IRI}. Next, we generate 100 similar IRI/IDNs for each of them by replacing similar characters in each original IRI/IDNs using UC-SimList T and UC-SimList_v T, where T∈{0.8, 0.85, 0.9, 0.95, and 1}. Note that we get 1,000 IRI/IDNs for each element of US_{IRI}, since there are 5 different T values, and we use each value in both UC-SimList T and UC-SimList_v T. We treat all of these generated IRI/IDNs as phishing IRI/IDNs, and we denote the 10 sets as $SUS_{IRI}(X)$, following the conventions defined in Section 5.5.4.

www.citibank.com	www.icbc.com
www.bank-of-china.com	www.wellsfargo.com
www.ebay.com	www.keybank.com
www.wamu.com	www.花旗银行.公司
www.usbank.com	www.シティバンク.会社

Figure 27. The 10 IRI/IDNs under protection

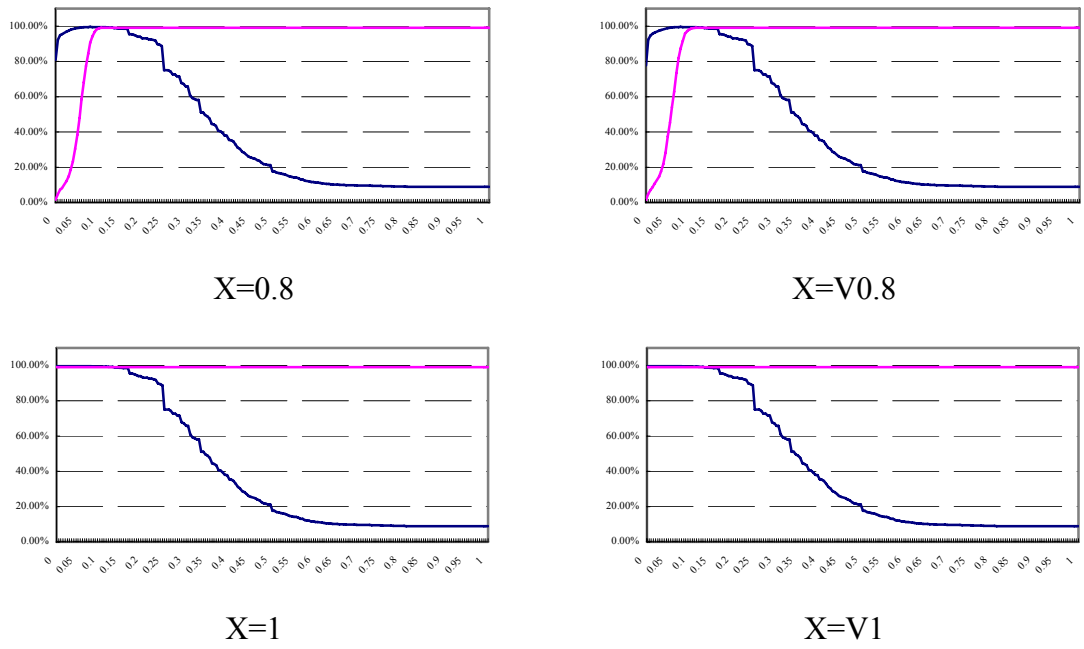


Figure 28. Precision and recall evaluation of detecting phishing IRI/IDNs to USIRI (the legend is the same as in Figure 24)

We mix the phishing IRI/IDNs up with 10,269 real web addresses that are randomly collected from the Internet. We then use the algorithm in Section 5.5.1 to perform phishing IRI/IDN detection. We use precision and recall values to evaluate our approach with varying edit distance thresholds. We also do experiments with UC-SimLists with various filtering thresholds (from 0 to 1, with a step of 0.01). We mix up $SUS_{IRI}(X)$ with 11,269 IRI/IDNs in random order to form the suspected Unicode strings and calculate the precision and recall value of detecting similar/fake Unicode strings to the 10 protected IRI/IDNs with varying thresholds. The partial experimental results are shown in Figure 28 and the complete results are in [5]. It is obvious that there is a clear threshold range, 0.08~0.17, where both high precision and recall values (both approaching 100%) can be achieved. It takes about 10^{-3} seconds to calculate the similarity of one pair of IRI/IDNs

using the same machine we used in Section 5.5.5. This performance is sufficient for online phishing IRI/IDN detection.

5.6 Discussion on Practical Use and Deployment Proposal

We have demonstrated in Section 5.5 that the prototype application performs well for Unicode attack detection. All it needs is a list of domain names to protect and the attack detection algorithms. However, deploying the technology in real-world situations raises issues that we would like to address in this section.

5.6.1 Domain Name Server

When end users want a new domain name, they get it from domain name registrars. The registrars are relatively autonomous. As defined in [39], "*the Internet Corporation for Assigned Names and Numbers (ICANN) is responsible for managing and coordinating the Domain Name System (DNS) to ensure that every address is unique and that all users of the Internet can find all valid addresses. ICANN is also responsible for accrediting the domain name registrars. "Accredit" means to identify and set minimum standards for the performance of registration functions, to recognize persons or entities meeting those standards, and to enter into an accreditation agreement that sets forth the rules and procedures applicable to the provision of Registrar Services.*" Hence we believe that if ICANN required the use of a tool such as ours prior to domain name registration, then the number of domain names registered for the purposes of phishing attacks would be significantly reduced. Users that are registering new domain names may worry that this limits their freedom to register any domain name, but it also protects any domain name that they do register from a variety of phishing attacks. As a matter of fact, ICANN is

realizing Unicode Attack problems. ICANN first added the “Unicode security condition”, which is exactly the problem we are addressing, in Guidelines for Implementation of IDNs [39], Ver. 2.0, Nov. 8, 2005, and continued to list it in Ver. 2.1, Feb. 22, 2006. However we have not seen a solution yet. We hope that the method proposed here can be used to address this issue.

Another solution is that domain name service registrars may provide the following service: once a domain name is registered, all similar domain names will automatically be registered to the same domain name owner. This service can be implemented by enumerating every way of substituting similar characters from UC-SimList for the characters in the domain name. The obvious drawback of this approach is that some domain names will have a prohibitively large number of similar names (since this number grows exponentially in the length of the name). However, some organizations may be willing to pay a premium for the resulting security if their name is small enough to secure in this manner.

5.6.2 Anti-Phishing Client Application

A defense based on our method of finding visual similarities in Unicode strings can be embedded into anti-phishing client applications or web browser plug-ins and installed in end user computers. These client applications and plug-ins could act as filters for the web links that users try to use to access the Internet. If any suspected web link is found, then they can provide alert information. One possible problem is that it may be hard for client applications or plug-ins to maintain one complete list of legitimate domain names. However they can be designed as only maintaining the most security sensitive domain

names, e.g., only maintain the list of banks, credit card companies, and online transaction services.

5.6.3 Registrar Applications

As we discussed, Web Identity could become a big target for malicious people. They can use Unicode attacks to create similar user names and accounts in the Internet. We would like to propose that username/account registrars in various online systems use Unicode attack detection systems to overcome possible attacks. For instance, when someone wants to register a new username/account in a BBS (Bulletin Board System) that allows users to register with Unicode names, we propose using Unicode detection systems to verify that the new username is not a homograph of any names in the system.

5.6.4 Content Filtering

It is possible that phishers can carry out attacks through systems like Email Servers, BBSs, Wikis, and Search Engines. There have been reports of phishers undermining Google results to direct unsuspecting users to their phishing websites, which is called *Page Rank Manipulation*. Such attacks not only cause network security problems, but also affect the reputations of decent service providers. Our method can be used by such systems to detect phishing Unicode attacks.

5.6.5 IRI/IDN SecuChecker

We implement IRI/IDN SecuChecker, which provides a mechanism to help domain name registrars check the validity of new registered domain names. We demonstrate SecuChecker usage and features in Appendix D.

5.7 Detect Phishing IRI/IDN with NFA

We provide another solution to Unicode attacks on both character and word level by using NFA. In this method we evaluate characters similarity with UC-SimList, and manually generate a word-word semantic similarity list at registration time.

We use NFA as the tool to construct potential phishing IRI/IDN patterns. We construct NFA with the keywords of IRIs that we need to protect. If an IRI/IDN is suspected (cf. the overall anti-phishing strategy in [48]), we use the NFA to check its acceptance. If it is accepted and not in protected IRI list then we report it as a phishing IRI. Otherwise, it is normal. For practical utilization of the method, we convert NFA to Regular Expression (RE) to represent the potential phishing IRI/IDN patterns. As NFA is equivalent to RE, the NFA is converted to RE to ease the utilization of this method to IRI/IDN based phishing detection platforms. We built the phishing pattern generator, which could be used to generate regular expressions of phishing patterns. We also propose a framework to build such anti-phishing systems.

5.7.1 Modeling the IRI/IDN based Phishing Patterns with NFA

We use NFA to model the potential IRI/IDN based phishing obfuscation patterns. In general, the modeling process is to convert the IRI/IDN list which we need to protect (e.g., login page IRIs of banks) into an NFA. It can be done by the following procedure, (1) Manually construct an NFA on the semantics level; (2) Replace each nonempty transition in the NFA with parallel-connected similar characters of this letter; (3) Replace each empty transition in the NFA with parallel-connected symbols that are valid in IRI/IDN. This final NFA is the one used for phishing IRI/IDN detection.

5.7.1.1 Construct NFA on the Semantics Level

Suppose we have a list of IRIs L_{IRI} that 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/IDN, 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/IDN based phishing obfuscation:

a. Pronunciation-based replacement: A phisher may replace some part of the keywords of real IRI/IDN to other string without changing the pronunciation. A phisher can change Chinese word to its Pinyin, Japanese to corresponding Romaji, and English to other common used form, as shown in Figure 29(a). All these cases are pronunciation based.
b. Abbreviation-based replacement: A phisher may use the abbreviations of the keywords of real IRI/IDN to other strings for obfuscation. A phisher may change the keywords to their abbreviations, or reversely by replacing abbreviations to full names, as shown in Figure 29(b).
c. Translation-based replacement: A phisher may translate some keywords in a real IRI/IDN to other languages for obfuscation. There are many mutations of each part of an IRI/IDN through translations-based replacement, as shown in Figure 29(c). There may be other semantics-based keywords replacement methods, which could be proposed by people in further study.

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

(a)

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

(b)

Language	Original	Replacement1	Replacement2	Replacement3
English	City Bank	City 银行	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

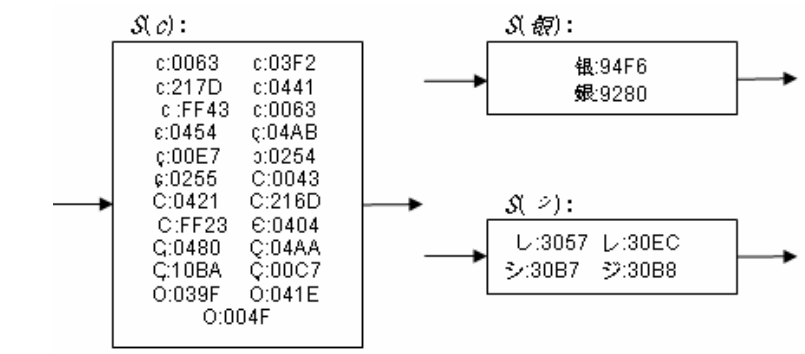
(c)

Figure 29. Demonstration to the three most potential types of semantic expansion of a keyword. (a) IRI/IDN based phishing obfuscation using the same pronunciations; (b) IRI/IDN based phishing obfuscation using abbreviation; (c) IRI/IDN based phishing obfuscation using translation[1].

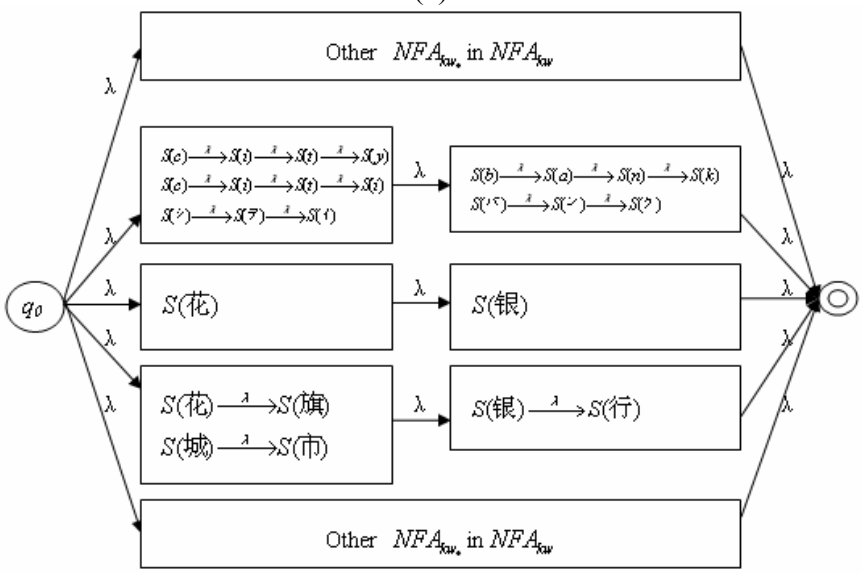
[1] "银行", "銀行" and "バンク" stands for "Bank", "シティ" stands for "Citi", "中国" stands for "China", and "東京" stands for "Tokyo".

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	Ȼ:0404	Ç:0480	Ç:04AA	Ç:10BA
	ɕ:0454	ç:04AB	ç:00E7	Ç:00C7	O:039F	O:041E	O:004F
	ɔ:0254	ȼ:0255		O :FF2F			
i:0069	i:0069	i:2170	i :FF49	I:0049	I:217C	I :FF29	I:0406
	i:0456	¡:00A1	í:1F77	I :FF4C	I:2160	I:006C	I:01C0
	í:03AF	ì:1F31	ì:1F30	I:04C0	I:0399	I:0049	ǀ:05C0
	I:006C	I:01C0	I:2160	l:0196	í:1F77	ì:1F30	í:1F31
	I:0049	I :FF4C	I:0406	ǀ:0140	Ị:1E37	Ị:1ECA	! :FF01
	I:04C0	ị:1ECB	I :FF29	¡:00A1	!:01C3	!:0021	ſ:017F
	I:0399	I:217C	ǀ:05C0	i:0456	i:2170	i:0069	ị:1ECB
	í:00ED	! :FF01	!:0021	i :FF49	í:03AF	Ị:013C	İ:0130
	!:01C3			í:00ED			
t:0074	t:0074	t :FF54	ţ:1E6D	T:0054	T:0422	T:03A4	T :FF34
	ŧ:0167	ţ:0163	ť:01AD	Ţ:1E6C	T:01AC	Ţ:0162	Ţ:04AC
	ṯ:1E6F	ƫ:01AB	f :FF46	T̲:1E6E			
	f:0066			Ŧ:0166	T̲:1E70		
y:0079	y:0079	y:0443	y :FF59	Y:0059	Y :FF39	Y:03A5	
	y̲:1EF5	ʏ:01B4		Y:04AE	Ỵ:1EF4	Ɏ:04B0	
bank	…			…			
花:82B1	花:82B1			*			
旗:65D7	旗:65D7			*			
城:57CE	城:57CE	珹:73F9	堿:583F	*			
市:5E02	市:5E02			*			
银:94F6	银:94F6			銀:9280			
行:884C	行:884C	行:FA08		*			
シ:30B7	し:3057	レ:30EC		シ:30B7	ジ:30B8		
テ:30C6	て:3066	で:3067		テ:30C6	〒:4E8D	〒:1195	
イ:30A4	い:3044			イ:30A4			
バ:30D0	ば:3070	ぱ:3071		バ:30D0	パ:30D1		
ン:30F3	ん:3093			ン:30F3			
ク:30AF	く:304F			ク:30AF			

Figure 30. 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 that Form2 is the same with Form1, and "…" denotes the omitted content)



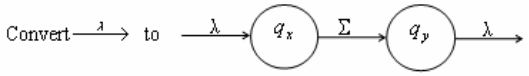
(a)



(b)

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

(c)



(d)

Figure 31. NFA Representation. (a) The NFA representation of $S(c)$, $S(銀)$, and $S(ｼ)$; $NFA_c^{'}$ generated from NFA_{kw}; (c) The complete valid symbol list of RFC3986 and RFC3987; Empty transition replacement (q_x and q_y denotes two states)

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 "$\cup$" 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. 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 Section 4.2 and section 4.3.

5.7.1.2 Replace the Nonempty Transitions

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 get the similar characters of each character of {c,i,t,y,b,a,n,k,花,旗,城,市,银,行,シ,テ,ィ,バ,ン,ク} from UC-SimList0.8. Figure 30 shows the similar character sets of these characters, each element is represented by a

character followed by its hexadecimal code. We denote the NFA formed by similar character set of character *char* as $S(char)$. Figure 31(a) 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 31(b). 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 $NFA_{kw_\bullet}$.

5.7.1.3 Replace the Empty Transitions

Replace each empty transition (i.e. $\xrightarrow{\lambda}$) 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 [8] and RFC3987 [21], as shown in Figure 31(c), 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 31(d). 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.

(
(((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(ン)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))))
)

(a)

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

(b)

Figure 32. Regular Expression Representation. (a) RE of NFA_{c_i} when $i = "www.citybank.com"$; (b) $\Gamma(char)$ (RE of $S(char)$, when $char = "c"$).

5.7.2 Regular Expression Generation

Regular expression (RE) is equivalent to NFA. We convert NFA_c to RE to ease the adaptation approach to the 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 32(a) 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 32(b) shows an example of $\Gamma(char)$ when $char = "c"$ where we use "u"+Unicode to represent a character.

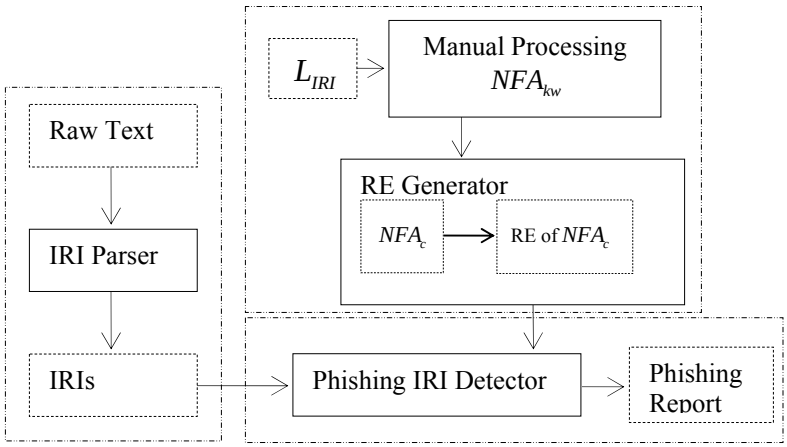
5.7.3 Anti-Phishing Framework and the Phishing IRI/IDN Pattern Generator

We propose an IRI/IDN 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/IDN parsing module and phishing IRI/IDN detection module, as shown in Figure 33(a). 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 [28]), 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 33(b) shows an NFA_{kw} definition with only one IRI/IDN, "*www.citibank.com*". Figure 33(c) demonstrates the generated RE, where delimiter-like character set Σ is defined in Figure 33(d). 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.

In the IRI/IDN 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/IDN Parser to collect all IRIs appearing in the raw text. In the phishing IRI/IDN detection module, the Phishing IRI/IDN 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/IDN can be accepted by the RE and is not an exact IRI/IDN

in the protected IRI/IDN set, it is considered as a potential phishing IRI. Finally, the potential phishing IRIs are reported.



(a)

```
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>
```

(b)

```
"((([C|C| C |C|c|c| c |c|c]Σ{0,2}[I|I|I|I|I| I |I|I|I| I |i|i|i| i ]Σ{0,2}[T| T |T|t| t
|T]Σ{0,2}[Y| Y |Y|Y| y |y|y])|([C|C| C |C|c|c| c |c|c]Σ{0,2}[I|I|I|I|I| I |I|I|I| I |i|i|i|
i ]Σ{0,2}[T| T |T|t| t |T]Σ{0,2}[I|I|I|I|I| I |I|I| I | I |i|i| i |
i ])|([城])|([城]Σ{0,2}[市])|([花]Σ{0,2}[旗])|([し|シ]Σ{0,2}[て|テ]Σ{0,2}[い|
イ ]))Σ{0,2}(([B|B| B |B|b| b ]Σ{0,2}[ A |A|A|A|a|a| a ]Σ{0,2}[ N |N|N|n|
n ]Σ{0,2}[ K |K|K|K|k| k ])|([ 銀 | 银 ])|([ 行 ])|([ 銀 | 银 ]Σ{0,2}[ 行 ])|([ ば |
バ]Σ{0,2}[ん|ン]Σ{0,2}[く|ク])))"
```

(c)

```
Σ="[-| - |-|-| -|- |¯|_|\-|˙| . |·| · |·|.|·|\.|_|¯| _ |¯| ¯ | ~ |~| : |:| : |:| / |/|\?| ? | # |#|
[ |\[|\]| ] |@|@|!|!| ! | $ |\$|&| & |'| ' | ( |\(|\)| ) | * |\*| + |\+| , |,|;|;| ; | = |=]"
```

(d)

Figure 33. Anti-Phishing System and Regular Expression based Pattern Generator. (a) Framework for IRI/IDN based Phishing Obfuscation Detection System; (b) An NFA_{kw} definition example of REGAP; (c) RE generated by REGAP from the NFA_{kw} definition in Figure 33(b); (d) Delimiter-like character set Σ.

5.8 Conclusion and Future Works

We identify a severe problem threatening computer security and privacy, Unicode attack. The problem is caused by the phenomenon that computer screens are hiding the real character representation from users. To detect Unicode attack, we propose a methodology, which is a good guideline for building Unicode attack detection systems. Following this methodology, we provide a demo implementation as a paradigm study to detect Unicode attacks using vision and semantic based algorithms. Our experiments show that both the classification effectiveness and time efficiency of the proposed method are satisfactory. The threshold for VSED could reach an optimum around 0.12. We also implemented VSMKP, where we showed the threshold for char-char similarity threshold of 0.80 could make it an empirical optimization of VSKMP. We also built up IRI/IDN SecuChecker using the two algorithms to perform IRI/IDN based phishing and fake web identity detection. The demo of Web Link Illustrator [25] sampled a possible improvement for web browsers.

The basic cornerstone to detect/discover *Unicode attack* is to construct the UC-SimList. We constructed the prototype UC-SimList_s of English, Chinese, and Japanese.

We use pixel-overlapping method to calculate the UC-SimList_v. This method is fast but not quite effective. Hence, we proposed a better symbol similarity assessment measure, KDE, to constructed UC-SimList_v. However KDE is slow. Therefore, we would like to propose to use Quadratic Spline similarity method as discussed in Section 5.3.4. We put all of the lists available for researchers on the Web [5]. We are also expecting other better methods can be proposed to calculate UC-SimList_v. The construction of UC-SimList_s is not easy, because (1) there are character sets of many more languages in UCS other

than English, Chinese, and Japanese, (2) more semantic similarity relationships need to be considered as well, for instance, we should consider "一" and "壹" semantically similar. We consider it as a valuable work that should be carried out in future works.

On the word semantic similarity level, we used information based measure to calculate the word-word semantic similarity (WWSS) List. We used the British National Corpus based Word Frequency of Written and Spoken English to perform the WWSS List calculation. The contribution in this section mainly focuses on the WWSS List generation which is a very important Unicode attack detection component. The WWSS List generation is really time-consuming but we only need to generate it once. WWSS List generation is still an on-going study. As a matter of fact, we only generated noun-noun semantic similarity. Future studies are still required to complement the list.

VSED and VSKMP are not the only algorithms we can use. We propose a more effective detection approach to Unicode attack problems using NFA, and address its construction detail. We built up a potential phishing IRI/IDN 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.

Chapter 6

Human Computer Interaction Enforcement

There are many techniques for anti-phishing. However, focusing on solving phishing problems at the User Interface (UI) level aspect has been highly recommended. Due to perceived problems with various anti-phishing toolbars, Ross et al proposed *PWDHash* [61], Dhamija et al proposed *Dynamic Security Skin* [20], and Wu et al proposed *Web Wallet* [73] to improve the user interface design. However these methods cannot solve the Screenjacking problem.

In this chapter, we propose several methods that may help solve Screenjacking problems. We use *Web Wallet* as a platform, and propose to use Genuine Skin, Application Trace, Spring-Loaded Button, and Interactive Image Filter as candidate methods for anti-Screenjacking. We also propose to use the merged approach of these methods to study the final effectiveness of an integrated method. All of these proposed methods use visual indicators. However, Wu et al [74] argued that visual indicators are not effective to stop users from inputting their private information into faked websites, because users do not always bother to look at these visual indicators. Users' goals are to input their information into web pages and finish their job. Hence, it is unfortunate that we cannot find a none-indicator based anti-Screenjacking method without changing traditional HCI mechanisms.

We consider passwords as the most important privacy that users should protect. In this chapter, we try to change the HCI mechanism and propose two novel password designs:

Semi-Random Password, and Context Sensitive Password. These two novel ideas can change users' behavior by changing their goals. Semi-Random Password is a technique to protect users' passwords by randomly typing noise keystrokes. Context Sensitive Password is a method to guide users to embed the procedure of reading a character (or image) string into their workflow of them, such that users have to consider personalized visual indicators before knowing the proper way to input their real passwords.

However, we did not have the resource to carry out the further study on usability, memorobility, and security of above anti-phishing methods. We expect to carry out more study on these when we have a chance. Hence, we still want to address them as a chapter.

6.1 Anti-Screenjacking

In this investigation, we use Web Wallet as the platform to carry out new UI designs and propose them for user study.

6.1.1 Proposed UI Designs

6.1.1.1 Referential UI for Control Study

We provide referential UI, as shown in Figure 65 (Appendix E), to make other UI *anti-Screenjacking* designs comparable. The difference from the original *Web Wallet* design in [73] is that we use an isolated window rather than an embed one in IE. Although everywhere on the computer screen can be faked, it is most common for phishers to put faked UI into the client window of web browsers. Hence, we believe it will be better to build *Web Wallet* outside of the web browser, rather than as a part of it.

6.1.1.2 Spring-Loaded Button

Web Wallet requires all security sensitive information to be inputted to web pages through its GUI. However, it is hard to prevent users from exposing their security sensitive information to other websites. Blake Ross et al [61] first propose educating users to press the "*F2*" key as the security key to input passwords. *Web Wallet* also uses the "*F2*" key as a security key to open the GUI and requires users to always remember to press the "*F2*" key before inputting security sensitive information. The "*F2*" key may work well at handling normal web pages. However it does not work for the faked *PWDHash* or *Web Wallet*. If users see *PWDHash* or *Web Wallet*, they may not bother to press the "*F2*" key. Therefore, it is tricky and we certainly do not want to design the *PWDHash of PWDHash* or *Web Wallet of Web Wallet*. Otherwise, it will recursively derive new problems. The "*F2*" key in *PWDHash* and *Web Wallet* only indicates "*finishing one task*". However, we really need the actions to indicate both "*starting one task*" and "*finishing one task*". Hence, we propose Spring-Loaded Button, which uses "*F2-key-holding-on*" to indicate the "*to start*", and "*F2-key-releasing*" to indicate "*finishing*". When we are holding the "*F2*" key, only the real *Web Wallet* can accept keystrokes, and the remaining area will be disabled. The disabled area will be set to gray, as shown in Figure 66 (Appendix E). However, the spring-loaded button is not necessarily to be "*F2*" or a key on the keyboard. If so, it could be difficult to hold spring-loaded button to input a password that you are used to input with two hands. We could design the spring-loaded button as an accessory device, such as a pedal, to make it easier.

6.1.1.3 Application Trace

Web browsers use small lock icons to indicate *Secure Socket Layer* (*SSL*) [57] connections. *Dynamic Security Skin* [20] also tries to indicate the genuineness of web pages with hash visualization. However, if the lock icon or other visual indicator is faked, we may have no idea whether our personal information is still secure. Ye et al [76] proposed the *Trusted Path* to solve such problems. However, we argue that these visual indicator based methods do not work under the assumption in Section 2.4.1, unless the genuineness of the visual indicators can be guaranteed. We redesign a visual indicator, *Application Trace*, to solve this problem. Through investigation, we found that we have to place the indicators somewhere on the screen which can be assured to be secure (we cannot put the visual indicators on taskbars, because the taskbar can also be faked). Hence, we propose to put the visual indicator at a fixed place in the physical area of the computer screen, e.g. at one specific corner or one specific side of the screens. Figure 67 (Appendix E) shows a prototype that the visual indicator is put at the top side of the screen. We use animate hash visualization [18] [58] as the visual indicator. We also add decoration to the *Web Wallet* with the same style as the visual indicator. The decoration in the Web Wallet GUI should synchronize with the visual indicator. Users should know that the visual indicator always appears at the top side of the screen. If the decoration in the Web Wallet GUI cannot synchronize with the visual indicator at the side of the screen then users can detect it. We do not think visual indicator is a good way to strengthen the HCI security since Wu et al [74] demostrated that user factor is very important for the security of visual indicator. However, we believe visual indicator is the only way to help users notice the genuineness of the visual contents in the PC screen without using other

output devices. Normal visual indicators can be faked. The “Application Trace” is designed to solve this problem.

6.1.1.4 Genuine Skin

One way to detect faked *Web Wallet* is to use image recognition techniques to detect its visual similarity to the real *Web Wallet* GUI on the screen. However, the detection processes could be complicated. There are difficult problems need to be solved before we can use this method, i.e. *How do we find the position of the faked GUI? What if malicious people adjust some items in the GUI to differentiate it from the real one? What if malicious people only allow a part of the GUI to appear on the screen while users still think it is similar to the real one?*

(1) Genuine Skin Pattern Example 1

(2) Genuine Skin Pattern Example 2

Figure 34 Genuine Skin Patter Examples

Genuine Skin is a method we would like to propose to mitigate the difficulty of automating faked GUI detections. We propose these guidelines for the Genuine Skin:

Genuine Skin is like a trade mark. It is identical for a certain company, and a series of products of this company can use the same Genuine Skin. It is illegal to use images identical or similar to the Genuine Skins of any companies without authentication.

There is one Anti-Screenjacking engine running in the back ground of the client computer and it watches if patterns similar to the Genuine Skin appear on the unauthorized area of the computer screen. If a similar pattern is found, the suspected area will be circled to give alert, as shown in Figure 69 (Appendix E). If the faked Web Wallet has no Genuine Skin, then users will not tend to believe it is real, as shown in Figure 70 (Appendix E).

The design of Genuine Skin should be easy to be recognized by both human eyes and computers. We need to train the anti-Screenjacking engine to simulate the human classification curve, which can minimize the total false classification number.

Figure 34 shows two examples of Genuine Skin patterns for the Web Wallet. Figure 68 (Appendix E) shows the paper prototypes of the Web Wallet using the two Genuine Skin patterns.

6.1.1.5 Merged Solution

Anti-Phishing is so difficult that no anti-phishing method can shield users from 100% of phishing attacks. However, we believe that if we incorporate more anti-phishing techniques into a single tool, then this tool could have better security effectiveness. Hence, we may integrate all the above techniques together, and carry out a user study to learn the effectiveness.

6.1.1.6 Interactive Image Filter (User Challenge)

Another interesting approach we may want to test is the Interactive Image Filter (or User Challenge). Users can select personalized images (e.g., pictures that users are familiar with) as backgrounds for the Web Wallet. When one user uses the Web Wallet for the first time, he will be required to select the personalized image and image filter. An image filter is an algorithm that can apply to a given image to generate another image with a special visual effect. Figure 72 (Appendix E) shows the example of a statue as a background image, and a "whirl like effect" algorithm as an image filter. When a user is typing into the Web Wallet, the predefined image filer will be continuously applied to the background image. Hopefully, the user will feel something wrong when they see the faked image and image filter. A phAsher must guess what kind of image and image filter the user is using. According to the assumption of Web Wallet [73], all security sensitive information should be inputted through Web Wallet, such that each time the user inputs into Web Wallet, challenges will be in progress. The reason that we would like to propose the User Challenge method is we would like to see the effectiveness of it to help users pay attention to the visual indicators when users are interacting with the visual indicators.

6.1.2 User Study Design

As introduced in Section 6.1, we have 6 anti-Screenjacking methods to be considered in a user study. We may recruit 10 users to test each type of anti-Screenjacking method. We may also design 40 "yes/no" questions for each user to answer. Each question will ask the user whether the given Web Wallet is trustable or not. In the 40 questions, we may include 10~15 faked Web Wallet cases.

6.2 Secure Passwords

Passwords may be the most security critical part of private information. We propose Semi-Random Password to fight against key logger attacks, and Context Sensitive Password to fight against phishing attacks. Context Sensitive Password is also an excellent anti-Screenjacking method.

6.2.1 Semi-Random Password

A traditional password is a string that users ssupply to (or get from) a registration system. It is very straightforward. In that, it uses a character string to verify whether you are the correct secrete holder. When users want to log into the system again, they should input the correct password into the "password field" to verify their identities. A key logger can monitor the keystrokes to a computer. Once the malicious people get the key logs, they can easily find the password(s). Sometimes, we also need to use public computers or other people's computers to log into some systems (e.g. email, on-line banking systems). Here, one important and interesting problem we want to ask is, "Can we devise a new password mechanism to improve the security of the passwords, even if a user is using a machine with a key logger?" Industry is moving quickly to build up many anti-spyware tools. However, we consider this to be a passive reaction, which can only detect the spyware after we know about it (like anti-virus applications). We would like it to be possible to take active action. We propose one such method, Semi-Random Password.

In this design, we insert random characters into the original password strings to form a longer string. A single semi-random password cannot be used more than once in a period of time. The semi-random password should also be sufficiently different from recently

inputted semi-random passwords, such that attackers will have difficulty knowing exactly what the password is, even if they know the semi-random password. Since attackers may use the semi-random password to carry out brute force attack, the login systems should only allow a limited number of failed attempts (they can freeze an account for a while if too many failed attempts are made to access it).

6.2.1.1 Semi-Random Password Design

6.2.1.1.1 Create Semi-Random Password

Suppose we have the original password string P_O as shown in Eq. 31.

$$P_O = C_{O,1}C_{O,2}.....C_{O,(|P_O|-1)}C_{O,|P_O|} \quad 31)$$

where $C_{O,i}$ is the i^{th} character in password P_O, and $|P_O|$ is the length of P_O. The semi-random password of P_O can be represented as P_S, as shown in Eq. 32.

$$\begin{cases} P_S = C*C_{O,1}C*C_{O,2}C*..... \\ \qquad C*C_{O,(|P_O|-1)}C*C_{O,|P_O|}C* \\ |P_S| = N \end{cases} \quad 32)$$

where, $C*$ is a regular expression, which represents "*a random character string with any length (including 0)*". However, another constrain is that the total length of the string P_S should be exactly N.

Users can arbitrarily separate the original password P_O into several sections, and insert randomly inputted strings as long as the total length is N. Figure 35 shows the input box designed for P_S. It should be fully filled to satisfy the length constraint of P_S.

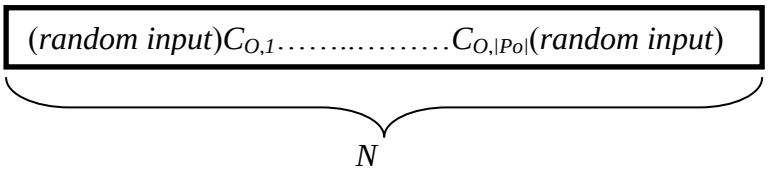


Figure 35. Semi-Random Password Input Box

The password verification on the server side should be designed to reject the P_S which is similar to a password that has been used to log into the system recently. Therefore, even if attackers get the P_S, it is not easy to guess P_O. The system will be inactivated when the number of failed attempts reaches a certain number, such that we can reduce the possibility of successful brute force attacks.

There are $C_N^{|P_O|}M^{(N-|P_O|)}$ possible P_S s, where M is the number of possible choices of C. $C_N^{|P_O|}M^{(N-|P_O|)}$ is a very large number. Hence, we consider the probability of users to generating two similar passwords in a short period of time to be very small.

6.2.1.1.2 Difference Sufficiency

We need to evaluate the differences in the current P_S to the recently used ones to avoid attackers stealing the password with key loggers and changing a little bit of the password to make it different. Hence, the currently used P_S should be sufficiently different from the recently inputted ones. We consider edit distance as a good assessment method, which exactly represents the semantic of the least number of revisions you need to make to differentiate the current password from the recently inputted ones.

If we select *n* as the minimum steps of revision from one recently inputted P_S, then attackers have to make *n* steps of revision to overcome the edit distances assessment.

$$p = \prod_{k=0}^{n-1} \frac{N - |P_O| - k}{N - k} \qquad 33)$$

The probability of a successful attack for each try will be less than or equal to p. It is a very small number if n is big enough.

6.2.1.1.3 Verification

We use a Regular Expression (RE) as shown in Eq. 32 at the server side to check the validity of the password. If P_S can be accepted by the corresponding RE, then the system passes the verification, otherwise it responds to users with retry/inactive.

6.2.1.2 Example

We provide an example of Semi-Random Password. Suppose we have a P_O (as shown in Eq. 34), P_S (as shown in Eq. 35), $|P_O|$=6, and N=15. Figure 36a shows the plain text of the users' input. Figure 36b shows how P_S really appears in the input box.

$$P_O = \mathbf{abc123} \qquad 34)$$

$$P_S = z\mathbf{a}b\mathbf{bc}h@ui\mathbf{123}n\&I \qquad 35)$$

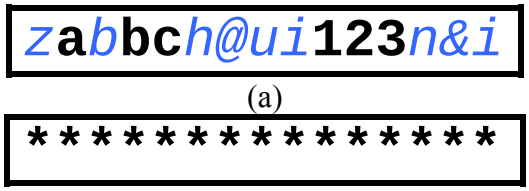


Figure 36. Semi-Random Password Demonstration

M = (26×2 letters) + (10 numbers) + (32 symbols) + (1 space) = 52 + 10 + 32 + 1 = 95. Hence, there are $C_N^{|P_O|} M^{(N-|P_O|)} = C_{15}^{6} 95^{15-6}$ = 3,154,398,295,671,669,921,875 possible choices

of P_S that can be used as semi-random password candidates. We also calculate p in Eq. 33. When n=1, 2, 3, 4, 5, 6, p=0.6, 0.32, 0.149, 0.06, 0.02, 0.005. We can see p decreases exponentially when n increases. Preliminary experiments show that n=6 is quite easy to reach, such that there is a very limited probability for attackers to break into the systems. We verify this hypothesis in a user study in Section 6.2.1.3.

6.2.1.3 Discussion

One major concern with semi-random passwords' security is the time attack. In the time attack, the assumption is that when a user types in a semi-random password, they are likely to pause between the password parts and the random parts, while he can type other parts (random parts and password parts) much faster. We would like to demonstrate how secure the semi-random password could be under time attack.

We used 15 as the fixed semi-random password length and 6 as the length of the original password. Greg Little, a graduate student in MIT, inputted 20 semi-random passwords. Figure 37 shows the key log. Characters in red cells are the ones in the original password, "p3k&vw". Characters in the yellow cells are the random ones. Each row is a semi-random password. The number following each typed character is the typing time-interval between the previous keystroke and the current one. LeastED is the least edit distance from the current semi-random password to all of the previously inputted semi-random passwords. We can see that it was very easy to achieve LeastED>=6. Hence, 15 as a total length is good enough to keep the system secure.

There are four types of transitions in semi-random passwords. We use "O" to represent an original password character, and "R" to represent a random character, and then the four types of transitions are O→O, O→R, R→O, and R→R. Under the attackers'

hypothesis, users will stop a little bit longer at O→R and R→O. Hence, O→O and R→R should be much faster than R->O and O->R. We calculated the average key stroke intervals for each type of transitions to be O→O:666, O→R:646, R→O:421, and R→R:327 (unit: millisecond). Obviously, O→O is slower than O→R, and R→R is faster than any others.

Suppose we use 500 (milliseconds) to classify R→R and O→O. There are 14 time intervals in the 15 keystrokes. Figure 38 shows the time distribution for each of the transitions. We can see O→O and R→R are the types that are distinguished from each other the most. Hence, in the worst case, we have 5 O→O, 1 O→R, and 8 R→R transitions in the password. The probability of making a correct guess of 5 OO is (38/116)^5=0.00377 (there are 116 intervals that are less than 500 milliseconds, and 38 of them are O→O). The probability of making a correct guess of 8 R→R is (154/272)^8=0.01056 (there are 272 intervals that are less than 500 milliseconds, and 154 of them are R→R). Therefore, the success probability for each attack is 0.00377*0.01056=1e-5 without counting the O→R transitions. Suppose we allow trying k times a day, then the probability to hack in is 1-(1-(1e-5))^k in the most lucky case for the attackers. When k=10, the probability is 1e-4.

Therefore, we conclude that the time attack does not work for semi-random password. The intuitive reasons could be:

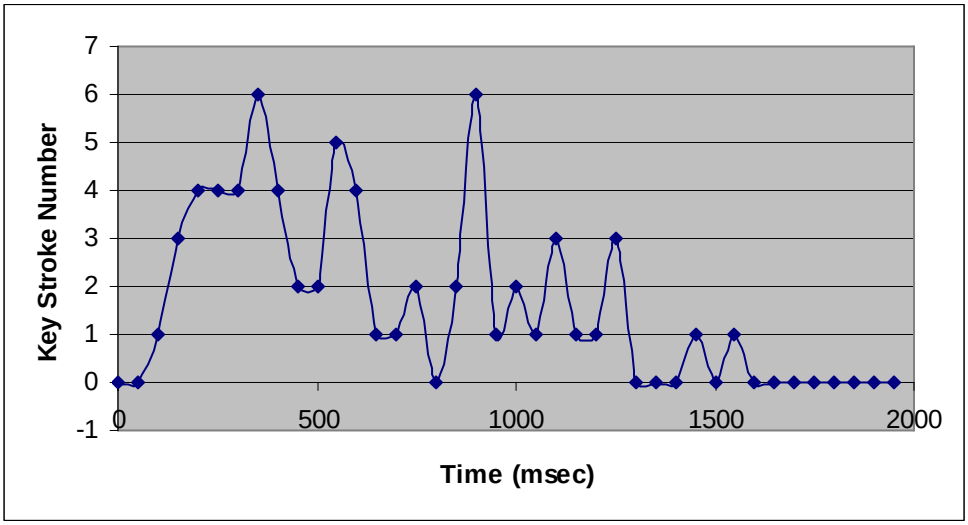
1. Different letters have different difficulties to be inputted.
2. The same user will have different behaviors when the previous input changes. For example, “H” is more difficult to type than “h”, but “H” following “G” will be

easier to type than “h” following “G”, because you have to release “Shift” key first.

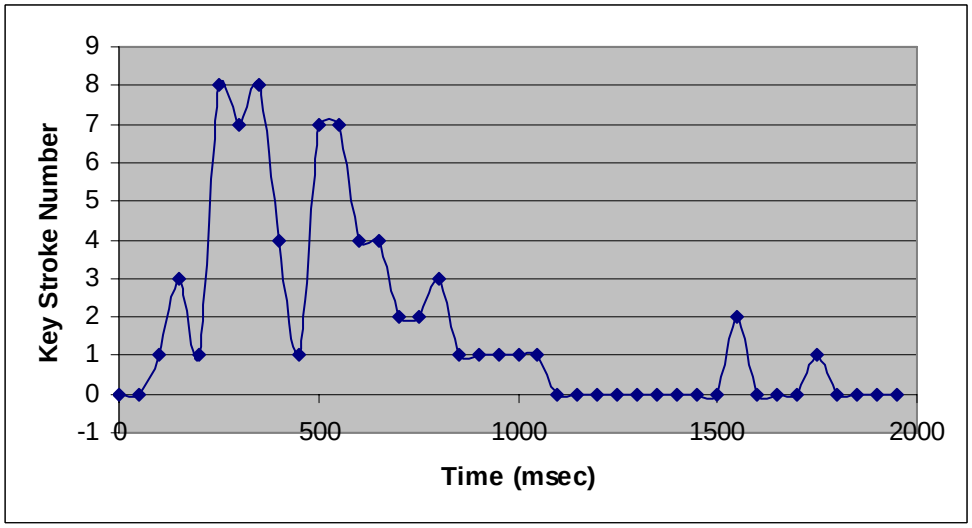
3. Different people have different hand behavior pattern when inputting, e.g. typically, there are many “two finger typing” computer users.
4. Users are using many types of keyboards. The keyboards’ properties could also affect the transition patters.

Input	Time Interval	Input	Time Interval	Input	Time Interval	Input	Time Interval	Input	Time Interval	Input	Time Interval	Input	Time Interval	Input	Time Interval	Input	Time Interval	Input	Time Interval	Input	Time Interval	Input	Time Interval	Input	Time Interval	Input	Time Interval	Input	Time Interval	Input	Time Interval	Input	Time Interval	LeastED	Correct ?
p	0	3	190	k	311	&	911	v	401	w	280	h	952	h	1672	h	230	h	160	h	181	h	160	h	140	h	150	h	141	15	T
y	0	o	200	p	491	3	1472	a	541	k	250	&	1262	a	1042	a	330	v	341	w	1282	k	290	a	210	l	161	s	140	14	T
a	0	s	160	p	691	3	931	k	191	s	620	d	181	&	591	v	2012	a	571	l	180	s	141	l	200	w	160	l	291	11	T
d	0	k	160	p	1002	3	1001	s	391	k	140	k	380	&	1242	a	861	k	541	s	120	v	461	w	391	l	540	s	101	8	T
d	0	f	230	j	171	p	340	3	1092	a	330	s	160	k	351	s	1062	d	140	f	100	&	300	l	691	a	161	v	1191	11	T
a	0	k	160	s	161	p	180	3	1142	a	360	s	160	k	351		681	a	420	s	201	&	591	a	400	s	140	v	361	9	T
k	0	a	210	p	731	3	1162	l	300	a	211	k	190	&	901	a	341	v	340	w	501	l	270	a	111	s	160	k	120	7	T
a	0	s	270	p	271	3	851	k	120	&	581	k	921	a	30	s	240	v	391	k	561	a	180	w	401	k	561	a	280	9	T
a	0	k	160	s	241	j	190	w	651	e	420	p	511	3	721	k	421	c	220	k	2494	&	631	v	300	w	210	a	581	10	T
p	0	a	190	s	220	3	331	D	1552	D	190	k	1452		611	&	361	a	560	s	291	l	80	V	1212	v	300	w	581	10	T
k	0	k	1092	a	160	p	160	3	772	a	260	f	300	k	211	&	591	a	340	s	130	v	311	w	240	l	361	k	180	6	T
a	0	k	130	s	171	p	200	3	491	a	290	s	120	d	151	k	260	&	691	a	400	v	291	w	561	l	290	a	90	6	T
a	0	p	361	3	761	a	611	s	210	d	130	k	170		832	k	220	a	250	&	441	v	641	w	250	k	251	j	220	7	T
a	0	j	191	p	250	3	361	a	300	s	230	k	201	&	941	l	571	a	120	l	260	v	241	H	1592	w	401	H	530	7	T
p	0	a	411	k	150	3	321	H	771	L	781	L	160	K	511	k	841	&	1122	v	600	w	201	a	320	l	90	k	221	9	T
p	0	a	150	s	140	d	130	p	331	a	641	k	140	3	330	a	671	k	441	&	1111	a	531	k	130	v	251	w	250	7	T
a	0	k	170	s	151	p	190	3	941	k	922	&	1292	a	380	k	130	v	311	w	530	w	151	l	160	a	200	k	160	7	T
a	0	k	150	a	191	p	120	3	350	k	391	a	501	a	300	&	350	a	561	k	161	j	250	l	140	v	140	w	361	8	T
a	0	k	70	j	170	p	240	3	951	l	451	a	190	k	771	a	351	k	250	s	160	d	171	f	110	&	260	v	571	8	T
j	0	l	421	p	981	a	340	k	151	3	1512	a	290	k	291	&	1001	k	531	a	180	j	160	v	261	w	260	A	381	7	T

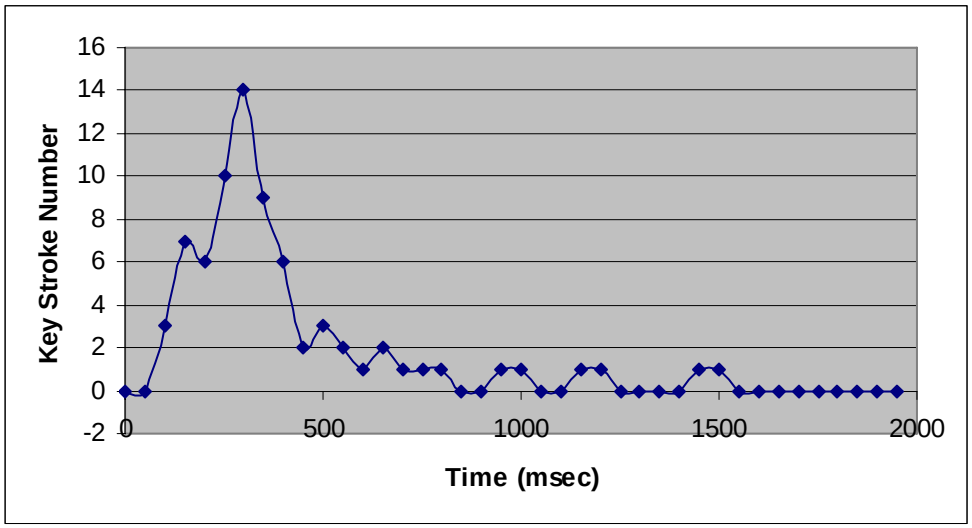
Figure 37. Twenty Semi-Random Passwords for “p3k&vw”. Original Password is in Red and Random Part is in Yellow.


Key Stroke Number
7
6
5
4
3
2
1
0
-1
0
500
1000
1500
2000
Time (msec)

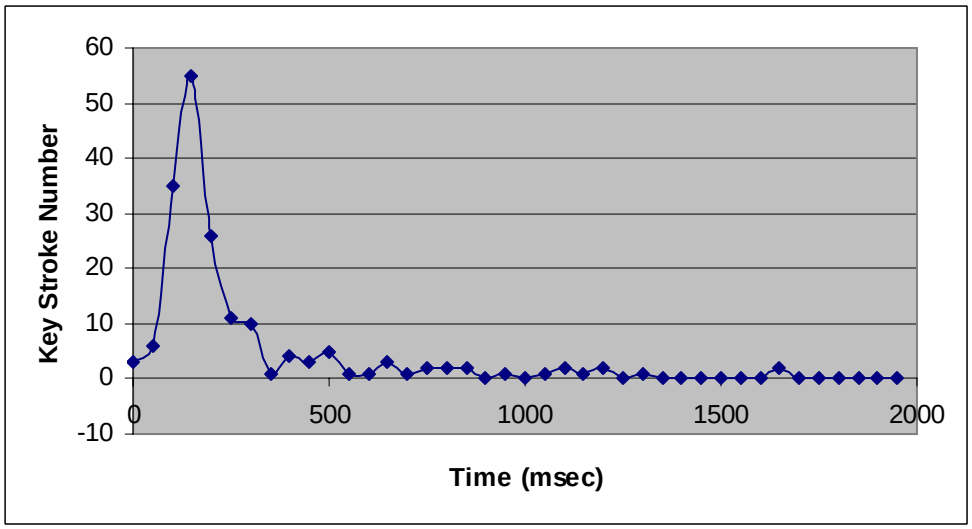
(a) O→O


Key Stroke Number
9
8
7
6
5
4
3
2
1
0
-1
0
500
1000
1500
2000
Time (msec)

(b) O→R



(c) R→O



(D) R→R

Figure 38. Time Distribution for Semi-Random Password Transitions

6.2.2 Context Sensitive Password

In this section, we propose a secure and easy-to-use password approach that keeps users away from password related phishing and Screenjacking attacks. We invent a way to guide users to generate correct passwords with the help of a Context-Sensitive

PasswordBox. Under such protection, attackers have low probability of breaking into the systems, even if they get the passwords. This method does not need any additional work on the password verification modules because the context can simply become a part of the password that will be saved to the database which manages passwords.

6.2.2.1 CSPass Design

6.2.2.1.1 Generate CSPass

The generation of context sensitive password has two steps. We choose or input one character string as personalized context string, S, as shown in Eq. 36.

$$S = C_{S,1}C_{S,2}.....C_{S,(|S|-1)}C_{S,|S|} \quad 36)$$

where $C_{S,i}$ is the i^{th} character in password S, and $|S|$ is the length of S. Suppose we have the original password, which is gotten from the registration process, to verify our identity in a certain system, and we denote it as P_O, as shown in Eq. 37.

$$P_O = C_{O,1}C_{O,2}.....C_{O,(|P_O|-1)}C_{O,|P_O|} \quad 37)$$

where $C_{O,j}$ is the j^{th} character in password P_O, and $|P_O|$ is the length of P_O. There is an additional constrain that $|S|$ should be longer than $|P_O|$.

The second step is to generate the password that is used by the password verification modules in the server (if you are logging into servers), or client applications (if you are using a client application such as Web Wallet) by combining S and P_O. The password holder should segment P_O into several substrings and replace such substrings into the substrings of S as shown in Figure 39 and denote the generated context sensitive password as P_S. We call the starting position(s) of segmented P_O substrings in P_S as init-

position(s), i.e. i is one of the init-positions in P_S if substring, $C_{O,j}C_{O,j+1}\cdots$, starts to replace in S that start from $C_{S,i}$, as shown in Figure 39.

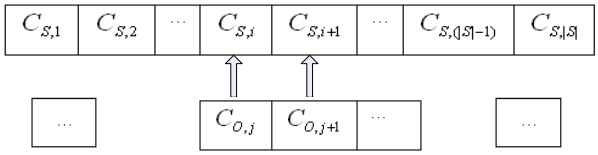


Figure 39. Generate Context Sensitive Password

Users should remember how they have sectioned P_O and all of the init-positions. P_S is saved in the server side database (or client application) as a traditional password.

6.2.2.1.2 Use CSPass

If Context Sensitive Password is used for identity verification in a server, then the server should provide the correct S in the password box according to the username. If the server can recognize a users' computer or web browser through certain identities, then they can provide a corresponding S according to such identities. We can also a use hashed user name to verify the users' identity. However, they may not be feasible in cases where different people are using the same machine.

If Context Sensitive Password is used on a client application (such as a client side banking system or Web Wallet), then it is proper to provide users the S (es) according to the user account(s) or user name(s).

Users should find out the init-position(s) manually and replace corresponding substring(s) in S with the ones from P_O. If S is a meaningful string, then it is quite easy to remember the init-positions.

Suppose the number of the segments is N, then the probability for attackers to successfully guess the correct position is $1/C_{|S|}^{N}$, so when $\left\| S \right| /2-N\right|$ is smaller, it is

harder to attack the password. However, if N is larger than 3, we could get bad usability for normal users.

6.2.2.1.3 Verification

The verification is very simple. We compare P_S to the corresponding password for the account/username you are using.

6.2.2.2 Example

6.2.2.2.1 Textual Version

Suppose we have S as shown in Eq. 38 and P_O as shown in Eq. 39. We select 2 (at "O"), 6 (at the first "X"), and 10 (at the second "X") as init-positions in S. P_O is segmented into 3 parts "aYF", "mW", and "RcM". Figure 40a shows the initial Context Sensitive PasswordBox after you input the user name, Figure 40b shows the plain text password in PasswordBox that will be verified, and Figure 40c shows the semi-hidden password mode, which would actually be displaying in the PasswordBox.

We can also use only one init-position, 4 (at the first "T"), and we can not segment P_O to improve the usability, as shown in Figure 40d.

$$S=\text{CONTEXTEXAMPLE} \tag{38}$$

$$P_O=\text{aYFmWRcM} \tag{39}$$

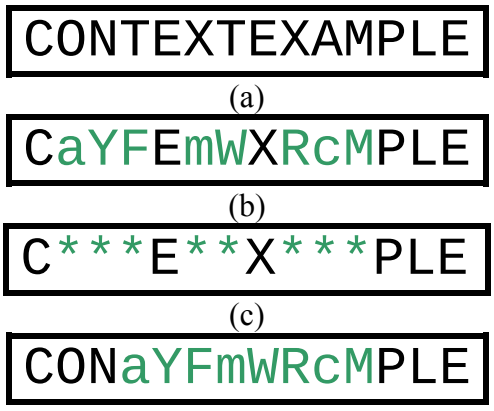


Figure 40. Context Sensitive Password

6.2.2.2.2 Icon Version

To further improve the usability, we can use icons to map characters in S. There are totally (26×2 letters) + (10 numbers) + (32 symbols) + (1 space) = 95 icons to use. Figure 41 shows the mapping samples from icon to characters (the icons are from[3]. The complete table should have 95 of them. We replace characters in S of Eq. 38 with icons, thus we have the initial mode, plain text mode, and hidden mode of the PasswordBox, as shown in Figure 42 a, b, and c respectively. They are corresponding to the example in Figure 40. Obviously, the user only needs to find the first "Red Hat" icon, which is at the upper-middle position and start to input their password.

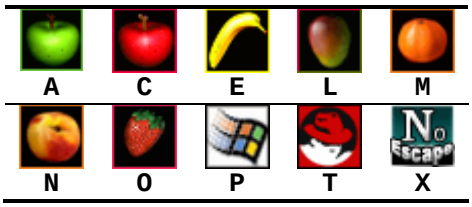


Figure 41. Icon to Character Mapping Samples

(a)

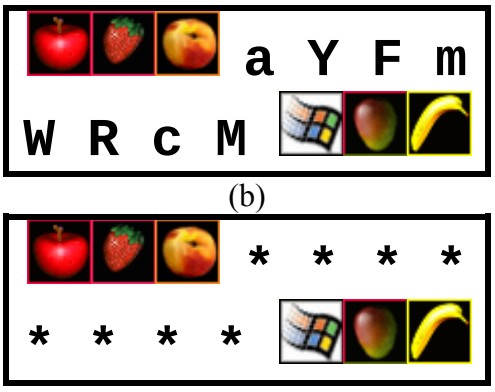


Figure 42. Icon Version Context Sensitive Password

6.2.2.3 Discussion

We can use CSPass at server side or client side. If CSPass is used at server side, it can help improve the security of current password mechanisms without requiring users to install any plug-ins. The correct context strings should be provided to users while they input the correct user names. However, this method is not immune to "man-in-the-middle" attacks for the server side version of CSPass. If CSPass is used at the client side, the "man-in-the-middle" attack can be solved. However, here are two technologies that we may expect to compare with, PWDHash [61] and Graphical Password [70].

6.2.2.3.1 Comparison with PWDHash

The client side version is superior to PWDHash, the famous bi-partial verification password mechanism. We abstract the mechanisms of the two methods and discuss the differences.

CSPass:

Client provides its identity (such as a user name, we can also use PWDHash here to submit user names) to the server.

Server verifies its legitimacy by displaying the correct Context String according to the client identity.

User input Original Password into the context string.

Server verifies the correctness of CSPass (NOT Original Password).

**note: attacker's goal is to get the CSPass, not Original Password, because they still cannot log into your system with only the Original Password, unless they crack the username.*

PWDHash:

Client provides both username and hashed password

Server verifies if the username and hashed password is correct.

**note: attacker's goal is to get the Original Password; they can login with the Original Password.*

There are obvious advantages of CSPass over PWDHash. CSPass submits username and password separately to verify server's legitimacy, while PWDHash does not; CSPass forces users to read the indicators, while PWDHash does not. Some users may ignore the alerts raised by PWDHash; CSPass uses a hashing algorithm in user's brain (if you consider the generation of CSPass as a kind of hashing), while PWDHash uses a standard hashing algorithm. Although the PWDHash algorithm is more difficult to crack, such algorithm is easy to find (just download the same version of PWDHash), unless PWDHash use different hashing algorithms for all plug-ins (but this seems very hard). Although the authentication procedure of CSPass is more complicated than PWDHash, it is still hard to say which one can provide a better user experience. Therefore we can say that CSPass is better than simply using PWDHash.

6.2.2.3.2 Comparison with Graphical Password

Graphical Password [70] is a password authentication method involving clicking particular spots in an image instead of inputting alphabetical password. For example, a user needs to select a personalized picture and click a sequence of objects in the picture with the mouse during registration, and select the same personalized picture and repeat to click the sequence of objects to login next time. This method forces users to look at personalized images to click on. It solved the problem of putting the personalized image recognition into the critical path of users' workflow. However it could be tedious to click many times on given images to achieve reasonable security. The interval of clicking actions is much slower that keyboard typing, which causes the graphical password to be much easier to steal by simply peeking at the screen.

6.3 Conclusion

We introduce a new concept, Screenjacking, and try to solve such problems by enforcing human computer interactions. We prove that everywhere on the computer screen is able to be faked. We propose various novel UI designs (Spring-Loaded Button, Application Trace, Genuine Skin, etc.) that could help solve Screenjacking problems.

Passwords are the most critical part of private information. We propose two novel password protection mechanisms: Semi-Random Password, and Context Sensitive Password.

Semi-Random Password is a method that can be used to verify users' personal identity without providing complete information of the private key that users are holding. We discuss the Semi-Random Password design and prove the feasibility and security of this

method through user study. This method is so simple that it could have a good learning curve. Our experiment also showed that users can adapt to this kind system quickly and feel comfortable with it. We would get more specific user study results in the future work. We would also like to provide a Semi-Random Password API which can be used by login systems. This API should also include a control that can be easily used to create login GUIs. We also see that the probability of successful attacks will increase dramatically when attackers get more than 2 semi-random passwords of the same original password. We would also address this problem in future work. One promising solution is to make the original password P_O relatively long, and only use part of it as "P_O" in the method of this chapter to generate P_S.

Context Sensitive Password provides a mechanism to put personalized text/image recognition into the critical path of a users' workflow. There are 3 preconditions for attackers to log into CSPass protected systems: 1. attacker knows users' personalized string; 2. attacker knows users' init-position(s); 3. attacker knows users' original password. The attacker cannot carry out successful attacks if they lack any of the three preconditions.

We demonstrate the textual version and icon version of CSPass. They are essentially equivalent. However the different forms of representation could affect the effectiveness of usability. We would like to carry out related user studies about that. We also consider providing an easy to use programming API package of CSPass as a great interest for our future work.

Chapter 7

Distinguish Originality for Web Identity

In this dissertation, we discuss much about the advanced phishing attack detection methods. However there is another import problem to be solved: websites should be able to prove that they have created the original web pages, such that we can have a basic idea of who is actually phishing whom. We design and implement a real system to provide strong evidence to original authors of websites to prove their originality.

7.1 Motivation and Background

Before the anti-phishing approach, we need to know who are the ones that should be protected, i.e., we want be sure we are protecting legitimate web pages, and not phishing sites. Although CitiBank is legitimate company, it is also possible to copy other people's webpages. We are not expected to dogmatically decide that CitiBank is always the victim. Surely CitiBank is famous, and we are eager to intuitively decide it is the victim. However, if it is "XYZTrust", which is not so famous, it will not be easy to decide say it is the victim. The fact is that there are more and more web pages (web sites) require to be protected. If hundreds of them are created a day, the automation of originality protection turns out to be very important. Hence, we would like to ask:

- *How do we prove some web pages, research work (or articles) to be original? (Or how do we provide an effective counter measure against the article plagiarism problems?)*

- *Suppose we have just created a web page, can we just "distinguish" it to provide evidence that you have created it at a certain time by simply clicking a button?*

There are some methods to help you prove that you have done something at a certain time. The most popular methods are to use witnesses to "prove" it. We first introduce two of the related systems. One is Cryptology ePrint Archive [16], which is a web based system that can be used to submit unpublished research work. There is somebody at the background of this system reviewing new submissions. The system wants new submissions to be reviewed to make sure their topics are relevant to be put into the system. Finally, you can prove that you have done some research work before a certain time with this system. The other one is a technical report in the libraries of universities (such as MIT library). If you have done some research work that worth to be archived and proved, you can draft technique reports and submit them to the libraries. They will help you make records and give the technical report an identical number. However, both of the two systems need to be processed by people in plain text, and your research works have to be disclosed. Hence, we cannot simply assume people in either Cryptology ePrint Archive or university libraries are completely reliable. We need to design a system that can provide strong evidences of what you have done without disclosing the content of it. This idea is quite similar to Time Stamp Protocol (TSP, RFC 3161[2]) published by IETF in august of 2001. The goal is to provide strong evidence to prove the time feature of information. A company, E-Time Stamp [23], is also providing similar services through the time sampling service. However, this system requires trust of the client side software. We argue that the system is not secure unless the client side application's source code is

open. We design and implement an open source project, DistAca, to accomplish the strong evidence creation task for web pages (or any other types of files).

7.2 System Design

DistAca is composed of 3 modules, Public Key Infrastructure (PKI), DistAca Server, and DistAca Client, as shown in Figure 43. All of them are open source for security reasons. All parts of them are written in C# which can be downloaded from [5]. People can read and compile the code. so they know what is happening in the system.

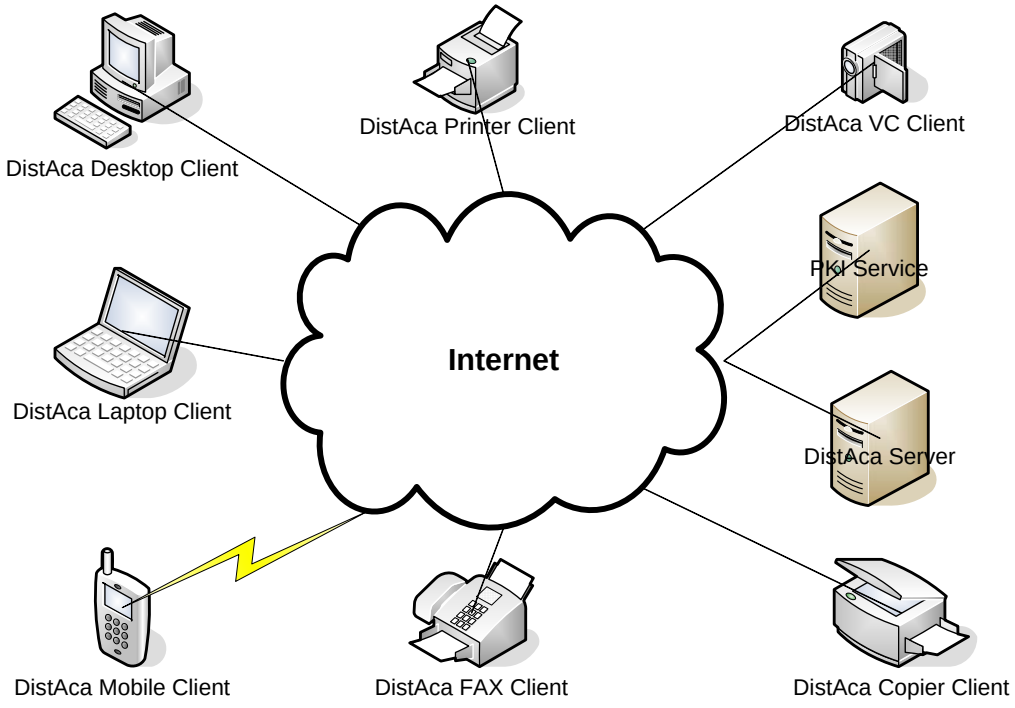


Figure 43. DistAca System Framework

7.2.1 PKI

PKI is the infrastructure that provides public key services. There are many PKI service providers, e.g. Entrust [22], NIST[55], VeriSign [69], Cybertrust [17], Tumbelweed [68],

and RSASecurity [62]. In the current DistAca system we created our own PKI system. In future development, users may be able to choose any of the PKI service providers.

7.2.2 DistAca Client

DistAca Client is a set of pervasive applications. It includes many different versions that can be installed /embedded into various devices and operating systems. It can be used in a PC or PDA to distinguish a single or a set of documents; a printer, FAX, or copier to distinguish that a single or a set of documents have been printed, transmitted, or copied; or a video camera or digital camera to distinguish that video or image has been recorded, etc.

7.2.3 DistAca Server

DistAca Sever plays a role like a broker. When DistAca Client requests to distinguish a set of information, DistAca Server interacts with both DistAca Client and PKI to accomplish the processes. DistAca Server maintains a database of the distinguished information.

7.2.4 Working Flow Design

[To simplify the discussion, we use an example that a user distinguishes a new web page. In other usages, a user may distinguish printed, FAXed, copied, or recorded documents]

The DistAca Server maintains users' information for all DistAca Clients. DistAca Client has a button. When somebody has just finished one web page, he can just simply clicks the button "Distinguish" to get the time stamp to this webpage. Suppose we have a user A. A chooses the new web page and click the "Distinguish" button. DistAca Client will use [file stream | user identity] to generate a signature, i.e. FileSignature = SHA512 ([file stream | user identity]). Then the Client will request DistAca Server to distinguish FileSignature to A's account. DistAca Server will return A one string in the form of

TimeProposal = [FileSignature | Proposed Time Label | DistAca Server Signature] for A to confirm. Hence, DistAca Server cannot deny that it has sent the time label to A. If A confirms the proposal from DistAca Server, the user should click a button to confirm and DistAca Client will send the signed confirmation, Confirmation = [TimeProposal | Confirmation | DistAca Client Signature], to DistAca Server. Therefore, A cannot deny the fact that he accepts the time label. When DistAca Server receive A's confirmation, the new web page is distinguished. At the same time, DistAca Server will send the signed "successful" information, SuccessFeedback = [Confirmation | Successful Information | DistAca Server Signature], back to DistAca Client.

7.3 System Usage Demo

We built the three basis modules of DistAca, which includes PKI Server, DistAca Server, and a PC version of DistAca Client. PKI Server and DistAca Server are web service based and they run in MS IIS (Microsoft Internet Information Server) with Asp.Net.

Figure 44 shows the DistAca Client UI. Users only need to select a file that they wish to distinguish and click the "Distinguish" button.

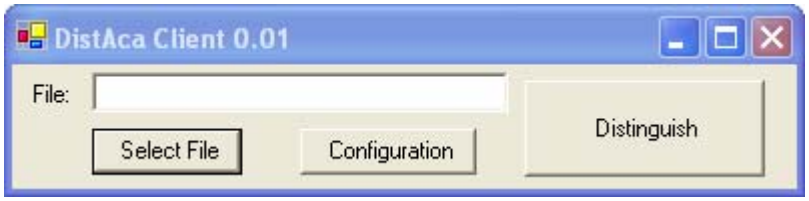


Figure 44. DistAca Client

Figure 45 shows the configuration UI for DistAca Client. There are two accounts users need to take care of: DistAca account in DistAca Server, and PKI account in PKI Server. PKI Server uses ID to represent the identity of the three parts in a distinguishing process.

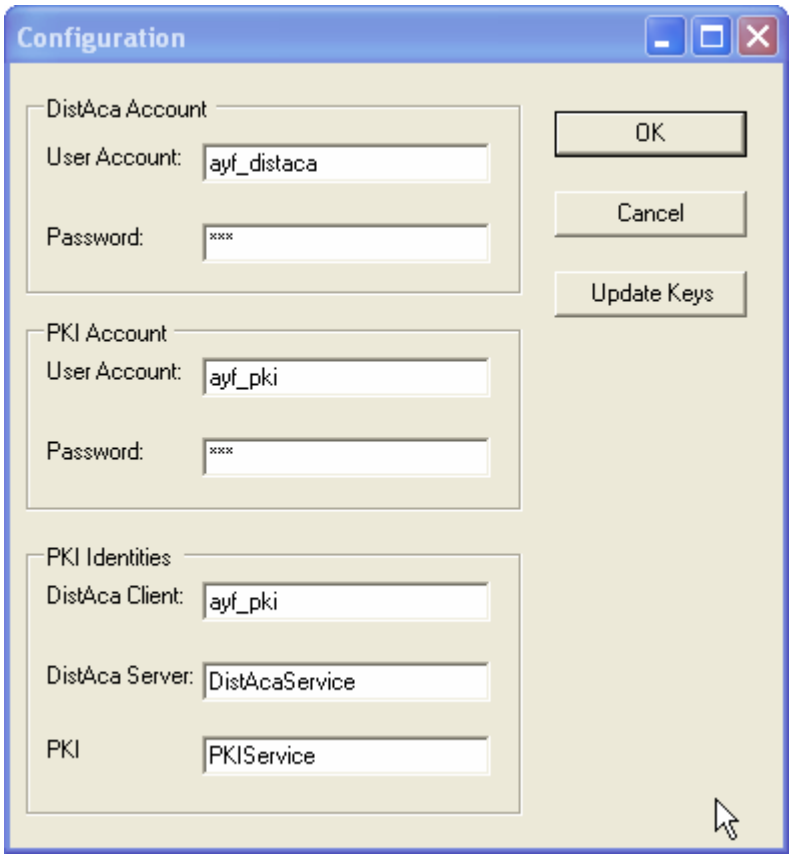


Figure 45. DistAca Client Configuration

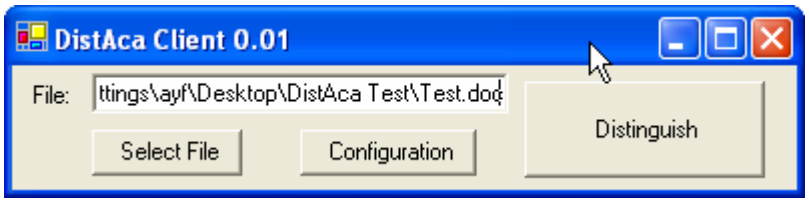


Figure 46. Select File to be Distinguished

Users can choose a file to distinguish as shown in Figure 46 and the file signature will be generated as shown in Figure 47.

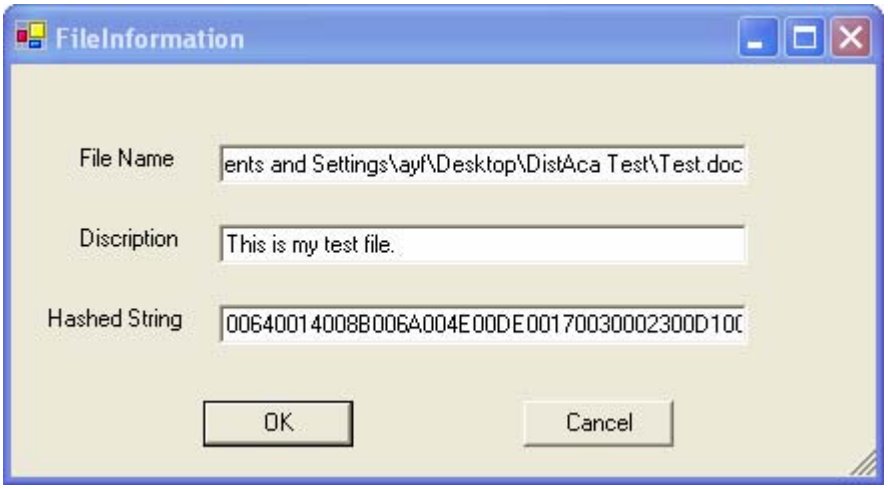


Figure 47. The File Signature

After identity verification of the DistAca Server through PKI Server, a time proposal will be provided to DistAca Client as shown in Figure 48.

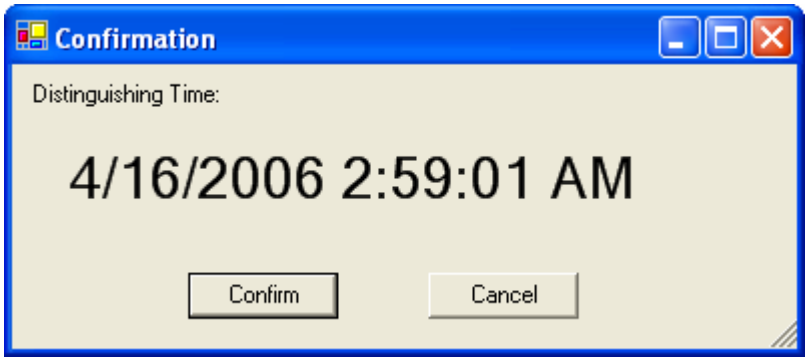


Figure 48. DistAca Server Time Label Proposal

If the user thinks the proposed time is acceptable, he can press the “Confirm” button. When the file is successfully distinguished, the user will be reminded about the success as shown in Figure 49.

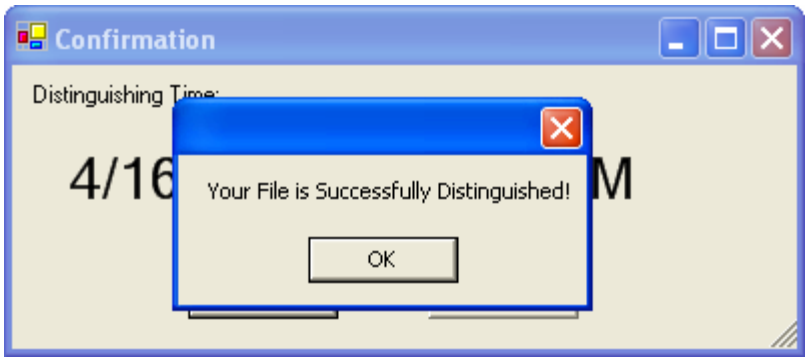


Figure 49. File is Successfully Distinguished

7.4 Conclusion

We design a system that can help people get strong evidence for the originality of a set of information. Websites can use it to prove that they have the original design and researchers can use it to prove they have done the original research work. We argue that a time stamp system is not secure unless it is open source. We built up a system which includes PKI Server, DistAca Server, and PC version DistAca Client. The source code is available at [5] for free download. We also demo the working process of the system.

Chapter 8

Conclusion

Phishing causes severe problems for Web security and privacy. In this dissertation, we address several advanced anti-phishing technologies systematically. We address the potential phishing attacks and propose to solve these phishing attacks with visual assessment approach, semantic assessment approach, human computer interaction enforcement, and web page originality verification. We summarize the conclusion in this section while we have more detailed conclusion and future work disscution in the end of Chapter 4~Chapter 7.

Phishers usually make web pages visually similar to real web pages to spoof users. We use Earth Mover's Distance to evaluate the visual similarity of the suspected web pages to the protected ones. The suspected web pages which are similar to the protected ones will be reported as phishing. Our experiments show that this method performs quite well. This method may fail to work when phishers intend to make the phishing web pages different from the real ones. However, this kind of attack is unusual. Following this method, we implemented SiteWatcher, which is an online anti-phishing system, which includes both a server solution and a client solution. Our experiments show that this method performs the best among all visual based assessment methods.

We propose the earliest methodology and counter measures to fight against Unicode attacks (a kind of homograph attack). We propose three methods to calculate UC-SimList, and one method to construct WWSS list. We also propose to apply the two lists to visual

semantic similarity evaluation algorithms. We construct several such algorithms and implement IRI/IDN SecuChecker, which is the first Unicode attack detection application. Our experiments show that our Unicode attack detection methodology is effective and IRI/IDN SecuChecker behaves well at catching Unicode attacks.

HCI plays an important role in computer security and privacy. HCI mechanism is the platform for visual and semantic level interaction between user and computer. We induce a new concept, Screenjacking, i.e., phishing of client side applications. We demonstrate that everywhere on the computer screen can be faked and propose several methods (Application Trace, Genuine Skin, etc.) to solve Screenjacking problems. Although most of anti-phishing applications use visual indicators to alert users, user studies show that such indicators do not work well, because these indicators are not in the critical path of users' workflow and users do not always bother to look at them. We propose Context Sensitive Password, which is a novel password mechanism in the critical path of users' workflows that can solve this problem. To improve the security against key logger attacks, we designed Semi-Random Password. Both of the new password mechanisms can be used as anti-Screenjacking approaches.

Originality verification of web pages should be the first step to define phishing. We provide a method to verify the originality of web pages. We provide a method to produce strong evidence for web page verification. We argue that a system having the function of making time stamps must use an open source client application. However, investigation shows there is no such system. We develop an open source system, DistAca (including PC version DistAca Client, DistAca Server, and PKI). This system also requires users to confirm the proposed time stamp to improve the trust level.

Phishing attacks have severe negative impacts for the Web and are also are evolving criminal techniques. We address several advanced anti-phishing methods in this dissertation and their counter measures. However, we are not expecting that all phishing problems can be solved by these methods. Other phishing methods would be carried out in the future. Hence, new specific anti-phishing methods should be proposed.

Bibliography

[1]. ActiveX Attack, http://www.mit.edu/~ayf/my_free_sw/activeXAttack.zip

[2]. Adams C., Cain P., Pinkas D., Zuccherato R., Internet X.509 Public Key Infrastructure Time-Stamp Protocol (TSP), Network Working Group Request for Comments: 3161, Category: Standards Track, http://www.ietf.org/rfc/rfc3161.txt

[3]. AlienEntity Free Icons, http://www.entity.cc

[4]. ANSI, American Standard Code for Information Interchange, http://www.ansi.org

[5]. Anti-Phishing Group of City University of Hong Kong, http://antiphishing.cs.cityu.edu.hk

[6]. Anti-Phishing Working Group, http://www.antiphishing.org

[7]. Ashok A., Jakobsson M., Detecting Obfuscation of malicious email by Virtual Rendering, manuscript in preparasion, School of Informatics, Indiana University at Bloomington

[8]. Berners-Lee T., Fielding R., Masinter L., RFC 3986: Uniform Resource Identifier (URI): Generic Syntax, The Internet Society (2005), 2005

[9]. British National Corpus, http://www.natcorp.ox.ac.uk

[10]. Broder A., Glassman S., Manasse M., and Zweig G., Syntactic Clustering of the Web, in Proceedings of the Sixth International World Wide Web Conference, pages 391–404, 1997.

[11]. Chen Y., Ma W. Y., and Zhang H. J., Detecting Web page Structure for Adaptive Viewing on Small Form Factor Devices, in Proceedings of the 12th International Conference on World Wide Web, pages 225–233, 2003.

[12]. Chowdhury A., Frieder O., Grossman D., and McCabe M., Collection Statistics for Fast Duplicate Document Detection, ACM Transactions on Information Systems, Volume 20(2), pages 171–191, 2002

[13]. Cohen S., Guibas L., The Earth Mover's Distance under Transformation Sets, in Proceedings of the IEEE International Conference on Computer Vision, Volume 2, pages 1076-1083, 1999.

[14]. Consumer Sentinel, www.consumer.gov/sentinel

[15]. Cover T. and Thomas J., Elements of Information Theory. John Wiley, 1991

[16]. Cryptology ePrint Archive, http://eprint.iacr.org

[17]. Cybertrust, http://www.cybertrust.com

[18]. Dhamija R. Hash visualization in user authentication. In Proceedings of the Computer Human Interaction 2000 Conference, April 2000.

[19]. Dhamija R., Tygar J. D., and Hearst M. Why Phishing Works, to appear in Proceedings of CHI-2006: Conference on Human Factors in Computing Systems, April 2006.

[20]. Dhamija R., Tygar J. D., The Battle against Phishing: Dynamic Security Skins, Symposium on Usable Privacy and Security 2005, pages 77-99, 2005.

[21]. Duerst M., Suignard M., RFC 3987: Internationalized Resource Identifiers (IRIs), the Internet Society (2005), Jan. 2005

[22]. Entrust, http://www.entrust.com

[23]. E-Time Stamp, http://www.e-timestamp.com

[24]. Federal Trade Commission, www.ftc.gov

[25]. Fu A. Y. Web Link Illustrator Demo for Microsoft IE, www.mit.edu/~ayf

[26]. Fu A. Y., Deng X., Liu W., A Methodology for Unicode String Similarity Assessment, Technical Report, Detp. of Computer Science, City Univ. of Hong Kong, Oct. 2005

[27]. Fu A. Y., Deng X., Liu W., A Potential IRI based Phishing Strategy, in the Proceedings of 6th International Conference on Web Information Systems Engineering, Lecture Notes in Computer Science Volume 3806, pages 618 - 619 , WISE 2005, New York, USA, Nov. 20-22, 2005

[28]. Fu A. Y., http://www.cs.cityu.edu.hk/~anthony/AntiPhishing/IRI, Jun. 2005

[29]. Fu A. Y., Liu W., Deng X., EMD based Visual Similarity for Detection of Phishing Web pages, in Proceedings of International Workshop on Web Document Analysis, 2005

[30]. Fu A. Y., www.cs.cityu.edu.hk/~anthony/AntiPhishing

[31]. Gabrilovich E. and Gontmakher A., The Homograph Attack, Communications of the ACM, Volume 45(2), page128, February 2002

[32]. Garfinkel S., Miller R., Johnny 2: A User Test of Key Continuity Management with S/MIME and Outlook Express. SOUPS, 2005.

[33]. Grauman K., Darrell T., Fast Contour Matching Using Approximate Earth Mover's Distance, in Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, Volume 1, pages 220-227, 2004.

[34]. Gu X. D., Chen J. L., Ma W. Y., and Chen G. L., Visual based Content Understanding towards Web Adaptation, in Proceedings of the Second International Conference on Adaptive Hypermedia and Adaptive Web-based Systems, pages 29–31, 2002.

[35]. Herzberg A., TrustBar: Re-establishing Trust in the Web. 2006. http://www.cs.biu.ac.il/~herzbea/TrustBar

[36]. Hillier F. S., Liberman G. J., Introduction to Mathematical Programming, McGraw-Hill, 1990.

[37]. Hitchcock F. L., the Distribution of a Product from Several Sources to Numerous Localities, Journal of Mathematical Physics, Volume 20, pages 224-230, 1941.

[38]. Hoad T.C. and Zobel J. Methods for Identifying Versioned and Plagiarized Documents, Journal of the American Society for Information Science and Technology, Volume 54(3), pages 203-215, 2003.

[39]. ICANN. http://www.icann.org

[40]. Jakobsson M., Modeling and Preventing Phishing Attacks, in Proceedings of Phishing Panel of Financial Cryptography 2005, 2005

[41]. John C. R., the Image Processing Handbook Second Edition, CRC Press, 1995.

[42]. Kantor P., Voorhees E., The TREC-5 Confusion Track: Comparing Retrieval Methods for Scanned Text, Information Retrieval, Volume 2(2/3), pages 165-176, 2000.

[43]. Knuth D., Morris J. H., and Pratt V., Fast Pattern Matching in Strings. SIAM Journal on Computing, 6(2):323¨C350. 1977. Citations. Original publication.

[44]. Leech G., Rayson P., Wilson A. Word Frequencies in Written and Spoken English: based on the British National Corpus. (2001) pp. 320, Longman, London. ISBN 0582-32007-0

[45]. Levenshtein V.I., Binary Codes Capable of Correcting Deletions, Insertions, and Reversals, Cybernetics and Control Theory, Volume 10, pages707-710, 1966.

[46]. Levina E., Bickel P., the Earth Mover's Distance is the Mallows Distance: Some Insights from Statistics, in Proceedings of the IEEE International Conference on Computer Vision, Volume 2, 2001.

[47]. Linz P., An Introduction to Formal Languages and Automata Third Edition, Jones and Bartlett Publishers Inc.,2001

[48]. Liu W., Deng X, Huang G., Fu A. Y., An Anti-Phishing Strategy based on Visual Similarity Assessment, IEEE Internet Computing, Vol. 10, No. 2, pp. 58-65, 2006

[49]. Crowe M., C# port to WordNet 2.0, http://cis.paisley.ac.uk/crow-ci0

[50]. MathWord, http://mathworld.wolfram.com/NormalDistribution.html

[51]. Microsoft, Arial Unicode MS, Version 1.01, 2000

[52]. Microsoft, Description of the Arial Unicode MS font in Word 2002, http://support.microsoft.com/kb/q287247

[53]. MIT Libraries, http://libraries.mit.edu

[54]. Nanno T., Saito S., and Okumura M., Structuring Web Pages Based on Repetition of Elements, in Proceedings of the 7th International Conference on Document Analysis and Recognition, 2003.

[55]. National Institute of Standards and Technology (NIST), http://csrc.nist.gov

[56]. Needleman S., and Wunsch C. A General Method Applicable to the Search for Similarities in the Amino acid Sequence of Two Proteins, Journal of Molecular Biology, 48(3):443_453, 1970.

[57]. Netscape Corp., the SSL Protocol, http://wp.netscape.com/eng/ssl3

[58]. Perrig A. and Song D., Hash Visualization: A New Technique to Improve Real-World Security, in Proceedings of the 1999 International Workshop on Cryptographic Techniques and E-Commerce

[59]. Reasonable Software, http://www.reasonablesw.com

[60]. Resnik P. (1999): Semantic Similarity in a Taxonomy: An Information-Based Measure and its Application to Problems of Ambiguity in Natural Language. In Journal of Artificial Intelligence Research, vol. 11, pp. 95-130.

[61]. Ross B., Jackson C., Miyake N., Boneh, D., Mitchell, J. Stronger Password Authentication Using Browser Extensions. Proceedings of the 14th Usenix Security Symposium, 2005.

[62]. RSASecurity, http://www.rsasecurity.com

[63]. Rubner Y., Tomasi C., and Guibas L. J., The Earth Mover's Distance as a Metric for Image Retrieval, Technical Report STAN-CS-TN-98-86, Department of Computer Science, Stanford University, 1998.

[64]. Rubner Y., Tomasi C., and Guibas L.J., A Metric for Distributions with Applications to Image Databases, in Proceedings of the IEEE International Conference on Computer Vision, pages 59-66, 1998.

[65]. Salton G., Wong A., and Yang C.S., A Vector Space Model for Information Retrieval, Journal of the American Society for Information Science, Volume 18(11), pages 613–620, 1975.

[66]. Sergei Winitzki, A Handy Approximation for the Error Function and its Inverse, http://www.theorie.physik.uni-muenchen.de/~serge, January 20, 2006

[67]. The Unicode Consortium, http://www.unicode.org

[68]. Tumbelweed, http://tumbleweed.com

[69]. VeriSign, https://www.verisign.com

[70]. Wiedenbeck S., Waters J., Birget J., Brodskiy A., Memon N., Authentication using Graphical Passwords: Eeffects of Tolerance and Image Choice, in SOUPS 2005

[71]. Wood L., Document Object Model (DOM) Level 1 Specification, http://www.w3.org/TR/REC-DOM-Level-1

[72]. WordNet, Cognitive Science Laboratory of Princeton University, http://wordnet.princeton.edu

[73]. Wu M., Miller R. C., and Little G., Web Wallet: Preventing Phishing Attacks by Revealing User Intentions, Symposium on Usable Privacy and Security 2006.

[74]. Wu M., Miller, R. Garfinkel S., Do Security Toolbars Actually Prevent Phishing Attacks?, CHI 2006, 2006

[75]. www.sitewatcher.biz

[76]. Ye Z., Smith S.W., Anthony D. Trusted Paths for Browsers. ACM Transactions on Information System Security. 8 (2): 153-186. May 2005.

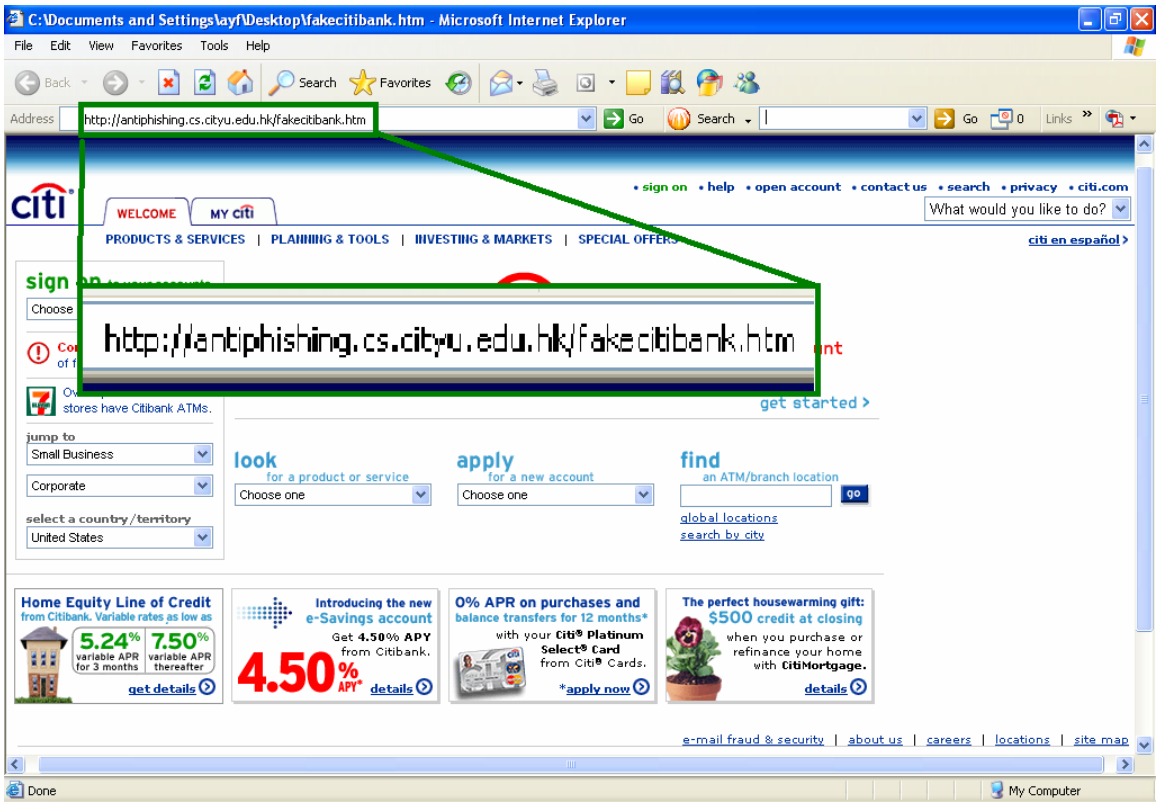
[77]. Yu S., Cai D., Wen J.R., and Ma W.Y., Improving Pseudo-relevance Feedback in Web Information Retrieval using Web Page Segmentation, in Proceedings of the 14th International Conference on World Wide Web, pages 11–18, 2003.

[78]. Zhang W., Liu W., and Zhang K., Symbol Recognition with Kernel Density Matching, to appear in IEEE Transaction on Pattern Analysis and Machine Intellegence, 2006.

Appendices

A. Visually Identical Web Pages Can be Totally Different

Figure 50 shows an example of faked web page of www.citibank.com and the real one. As we can see, the real web page contains 320 lines of HTML and Java Script code (we deleted part of them to save space) while the faked web page only contains one line of HTML code. However they look exactly the same (except the links in the address bars).

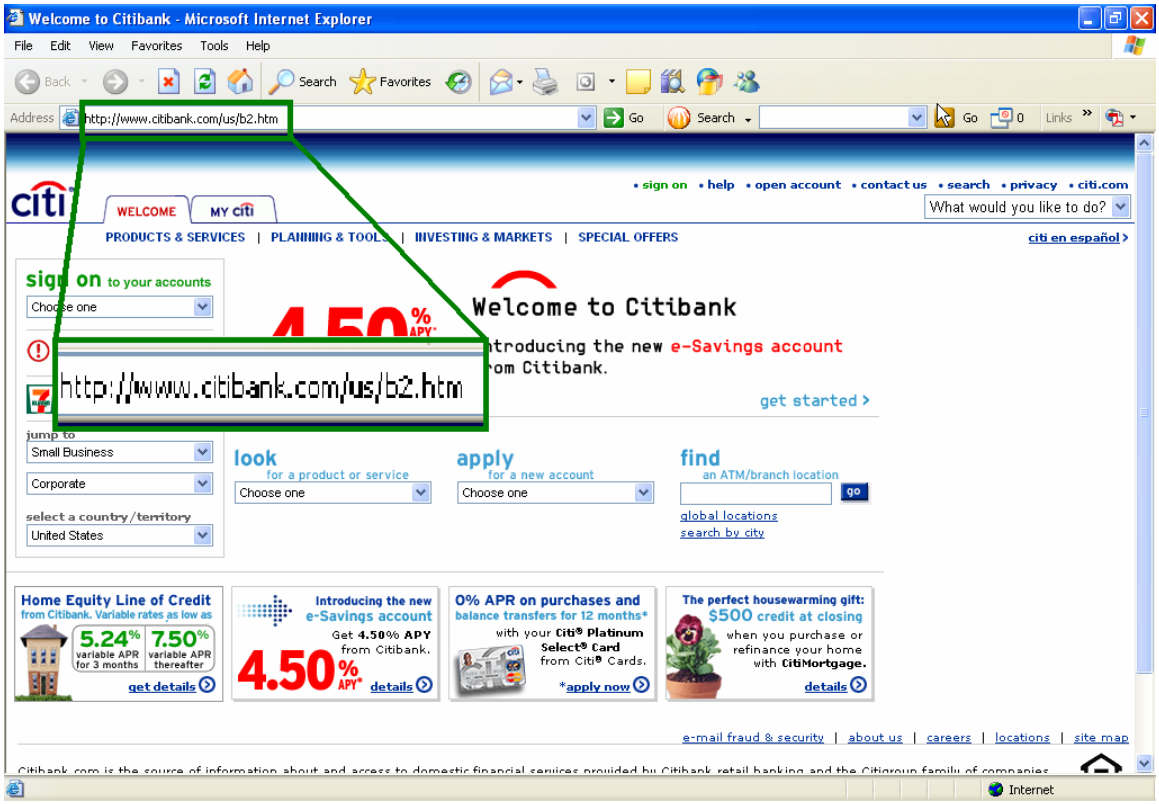


(a) Real CitiBank web page

```
<html>
<head>
<title>Welcome to Citibank</title>
<script type="text/javascript" src="/domain/scripts/sniffer.js"></script>
<script type="text/javascript">
<!--
//Functions here. Deleted
//-->
</script>
<script language="Javascript" src="/domain/scripts/bsa_client_src.js"></script>
```

```
<script type="text/javascript">
<!--
var pixelBeacon = new Image();
pixelBeacon.src=PGI+'/track/?id=8706&r='+Math.random();
// site analytics pixel
var szPixURL = new Image();
szPixURL.src=PGI+'/site/?adv=1&c=CitibankHome&sc='+BVU+btsa();
//-->
</script>
</head>
<script type="text/javascript">
<!--
PGI="http://citi.bridgetrack.com";
var CITI_PG_CT_PID='Citibank.comHomePageBAU';
var CITI_PG_isHome = true;
var  CITI_PG_txtBottomDisclaimer='<table  border=0  cellspacing=0  cellpadding=0><tr><td
align=left valign=top><br>Citibank.com is the source of information about and access to
domestic financial services provided by Citibank retail banking and the Citigroup family
of companies. Citibank, N.A., Citibank (West), FSB, Citibank, F.S.B., Citibank Texas,
N.A.  Member  FDIC.</td><td  align=right><img  src=/domain/images/lender.gif  width=37
height=42 alt="An Equal Housing Lender" border=0 hspace=6 vspace=2></td></tr></table>';
//-->
</script>
<body    onload="initLists();"    bgcolor="#ffffff"    bottommargin="0"    leftmargin="0"
marginheight="0" marginwidth="0" topmargin="0" link="#003399" vlink="#003399">
<script                  type="text/javascript"                  language="JavaScript1.2"
src="/domain/scripts/branding.js"></script>
<style type="text/css">
//Styles here. Deleted
</style>
<table border="0" width="776" cellspacing="0" cellpadding="0" ID="Table1">
//table here. Deleted
</table>
<!-- footer -->
<script type="text/javascript">
<!--
footer()
//-->
</script>
</body>
</html>
```

(b) Source Code of Real CitiBank web page (320 lines, part of them are deleted)



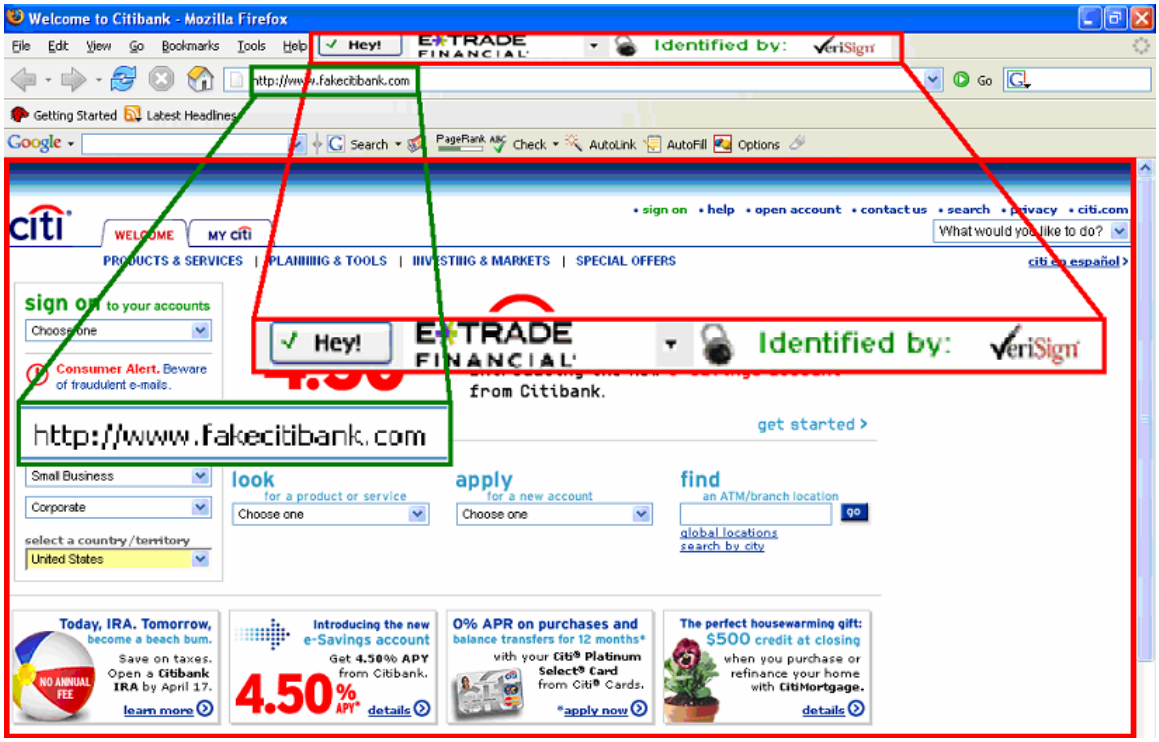
(c) Fake CitiBank web page

```
<p><img border="0" src="a.png" width="1005" height="658"></p>
```

(d) Source Code of Fake CitiBank web page (only 1 line)

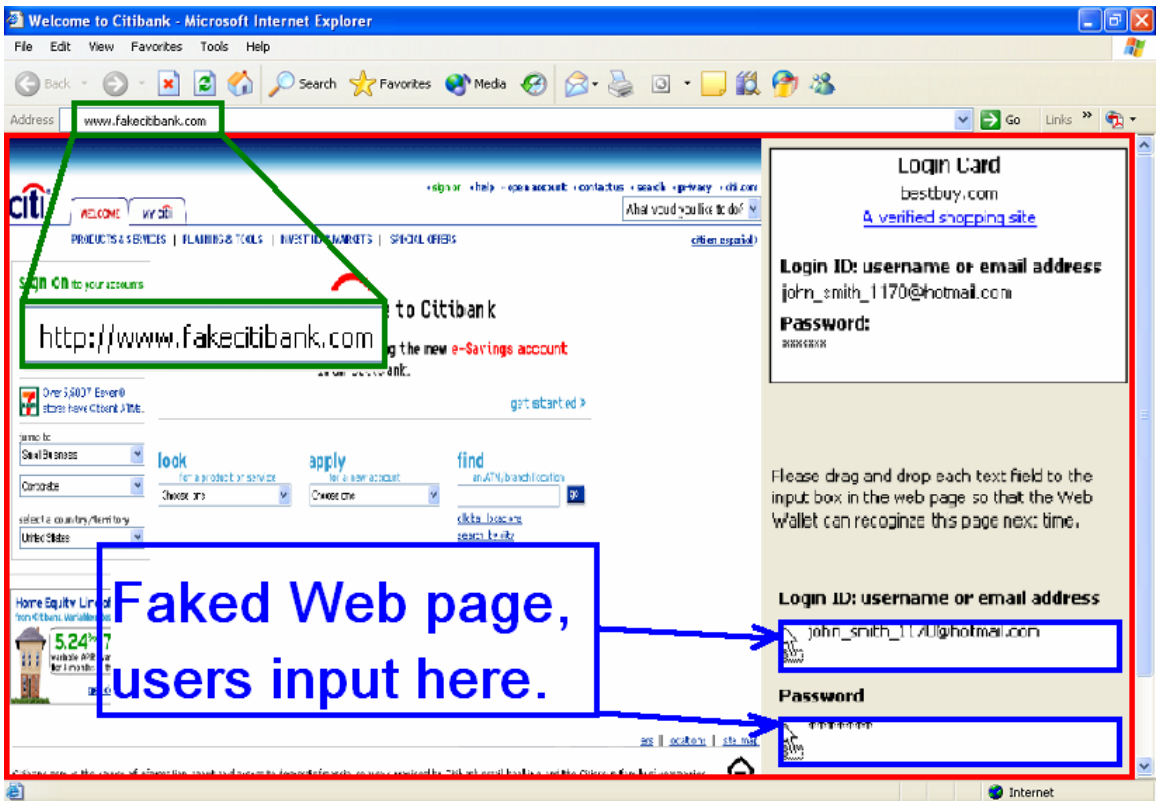
Figure 50. Visual and code comparisons of faked www.citibank.com to the real one.

B. Faked TrustBar, Dynamic Secure Skin, and Web Wallet.



(a) Faked TrustBar

(b) Faked Dynamic Secure Skin



(c) Faked Web Wallet

Figure 51 Demonstrations of faked TrustBar, Dynamic Secure Skin, and Web Wallet. The contents in red rectangles are faked areas.

C. Similar Unicode String Generation

Figure 52 and Figure 53 demonstrate the similar text strings of "Welcome to CitiBank!" using UC-SimList_v1 and UC-SimList1 respectively, while Figure 54 and Figure 55 demonstrate the samples of the similar IRIs of "www.citibank.com" using UC-SimList_v1 and UC-SimList1 respectively. The number under each character is the corresponding Unicode of this character in Hexadecimal format. Similarly, there are various similar/fake Unicode strings of Chinese and Japanese etc., as shown in Figure 56 and Figure 57 respectively. We demonstrate the examples of Chinese and Japanese using

UC-SimList only, which can generate more variety of a certain given string. We also demonstrate the experiments using UC-SimList_v and UC-SimList with different similarity thresholds (0.8, 0.85, 0.9, 0.95, and 1) as shown in Figure 58 and Figure 59.

Original Text	W	e	l	c	o	m	e		t	o		C	i	t	i	B	a	n	k	!
	57	65	006C	63	006F	006D	65	20	74	006F	20	43	69	74	69	42	61	006E	006B	21
Faked Text 1	W	e	l	c	o	m	e		t	o		C	i	t	i	B	a	n	k	!
	FF37	FF45	04C0	217D	043E	006D	FF45	20	74	03BF	20	421	69	FF54	FF49	392	430	FF4E	006B	FF01
Faked Text 2	W	e	l	c	o	m	e		t	o		C	i	t	i	B	a	n	k	!
	57	FF45	FF4C	63	006F	217F	FF45	20	74	043E	20	43	456	FF54	2170	392	61	FF4E	006B	01C3
Faked Text 3	W	e	l	c	o	m	e		t	o		C	i	t	i	B	a	n	k	!
	57	435	2160	217D	FF4F	006D	FF45	20	FF54	FF4F	20	43	456	74	69	412	430	006E	006B	FF01
Faked Text 4	W	e	l	c	o	m	e		t	o		C	i	t	i	B	a	n	k	!
	FF37	FF45	FF29	FF43	FF4F	217F	FF45	20	FF54	FF4F	20	216D	2170	FF54	456	FF22	FF41	FF4E	006B	01C3
Faked Text 5	W	e	l	c	o	m	e		t	o		C	i	t	i	B	a	n	k	!
	57	435	FF29	217D	006F	006D	435	20	FF54	043E	20	43	FF49	FF54	456	412	FF41	006E	006B	01C3
Faked Text 6	W	e	l	c	o	m	e		t	o		C	i	t	i	B	a	n	k	!
	FF37	435	217C	63	FF4F	006D	435	20	74	03BF	20	43	2170	74	2170	42	61	FF4E	FF4B	01C3

Figure 52. Similar/Faked text strings of "Welcome to CitiBank!" using UC-SimList_v1

Original Text	W	e	l	c	o	m	e		t	o		C	i	t	i	B	a	n	k	!
	57	65	006C	63	006F	006D	65	20	74	006F	20	43	69	74	69	42	61	006E	006B	21
Faked Text 1	w	e	l	C	o	M	E		T	o		c	l	T	i	B	A	N	K	!
	FF57	435	01C0	FF23	03BF	004D	395	20	03A4	FF4F	20	63	2160	03A4	69	FF22	391	004E	FF2B	01C3
Faked Text 2	w	E	l	C	o	M	E		T	O		C	l	t	l	B	a	n	K	!
	FF57	395	04C0	216D	006F	216F	395	20	FF34	004F	20	216D	399	FF54	04C0	392	61	FF4E	004B	FF01
Faked Text 3	w	E	L	C	O	M	E		T	o		C	l	t	l	B	a	N	k	!
	FF57	415	004C	421	004F	039C	45	20	03A4	03BF	20	43	49	74	006C	392	FF41	FF2E	FF4B	21
Faked Text 4	W	E	l	c	O	M	e		T	o		C	l	t	l	b	A	N	k	!
	57	45	04C0	217D	041E	216F	FF45	20	FF34	006F	20	43	2160	74	399	62	FF21	039D	006B	21
Faked Text 5	W	E	l	c	O	m	e		t	O		C	l	t	l	B	A	N	K	!
	57	395	399	63	041E	217F	65	20	74	FF2F	20	FF23	406	FF54	FF4C	FF22	391	004E	212A	FF01
Faked Text 6	w	e	l	c	o	M	E		T	o		c	l	T	l	B	a	N	K	!
	77	65	FF4C	441	03BF	039C	45	20	54	FF4F	20	63	217C	FF34	04C0	FF22	61	FF2E	212A	01C3

Figure 53. Similar/Faked text strings of "Welcome to CitiBank!" using UC-SimList1

Original Text	w	w	w	.	c	i	t	i	b	a	n	k	.	c	o	m
	77	77	77	002E	63	69	74	69	62	61	006E	006B	002E	63	006F	006D

Faked Text 1	w	w	w	·	c	i	t	i	b	a	n	k	·	c	o	m
	FF57	FF57	77	2027	FF43	69	74	69	62	61	006E	FF4B	02D9	441	FF4F	217F
Faked Text 2	w	w	w	·	c	i	t	i	b	a	n	k	.	c	o	m
	FF57	FF57	77	02D9	FF43	456	FF54	2170	FF42	FF41	FF4E	FF4B	2024	217D	006F	FF4D
Faked Text 3	w	w	w	·	c	i	t	i	b	a	n	k	.	c	o	m
	77	FF57	77	387	217D	456	FF54	2170	FF42	61	FF4E	006B	FF0E	63	03BF	FF4D
Faked Text 4	w	w	w	·	c	i	t	i	b	a	n	k	·	c	o	m
	77	FF57	77	FF65	441	2170	FF54	2170	62	FF41	FF4E	006B	02D9	217D	043E	006D
Faked Text 5	w	w	w	.	c	i	t	i	b	a	n	k	·	c	o	m
	FF57	FF57	FF57	2024	63	456	FF54	456	FF42	430	FF4E	FF4B	2027	03F2	FF4F	FF4D
Faked Text 6	w	w	w	·	c	i	t	i	b	a	n	k	.	c	o	m
	77	FF57	77	2027	03F2	69	74	456	FF42	FF41	006E	FF4B	2024	FF43	03BF	006D

Figure 54. Similar/Faked IRI/IDN strings of “www.citibank.com” generation using UC-SimList_v1

Original Text	w	w	w	.	c	i	t	i	b	a	n	k	.	c	o	m
	0077	0077	0077	002E	0063	0069	0074	0069	0062	0061	006E	006B	002E	0063	006F	006D
Faked Text 1	W	w	w	·	c	l	t	l	B	A	N	k	·	c	O	m
	0057	0077	0077	2027	217D	01C0	0074	04C0	0042	0410	039D	006B	2027	0063	039F	006D
Faked Text 2	W	W	W	·	c	l	T	i	b	a	N	K	.	c	O	M
	FF37	FF37	0057	0387	03F2	FF4C	0054	0456	0062	FF41	004E	004B	2024	0063	FF2F	FF2D
Faked Text 3	W	w	w	.	c	i	t	l	B	a	N	K	·	c	O	M
	FF37	0077	0077	2024	0063	FF49	0074	006C	FF22	FF41	FF2E	004B	02D9	0441	039F	004D
Faked Text 4	w	w	w	·	C	i	t	i	B	a	n	K	·	C	o	m
	FF57	FF57	FF57	2027	FF23	FF49	FF54	0069	0392	FF41	006E	FF2B	00B7	0043	FF4F	FF4D
Faked Text 5	W	W	W	·	C	l	t	l	b	a	N	K	·	c	O	M
	FF37	FF37	FF37	02D9	FF23	217C	FF54	217C	0062	FF41	039D	039A	00B7	FF43	004F	FF2D
Faked Text 6	W	w	W	.	C	l	t	l	B	a	n	k	.	c	o	M
	FF37	FF57	0057	2024	0421	006C	FF54	217C	0392	0061	006E	006B	2024	0441	FF4F	FF2D

Figure 55. Similar/Faked IRI/IDN strings of “www.citibank.com” using UC-SimList1

Original Text	欢	迎	光	临	花	旗	银	行	！
	6B22	8FCE	5149	4E34	82B1	65D7	94F6	884C	FF01
Faked Text 1	欢	迎	光	臨	花	旗	銀	行	！
	6B22	8FCE	5149	81E8	82B1	65D7	9280	884C	01C3
Faked Text 2	欢	迎	光	临	花	旗	银	行	！
	6B22	8FCE	5149	4E34	82B1	65D7	94F6	884C	01C3
Faked Text 3	欢	迎	光	临	花	旗	銀	行	﹗
	6B22	8FCE	5149	4E34	82B1	65D7	9280	884C	FE57
Faked Text 4	欢	迎	光	臨	花	旗	銀	行	！
	6B22	8FCE	5149	81E8	82B1	65D7	9280	884C	FF01
Faked Text 5	歡	迎	光	臨	花	旗	銀	行	﹗

	6B61	8FCE	5149	81E8	82B1	65D7	9280	884C	FE57
Faked Text 6	歡	迎	光	臨	花	旗	銀	行	﹗
	6B61	8FCE	5149	81E8	82B1	65D7	9280	884C	0021

Figure 56. Similar/Faked text strings of "欢迎光临花旗银行！" (in Chinese, for "Welcome to CitiBank!") using UC-SimList1

Original Text	よ	う	こ	そ	シ	テ	ィ	バ	ン	ク	へ	！
	3088	3046	3053	305D	30B7	30C6	30A3	30D0	30F3	30AF	3078	FF01
Faked Text 1	よ	ウ	コ	ソ	シ	テ	ぃ	バ	ン	く	へ	！
	3088	30A6	30B3	30BD	30B7	30C6	3043	30D0	30F3	304F	30D8	0021
Faked Text 2	ヨ	う	コ	ソ	し	て	ぃ	バ	ン	ク	へ	﹗
	30E8	3046	30B3	30BD	3057	3066	3043	30D0	30F3	30AF	3078	FE57
Faked Text 3	よ	ウ	コ	そ	シ	て	ぃ	バ	ン	く	へ	！
	3088	30A6	30B3	305D	30B7	3066	3043	30D0	30F3	304F	3078	0021
Faked Text 4	よ	ウ	こ	ソ	シ	て	ぃ	バ	ン	ク	へ	！
	3088	30A6	3053	30BD	30B7	3066	3043	30D0	30F3	30AF	3078	01C3
Faked Text 5	ヨ	う	こ	そ	し	て	ィ	ば	ん	く	へ	﹗
	30E8	3046	3053	305D	3057	3066	30A3	3070	3093	304F	30D8	FE57
Faked Text 6	ヨ	う	コ	ソ	し	テ	ィ	バ	ン	ク	へ	！
	30E8	3046	30B3	30BD	3057	30C6	30A3	30D0	30F3	30AF	30D8	01C3

Figure 57. Similar/Faked text strings of "よ う こ そ シ テ ィ バ ン ク へ ！ "(in Japanese, for "Welcome to CitiBank!") using UC-SimList1

Original Text	W	e	l	c	o	m	e		t	o		C	i	t	i	B	a	n	k	!
	57	65	006C	63	006F	006D	65	20	74	006F	20	43	69	74	69	42	61	006E	006B	21
UC- SimList$_v$ 0.8	W	o̩	!	c	e	ɱ	o		t̲	o		C	!	f	¡	B	ạ	ŋ	k	ı
	FF37	1ECD	01C3	63	04D9	271	FF4F	20	1E6F	FF4F	20	43	FF01	01AD	00A1	392	1EA1	014B	006B	05C0
UC- SimList$_v$ 0.85	W	e̩	!	ɛ	o	m	o		t̪	ɒ		C	l	t̪	i	Б	a	n	ķ	l
	57	1E+19	01C3	454	047B	217F	FF4F	20	163	252	20	216D	FF4C	163	1F30	182	FF41	FF4E	1E33	217C
UC- SimList$_v$ 0.9	Ẉ	e	ı	c	o	m	e		t	o		C	i	t	i	B	a	n	ķ	!
	1E88	435	05C0	FF43	006F	217F	65	20	74	03BF	20	421	456	74	FF49	412	61	FF4E	1E33	FF01
UC- SimList$_v$ 0.95	Ẉ	e	l	c	o	m	e		t	o		C	i	t	i	B	a	n	k	!
	1E88	65	49	217D	006F	217F	435	20	74	043E	20	216D	FF49	FF54	2170	412	430	FF4E	006B	FF01
UC-SimList_v 1.0	W	e	l	c	o	m	e		t	o		C	i	t	i	B	a	n	k	!
	57	435	2160	FF43	043E	006D	65	20	74	03BF	20	FF23	69	74	FF49	FF22	FF41	006E	FF4B	01C3

Figure 58. Similar/Faked text strings of "Welcome to CitiBank!" using UC-SimList_v T, where T∈{0.8, 0.85, 0.9, 0.95, and 1}

Original Text	W	e	l	c	o	m	e		t	o		c	l	T	I	B	A	n	K	!

	0057	FF45	FF4C	0441	043E	FF4D	0065	0020	0074	006F	0020	03F2	0399	03A4	0049	0042	0410	006E	004B	FF01
UC-SimList 0.8	W	ɐ	L̲	ç	C	M	ɵ		ţ	Ç		O	I	ţ	i	Б	Ạ	ŋ	K	i
	0057	0250	1E3A	00E7	FF23	041C	01DD	0020	0163	04AA	0020	004F	01C0	0163	0456	0183	1EA0	019E	FF2B	0456
UC-SimList 0.85	Ẉ	E	I	Ç	o	M	o		ţ	ɵ		c	I	t	Ị	ь	Ạ	ŋ	ĸ	I
	1E88	FF25	04C0	04AA	047B	039C	043E	0020	0163	0275	0020	03F2	FF4C	FF54	1ECA	1E05	1E00	0146	0199	04C0
UC-SimList 0.9	w	E	I	c	Ọ	M	e		T	o		c	I	t	I	B	A	n	K̲	!
	FF57	0395	2160	03F2	1ECC	216F	FF45	0020	01AC	043E	0020	217D	0399	0074	2160	0042	0041	043F	1E34	01C3
UC-SimList 0.95	ẉ	e	I	C	o	m	e		T	o		c	I	T	i	B	A	Ṇ	K̲	!
	1E89	0065	006C	0043	043E	FF4D	0065	0020	0422	FF4F	0020	0063	217C	0422	FF49	FF22	0041	1E46	1E32	FF01
UC-SimList 1.0	W	e	I	c	o	m	e		t	o		c	I	T	I	B	A	n	K	!
	0057	FF45	FF4C	0441	043E	FF4D	0065	0020	0074	006F	0020	03F2	0399	03A4	0049	0042	0410	006E	004B	FF01

Figure 59. Similar/Faked text strings of "Welcome to CitiBank!" using UC-SimList T, where T∈{0.8, 0.85, 0.9, 0.95, and 1}

D. SecuChecker Usage and Features

With the internationalization of information processing, the use of IRI/IDN is becoming a trend. However, we have shown that IRI/IDNs can be deceptive; in particular, it is possible to have two distinct IRI/IDNs which are hard (or impossible) to distinguish visually. This problem has already been reported for URIs in [31], which notes the possibility of mimicking English "microsoft.com" with Cyrillic "microsoft.com". However, there is no application or tool available for IRI/IDN detection. The domain name registration regulations are made by ICANN [39]. It can improve the domain name registration guidelines and ask its authorized registrar companies to follow them. ICANN first added the related section in "Additional Remark" in IRI/IDN Ver. 2.0, Nov. 8, 2005, and continued listing it in the "Additional Remark" of Ver. 2.1, Feb. 22, 2006. However there are no counter measure has been provided yet. Hence implement IRI/IDN SecuChecker as one possible solution.

Figure 60 shows the interface of IRI/IDN SecuChecker. It includes a textbox to input a new registered IRI/IDN, a display showing the Unicode form of the inputted IRI/IDN, an

option pane for selecting the Kernel Algorithm, a “Search” button, a “Clear” button, and a listbox to show the detected similar/fake IRI/IDN(s). There are two important lists in the database running behind the application: the UC-SimList and the registered IRI/IDN list. According the methodology in Section 5.2.4, various string matching algorithms (including string similarity algorithms and substring searching algorithms, etc.) could be applied. We have implemented visual and semantic based edit distance (VSED), where we use the UC-SimList to evaluate the cost of replacing one character into another. We choose the threshold to be 0.12, where we can achieve the best recall and precision values at the same time, as already shown in Figure 27. We also implement the visual and semantic based Knuth Morris Pratt (KMP) algorithm (VSKMP), where we use UC-SimList to assess the similarity between two given characters. We use a character similarity threshold of 0.08 to evaluate the two given characters—empirically, we find that this threshold can classify the characters well, as shown in Figure 61. Characters to the left of the thick vertical bars have the similarities to the given characters (GCs) of more than 0.8.

VSED shows better performance in phishing IRI/IDN detection. It can detect “www.bankofthevvest.com” (double “v” to mimic “w”) from “www.bankofthewest.com”, while VSKMP cannot, as shown in Figure 62. However VSKMP has better time complexity, namely $\Theta(m+n)$ compared to $\Theta(mn)$ for VSED (where m and n are the lengths of the two strings under assessment). In real experiments, VSKMP tends to be fast, and it also behaves well when the protected IRI/IDN is a substring of the new domain name under registration, e.g. VSKMP can detect the phishing IRI/IDN “www.citibank.com.info123.biz”, while VSED cannot, as shown in Figure 63. Each pair

of domain name evaluations takes about 0.0012 seconds with VSED and about 0.0005 seconds with VSKMP (on average, on a P4 2.4G PC with 512MB RAM). Therefore, if the user thinks the system is fast enough, we recommend selecting "Both" in the kernel algorithm group before pressing the "Search" button. In this way, the phishing IRI/IDN(s) detected by any of VSED or VSKMP will be reported.

IRI/IDN SecuChecker is not just limited to checking the validity of domain names; it can be used in any instance where visually and semantically unique Unicode identifiers are desired. For instance, it can be used in a user name (account) registration server, and prevent users from spoofing the identities of other users.

In the new registration textbox, we used different colors to represent characters from different language character sets. It is another feature of IRI/IDN SecuChecker and we call it Web Link Illustrator. We can simply add this web address convention to web browsers' address bars. A demo of the Web Link Illustrator for Microsoft IE is available at [25] as shown in Figure 64. ICANN classifies characters that can be used in many different languages, which are listed in [39]. To implement a Web Link Illustrator, one can simply study and implement the character code ranges and program add-ins for IE, FireFox, etc. In this demo, the fake address for CitiBank contains an "a" from a different character set, which is highlighted in a different color (red) to remind users to be cautious (we choose to change the color because human eyes are sensitive to colors). We consider this a simple but effective potential feature for web browsers.

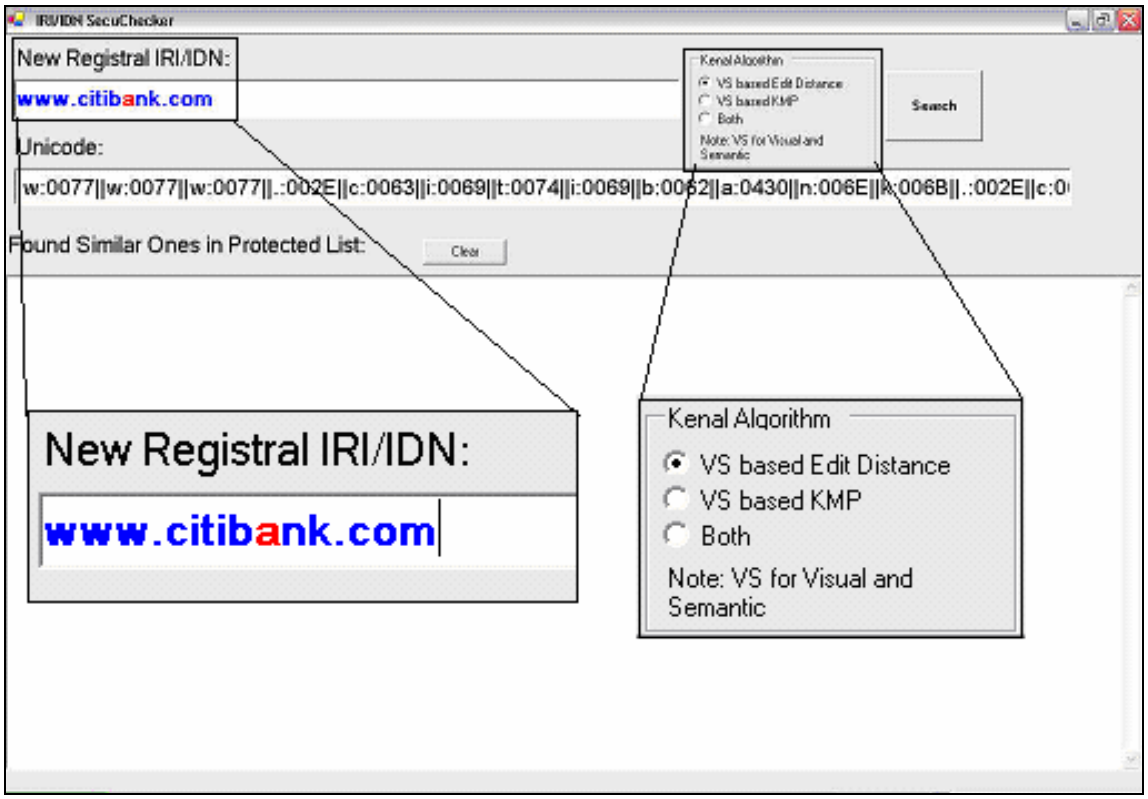
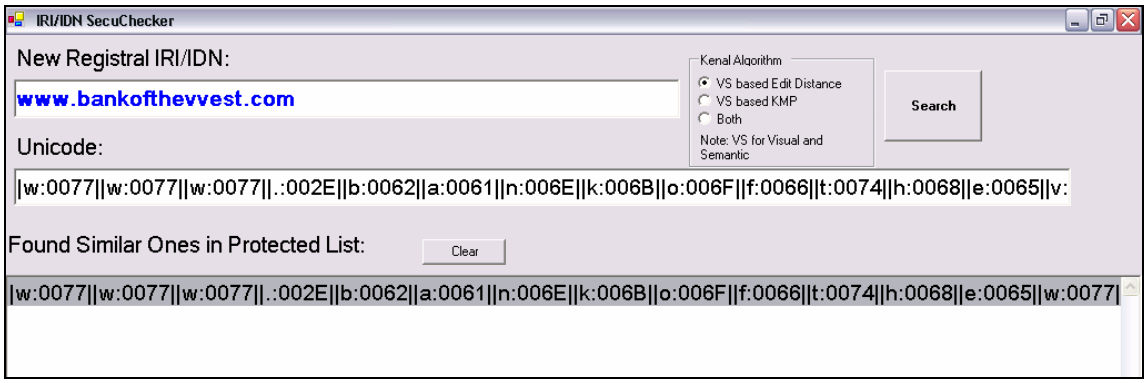


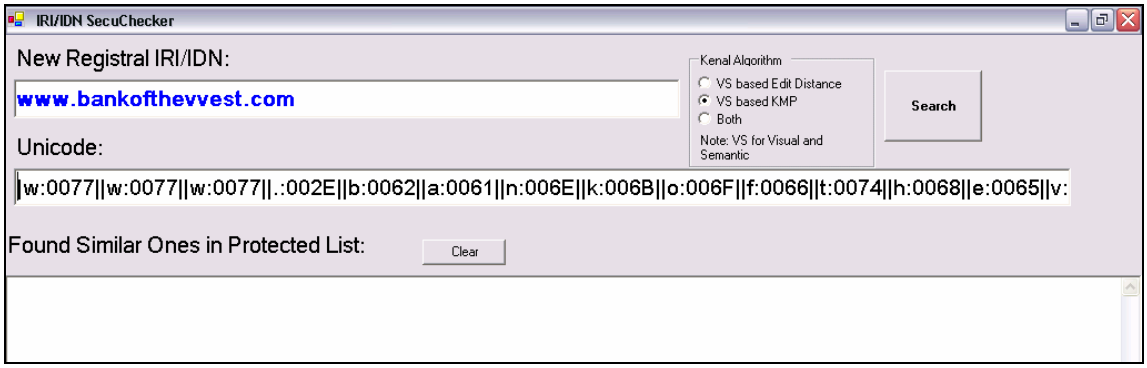
Figure 60. The interface of IRI/IDN SecuChecker

GC \ Rank	1	2	3	4	5	6	7
M	M	M	M	M	Ṃ	Ṁ	IX
004D	FF2D	039C	041C	216F	1E42	1E40	2168
N	N	N	Ṇ	Ņ	N̲	N̂	Nj
004E	FF2E	039D	1E46	0145	1E48	1E4A	01CB
Z	Z	Z	Ẓ	Z̲	Ƶ	Σ	Σ
005A	0396	FF3A	1E92	1E94	01B5	01A9	03A3

Figure 61. The threshold selection for characters' visual similarity. Note: GC for given character. In each row, the rank the higher, the visual similarity the higher to GC.

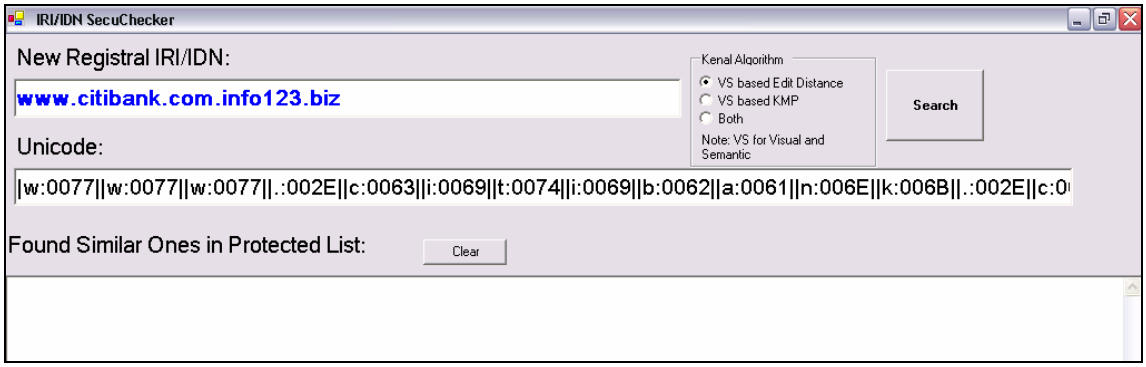


a) Phishing IRI/IDN detected using VSED

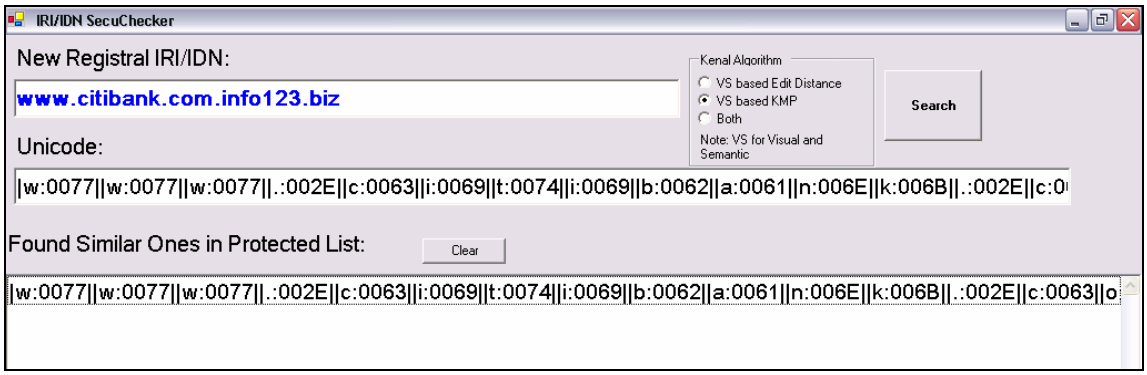


b) Phishing IRI/IDN not detected using VSKMP

Figure 62. VSED can detect the string“www.bankofthevvest.com ” (double “v” to mimic “w”), while VSKMP cannot.



a) Phishing IRI/IDN not detected using VSED



b) Phishing IRI/IDN detected using VSKMP

Figure 63. VSKMP can detect the string “www.citibank.com. info123.biz”, while VSED cannot

a) Correct web link. Web Link Illustrator paint all characters to blue, which is the color corresponding to Latin characters

b) Web Link Illustrator paint the “a” from another character set into red.

Figure 64. Demo for Web Link Illustrator

E. Figures for Human Computer Interaction Enforcement

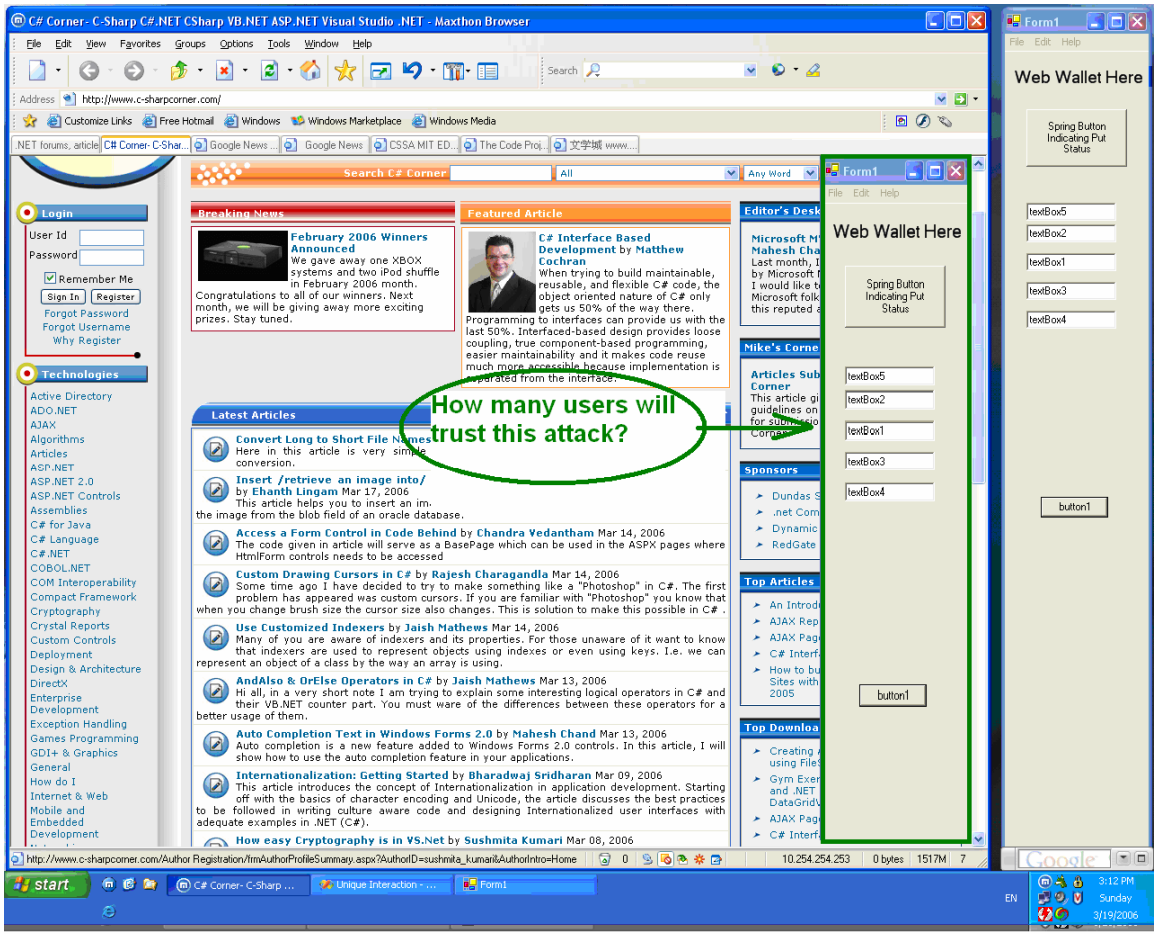


Figure 65 Referential UI

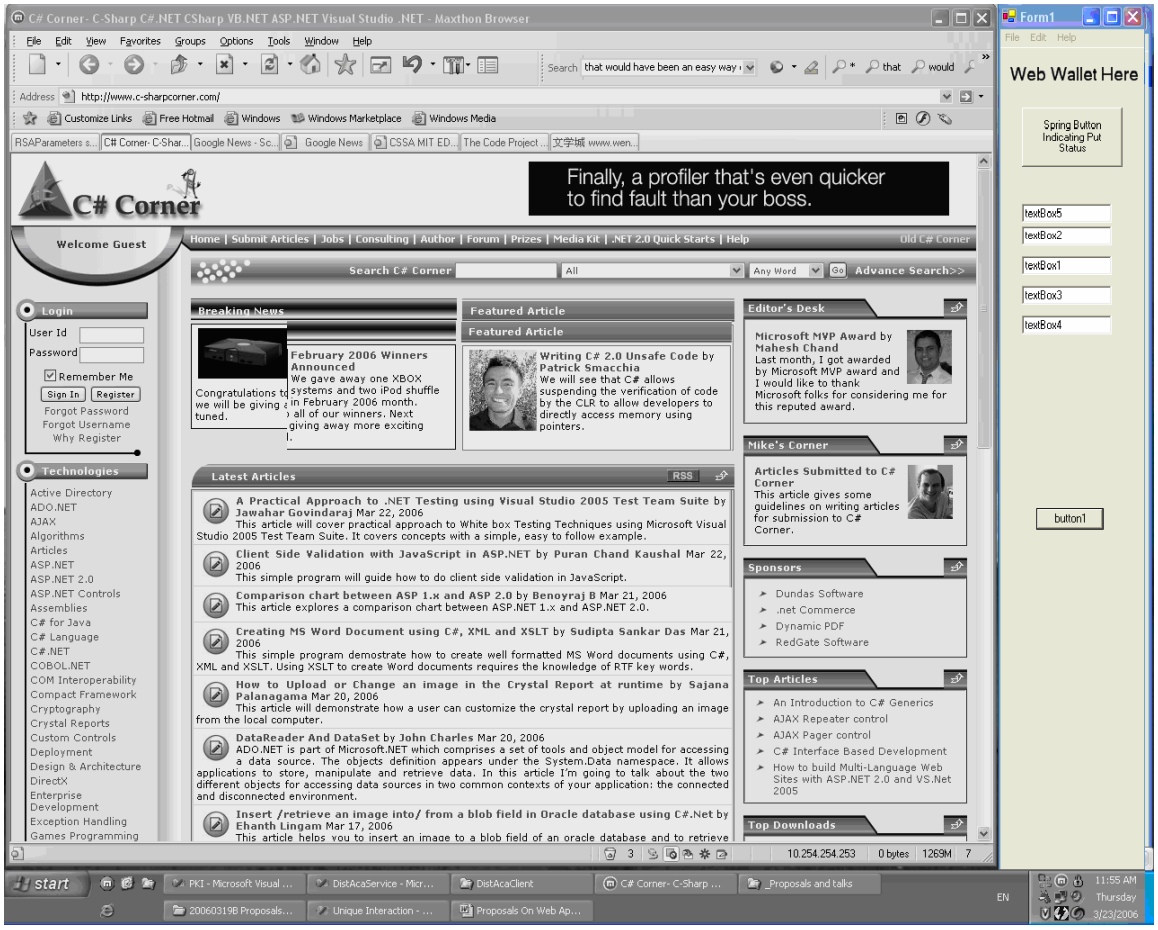
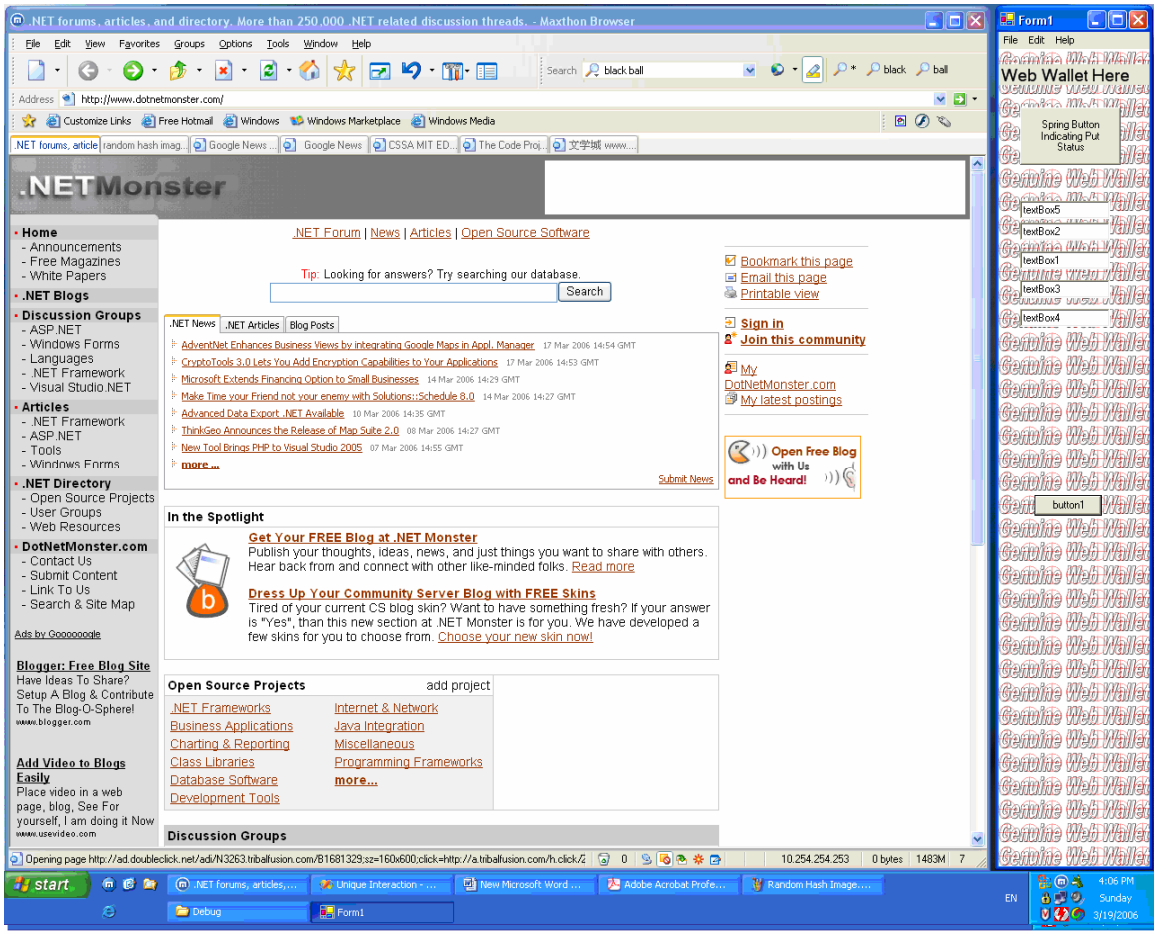
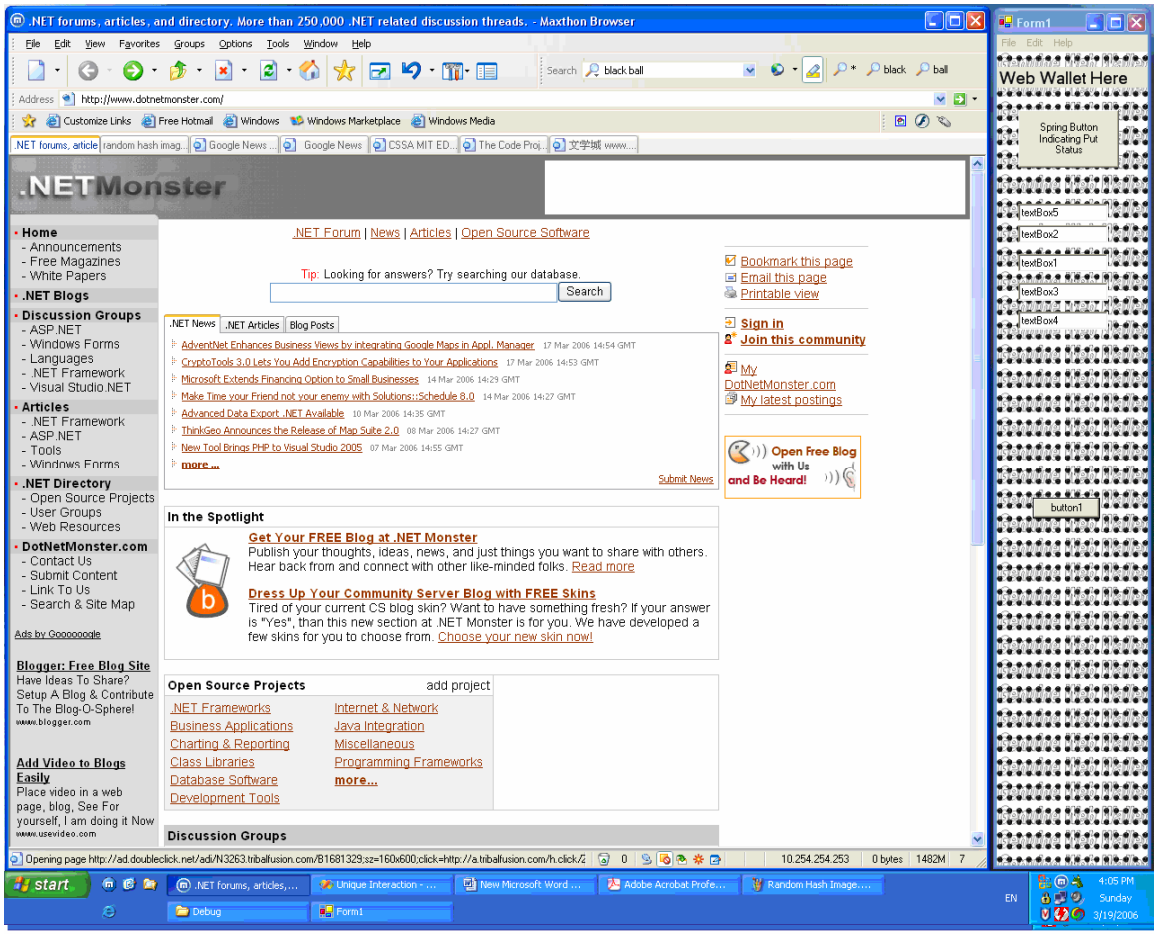


Figure 66. Spring-Loaded Button in Active Mode

Figure 67 Application Trace Prototype



(1) Genuine Skin Prototype 1



(2) Genuine Skin Prototype 2

Figure 68 Genuine Skin Prototypes (on Web Wallet)

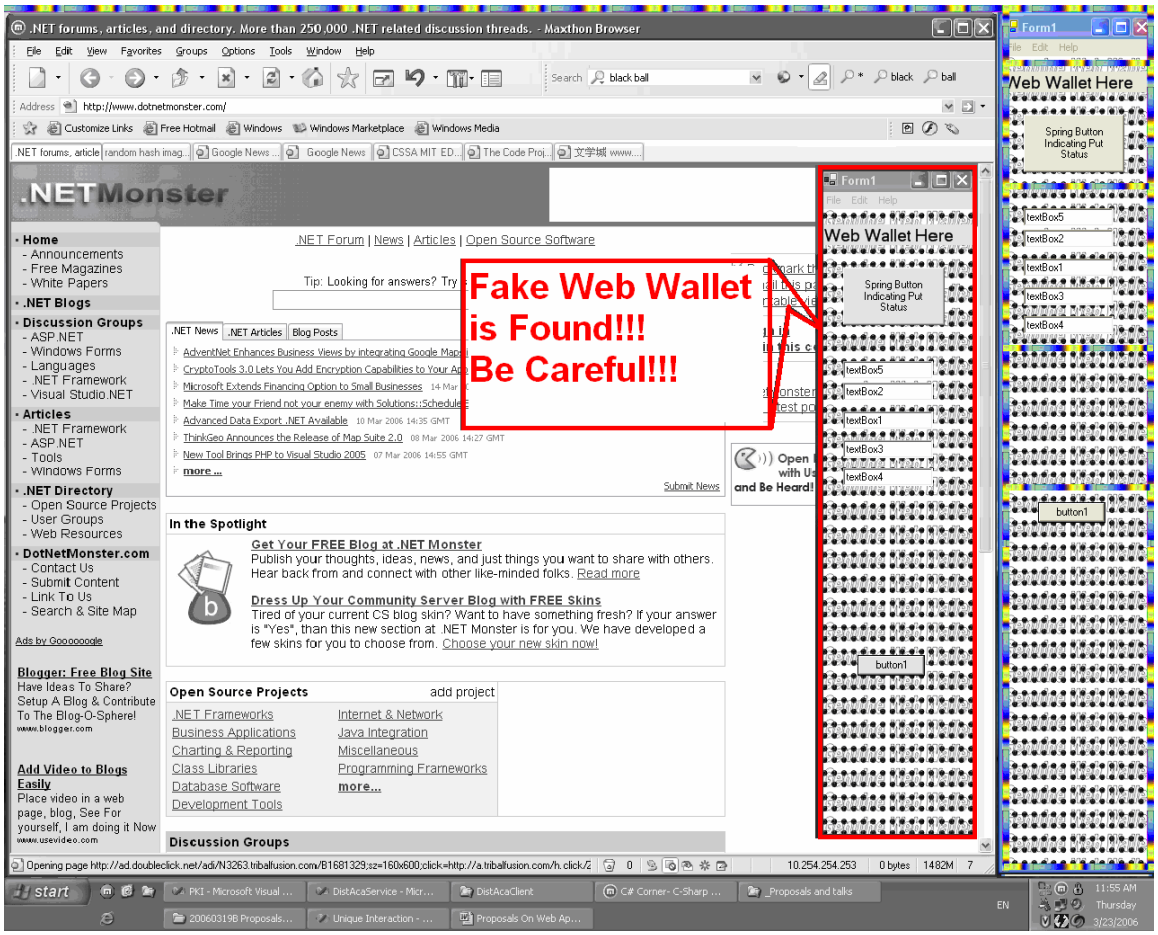


Figure 69 Anti-Screenjacking engine found suspected pattern and gives out alert.

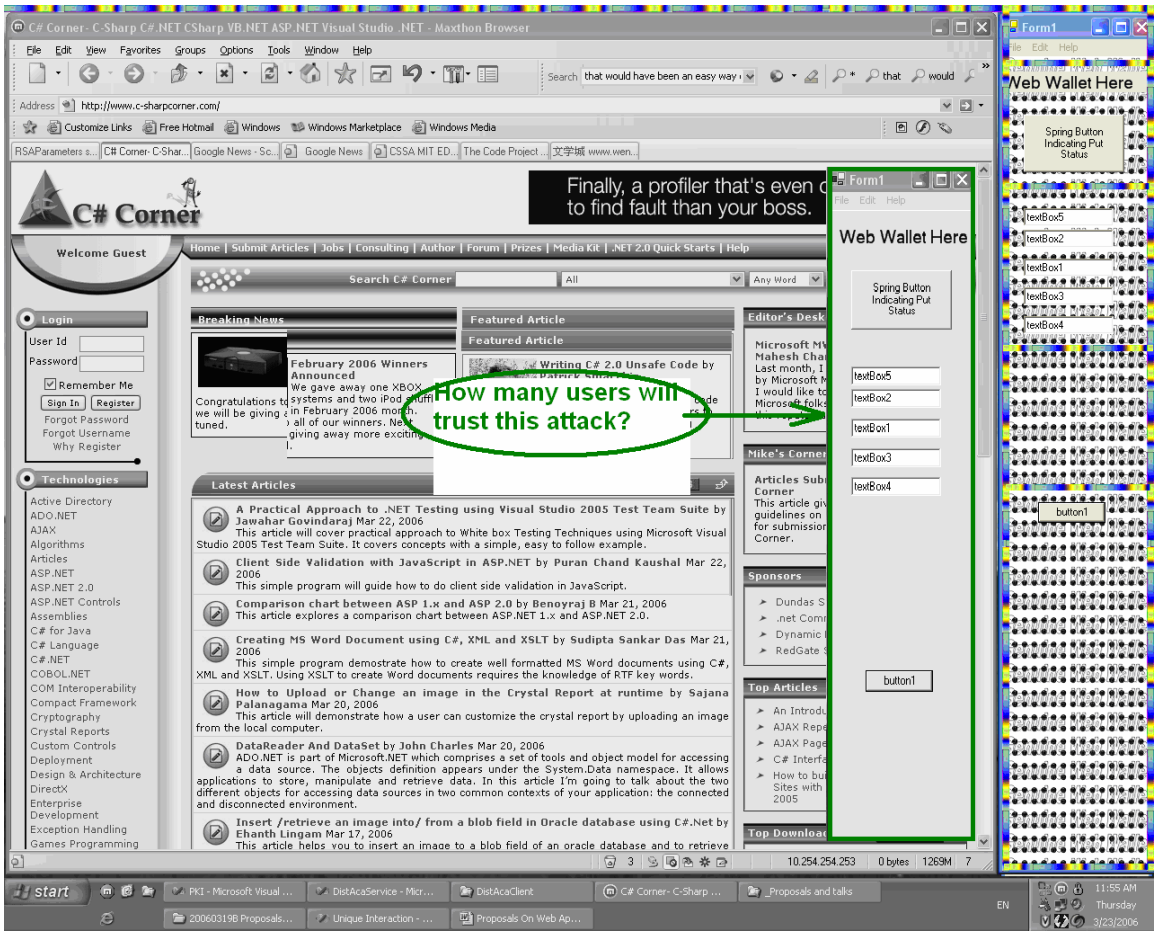


Figure 70 Users can easily recognize the faked Web Wallet

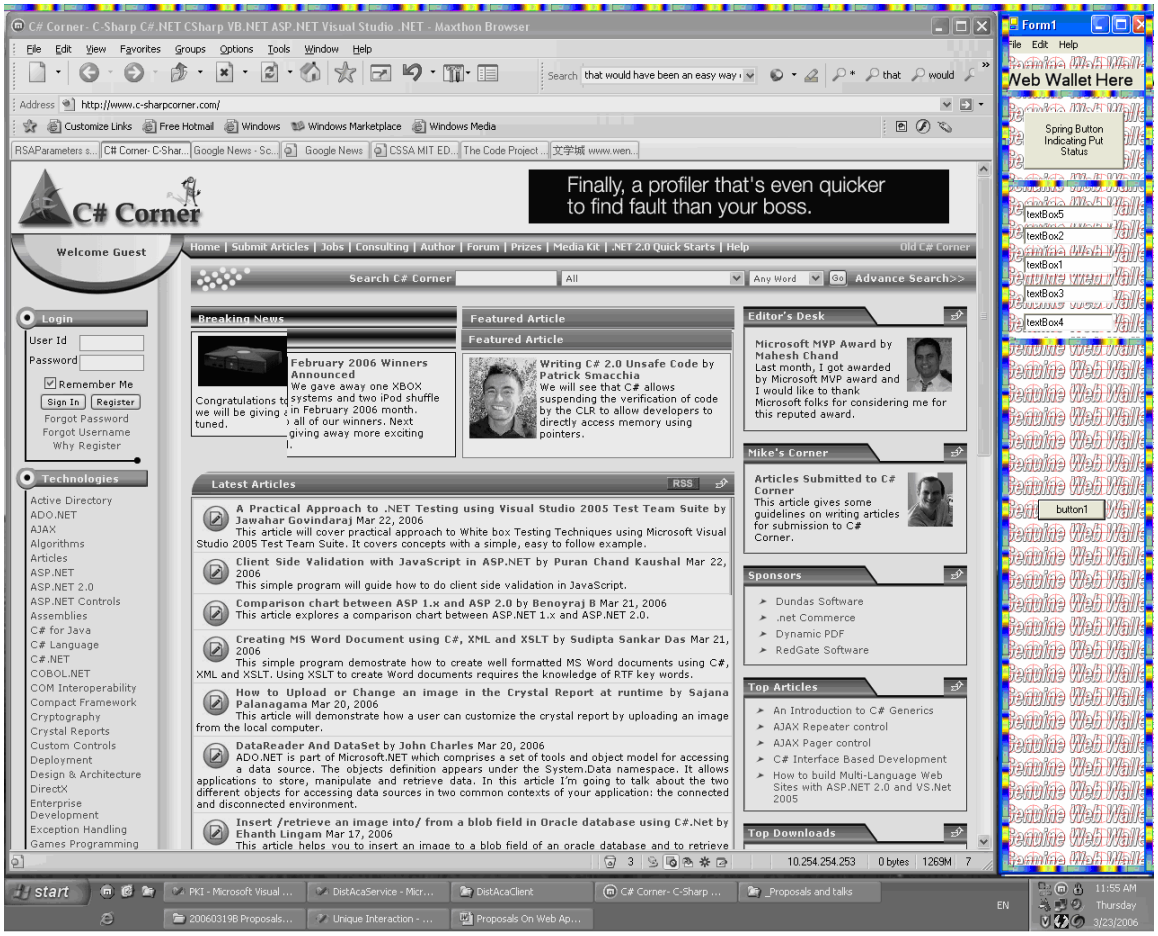


Figure 71 Prototype for merged anti-Screenjacking approach

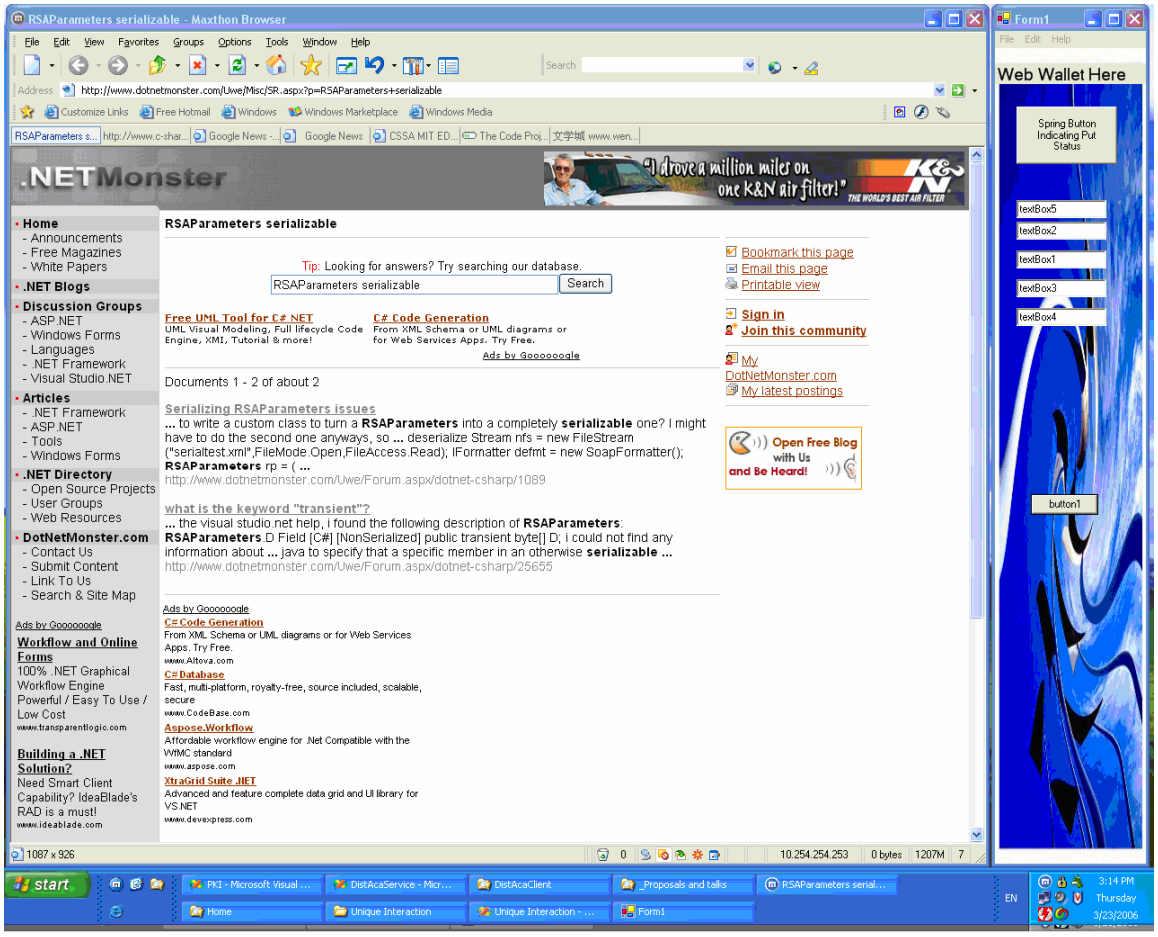


Figure 72 Prototype for interactive image filters (user challenge).

F. Publications

Fu A. Y., Deng X., Liu W., "*REGAP: A Tool for Unicode-based Web Identity Fraud Detection*", to appear in Journal of Digital Forensics Practice, Special Edition on Anti-phishing and Online Fraud, Publisher: Taylor & Francis

Fu A. Y., Liu W., Deng X., "*Detecting Phishing Webpages with Visual Similarity Assessment based on Earth Mover's Distance (EMD)*", to appear in IEEE Transactions on Dependable and Secure Computing

Fu A. Y., Deng X., Liu W., Little G., "*The Methodology and an Application to Fight against Unicode Attacks*", in Proceedings of SOUPS 2006, Jul, 2006, CMU, Pittsburgh, USA

Fu A. Y., Miller R. C., Little G., and Wu M., "*Context Sensitive Password*", in Proceedings of SOUPS 2006, Jul, 2006, CMU, Pittsburgh, USA

Hu B. C., Fu A. Y., and Deng X., "*Future Decryption: Secure the Time Sensitive Information*", in Proceedings of SOUPS 2006, Jul, 2006, CMU, Pittsburgh, USA

Fu A. Y., "*Anti-PhAshing: User Study Design*", in Security User Studies Workshop of SOUPS 2006, Jul, 2006, CMU, Pittsburgh, USA

Fu A. Y., Zhang W., Deng X., Liu W., "*Safeguard against Unicode Attack: Generation and Application of UC-SimList*", in Proceedings of the 15th International World Wide Web Conference, May, 2006

Fu A. Y., Deng X., and Liu W., "*Homograph Attach and Counter Measures*". Book chapter in "Phishing and Counter-Measures", publisher: Wiley, to be published in the third quarter of 2006

Liu W., Deng X., Huang G., Fu A. Y., "*An Anti-Phishing Strategy based on Visual Similarity Assessment*", IEEE Internet Computing, Vol. 10, No. 2, pp. 58-65, 2006. Also appears in IEEE Distributed Systems Online, Volume 7 (4), April 2006

Fu A. Y., Deng X., Liu W., "*A Potential IRI based Phishing Strategy*", in the Proceedings of 6th International Conference on Web Information Systems Engineering, Lecture Notes in Computer Science Volume 3806, pages 618 - 619 , WISE 2005, New York, USA, Nov. 20-22, 2005

Fu A. Y., Liu W., Deng X., "*EMD based Visual Similarity for Detection of Phishing Webpages*", Proceedings of International Workshop on Web Document Analysis, WDA 2005, Seoul, Korea, Aug. 28, 2005

Fu A. Y., Wu X., Fu H., "*Statistics-based Web Cache Prediction Model*", Proceedings of ACM The 6th Postgraduate Research Day 2005, pages 85-93, Poly University of Hong Kong, Mar. 12, 2005

Fu A. Y., Lin D., Liu G., "*Neural Network Enhanced Sound Source Localization Model for Mobile Terminals*", in Proceedings of the 10th Conference on Artificial Intelligence and Applications, TAAI2005, Kaohsiung, Taiwan, Dec. 2-3, 2005

Fu A. Y., Au P., "*Waiting time prediction: an enhancement for call admission control of mobile system using neural networks*", Proceedings of IEEE The 7th International Conference on Advanced Communication Technology, Volume 2, pages 749- 754, ICACT 2005, Phoenix Park, Korea, Feb. 21-23, 2005

Fu A. Y., Fu H., Au P., "*An Integration Approach of Data Mining with Web Cache Pre-fetching*", Lecture Notes in Computer Science Volume 3358, pages 59-63, Publisher: Springer-Verlag, ISPA 2004, Hong Kong, China, Dec. 13-15, 2004