5-1-2008

Using Web bugs and honeytokens to investigate the source of phishing attacks

Craig Michael McRae

USING WEB BUGS AND HONEYTOKENS TO INVESTIGATE THE SOURCE OF PHISHING ATTACKS

By

Craig Michael McRae

A Thesis
Submitted to the Faculty of
Mississippi State University
in Partial Fulfillment of the Requirements
for the Degree of Master of Science
in Computer Science
in the Department of Computer Science and Engineering

Mississippi State, Mississippi

May 2008

USING WEB BUGS AND HONEYTOKENS TO INVESTIGATE

THE SOURCE OF PHISHING ATTACKS

By

Craig Michael McRae

Approved:

Rayford B. Vaughn Billie J. Ball Professor of Computer Science and Engineering (Major Professor)	David A. Dampier Associate Professor of Computer Science and Engineering (Committee Member)
Mahalingham Ramkumar Assistant Professor of Computer Science and Engineering (Committee Member)	Edward B. Allen Associate Professor of Computer Science and Engineering, and Graduate Coordinator

W. Glenn Steele
Dean
Bagley College of Engineering

Name: Craig Michael McRae

Date of Degree: May 2, 2008

Institution: Mississippi State University

Major Field: Computer Science

Major Professor: Dr. Rayford B. Vaughn

Title of Study: USING WEB BUGS AND HONEYTOKENS TO INVESTIGATE THE SOURCE OF PHISHING ATTACKS

Pages in Study: 71

Candidate for Degree of Master of Science

Phishing is the use of social engineering and electronic communications such as emails to try and illicit sensitive information such as usernames, passwords, and financial information. This form of identity theft has become a rampant problem in today's society. Phishing attacks have cost financial institutions millions of dollars per year and continue to do so.

Today's defense against phishing attacks primarily consists of trying to take down the phishing web site as quickly as possible before it can claim too many victims. This thesis demonstrates that is possible to track down a phisher to the IP address of the phisher's workstation rather than innocent machines used as intermediaries. By using web bugs and honeytokens on the fake web site forms the phisher presents, one can log accesses to the web bugs by the phisher when the attacker views the results of the forms.

Key words: anti-phishing, phishing, honeytokens, investigation, tracking, web bugs

ACKNOWLEDGMENTS

I thank my committee for their comments on this thesis, and I thank Dr. Rayford B. Vaughn for directing this research, and also Robert Wesley McGrew for assisting and advising in this research.

I would also like to thank Dr. Edward Allen for the LaTeX template used in creating this document.

Also I would like to thank all of the CSE faculty, graduate students, and all others for their help in providing me with phishing e-mails for this research.

TABLE OF CONTENTS

LIST OF TABLES

CHAPTER 1

INTRODUCTION

Phishing, or password harvesting fishing to some [2], is the use of social engineering and spam e-mails to try to elicit sensitive information, such as usernames, passwords, and financial information. This form of identity theft is a major problem in information security today. Phishing attacks cost financial institutions millions of dollars per year. In a one year period that ended in April of 2004, there were an estimated 1.8 million phishing attacks that generated $1.2 billion USD in losses. When people are victimized by phishing attacks, their bank or other financial institution typically reimburses them for damages just like any other form of identity theft. Even though this institution has done nothing wrong, they are the ones that ultimately pay the price [3].

Financial institutions are also victimized in another way that is often overlooked. When people become victims of phishing attacks or become aware of the threat of phishing attacks, the financial institution loses the ability to communicate with their customer base through email. Their customers no longer trust emails that come from those institutions and usually assume it to be a phishing email [4]. Customers also tend to blame the legitimate institution for phishing attacks.

It is becoming increasingly important to try to protect against phishing attacks. These phishing attacks need to be tracked down to their source, so that attackers can be identified

and stopped. This thesis aims to introduce a new way of investigating phishing attacks to the actual IP address of the phisher's machine, rather than simply the IP address of one of the computers that the attacker has compromised for use in attacks. The set of compromised machines that an attacker uses is known as a botnet. The experiment is explained in section 3.1.

1.1 Introduction to Phishing and Social Engineering

The Anti-Phishing Working Group defines phishing as attacks that "use both social engineering and technical subterfuge to steal consumers' personal identity data and financial account credentials" [4]. The first known instance occurred in the mid-1990s when phishing schemes obtained America Online user names and passwords [12].

Phishing typically involves an email posing as a legitimate company, usually a financial institution. Graphical elements, such as company logos identical to the ones on the institution's web site, increase the apparent legitimacy of the email. The user is presented with a message, that purports to be from the legitimate company, that calls the user to some action. For example, this may be a request for updated information, verification of identity, or a notification something has changed. By combining the problem with the account, the phisher preys on the victim's emotions affecting their ability to process what is happening. When the user accesses the link, it then takes them to a website that looks authentic since the attacker again uses the same graphics and layout as the legitimate web site but is designed to elicit personal information from the user which is sent to the phisher.

Attackers have long been trying to obtain secret information from others. Originally this was done solely through the process of "social engineering". Kevin Mitnick defines social engineering as "getting people to do things they wouldn't ordinarily do for a stranger" [8]. A social engineer tries to deceive a trusted user of a system to reveal secret information or tricks the social engineer would not otherwise know. He describes how the individual is the weakest link of security. Social engineering attacks prey on users who are typically ignorant about good security practices. This is the one attack that technology cannot prevent [8].

To demonstrate how easy this attack can happen, Mitnick tells the story of Stanley Mark Rifkin and his attack on Security Pacific National Bank. In 1978, Rifkin walked into the bank's authorization-personnel-only wire-transfer room. He was working for a company contracted to develop a backup system for the room's data. This gave him access to the transfer procedures. This allowed him to learn that the authorized people received a daily code every morning. These people did not bother memorizing the code. They instead posted it inside the room. Rifkin was then able to call the room from a pay phone posing as Mike Hansen, a member of the bank's International Department. After giving an office number and the code he was able to give instructions to wire ten million two hundred thousand dollars to a bank account he had already set up. The wire room then asked for the inter-office settlement number. Rifkin did not anticipate that question and therefore did not have the answer. He simply stated that he would check the number and call right back. Rifkin then changed hats and called the International Department posing to be the

wire room and asked for the number. He then called the wire room back and finished the transaction [8].

Today's social engineer does not require direct communication with their victim. Thanks to the widespread use of the internet, social engineering is easier than ever before. A social engineer can victimize anyone from anywhere. Also, in Mitnick's story, Rifkin had to keep his nerves and wits about him in order to not give away the scam. With the Internet, the social engineer does not have direct contact with his victim. The social engineer does not have to worry about the inflection in his voice or any other physiological clues giving away the scam. The interpersonal skills used in attacks such as those described by Mitnick are not necessary to execute phishing attacks on the Internet.

Dr. Max Kilger believes that the motives of computer attackers can be explained with the acronym MEECES. This stands for Money, Ego, Entertainment, Cause, Entrance to a social group, and Status [9]. The CEO of Cyota, Naftali Bennett, believes that phishing has become popular because as it stands right now, there is little risk but potentially great rewards [3]. The risk is low due to the lack of effective methods for tracking down phishing attacks to their source and the low probability of being prosecuted. Most phishing investigative techniques can only track the attack to the server where the phishing site was hosted or to a botnet used to relay the emails. The server hosting the site is usually a compromised server that has no direct connection to the attacker.

1.2 Introduction to Web Bugs

A web bug can be defined as any HTML element that is used to, at least in part, secretly gather information about users who navigate to a site that contains that element. Usually these come in the form of images. For that reason, they are sometimes referred to as pixel tags and clear GIFs [6]. These images could be as large as the web site's logo or as small as one square pixel.

David Martin, et al. mentions several different things that web bugs can do. A few of these include:

- Count web site hits
- Send demographic data to a internet marketing company such as gender, age, and zip code with the help of a cookie
- Track web browsing habits of a user
- Report browser information back to a web site [6]

Web bugs can accomplish some of this by explicitly giving an image a unique file name for that particular web user. An example of a web bug in use can be seen in emails. This technique has been used by spammers in the past and by a company sending out legitimate promotional emails as well. A 1x1 pixel white image can be created with a particular email address embedded into the filename such as joesmith_example_com.gif. The spammer will then embed this image into that email to joesmith@example.com. When Joe Smith views his email, the small white image will go unnoticed. The image is loaded from the spammer's web server, and will create an entry on the server log for the filename joesmith_example_com.gif. The spammer now knows that Joe Smith received his email and will continue to use that email address in future spam attacks. Some web-based email

providers and vendors of desktop email software have disabled the retrieval of images in incoming emails for this reason.

1.3 Introduction to Honeytokens

Lance Spitzner, the leader of the Honeynet Project, has defined the term honeytoken as "a digital or information system resource whose value lies in the unauthorized use of that resource" [10]. Honeytokens can be any digital data. They can consist of documents, images, or even data such as a phony login/password combination. A honeytoken may be any data on a system for which accesses can be logged, and whose access automatically implies unauthorized activity.

While the term 'honeytoken' is new, the concept is not. The term 'honeytoken' was created by Augusto Paes de Barros on February 21, 2003. He used it in a email that went to a list of security professionals [11]. Spitzner further mentions that some map makers insert phony roads or cities on their maps for proof when competitors sell copies of their maps. Spitzner gives other hypothetical examples of honeytokens in use. One such example shows how a honeytoken could possibly help in database security. Hospitals could create a bogus medical record for John F. Kennedy and then track the access to that tuple of the database. Anyone who views that record is in violation of regulations and rules that are meant to protect the privacy of patient data. This is because stored procedures and other layers of database access can be designed to avoid accessing honeytokens. In this situation, access to a honeytoken implies that the database is not being accessed through approved

means. Also, financial institutions can create bogus accounts. If one tries to access those accounts, then the institution's system has been compromised [10].

The key to using honeytokens is to give the token unique identifiable elements to guarantee that the only access to that token is by unauthorized parties. If the token could be viewed in normal interaction with a system, the token's tracking ability is compromised. Honeytokens' greatest advantages lie in their flexibility and their minimal cost.

1.4 Hypothesis and Methodology

This research combines the concepts of web bugs and honeytokens and uses them against phishing attacks. Web bugs are sent to phishers via the web forms used in attacks. Since these web bugs are named specific to each phishing attack and should not be accessed in normal web browsing, then they also function as honeytokens. These web bugs should only be accessed by the phisher and no one else. This leads to the following hypothesis:

> The combination of HTML web bugs and honeytokens can be used against phishers to track the phishing attacks to their source thus being able to help stop phishing where the problem starts.

This research is not simply testing a phishing investigation technique, it looks at phishing attacks as a whole. This will lead to new discoveries beyond the success of the phishing investigation technique. Other goals of this research are to answer the following questions:

1. What methods and motives of phishing attacks can be discovered?
2. What other vulnerabilities are available to use against phishing attacks?
3. Can new phishing defenses be uncovered?

In addition to investigating phishing attacks, this research aims to develop a profile for phishing attacks similar to what honeypots have done for general network security. By walking through their entire attack and investigating where the results are being viewed, lots of information can be discovered about a phisher. This could help in developing a profile on typical phishing attacks.

This investigative method does has its shortcomings which will be discussed later. This is why discovering other vulnerabilities in phishing attacks can lead to a much more effective phishing investigation strategy. This is based on the totality of computer security. When a person only secures one aspect of their system, then an attacker simply uses another method. If you only attack one of a phishing scheme's vulnerabilities, then they will eventually fix the vulnerability, reducing the effectiveness of the attack.

This research presents a new investigative technique, rather than a defense against phishing attacks. However, by exploring the totality of phishing attacks, profiling the attackers, and exploring the vulnerabilities of phishing attacks, it is possible that other methods of phishing investigations and defenses could be developed.

CHAPTER 2

CURRENT EFFORTS

2.1 Defenses Against Phishing

There are currently numerous approaches to protecting users from phishing schemes. As previously mentioned, these approaches do not get to the root of the problem. These methods do not track down the phishing schemes to the individual or group running the scam.

The simplest of these methods is to attempt to take down phishing sites quickly. Web crawlers similar to those used for search engines can also be used to find phishing sites. The information is then passed on to the appropriate Internet Service Provider (ISP) in order to take the site down. This method is flawed because of the difficulty in finding these sites quickly. Also, some websites hosted in foreign countries do not have similar laws that can justify removing the site.

One defense is to flood a phisher's database with false information. This flood of information is not intended on being a denial of service (DOS) attack. This flood is designed to make the phisher unable to distinguish correct data from incorrect data in their database. This makes the user's database virtually unusable. This method does nothing to prevent

phishing attacks. It simply tries to minimize the ability to carry through with the theft of financial data [3].

Another defense is to provide a two-phase authentication process for users in e-mail and on the corporate website. An example of this would include the normal username/password combination combined with a authenticating image and/or phrase chosen by the user. Any time the corporation sends out an e-mail to a user, they would include this personal phrase and image in the e-mail. Since a phisher would not know this information, then the e-mail is validated. By the same token, including the image and phrase on the corporation website as part of the login process would also validate the website to the user as well. This can help prevent phishing attacks by providing authentication of communication between a user and a corporation [3].

One of the few current approaches that can possibly target the root of the problem is to watch corporation's web logs for users downloading their images. Creators of phishing websites usually use the actual images from the corporate website to make the phishing website more believable. If users are downloading images to their personal computer, then their IP address will show up in the server logs. The Corillian Fraud Detection System (CFDS) is a commercial server that looks for such a behavior in web logs. It then investigates further to find the phishing site that is illegally using those images. Corillian then notifies the administrator of the compromised server and the authorities [3].

Internet browser creators are trying to help out as well. This usually involves comparing the address of a web site to a list of 'blacklisted' websites. This blacklist can come from the browser's creators, users, and other sources. The sites may also have a list of

'whitelist' websites as well. If a site does not apply to either list then some security measures may be put in place such as disabling mobile code like ActiveX and JavaScript [3]. Internet Explorer 7.2 disallows mobile code to disable the address bar. Phishers often do this to hide the actual location of the site. They can even include a fake address bar showing a fake URL to further fool the user. This version of Internet Explorer also includes a phishing filter which allows users to report a phishing site. This reporting eventually can lead to the site becoming blacklisted and all users of IE are notified at any future visits to the site. Users are notified of both confirmed and suspicious phishing sites [7].

2.2 Uses of Honeytokens and Web Bugs

Both honeytokens and web bugs are still relatively new ideas. The full potentials of honeytokens and web bugs have yet to be realized. Especially in the case of honeytokens, most ideas are still in the hypothetical stage.

One early example of the use of honeytokens occurred in 1986. Clifford Stoll, working as a programmer for Lawrence Berkeley National Laboratory, placed phony records for a fictitious organization named Strategic Defense Initiative Network deep into the lab's server. After the files were downloaded by intruders, Stoll received a letter about the company. With the help of federal investigators, Stoll traced the intruders to East German and Soviet intelligence agencies. ForeScout Technologies actually uses honeytokens to help as an intrusion detection system (IDS). Their software detects 'surreptitious reconnaissance', i.e. port scans. Then the software announces a false message of vulnerability to

the intruder. If the intruder follows up with an attack, then the connection is terminated [11].

ForeScout Technologies' use of honeytokens demonstrates another advantage. When used with an IDS, it can help eliminate false positives. When an IDS is designed to trigger on honeytokens as opposed to normal rules, then false positives can be minimized, and more intrusions can be detected.

As previously mentioned web bugs can be used for primitive e-mail tracking. This technique could be applied to track down the source of the 'Nigerian Scam', or '419 scam', emails that try to convince users to supply their bank account information in order to receive a large charitable donation. These emails want you to reply to the e-mail to swap personal information. A web bug picture could be simply added to the reply in order to try and investigate the source.

Web bugs are not just limited to web pages and e-mail. Web bugs can also be hidden in Microsoft Word documents. Microsoft Word allows a document to link to an image on a remote server. The document only stores the link to the image and not the image itself; therefore, every time the document is opened, the image has to be loaded from the remote server. Harding, et al. lists a few instances where web bugs would be useful in Microsoft Word documents:

- Discovering and keeping up with any possible leaks of confidential documents
- Keeping up with any possible copyright infringement of newsletters and reports
- Keeping up with the distribution of a press release
- Tracking quoted text that is copied and pasted from one word document to another.

Harding adds that web bugs can also be used in Excel 2000 and PowerPoint 2000 files [5].

The biggest security risk with web bugs is when they are combined with cookies to obtain information about a user. These two tools combined are a threat to internet privacy and anonymity. This threat is realized when cookies from two or more sites belonging to the same ad agency are stored on the same computer. These sites each contain web bugs that point back to the ad agency. When a user visits one of those sites, the web bug will be loaded and information will be sent back to the host server. Included in this information would be a previously set cookie value from the user's hard drive. When the user then goes to another site, the ad agency can cross-reference the visits to the same user based on the cookie's values that are sent with the web bugs. The ad agency that can begin to gather information on the user's browsing habits within their ad network [5].

The use of web bugs by websites is much more prevalent than most people realize. David Martin, et al. did a study on web bugs. In their study they analyzed two lists of websites. One list contained 84 web sites that were marked as the most popular in January 2000. The second list contained a random list of 298 consumer-oriented web sites. The study used Bugnosis to analyze an average of approximately 90 pages from each website. This study found 58% of 'the popular websites contained web bugs and 36% of the consumer-oriented websites [6].

CHAPTER 3

PRELIMINARY EXPERIMENT

3.1 Initial Experiment

Using web bugs and honeytokens, this research attempts to describe a method for tracking down phishing attacks to the source. This involves filling out the phishers' web forms with HTML image tags of one square pixel images and HTML web page links. Both the image files and the web page links were hosted on a machine in a laboratory environment with an unrestricted internet connection. This machine was on its own dedicated subnet separate from the rest of the campus network. In the event that a phisher decides to attack the research system, this protects the departmental and campus network from being compromised. The images in this project can be considered web bugs since they are named uniquely for each phishing e-mail and can be used to gather information about the individual or group that views the data collected from the phishing scheme. Both the image files and the HTML files can also be viewed as honeytokens since they are pieces of data that are unlikely to be accessed by users other than those involved in specific phishing schemes.

The image and HTML files are both named using a timestamp in order to give them a unique name. That filename is stored, along with the URL of the attack, to associate the

files with each particular phishing attack. As previously mentioned, if the phisher views the results in an HTML enabled environment that does not filter or block third party images from being loaded, such as many webmail applications, browsers, and applications such as Outlook, the images will be retrieved from the server unknowingly by the attacker. The phisher will likely know that something is not right with the data due to the links being displayed and the lack of personal information. By the time the phisher realizes this, the image is already loaded. Any further investigation by the phisher will likely result in more information being logged about them. If the phisher's client software loads the images or the phisher accesses the hyperlinks, a referral will be generated in the web logs on the tracking server. Some of the information included in the log file is:

- The IP address of the actual box where the results were viewed
- The webpage where the phisher viewed the results including webmail accounts like GMail, Yahoo Mail, or Hotmail
- The browser type that was used to view these results
- A guess at the operating system the phisher is working from

An analysis of an actual experimental referral that was generated is shown in Section 3.3.

3.1.1 Example Case

In order to test this concept at tracking phishing schemes, two simple webpage forms were created for testing the process. One form to test the 'get' form request, and the second to test the 'post' form request. Both of these files are shown in Appendix B. Most phishing

attacks begin with a login page, so the two forms were created as login forms for testing. The form was then fed HTML tags similar to the following:

- myspam@hotmail.com
- <img src=``http://192.168.0.0/images/ 2005_Nov_28_11_20_05.gif''>

Lastly, a web page was created that displayed the results of the fictitious phishing site. This file is shown in Appendix B as well. This web page was opened up in a web browser that allowed the viewing of third-party images. To guarantee that a referral had been made, some of the web page links were accessed as well. At this point the web server log files on the experimental machine were checked, and it was confirmed that the referral was made from the phishing investigation.

3.1.2 Real World Cases

E-mails were sent to all the faculty and grad students in the CSE department asking them to submit phishing e-mails they had received for use in the experiment. Other individuals that were capable of identifying phishing schemes were asked to do the same.

Originally the plan was to automate the process with a single Perl script or series of scripts. The scripts used the Unix utility wget to automate the form submissions. A problem arose when it was noticed the script was not able to handle redirect pages in the case of nested forms. Due to the limited time available for this test case, it was decided to continue doing the form submission manually to achieve results and to prove tracking was possible.

The Perl script used to automate this process was modified to simply take the input of a particular web site URL, generate the appropriate image and HTML files, and store the web site URL with the image and HTML filenames into a comma-separated values (csv) file. The HTML values could then be copied and pasted into the individual fields of the phisher's forms. If the entire HTML tag could not fit into a form field, then the values were simply created with fictitious names, or in the case of number fields such as credit card numbers, the appropriate number of 5's were input into the field. This problem can be bypassed once the process is automated. An automation script could bypass the HTML form restrictions and any possible JavaScript-based error checking. Another alternative solution is to use a proxy server designed for web application security testing, such as BurpSuite [1]. This will allow the outgoing data stream to be filtered in order to allow the HTML tags to be entered into the outgoing data. After submitting form data, the packet can be changed to include the HTML tags. This would also bypass the HTML form restrictions and JavaScript-based error checking.

During the week of November 11 through November 18, eleven phishing sites were investigated and two referrals were found. Some of the other nine attacks may not have generated referrals since the only fields that would accept the full HTML tag were the login fields. If the phisher was not worried about the login parameters and only looked at the financial data from later forms, then the referral would not occur. In some cases, none of the fields would accept the full HTML tag. This problem could again be fixed by script automation or by the use of a proxy server. A sanitized version of one of the referrals that was generated during this week is shown below:

```
82.79.137.22 - - [11/Nov/2005:12:08:12 -0600] ``GET
/images/2005_Nov_12_11_52_53.gif/ HTTP/1.1'' 404 309
``http://mail.google.com/mail/h/15zatsmc8x2ql/
?th=1078083bcd28d7d6&v=c'' ``Mozilla/4.0
(compatible; MSIE 6.0; Windows 98) Opera 7.50  [en]''
```

We can infer the following from this referral:

- The results of the attack were e-mailed to the attacker via a GMail e-mail account. Attempts to reproduce the GMail session by entering the URL into a browser were unsuccessful. The user would have had to allow the images to be loaded since GMail does not load images by default.
- Using the IP address, a whois query can be executed, possibly revealing information about the attacker's physical location, or another machine or proxy under the attacker's control.
- The Opera web browser was used by the attacker to view his information.

Actually tracking the source of the phishing scheme occurs with a whois query of the IP address given in the referral. Using the IP address from the previous referral returned the following:

```
inetnum:        82.79.137.0 - 82.79.137.63
netname:        RO-BZ-METRONETWORK
descr:          Metronetwork SRL
country:        RO
admin-c:        IS1460-RIPE
tech-c:         IS1460-RIPE
tech-c:         RDS-RIPE
status:         ASSIGNED PA
remarks:        +----------------------------------------+
remarks:        | ABUSE CONTACT: abuse@rdsnet.ro IN CASE |
remarks:        | OF HACK ATTACKS, ILLEGAL ACTIVITY,     |
remarks:        | VIOLATION, SCAMS, PROBES, SPAM, ETC.   |
remarks:        +----------------------------------------+
mnt-by:         AS8708-MNT
mnt-lower:      AS8708-MNT
source:         RIPE # Filtered

role:           Romania Data Systems NOC
```

```
address:        71-75 Dr. Staicovici
address:        Bucharest / ROMANIA
phone:          +40 21 30 10 888
fax-no:         +40 21 30 10 892
e-mail:         contact-tech@rdsnet.ro
admin-c:        CN19-RIPE
tech-c:         CN19-RIPE
tech-c:         GEPU1-RIPE
nic-hdl:        RDS-RIPE
mnt-by:         AS8708-MNT
remarks:        +----------------------------------------+
remarks:        | ABUSE CONTACT: abuse@rdsnet.ro IN CASE |
remarks:        | OF HACK ATTACKS, ILLEGAL ACTIVITY,     |
remarks:        | VIOLATION, SCAMS, PROBES, SPAM, ETC.   |
remarks:        +----------------------------------------+
source:         RIPE # Filtered

person:         Iordache Silviu
address:        Str Traian Vuia, bl 16,sc. C, et3, ap.54
address:        Buzau / Romania
phone:          +4-0744821745
fax-no:         +4-338401143
e-mail:         iorsior@yahoo.com
nic-hdl:        IS1460-RIPE
mnt-by:         AS8708-MNT
mnt-by:         AS8708-MNT
remarks:        +----------------------------------------+
remarks:        | ABUSE CONTACT: abuse@rdsnet.ro IN CASE |
remarks:        | OF HACK ATTACKS, ILLEGAL ACTIVITY,     |
remarks:        | VIOLATION, SCAMS, PROBES, SPAM, ETC.   |
remarks:        +----------------------------------------+
source:         RIPE # Filtered

% Information related to '82.76.0.0/14AS8708'

route:          82.76.0.0/14
descr:          RDSNET
origin:         AS8708
mnt-by:         AS8708-MNT
source:         RIPE # Filtered
```

From this whois query, quite a bit of information can be gathered:

- The entire range of IP addresses that this organization owns is shown.

- The actual street address, city, state and zip of this organization that owns that IP address is given. In this case, the owner of this IP address resides in Bucharest, Romania.
- Phone numbers for the organization are given.
- A e-mail contact for general communications is provided.
- Another e-mail contact where abuse is reported.
- A named contact who is a Senior IP engineer in this organization is given with his e-mail and phone number.

At this point, the investigation moves to the ISP that owns the particular IP address. In order to trace the source of the IP, the ISP would have to check their logs to try and determine what machine was logged in at the particular IP address at a given time.

3.2 Technique Advantages and Disandvantages

There are limitations of tracking by IP addresses. Many ISPs use dynamic IPs with their customers. Every time a machine logs on to the ISP's network, they could have a different IP address. This makes it very difficult to determine who had a particular IP address at a particular time. Some ISPs do not actually log IP assignments; therefore, the machine that was using a particular IP at a particular time cannot be determined. Other limitations include the use of public wireless access points at coffee shops and other public venues, the use of anonymous proxies which hide the actual IP address of a particular machine, and cooperation of foreign ISPs in providing information on the possible phisher.

Another disadvantage with this technique is that the images may not be loaded by the phisher when he checks his results. If the results are viewed in a non-HTML environment, then the images will not be loaded, and no log entry will be generated. Also many web

based e-mail, which it is believed the phishers' use, allow the blocking of images in e-mails. An e-mail user might have to explicitly tell the browser to load the images.

3.3 Experiment Conclusions

In conclusion, this trial experiment did show that this investigative method could be a valid way of tracking down phishing attacks. With this type of investigation, the attacks can actually be traced to the source of the problem in some situations.

Since most of the data in this project came from members of the CSE department both students and faculty, a very limited variety of phishing attack methods were seen. A larger variety in phishing attack methods would more thoroughly test the investigation method.

CHAPTER 4

FINAL EXPERIMENT

4.1 Experiment Setup

This experiment was run from June 20, 2007 through October 5, 2007. Fifty-one sites were investigated using emails forwarded from Mississippi State University Computer Science and Engineering faculty, grad students, BullyLUG (the local Linux Users Group), others capable of identifying phishing emails and web page links from the web site http://www.phishtank.com.

Initially phishing sites were gathered solely from e-mail traffic. This was proven to be an ineffective method due to the lack of new phishing emails received. When the e-mail was received, the website URL, and the company that was being phished was recorded. Once the site was loaded, the IP of the site compromised was recorded along with the country of the ISP hosting the compromised site. The honeytokens were manually entered into the site if possible. If not, the values were entered after submitting the data via a web proxy. While browsing the site and filling out the forms, anything unique about the site was recorded along with the scripting language used with the forms.

Ultimately acquiring phishing sites via email traffic was inefficient and did not produce enough data. PhishTank.com was given as another source for phishing attacks. The site

allows users to submit phishing sites. Registered members to the site can confirm or reject the site as a phishing site. Once enough votes are gathered, the site is either confirmed or rejected. Users can also submit a site to see if it has already been submitted, confirmed, or rejected as a phishing site. This site was then used to obtain links to phishing sites for the remainder of the experiment. Data was still collected in the same manner.

The following three tables show a breakdown in the countries hosting compromised servers, companies that were phished, and the scripting languages used on these sites. The phishing sites investigated revealed nineteen different country locations. Out of these nineteen countries, the top four countries with phishing sites were the United States with fifteen, Korea with six, and Australia and the Netherlands with five. These sites also revealed eighteen companies being phished. The top four companies phished were PayPal with thirteen, Ebay with eleven, and Bank of America and Citizen's Bank with five. PHP was an overwhelming majority among the scripting languages used. Forty-two of the fifty-one sites used PHP. Nineteen of those forty-three used JavaScript for their error checking.

The web server logs were checked every few minutes after the investigation for the first 20 - 30 minutes and then ever few hours to determine if a referral had been made. Once a referral was found, the referral was recorded along with the IP of the phisher, country of origin of the IP, website viewed (if applicable), OS, browser (if applicable), and turn around time.

Table 4.1

Countries with Compromised Web Servers

Country of Compromised Server	Count
United States	15
Korea	6
Australia	5
Netherlands	5
Taiwan	3
Germany	2
Russia	2
United Kingdom	2
Bahrain	1
Canada	1
Denmark	1
Ecuador	1
France	1
Japan	1
Mexico	1
Portugal	1
San Salvador	1
Serbia and Montenegro	1
Thailand	1

Table 4.2

Companies that were Phished

Companies Phished	Count
PayPal	13
Ebay	11
Bank of America	5
Citizen's Bank	5
Capital One	3
NatWest Bank	2
Wachovia	2
Amazon	1
Bank of the Cascades	1
BankTennesee	1
Citadel FCU	1
Citibank	1
E*TRADE	1
Halifax	1
MidAmerica Bank	1
NetBank	1
Yahoo Mail	1

Table 4.3

Scripting Languages that were used

Scripting Language Used	Count
PHP	23
PHP & JavaScript	19
JavaScript	4
ASP	2
CGI	2
ASP & JavaScript	1

4.2 Results

Part of the research goals was to create automation scripts to perform the investigation. Initial scripts were created using the Python programming language. Some basic testing was done on the scripts with simulated phishing sites initially with great success. Once the script began to be used on live phishing sites, new reasons for the script to fail were found with each email. Some of these methods include, but are not limited to:

- Encoded HTML files
- Some types of HTML and other script redirects
- HTML tags spread over multiple lines

Initially, the script was modified to handle changes that were discovered, but each phishing email brought new bugs. During these testing runs, a referral was never generated from any site. Eventually it was determined that the automation script was too large of a project to handle an efficient variety of phishing sites to be useful in this investigation. It was deemed that the final data was more important than the automation process. The only way to guarantee the result data was to run the investigation manually. Therefore, the final investigation was done manually as outlined in the real world preliminary experiment discussed in Section 3.1.2. The automation script is included in Appendix C as a starting point for any future work.

The investigation set out to accomplish two different tasks. First and foremost was to quantify the ability of the investigation technique to obtain a single source IP of the phisher. The second task was to begin to develop a profile of a typical phisher for use in developing further investigation or protection techniques.

For the the first task, fifty-one sites were investigated with thirteen returning at least one HTML referral for a 25.5% success rate. The peak success rate over the course of the investigation was approximately 27 - 28%. The full details of the sites investigated and the referrals generated are listed in Appendix E. Of the thirteen sites, eight had referrals from the html links, eleven had referrals from the image links, and six had referrals from both. Among the seven that did not have referrals from both the images and the HTML links, five were image referrals.

As can be seen, most of the referrals were generated by the image links which was expected. The number of referrals from HTML links does lessen the concern of images being blocked by browsers or email applications. This gives further hope that the referral percentage can be increased as the method is refined.

In order to begin to profile the typical phisher, several factors were gathered from the HTML referrals and the sites being investigated. These factors are:

- Factors listed in previous section
- Location of phisher
- Browser used by phisher
- Operating system used by phisher

Phishers were indentified from only 3 countries. United States and the Netherlands each had 6 phishing attacks traced back to them, with Romania being the third country. Most of the United States IP belonged to IP addresses owned by AOL. This could be phishers who used free AOL hours from CDs to carry out their phishing attack. This could make tracking down the phisher difficult depending on the accuracy in the details registered with the account.

Table 4.4

Countries of IPs Used by Phishers

Countries of IPs Used by Phishers	Count
Netherlands	6
United States	6
Romania	1

Firefox and Internet Explorer came back as the only browsers used by phishers. Internet Explorer led with eight referrals, with Firefox having the other five. These results were fairly expected due to the vast popularity of these two browsers.

Table 4.5

Browsers Used by Phishers

Browsers Used by Phishers	Count
Internet Explorer	8
Firefox	5

Windows was an overwhelming choice of operating system. Twelve of the thirteen referrals used it as their Operating System, with Linux taking the other referral.

The results of this investigation show a typical phisher was using an ISP in the United States to hack a U.S. webserver. The phisher will then use that web server to host either a PayPal or Ebay website scripted with the PHP language and possibly JavaScript error checking. The phisher will then view these results using a computer with the Windows operating system and the Internet Explorer web browser.

Table 4.6

Operating Systems used by Phishers

Browsers Used by Phishers	Count
Windows NT 5.1	12
Linux	1

CHAPTER 5

CONCLUSIONS AND FUTURE WORK

5.1 Conclusions

This chapter returns to the hypothesis and research questions detailed in section 1.4. The data outlined in the previous section is used to either confirm or refute the hypothesis. Also, the data is used to answer the original research questions.

The hypothesis from section 1.4 was given as follows:

> The combination of HTML web bugs and honeytokens can be used against phishers to track the phishing attacks to their source thus being able to help stop phishing where the problem starts.

In addition the research questions were as follows:

1. What methods and motives of phishing attacks can be discovered?
2. What other vulnerabilities are available to use against phishing attacks?
3. Can new phishing defenses be uncovered?

The hypothesis was proven in this particular experiment by the fact that 1 out of every 4 phishing attacks were successfully traced back to a single IP source address. With additional work on automated scripts, we believe this percentage could improve.

As addressed in section 4.2, the typical phisher method used here was to use the scripting language PHP to handle client-server interactions and to create the phishing website

and JavaScript to handle error checking on the phishing site. The phisher would then check for results using a Windows operating system and the Internet Explorer web browser.

The motives of the phisher varied greatly among the sites investigated. The level of detail in information that the phisher attempted to acquire varied from basic credit card information to asking for social security numbers, checking account numbers, routing numbers, and even the mother's maiden name. Ten of the fifty-one phishing sites only attempted to acquire username/password combinations from potential victims as opposed to asking for more detailed financial information.

Other vulnerabilites and new phishing defenses can possibly be uncovered by looking at the tools used by the majority of phishers. Security vulnerabilities particular to the Windows operating system and the Internet Explorer web browser could be leveraged against phishers in order to attack vulnerabilites of the phisher's host. For possible defenses a larger database of phisher IP addresses could be used to create black lists within web browsers to block access to potential phishers. This black list could also be used by financial institutions as a red flag if account information is accessed from one of these IP addresses. Other possible vulnerabilites and defenses could possibly be inferred from the results information detailed in Section 4.2.

5.2 Future Research

Future work on this project should go further than just the HTML form of phishing schemes. By looking at the server that was compromised to host the attack, vulnerabilities that were exploited could be determined.

While looking at different phishing schemes, it was noticed that the directory structure of the schemes were very similar. This is due to the fact that phishers use premade 'kits' to set up their schemes across a large number of systems. By communicating with administrators of the compromised host after a phishing site is taken down, these phishing kits could be recovered and studied for other insights into phishing schemes. In addition to insights into phishing schemes, the phishing kits could be studied for exploits and other vulnerabilities in their code.

Web bugs are not the only way to exploit web browsers and other client software used for viewing results of phishing schemes. Other exploits in these types of software can be studied as well to use in tracking down the source of phishing schemes. As discussed in the previous section, vulnerabilities of a particular operating system, web browser, or scripting language could be explored as futher attacks against phishing as well as future defenses using other results data.

As mentioned previously, automating the process was unsuccessful. If this were addressed in future work, it is possible that greater success could be achieved. This would allow the process to be used on a larger scale against a much larger data set.

Lastly, this process could possibly be applied to other computer security areas as well. One example of this would be to embed web bugs into a Nigerian 411 scam reply e-mail.

REFERENCES

[1] "PortSwigger.net - web application security," 2007, http://portswigger.net/suite/index.html (current Oct. 22, 2007).

[2] T. H. P. . R. Alliance, "Know Your Enemy: Phishing Beyond the Scenes of Phishing Attacks," 2005, http://www.honeynet.org/papers/phishing/index.html (current Feb. 6, 2005.

[3] D. Geer, "Security Technologies Go Phishing," *Computer*, vol. 38, no. 6, June 2005, pp. 18–21.

[4] G. Goth, "Phishing Attacks Rising, but Dollar Losses Down," *IEEE Security & Privacy Magazine*, vol. 3, no. 1, January-February 2005, p. 8.

[5] W. T. Harding, A. J. Reed, and R. L. Gray, "Cookies and Web Bugs: What They Are and How They Work Together," *Information Systems Management*, vol. 18, no. 3, 2001, pp. 17–24.

[6] D. Martin, H. Wu, and A. Alsaid, "Hidden Surveillance by Web sites: Web Bugs in Contemporary Use," *Communications of the ACM*, vol. 46, no. 12, 2003, pp. 258–264.

[7] Microsoft.com, "Microsoft Phishing Filter: A New Approach to Building Trust in E-Commerce Content," 2005, http://www.microsft.com/downloads/details.aspx?family-id=b4022c66-99bc-4a30-9ecc-8bdefcf0501d&displaylang=en (current Mar. 31, 2005).

[8] K. D. Mitnick and W. L. Simon, *The Art of Deception: Controlling the Human Element of Security*, Wiley Publishing, Indianapolis, Indiana, 2002.

[9] L. Spitzner, *Honeypots: Tracking Hackers*, Pearson Education, Inc., Boston, Massachusettes, 2002.

[10] L. Spitzner, "Honeytokens: The Other Honeytokens," 2003, http://www.securityfocus.com/infocus/1713 (current Nov. 18, 2005).

[11] N. Thompson, "New Economy; The "Honeytoken," an Innocuous Tag in a File Can Signal an Intrusion in a Company's Database," *NY Times*, April 2003.

[12] A. van der Merwe, M. Loock, and M. Dabrowski, "Characteristics and Responsibilities Involved in a Phishing Attack," *WISICT '05: Proceedings of the 4th International Symposium on Information and Communication Technologies*. ACM, 2005, pp. 249–254, Trinity College Dublin.

APPENDIX A

PRELIMINARY EXPERIMENT PERL SOURCE CODE

A.1 automate.pl

```
#!/usr/bin/perl

use strict;
use LWP::Simple;
use LWP::UserAgent;
use File::Copy;

#########################################################
#Create new gif and html files for this particular
#phish attack.
#########################################################

my $url = @ARGV[0];
my $root_url;

if($url =~ m/(http:\/\/(.+\/)+)/cgi) {
    $root_url = $1;
}

my @mArray = ("Jan","Feb","Mar","Apr","May","Jun",
              "Jul","Aug","Sep","Oct","Nov","Dec");

my $mon = (localtime)[4];
my $date = (localtime)[3]+1;
my $year = (localtime)[5]+1900;
my $hour = (localtime)[2];
my $min = (localtime)[1];
my $sec = (localtime)[0];

my $date_time = $year."_".$mArray[$mon]."_"
    .$date."_".$hour."_".$min."_".$sec;

my $templateGIFFile = "/var/www/html/images/template.gif";
my $gif_file        = $date_time.".gif";
my $newGIFFile      = "/var/www/html/images/".$gif_file;

my $templateHTMLFile = "/var/www/html/index.html";
my $html_file        = $date_time.".html";
my $newHTMLFile      = "/var/www/html/html_files/".$html_file;

copy($templateGIFFile,$newGIFFile);
copy($templateHTMLFile,$newHTMLFile);
#########################################################

#########################################################
#Add new attack to database file siteData.csv
#########################################################

open(DB,"+>> siteData.csv");
```

```
my $data = $url.",".$date_time.",auto\n";
print DB $data || die "Could not write to file.\n";

close(DB);
#########################################################

#########################################################
#Parse remote html file for form data.      Do this up
#to 3 times as long as the result is a form.
#########################################################

system("rm","-f","./output_html.txt");
open(HTML,"+>> output_html.txt");

my $action_root;
my $form_count = 1;
my $form_flag = 1;

print "URL is $url.\n";

while(($form_flag) && ($form_count <= 3)) {
    if($form_count == 1) {
        my $html      = get($url)
            or die "Couldn't open URL.";
        my $html_orig = $html;

        print HTML $html;
        close(HTML);
    }

    open(HTML,"< output_html.txt")
        or die "Couldn't open URL file.";

    my $action_url;
    my $method;
    my @input_names;
    my @types;
    my @values;

    print "Starting while loop.\n";
    while(my $line = readline(HTML)) {
        if($line =~
           m/<form.*method\s*=\s*[\"\']?(\w{3,4})[\"\']?/i) {
            $method = $1;
            print "Method is $method.\n";
        }

        if($line =~
           m/action\s*=\s*[\"\']?(\w*\.\w*)[\"\']?/i) {
            $action_root = $1;
            $action_url = $root_url.$action_root;
```

```
            print "Action is $action_url.\n";
            print "Action Root is $action_root.\n";
        }

        if($line =~ m/\<input/i) {
            my $input_type = 0;
            my $input_value = 0;
            do {
                if($line =~
                   m/type\s*=\s*[\"\']?(\w+)[\"\']?/i) {
                    push @types,$1;
                    print "Input type is $1.\n";
                    $input_type = 1;
                }
                if($line =~
                   m/value\s*=\s*[\"\']?(\w+)[\"\']?/i) {
                    push @values,$1;
                    print "Input value is $1.\n";
                    $input_value = 1;
                }
                if($line =~
                   m/name\s*=\s*[\"\']?(\w+)[\"\']?/i) {
                    push @input_names,$1;
                    print "Input name is $1.\n";
                }
                if($input_type == 0) {
                    push @types,"text";
                }
                if($input_value == 0) {
                    push @values," ";
                }
                $line = readline(HTML);
            } while($line !~ m/>/i)
        }

        if($line =~
           m/<select\s*name\s*=\s*[\"\']?(\w+)[\"\']?.*
           <option.*<option\.*value\s*=\s*[\"\']?(\w+)
           [\"\']?/cgi) {
            push(@types,"select");
            push(@input_names,$1);
            push(@values,$2);
            print "Select statment type, name, value ";
            print "equals select, $1, $2.\n";
        }
    }
    close(HTML);
##########################################################

##########################################################
#Creating form values for form submission
    my $size = @input_names;
```

```
    my $form_values = "?";
    my $input_flag = 0;

    for(my $i=0; $i<$size; $i++) {
        print "Current type is $types[$i].\n";
        if(($types[$i] eq "hidden")
           || ($types[$i] eq "checkbox")) {
            $form_values .= "$input_names[$i]=$values[$i]&";
        }
        elsif (($types[$i] eq "text") ||
              ($types[$i] eq "Text") ||
              ($types[$i] eq "password") ||
              ($types[$i] eq "Password")) {
            if($input_flag) {
                $form_values .= "$input_names[$i]=
%3Cimg%20src=\%22http%3A%2F%2F130%2E18%2E5%2E11%2F
images%2F$gif_file\%22%3E&";
            }
            else {
                $form_values .= "$input_names[$i]=
%3Ca%20href=\%22http%3A%2F%2F130%2E18%2E5%2E11%2F
html%5Ffiles%2f$html_file\%22%3Emyspam\@hotmail.com
%3C%2Fa%3E&";
            }
            $input_flag = !$input_flag;
        }
    }

    chop $form_values;

######################################################

######################################################
#Autofilling form with html tags
    if($method eq "POST" || $method eq "post") {
        my $str_length = length($form_values);
        my $post_values = substr($form_values,1,
                                 $str_length);
        print "Submitting POST data $post_values\n";
        system("wget","-O","output_html.txt","--post-data",
               "$post_values","$action_url");
    }
    elsif($method eq "GET" || $method eq "get") {
        my $new_url = $action_url.$form_values;
        print "Submitting GET data $new_url.\n";
        system("wget","-O","output_html.txt","$new_url");
    }
######################################################

######################################################
#Checking output to see if we still have a form
```

```
    sysopen(NEW_PAGE,"./output_html.txt","O_RDONLY") ||
die "Could not open new form.\n";

    while(my $line = readline(NEW_PAGE)) {
        if($line =~ m/<form/cgi) {
            $form_flag = 1;
            last;
        }
        else {
            $form_flag = 0;
        }
    }
    $form_count++;
}
#########################################################
```

A.2 imagesetup.pl

```
#!/usr/bin/perl

use strict;
use LWP::Simple;
use LWP::UserAgent;
use File::Copy;

#######################################################
# Create new gif and html files for this particular
# phish attack.
#######################################################

my $url = @ARGV[0];
my $rootUrl;

if($url =~ m/(http:\/\/(.+\/)+)/cgi) {
    $rootUrl = $1;
}

my @mArray = ("Jan","Feb","Mar","Apr","May","Jun",
              "Jul","Aug","Sep","Oct","Nov","Dec");

my $mon = (localtime)[4];
my $date = (localtime)[3]+1;
my $year = (localtime)[5]+1900;
my $hour = (localtime)[2];
my $min = (localtime)[1];
my $sec = (localtime)[0];

my $dateTime = $year."_".$mArray[$mon]."_".$date.
    "_".$hour."_".$min."_".$sec;

my $templateGIFFile = "/var/www/html/images/template.gif";
my $gifFile   = $dateTime.".gif";
my $newGIFFile      = "/var/www/html/images/".$gifFile;

my $templateHTMLFile = "/var/www/html/index.html";
my $htmlFile         = $dateTime.".html";
my $newHTMLFile     = "/var/www/html/html_files/".$htmlFile;

copy($templateGIFFile,$newGIFFile);
copy($templateHTMLFile,$newHTMLFile);
#######################################################

#######################################################
#Add new attack to database file siteData.csv
#######################################################

open(DB,"+>> siteData.csv");
```

```
my $data = $url.",".$dateTime.",man\n";
print DB $data || die "Could not write to file.\n";

close(DB);
####################################################

####################################################
#Print out form field values to cut and paste into
#the form.
####################################################

print "Image tag is ";
print "<img src='http://130.18.5.11/images/";
print "$gifFile'>\n";

print "Image tag in hex code is ";
print "%3Cimg+src%3D%27http%3A%2F%2F";
print "130.18.5.11%2Fimages%2F$gifFile%27%3E\n";

print "HTML tag is ";
print "<a href='http://130.18.5.11/htmlFiles/
print "$htmlFile'>myspam\@hotmail.com</a>\n";

print "HTML tag is ";
print "%3Ca+href%3D%27http%3A%2F%2F130.18.5.11%2F";
print "html_files%2F$htmlFile%27%3E";
print "myspam\@hotmail.com%3C%2Fa%3E\n";
####################################################
```

APPENDIX B

PRELIMINARY EXPERIMENT HTML/PHP SOURCE CODE

B.1 phishFormGet.php

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML
   4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>
  <title>Phishing Test Form</title>
  <meta http-equiv="content-type"
   content="text/html; charset=utf-8">
</head>

<body>
<?php
  $submit = $_POST['submit'];

  if($submit == 'y') {
        $name = $_POST['name'];
        $email = $_POST['email'];

    $query = "INSERT INTO project
             VALUES ('$name','$email')";

    $connection = mysql_connect('localhost', 'etsumc',
                                'etfwb32547');
    mysql_select_db('etsumc_cal', $connection);

    if((mysql_query($query, $connection)) &&
       mysql_affected_rows() == 1)
      echo "<center><font size=\"3\">";
      echo "Information Successfully Added.";
      echo "</font></center>";
    else
      echo "<center><font size=\"3\">";
      echo "Information NOT Successfully Added.";
      echo "</font></center>";
  }

?>

<br /><br /><br /><br /><br />
<br /><br /><br /><br /><br />

<center><h1>
  <form action="phish_form_post.php" method="POST">
    <input type="hidden" name="submit" value="y">
    <table border="0">
      <tr>
        <td>Name:</td>
```

```
        <td><input type="text" name="name"></td>
      </tr>
      <tr>
        <td>E−mail:</td>
        <td><input type="text" name="email"></td>
      </tr>
      <tr>
        <td> </td>
        <td><input type="submit"></td>
      </tr>
    </table>
  </form>
</h1></center>
</body>

</html>
```

B.2 phishFormPost.php

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
   Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>
  <title>Phishing Test Form</title>
  <meta http-equiv="content-type"
   content="text/html; charset=utf-8">
</head>

<body>

<br /><br /><br /><br /><br />
<br /><br /><br /><br /><br />

<center><h1>
  <form action="phish_form_get.php" method="POST">
    <input type="hidden" name="submit" value="y">
    <table border="0">
      <tr>
        <td>Name:</td>
        <td><input type="text" name="name"></td>
      </tr>
      <tr>
        <td>E-mail:</td>
        <td><input type="text" name="email"></td>
      </tr>
      <tr>
        <td> </td>
        <td><input type="submit"></td>
      </tr>
    </table>
  </form>
</h1></center>
</body>

</html>
```

B.3 showDB.php

```
<HTML>
<BODY>

<br /><br /><br /><br /><br />
<br /><br /><br /><br /><br />

<table align="center" border="0">
    <center><h1>Project Test:
    <?php

      $query = "SELECT * FROM project";

      $connection = mysql_connect('localhost', 'etsumc',
                                  'etfwb32547');
      mysql_select_db('etsumc_cal', $connection);

      $result = mysql_query($query, $connection);

      while($row = mysql_fetch_row($result)) {
        echo "<tr><td>$row[0] $row[1]</td></tr>\n";
      }
    ?>
    </h1></center>
</table>
</BODY>
</HTML>
```

APPENDIX C

ATTEMPTED AUTOMATION PYTHON SOURCE CODE

C.1 automateDQ.py

```
#!/usr/bin/python

import filecmp                          #Used for file comparison to see if
                                        #the same page is encountered
import os                               #imports os libraries
import re                               #imports regular expression library
import string
import urllib                           #imports web libraries

from HTMLParser import HTMLParser       #Used for processing HTML files
from time import localtime, strftime    #imports time libraries

class MyHTMLParser(HTMLParser):
    actionField       = ""
    currentSelectField = ""
    foundForm         = 0
    methodField       = ""
    optionCount       = 0
    params            = {}

    def handle_starttag(self, tag, attrs):
        if tag == "form":
            MyHTMLParser.foundForm = 1
            for value in attrs:
                if value[0] == "action":
                    MyHTMLParser.actionField = value[1]
                if value[0] == "method":
                    MyHTMLParser.methodField = value[1]

        elif tag == "input":
            inputVal    = ""
            nameValue   = ""
            typeValue   = ""

            for value in attrs:
                if value[0] == "name":
                    nameValue = value[1]
                elif value[0] == "type":
                    typeValue = value[1]
                elif value[0] == "value":
                    inputVal    = value[1]

            typeValList = (typeValue,inputVal)

            MyHTMLParser.params[nameValue] = typeValList

        elif tag == "select":
            for value in attrs:
                if value[0] == "name":
```

```
                    MyHTMLParser.currentSelectField = value[1]

        elif tag == "option":
            if MyHTMLParser.currentSelectField != "":
                MyHTMLParser.optionCount = MyHTMLParser.optionCount+1

            selectFieldValue = ""

            if MyHTMLParser.optionCount == 2:
                if MyHTMLParser.currentSelectField != "":
                    for value in attrs:
                        if value[0] == "value":
                            selectFieldValue = value[1]

                            MyHTMLParser.params[MyHTMLParser.currentSelectField] =
selectFieldValue
                            MyHTMLParser.currentSelectField = ""
                            MyHTMLParser.optionCount = 0

infile=open('./newAttacks.csv','r') #list of new attacks

for phish in infile:                        #Look at each phishing attack
    m    = string.split(phish,',')
    url  = m[0]                             #Get URL from infile
    site = m[1]                             #Get business being exploited

    site = site.rstrip('\n')

    filename = strftime("%Y_%m_%d_%H_%M_%S", localtime())
    filename = filename+'_dq'

    outfileName = './'+filename+'/log.txt'

    os.system("mkdir %s" % (filename))

    outfile = open(outfileName,'w')

    outfile.write('URL is '+url+'\n')
    outfile.write('Site is '+site+'\n')

    outfile.write('Filename is '+filename+'\n')
    outfile.write('Outfile name is '+outfileName+'\n')

    ##########################################################
    #Get directory structure from url for relative pathnames#
    #                    in the action field                 #
    ##########################################################

    m = re.search("http://((.*/)*)",url,re.IGNORECASE)
    if m:
        actionRoot = m.group(1)
        actionRoot = 'http://'+actionRoot
```

```
    ##########################################################
    #Create new gif files and html files for this particular#
    #Create new gif files and html files for this particular#
    #                    phishing attack                     #
    ##########################################################

    gifTemplate = '/var/www/html/images/template.gif'
    gifNewfile    = '/var/www/html/images/'+filename+'_auto.gif'
    os.system("cp %s %s" % (gifTemplate,gifNewfile))

    htmlTemplate = '/var/www/html/html_files/template.html'
    htmlNewfile   = '/var/www/html/html_files/'+filename+'_auto.html'
    os.system("cp %s %s" % (htmlTemplate,htmlNewfile))

    ##########################################################
    #   Writes url, site exploited, and filename to logfile  #
    #                    for reference later                 #
    ##########################################################

    datafile = open('./investigationList.csv','a')
    out = url+','+site+','+filename

    datafile.write(out)
    datafile.write('\n')
    datafile.close

    ##########################################################
    #            Creates links for form submission           #
    ##########################################################

    htmlLink = '<a href="http://130.18.5.11/html_files/'+filename+'_auto.html">
myspam@hotmail.com</a>'

    htmlHex = urllib.quote(htmlLink)

    imageLink = '<img src="http://130.18.5.11/images/'+filename+'_auto.jpg">'

    imageHex = urllib.quote(imageLink)

    ##########################################################
    #             Stores url locally for parsing             #
    ##########################################################

    count = 1;    #Keeps up with the number of pages
                  #being processed.    Used in filenames
                  #for html files stored locally

    countString = repr(count);       #Converts count to a string

    localFN = './'+filename+'/'+countString+'.html'
```

```
urllib.urlretrieve(url,localFN)
#Stores html file into a local file

##########################################################
#       Parses local file to get html form parameters    #
##########################################################

fileExtListFile = open('./fileExtensions.csv','w')

m = re.search("/(.*?\..*?)$",url,re.IGNORECASE)

if m:
    fileExtListFile.write(m.group(1))
    fileExtListFile.write('\n')

processForm = 1

while processForm:
    httpFile = open(localFN,'r')

    p = MyHTMLParser ( )

    outfile.write('Parsing the file ========'+localFN+'\n')
    p.feed(httpFile.read())

    if not p.foundForm:
        processForm = 0
    else:

        ##########################################################
        #       Creates name=value pairs for form submission     #
        ##########################################################

        dataDict = {}

        htmlImage = 0   #Boolean variable to swap between html and image tags

        for key in p.params.keys():
            dictList = p.params[key]
            if "hidden" == dictList[0]:
                dataDict[key] = dictList[1]
            else:
                phoneNo = re.search("phone",key,re.IGNORECASE)
                if phoneNo:
                    dataDict[key] = "5555555555"
                else:
                    if htmlImage:
                        dataDict[key] = imageHex
                        htmlImage = 0
                    else:
                        dataDict[key] = htmlHex
                        htmlImage = 1
```

```
##########################################################
#              Submits data to next web page             #
##########################################################

count = count+1

actionValue = p.actionField

httpSearch = re.search("http://",actionValue,re.IGNORECASE)

prevLocalFN = localFN

if not httpSearch:
    actionValue = actionRoot+actionValue

outfile.write('Action value is '+actionValue+'\n')

methodField = string.upper(p.methodField)
outfile.write(methodField+'**********************************\n')

if "POST" == methodField:
    request         = urllib.urlencode(dataDict)
    newWebHandle = urllib.urlopen(actionValue, request)

    countString    = repr(count);      #Converts count to a string
    localFN    = './'+filename+'/'+countString+'.html'

    newHttpFile    = open(localFN,'w')
    while 1:
        data = newWebHandle.read(512)
        if not data:
            break
        newHttpFile.write(data)
    newHttpFile.close()
    newWebHandle.close()
else:
    request         = urllib.urlencode(dataDict)
    newPage         = actionValue+'?'+request

    newWebHandle = urllib.urlopen(newPage)

    countString    = str(count);      #Converts count to a string
    localFN    = './'+filename+'/'+countString+'.html'
    urllib.urlretrieve(newPage,localFN)

httpFile.close

urllib.urlcleanup()          #Cleans up cache from urlretrieve calls

outfile.write(prevLocalFN+'and'+localFN+'\n')
```

```
            if filecmp.cmp(prevLocalFN,localFN):
                outfile.write('Equal Files\n')
                processForm = 0

            #p.foundForm = 0

            outfile.write('COUNT IS =======' +countString+'\n')

            if count == 6:
                processForm = 0
        #End of 'if not p.foundForm' loop
    #End of 'while processLoop' loop

    fileExtListFile.close()

#End of 'for phish in infile' loop

infile.close
```

C.2 automateSQ.py

```
#!/usr/bin/python

import filecmp                          #Used for file comparison to see if
                                        #the same page is encountered
import os                               #imports os libraries
import re                               #imports regular expression library
import string
import urllib                           #imports web libraries

from HTMLParser import HTMLParser       #Used for processing HTML files
from time import localtime, strftime    #imports time libraries

class MyHTMLParser(HTMLParser):
    actionField       = ""
    currentSelectField = ""
    foundForm         = 0
    methodField       = ""
    optionCount       = 0
    params            = {}

    def handle_starttag(self, tag, attrs):
        if tag == "frame":
            for value in attrs:
                if value[0] == "src":
                    MyHTMLParser.actionField = value[1]

        elif tag == "form":
            for value in attrs:
                if value[0] == "action":
                    MyHTMLParser.actionField = value[1]
                if value[0] == "method":
                    MyHTMLParser.methodField = value[1]

        elif tag == "input":
            inputVal  = ""
            nameValue = ""
            typeValue = ""

            for value in attrs:
                if value[0] == "name":
                    nameValue = value[1]
                elif value[0] == "type":
                    typeValue = value[1]
                elif value[0] == "value":
                    inputVal  = value[1]

            typeValList = (typeValue,inputVal)

            MyHTMLParser.params[nameValue] = typeValList
```

```
        elif tag == "select":
            for value in attrs:
                if value[0] == "name":
                    MyHTMLParser.currentSelectField = value[1]

        elif tag == "option":
            if MyHTMLParser.currentSelectField != "":
                MyHTMLParser.optionCount = MyHTMLParser.optionCount+1

            selectFieldValue = ""

            if MyHTMLParser.optionCount == 2:
                if MyHTMLParser.currentSelectField != "":
                    for value in attrs:
                        if value[0] == "value":
                            selectFieldValue = value[1]

                            MyHTMLParser.params[MyHTMLParser.currentSelectField] =
selectFieldValue
                            MyHTMLParser.currentSelectField = ""
                            MyHTMLParser.optionCount = 0

infile=open('./newAttacks.csv','r') #list of new attacks

for phish in infile:                        #Look at each phishing attack
    m    = string.split(phish,',')
    url  = m[0]                             #Get URL from infile
    site = m[1]                             #Get business being exploited

    filename = strftime("%Y_%m_%d_%H_%M_%S", localtime())
    filename = filename+'_sq'

    outfileName = filename+'/log.txt'

    os.system("mkdir %s" % (filename))

    outfile = open(outfileName,'w')

    outfile.write('URL is'+url+'\n')
    outfile.write('Site is'+site+'\n')

    outfile.write('Filename is'+filename+'\n')
    outfile.write('Outfile is'+outfileName+'\n')

    ##########################################################
    #Get directory structure from url for relative pathnames#
    #                  in the action field                   #
    ##########################################################

    m = re.search("http://((.*/)*)",url,re.IGNORECASE)
    if m:
```

```
    actionRoot = m.group(1)
    actionRoot = 'http://'+actionRoot

##########################################################
#Create new gif files and html files for this particular#
#                    phishing attack                     #
##########################################################

gifTemplate = '/var/www/html/images/template.gif'
gifNewfile  = '/var/www/html/images/'+filename+'_auto.gif'
os.system("cp %s %s" % (gifTemplate,gifNewfile))

htmlTemplate = '/var/www/html/html_files/template.html'
htmlNewfile  = '/var/www/html/html_files/'+filename+'_auto.html'
os.system("cp %s %s" % (htmlTemplate,htmlNewfile))

##########################################################
#   Writes url, site exploited, and filename to logfile  #
#                    for reference later                 #
##########################################################

datafile = open('./investigationList.csv','a')
out = url+','+site+','+filename

datafile.write(out)
datafile.write('\n')
datafile.close

##########################################################
#    Downloads whois query of the compromised webserver  #
##########################################################

whoisFile = './'+filename+'/whois.txt'
os.system("whois %s > %s" % (url,whoisFile))

##########################################################
#      Takes whois query and logs the physical address   #
##########################################################

whoisLogFile = open('./serverLoc.csv','a')
whoisFH      = open(whoisFile,'r')

for line in whoisFH:
    m = re.search("^address:\s*(.*)",line,re.IGNORECASE)
    if m:
        whoisLogFile.write(m.group(1))
    else:
        m2 = re.search("^role:\s*(.*)",line,re.IGNORECASE)
        if m2:
            roleLine = 1
            whoisLogFile.write(m2.group(1),re.IGNORECASE)
        else:
```

```
                roleLine = 0

            m3 = re.search("^person:\s*(.*)",line,re.IGNORECASE)
            if m3:
                personLine = 1
                whoisLogFile.write(m3.group(1))
            else:
                personLine = 0

    whoisLogFile.close()
    whoisFH.close()

    ##########################################################
    #             Creates links for form submission          #
    ##########################################################

    htmlLink = "<a href='http://130.18.5.11/html_files/"+filename+"_auto.html'>
myspam@hotmail.com</a>"

    htmlHex = urllib.quote(htmlLink)

    imageLink = "<img src='http://130.18.5.11/images/"+filename+"_auto.jpg'>"

    imageHex = urllib.quote(imageLink)

    ##########################################################
    #              Stores url locally for parsing            #
    ##########################################################

    count = 1;      #Keeps up with the number of pages
                    #being processed.    Used in filenames
                    #for html files stored locally

    countString = repr(count);        #Converts count to a string

    localFN = './'+filename+'/'+countString+'.html'

    urllib.urlretrieve(url,localFN)
    #Stores html file into a local file

    ##########################################################
    #      Parses local file to get html form parameters     #
    ##########################################################

    fileExtListFile = open('./fileExtensions.csv','w')

    m = re.search("/(.*?\..*?)$",url,re.IGNORECASE)

    if m:
        fileExtListFile.write(m.group(1))
        fileExtListFile.write('\n')
```

```
processForm = 1

while processForm:
    httpFile = open(localFN,'r')

    p = MyHTMLParser ( )

    outfile.write('Parsing the file ========'+localFN+'\n')
    p.feed(httpFile.read())

    ##########################################################
    #       Creates name=value pairs for form submission     #
    ##########################################################

    dataDict = {}

    htmlImage = 0   #Boolean variable to swap between html and image tags

    for key in p.params.keys():
        dictList = p.params[key]
        if "hidden" == dictList[0]:
            dataDict[key] = dictList[1]
        else:
            phoneNo = re.search("phone",key,re.IGNORECASE)
            if phoneNo:
                dataDict[key] = "5555555555"
            else:
                if htmlImage:
                    dataDict[key] = imageHex
                    htmlImage = 0
                else:
                    dataDict[key] = htmlHex
                    htmlImage = 1

    ##########################################################
    #              Submits data to next web page             #
    ##########################################################

    count = count+1

    actionValue = p.actionField

    httpSearch = re.search("http://",actionValue,re.IGNORECASE)

    prevLocalFN = localFN

    if not httpSearch:
        actionValue = actionRoot+actionValue

    methodField = string.upper(p.methodField)
    outfile.write(methodField+'****************************************\n')
    if "POST" == methodField:
```

```
            outfile.write('Action value is '+actionValue+'\n')

            request        = urllib.urlencode(dataDict)
            newWebHandle = urllib.urlopen(actionValue, request)

            countString    = repr(count);      #Converts count to a string
            localFN    = './'+filename+'/'+countString+'.html'

            newHttpFile    = open(localFN,'w')
            while 1:
                data = newWebHandle.read(512)
                if not data:
                    break
                newHttpFile.write(data)
            newHttpFile.close()
            newWebHandle.close()
        else:
            request        = urllib.urlencode(dataDict)
            newPage        = actionValue+'?'+request

            newWebHandle = urllib.urlopen(newPage)

            countString    = str(count);       #Converts count to a string
            localFN    = './'+filename+'/'+countString+'.html'
            urllib.urlretrieve(newPage,localFN)

        httpFile.close

        urllib.urlcleanup()          #Cleans up cache from urlretrieve calls

        outfile.write(prevLocalFN+' and '+localFN+'\n')

        if filecmp.cmp(prevLocalFN,localFN):
            outfile.write("Equal Files\n")
            processForm = 0

        #p.foundForm = 0

        outfile.write('COUNT IS ======='+countString+'\n')
        if count == 6:
            processForm = 0

    #End of 'while processLoop' loop

    fileExtListFile.close()

#End of 'for phish in infile' loop

infile.close
```

APPENDIX D

FINAL EXPERIMENT PYTHON SOURCE CODE

D.1 imagesetupDQ.py

```
#!/usr/bin/python

import os                                   #imports os libraries
import string
from time import localtime, strftime        #imports time libraries
import urllib

infile=open('./newAttacks.csv','r') #list of new attacks

for phish in infile:                        #Look at each phishing attack
    m=string.split(phish,',')
    url=m[0]                                #Get URL from infile
    site=m[1]                               #Get business being exploited
    print 'URL is',url
    print 'Site is',site

    filename = strftime("%Y_%b_%d_%H_%M_%S", localtime())
    filename = filename+'_dq'

    print 'Filename is',filename

    ##########################################################
    #Create new gif files and html files for this particular#
    #                    phishing attack                     #
    ##########################################################

    gifTemplate='/var/www/html/images/template.gif'
    gifNewfile='/var/www/html/images/'+filename+'_man.gif'
    os.system("cp %s %s" % (gifTemplate,gifNewfile))

    htmlTemplate='/var/www/html/html_files/template.html'
    htmlNewfile='/var/www/html/html_files/'+filename+'_man.html'
    os.system("cp %s %s" % (htmlTemplate,htmlNewfile))

    ##########################################################
    #Writes url and filename to logfile for reference later#
    ##########################################################

    datafile=open('./siteData.csv','w')
    out=url+','+filename
    datafile.write(out)
    datafile.close

    ##############################################
    #Print out html tags to use in phisher's form#
    ##############################################

    html_reg = '<a href="http://130.18.207.5/html_files/'+filename+'_man.html">
myspam@hotmail.com</a>'
```

```
    print "HTML link is "
    print html_reg

    html_hex = urllib.quote(html_reg)

    print "HTML link in hex code is "
    print html_hex

    img_reg = '<img src="http://130.18.207.5/images/'+filename+'_man.gif">'

    print "IMG tag is "
    print img_reg

    img_hex = urllib.quote(img_reg)

    print "IMG tag in hex code is "
    print img_hex

    #End of 'for phish in infile' loop

infile.close
os.remove('./newAttacks.csv')
```

D.2 imagesetupSQ.py

```
#!/usr/bin/python

import os                                   #imports os libraries
import string
from time import localtime, strftime        #imports time libraries
import urllib

infile=open('./newAttacks.csv','r') #list of new attacks

for phish in infile:                        #Look at each phishing attack
    m=string.split(phish,',')
    url=m[0]                                #Get URL from infile
    site=m[1]                               #Get business being exploited
    print 'URL is',url
    print 'Site is',site

    filename = strftime("%Y_%b_%d_%H_%M_%S", localtime())
    filename = filename+'_sq'
    print 'Filename is',filename

    ########################################################
    #Create new gif files and html files for this particular#
    #                   phishing attack                    #
    ########################################################

    gifTemplate='/var/www/html/images/template.gif'
    gifNewfile='/var/www/html/images/'+filename+'_man.gif'
    os.system("cp %s %s" % (gifTemplate,gifNewfile))

    htmlTemplate='/var/www/html/html_files/template.html'
    htmlNewfile='/var/www/html/html_files/'+filename+'_man.html'
    os.system("cp %s %s" % (htmlTemplate,htmlNewfile))

    ########################################################
    #Writes url and filename to logfile for reference later#
    ########################################################

    datafile=open('./siteData.csv','w')
    out=url+','+filename
    datafile.write(out)
    datafile.close

    ##############################################
    #Print out html tags to use in phisher's form#
    ##############################################

    html_reg = "<a href='http://130.18.207.5/html_files/"+filename+"_man.html'>
myspam@hotmail.com</a>"
```

```
    print "HTML link is "
    print html_reg

    print "HTML link is "
    print html_reg

    html_hex = urllib.quote(html_reg)

    print "HTML link in hex code is "
    print html_hex

    img_reg = "<img src='http://130.18.207.5/images/"+filename+"_man.gif'>"

    print "IMG tag is "
    print img_reg

    img_hex = urllib.quote(img_reg)

    print "IMG tag in hex code is "
    print img_hex

    #End of 'for phish in infile' loop

infile.close
```

APPENDIX E

INVESTIGATION DATA

Table E.1

Phishing Sites Investigated

Case No	Site
1	http://fr.ebayobjects.com/6k%3Bh=http://3731128842:82/ebayISAPII.dll-cgi/index.php
2	http://mckim.skhu.ac.kr/bbs/data/ai/1115632907/.military/efs/servlet/military/
3	http://signin.ebay.com.1234.tty.btd.com.sg/ws/eBayISAPI.php?cmd=SignIn&co_partnerId=2&pUserId=& siteid=0&pageType=&pa1=&i1=&bshowgif=&UsingSSL=&ru=&pp=&pa2=&errmsg=&runame=
4	http://towernet.capitalonebank.com.id22crlsdu.sing34.biz/verify/customerservice/formpage.html
5	http://210.66.55.11/Images/paypal/index.htm
6	http://www.liceymum.ru/images/smilies/index.htm
7	http://moneywireinc.selfip.net/paypal.html
8	http://75.23.0.9/MembersLogin/index.htm
9	http://1358148044/%66%61%6D%69%6C%79/media.renamed.vunerable/.https:/capitalone.com/login.php?/ objectclicked=navcustomercarelink&source=recentactivity&SID=200002227628582
10	http://www.burninggirlsonline.com/pics/cmd-run=/login.htm
11	http://intopsgroup.net/skin/red/mainimg/netbank.com.au/aubank/www3.netbank.commbank.com.au/update/index.htm
12	http://212.62.47.196//.Artikelnummer/18993457382/bitteEinloggen/index.php
13	http://moneymanagergps-id774144343.citizensbank.com.pal-neto.cn/gps/userdir/onlineform.aspx/
14	http://69.95.145.147:443/us.etrade.com/login.htm?Survey
15	http://www.sanorga.de/rdx/cache/index.html?ssl=promos/jump/checking/ ?cm_sp=ThankYou-Checking-_-Free%20Checking%20with%20Direct%20Deposit-_-Open%20SignOn
16	http://www.un3.net/imgbay/uploads/confirm-paypal/
17	http://61.19.233.130/~saifon/%20%20/mid/step1.html
18	http://207.36.160.15/icons/.../www.bankofamerica.com/sslencrypt218bit/online_banking/
19	http://onlinesession-724822614.natwest.com.palvica131.cn/updatemode/userdatadirectory/start.aspx/
20	http://surgicalservicesinternational.com/editables/flash/lipo/module.dll.php? customerid=rod@bradbury.worldonline.co.uk&co_partnerId=2&siteid=0&ru=&PageName=login_run& pp=pass&pageType=708XeMWZllWXS3AlBXVShqAhQRfhgTDrf= https://signin.ebay.com/ws/eBayISAPI.dll?SignIn&UsingSSL=1&pUserId=&co_partnerId=2&siteid=0&ru=&pp=& pageType=708&MfcISAPICommand=ConfirmRegistration&708XeMWZllWXS3AlBXVShqAhQRfhgTDrfQRfhgTDrfA
21	http://www.takemetothewu.com/x/View.htm
22	http://210.60.133.203/icons/small/ps1.gif/signin.ebay.com/ws/ebayISPP.dll/SignIn/
23	http://teencreek.org/phpBB2/language/lang_english/timbre/index1.php
24	http://moneymanagergps-id713480.citizensbank.com.oida17.cn/gps/userdir/onlineform.aspx/
25	http://66.49.130.245/~martin/-/PayPal/updates/us/webscr.php?cmd=_login-run
26	http://fondolisiados.gob.sv/error/include/capitalone/info.htm
27	http://wvps212-241-211-27.vps.webfusion.co.uk/www.halifax-online.co.uk/_mem_bin/ formslogin.asp_source=halifaxcoukHOME/
28	http://64.150.167.151/addssd/
29	http://diata.dk/modules/mod_as_category/images/service.htm
30	http://est-213-228-152-145.netvisao.pt/fonts/www.signin.ebay.com/beware.buyer.html
31	http://www.cnmd.org/bbs/.secure_ssl30/33.swf/militarybankonline.bankofamerica/efs/servlet/ military/login.htm?id=1191071916
32	http://v1338.ncsrv.de:84/bankofamerica/
33	http://home.frv.fr/dot/amazon.com/index.htm
34	http://securelogin-26805221.moneymanagergps.com.gts72.com/Online_Form.htm
35	http://id-199044007.citizensbankmoneymanagergps.com.bk4ft.zj.cn/securepage/challenge.aspx/
36	http://payrhpal.com/
37	http://78.38.172.3/citadel/
38	http://onlinesession-51728.natwest.com.miloe2r.zj.cn/updatemode/userdatadirectory/start.aspx/
39	http://140.8.233.220.exetel.com.au/.www.paypal.com/webscr.php?cmd=_login-run
40	http://www.csifanclub.com/coppermine/albums/userpics/newdir1/albacaza.htm
41	http://ernest-industries.com/Upload/webscrcmd=_account-run%5Cupdates-paypal%5Cconfirm-paypal/
42	http://signin.cgi3.com.wsbayisapi.dll.co-partnerid.signinusingssl.powersellers-cgi5bay.com.istemp.com/cgi/ www.ebay.com/3FMyeBay20pp-20pa220errmsg2runameruparams-ruproduct-sidfavoritenavconfirm0ebxPageTypeexisting Email20isCheckoutmigrateVisitor1SignIn/login.html
43	http://www.broeske.com/barb/www.paypal.com/cgi-bin/online-security=paypal/cgi-bin=_connexion/ online_security=information/webscrcmd=_login-run/webscrcmd=_account-run/updates-paypal/ confirm-paypal/update.php
44	http://www.motoplus.ro/sitemap/cgi-bin/webscrcmd/update.php
45	http://securelogin-22072011.moneymanagergps.com.fcf18.com/login.htm
46	http://paythpal.com/
47	http://217.17.228.148:82/eBaySignIn-eBayISAPI/ signin.ebay.com_ws_eBayISAPI.dllSignIn%20ru=http=www.ebay.com.htm
48	http://www.brugro.nl/%20./index.html
49	http://85.25.48.26:84/cascades/index.htm?ssl=433
50	http://w0qqrromzr4qqsacatz133236qwqsocm.smtp.ru/Great-Dell-C600-P3-laptop-CDROM-FDD-AC-complete.htm
51	http://59.188.28.202/index.htm
52	http://www.unl.edu.ec/feue/cache/p/

Table E.2

Phishing Sites Data

Case No	Site IP	Phishing Site Location	Investigation Filenames	Company Impersonated
1	222.100.130. 10	Korea	2007_Jun_20_11_28_52_dq, 2007_Jun_20_11_29_53_sq	Ebay
2	203.246.275.102	Korea	2007_Jun_21_16_36_13_dq, 2007_Jun_21_16_36_22_sq	Bank of America
3	77. 97.226. 19	Netherlands	2007_Jun_30_21_33_57_dq, 2007_Jun_30_21_34_13_sq	Ebay
4	58. 81.164. 82	Japan	2007_Jul_04_13_25_16_dq, 2007_Jul_04_13_25_26_sq	CapitalOne
5	210. 66. 55. 11	Taiwan	2007_Jul_04_13_58_47_dq, 2007_Jul_04_13_59_05_sq	PayPal
6	81.222.134. 49	Russia	2007_Jul_17_21_37_59_dq, 2007_Jul_17_21_38_06_sq	Wachovia
7	71.109.252.214	United States	2007_Aug_02_20_48_26_dq, 2007_Aug_02_20_48_31_sq	PayPal
8	75. 23. 0. 9	United States	2007_Aug_02_20_58_03_dq, 2007_Aug_02_20_58_08_sq	PayPal
9	80.243.177.204	United Kingdom	2007_Aug_08_19_05_49_dq, 2007_Aug_08_19_05_55_sq	Capital One
10	72.232. 72. 98	United States	2007_Aug_08_19_20_34_dq, 2007_Aug_08_19_20_48_sq	PayPal
11	210.223.132. 1	Korea	2007_Aug_08_19_41_20_dq, 2007_Aug_08_19_41_32_sq	NetBank
12	212. 62. 47.196	Serbia & Montenegro	2007_Aug_08_19_54_35_dq, 2007_Aug_08_19_54_46_sq	Ebay
13	60. 12.130.112	Austrailia	2007_Sep_10_20_14_12_dq, 2007_Sep_10_20_14_20_sq	Citizen's Bank
14	69. 95.145.147	United States	2007_Sep_10_20_30_13_dq, 2007_Sep_10_20_30_20_sq	ETRADE
15	62. 26.185.250	Germany	2007_Sep_10_20_43_39_dq, 2007_Sep_10_20_43_46_sq	Bank of America
16	208. 67.253. 2	United States	2007_Sep_16_15_11_05_dq, 2007_Sep_16_15_11_14_sq	PayPal
17	61. 19.233.130	Thailand	2007_Sep_16_15_24_28_dq, 2007_Sep_16_15_24_35_sq	MidAmerica Bank
18	207. 36.160. 15	United States	2007_Sep_16_15_32_18_dq, 2007_Sep_16_15_32_22_sq	Bank of America
19	200. 77.213. 15	Mexico	2007_Sep_16_15_57_26_dq, 2007_Sep_16_15_57_37_sq	NatWest Bank
20	72. 9.251. 74	United States	2007_Sep_16_16_17_44_dq, 2007_Sep_16_16_17_54_sq	Ebay
21	216. 39. 58.194	United States	2007_Sep_17_20_05_23_dq, 2007_Sep_17_20_05_30_sq	Citibank
22	210. 60.133.203	Taiwan	2007_Sep_17_20_20_42_dq, 2007_Sep_17_20_20_46_sq	Ebay
23	38.113.185.219	United States	2007_Sep_17_20_38_44_dq, 2007_Sep_17_20_38_49_sq	PayPal
24	60. 12.130.112	Australia	2007_Sep_17_21_04_04_dq, 2007_Sep_17_21_04_14_sq	Citizen's Bank
25	66. 49.130.245	Canada	2007_Sep_17_21_22_34_dq, 2007_Sep_17_21_22_40_sq	PayPal
26	168.243.199. 98	San Salvador	2007_Sep_28_20_13_44_dq, 2007_Sep_28_20_13_52_sq	Capital One
27	212.241.211. 27	United Kingdom	2007_Sep_28_21_47_47_dq, 2007_Sep_28_21_47_53_sq	Halifax FCU
28	64.150.167.151	United States	2007_Sep_28_21_57_36_dq, 2007_Sep_28_21_58_01_sq	Yahoo Mail
29	212. 70. 21. 86	Denmark	2007_Sep_28_22_12_37_dq, 2007_Sep_28_22_12_45_sq	Wachovia
30	213.228.152.145	Portugal	2007_Sep_29_07_46_44_dq, 2007_Sep_29_07_46_50_sq	Ebay
31	61.100. 9.211	Korea	2007_Sep_29_08_08_25_dq, 2007_Sep_29_08_08_33_sq	Bank of America
32	89.110.149.177	Netherlands	2007_Sep_29_08_26_21_dq, 2007_Sep_29_08_26_28_sq	Bank of America
33	82.165.119. 76	Germany	2007_Sep_29_08_40_11_dq, 2007_Sep_29_08_40_19_sq	Amazon
34	87. 20. 15.141	Netherlands	2007_Sep_29_08_57_36_dq, 2007_Sep_29_08_57_42_sq	Citizen's Bank
35	219.253.140.172	Korea	2007_Oct_03_20_27_21_dq, 2007_Oct_03_20_27_35_sq	Citizen's Bank
36	216. 39. 58.235	United States	2007_Oct_03_20_39_35_dq, 2007_Oct_03_20_39_40_sq	PayPal
37	78. 38.172. 3	Netherlands	2007_Oct_03_20_48_18_dq, 2007_Oct_03_20_48_29_sq	Citadel FCU
38	219.253.140.172	Korea	2007_Oct_03_20_56_51_dq, 2007_Oct_03_20_56_57_sq	NatWest
39	220.233. 8.140	Australia	2007_Oct_03_21_02_21_dq, 2007_Oct_03_21_02_27_sq	PayPal
40	64.202.166.214	United States	2007_Oct_03_21_22_02_dq, 2007_Oct_03_21_22_16_sq	Ebay
41	69. 89. 21. 81	United States	2007_Oct_03_21_29_37_dq, 2007_Oct_03_21_30_10_sq	PayPal
42	64.136. 25.175	United States	2007_Oct_03_21_42_56_dq, 2007_Oct_03_21_43_07_sq	Ebay
43	65.254. 74. 81	United States	2007_Oct_03_21_53_35_dq, 2007_Oct_03_21_53_44_sq	PayPal
44	208.109.181. 24	United States	2007_Oct_03_22_09_34_dq, 2007_Oct_03_22_09_42_sq	PayPal
45	58.140. 85.112	Australia	2007_Oct_03_22_20_23_dq, 2007_Oct_03_22_20_30_sq	Citizen's Bank
46	216. 39. 58.207	United States	2007_Oct_05_12_02_25_dq, 2007_Oct_05_12_02_29_sq	PayPal
47	217. 17.228.148	Bahrain	2007_Oct_05_12_17_31_dq, 2007_Oct_05_12_17_36_sq	Ebay
48	62.193.248. 62	France	2007_Oct_05_12_34_59_dq, 2007_Oct_05_12_35_15_sq	BankTennesee
49	85. 25. 48. 26	Netherlands	2007_Oct_05_12_46_18_dq, 2007_Oct_05_12_46_23_sq	Bank of the Cascades
50	82.204.219.231	Russia	2007_Oct_05_12_59_55_dq, 2007_Oct_05_13_00_00_sq	Ebay
51	59.188. 28.202	Australia	2007_Oct_05_13_07_29_dq, 2007_Oct_05_13_07_35_sq	PayPal
52	192.188. 49. 2	Ecuador	2007_Oct_05_13_20_14_dq, 2007_Oct_05_13_20_22_sq	Ebay

Table E.3

Referral Data

Case No	Referral Generated
5	172.176.239.46 - - [05/Jul/2007:20:59:50 -0500] "GET /html_files/ 2007_Jul_05_20_56_17_sq_man.html HTTP/1.1" 200 33 http://us.f622.mail.yahoo.com/ym/ShowLetter? MsgId=1681_27011077_8238_1357_447_0&Idx=0&YY=48392&y5beta=yes&y5beta=yes&inc=25&order=down&sort=date &pos=0&view=a&head=f&box=Inbox" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)"
7	71.109.252.214 - - [03/Aug/2007:00:34:52 -0500] "GET /html_files/2007_Aug_02_20_48_31_sq_man.html HTTP/1.1" 304 - "-" "Mozilla/5.0 (X11; U; Linux x86_64; en-US; rv:1.8.1.3) Gecko/20070208 Mandriva/2.0.0.3-2mdv2007.1 (2007.1) Firefox/2.0.0.3 71.109.252.214 - - [03/Aug/2007:00:42:22 -0500] "GET /images/2007_Aug_02_20_48_26_dq_man.gif%5C HTTP/1.1" 404 317 "-" "Mozilla/5.0 U; Linux x86_64; en-US; rv:1.8.1.3) Gecko/20070208 Mandriva/2.0.0.3-2mdv2007.1 (2007.1) Firefox/2.0.0.3
14	172.181.233.84 - - [11/Sep/2007:02:54:24 -0500] "GET /images/2007_Sep_10_20_30_13_dq_man.gif HTTP/1.1" 200 824 "-" "Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.7.12) Gecko/20050915 Firefox/1.0.7" 172.181.233.84 - - [11/Sep/2007:02:54:39 -0500] "GET /html_files/2007_Sep_10_20_0_20_sq_man.html HTTP/1.1" 200 33 "-" "Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.7.12) Gecko/20050915 Firefox/1.0.7"
15	62.231.97.162 - - [12/Sep/2007:11:02:02 -0500] "GET /images/2007_Sep_10_20_43_39_dq_man.gif%5C HTTP/1.0" 404 317 "-" "Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.1.6) Gecko/20070725 Firefox/2.0.0.6"
20	70.244.204.105 - - [17/Sep/2007:03:25:58 -0500] "GET /images/2007_Sep_16_16_17_54_sq_man.gif HTTP/1.1" 200 824 "-" "Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.1.6) Gecko/20070725 Firefox/2.0.0.6"
21	89.35.62.17 - - [17/Sep/2007:20:14:46 -0500] "GET /html_files/2007_Sep_17_20_05_23_dq_man.html/ HTTP/1.1" 404 322 "-" "Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1)"
28	86.123.193.105 - - [29/Sep/2007:03:54:21 -0500] "GET /html_files/2007_Sep_28_21_57_36_dq_man.html HTTP/1.1" 200 33 "-" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)" 86.123.193.105 - - [29/Sep/2007:03:54:38 -0500] "GET /images/2007_Sep_28_21_57_36_dq_man.gif HTTP/1.1" 200 824 "-" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)"
30	172.139.141.155 - - [30/Sep/2007:12:19:56 -0500] "GET /html_files/2007_Sep_29_07_46_44_dq_man.html/ HTTP/1.1" 404 322 "-" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)" 172.139.141.155 - - [30/Sep/2007:12:19:59 -0500] "GET /images/2007_Sep_29_07_46_44_dq_man.gif/ HTTP/1.1" 404 317 "-" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)" 172.139.141.155 - - [30/Sep/2007:12:20:07 -0500] "GET /html_files/2007_Sep_29_07_46_50_sq_man.html/' HTTP/1.1" 404 323 "-" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)" 172.139.141.155 - - [30/Sep/2007:12:20:10 -0500] "GET /images/2007_Sep_29_07_46_50_sq_man.gif/' HTTP/1.1" 404 318 "-" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)"
32	89.33.93.72 - - [29/Sep/2007:08:30:21 -0500] "GET /images/2007_Sep_29_08_26_21_dq_man.gif/ HTTP/1.1" 404 317 "-" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)" 89.33.93.72 - - [29/Sep/2007:08:34:22 -0500] "GET /images/2007_Sep_29_08_26_21_dq_man.gif/ HTTP/1.1" 404 317 "-" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)"
37	89.37.26.93 - - [03/Oct/2007:21:32:18 -0500] "GET /html_files/2007_Oct_03_20_48_29_sq_man.html HTTP/1.1" 200 33 "-" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)" 89.38.10.22 - - [04/Oct/2007:08:47:49 -0500] "GET /images/2007_Oct_03_20_48_18_dq_man.gif HTTP/1.1" 200 824 "http://78.38.172.3/citadel/cit.txt" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 2.0.50727)" 89.38.10.22 - - [04/Oct/2007:08:47:49 -0500] "GET /images/2007_Oct_03_20_48_29_sq_man.gif HTTP/1.1" 200 824 "http://78.38.172.3/citadel/cit.txt" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 2.0.50727)"
48	172.183.122.233 - - [05/Oct/2007:12:38:03 -0500] "GET /images/2007_Oct_05_12_34_59_dq_man.gif HTTP/1.1" 200 824 "-" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)"
49	85.121.49.230 - - [05/Oct/2007:13:51:18 -0500] "GET /images/2007_Oct_05_12_46_18_dq_man.gif HTTP/1.1" 200 824 "-" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)" 85.121.49.230 - - [05/Oct/2007:13:51:20 -0500] "GET /html_files/2007_Oct_05_12_46_18_dq_man.html HTTP/1.1" 200 33 "-" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)"
51	89.122.205.16 - - [05/Oct/2007:13:17:36 -0500] "GET /images/2007_Oct_05_13_07_29_dq_man.gif HTTP/1.1" 200 824 "-" "Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.1.7) Gecko/20070914 Firefox/2.0.0.7"

Table E.4

Phisher's Data (1)

Case No	Phishers IP	Phishers Location	Website Viewed
5	172.176.239. 46	United States	http://us.f622.mail.yahoo.com/ym/ShowLetter?MsgId=1681_27011077_8238_1357_447_0_1828_-1_0&Idx=0&YY=48392&y5beta=yes&y5beta=yes&inc=25&order=down&sort=date&pos=0&view=a&head=f&box=Inbox
7	71. 109.252.214	United States	None
14	172.181.233. 84	United States	None
15	62. 231. 97. 162	Romania	None
20	70. 244.204.105	United States	None
21	89. 35. 62. 17	Netherlands	None
28	86. 123.193.105	Netherlands	None
30	172.139.141.155	United States	None
32	89. 33. 93. 72	Netherlands	None
37	89. 37. 26. 93	Netherlands	None
	89. 38. 10. 22	Netherlands	http://78.38.172.3/citadel/cit.txt
48	172.183.122.233	United States	None
49	85. 121. 49. 230	Netherlands	None
51	89. 122.205. 16	Netherlands	None

Table E.5

Phisher's Data (2)

Case No	Browser	OS	Turn Around Time
5	IE	Windows NT 5.1	00:03:33
7	Firefox	Linux	04:46:19
14	Firefox	Windows NT 5.1	06:34:11
15	Firefox	Windows NT 5.1	50:18:23
20	Firefox	Windows NT 5.1	11:08:04
21	IE	Windows NT 5.1	00:09:23
28	IE	Windows NT 5.1	05:56:45
30	IE	Windows NT 5.1	28:33:12
32	IE	Windows NT 5.1	00:04:00
37	IE	Windows NT 5.1	00:43:49
	IE	Windows NT 5.1	11:59:31
48	IE	Windows NT 5.1	00:03:54
49	IE	Windows NT 5.1	01:05:00
51	Firefox	Windows NT 5.1	00:10:07