2012 IEEE Eighth World Congress on Services

Cloud Based Content Fetching: Using Cloud Infrastructure to Obfuscate Phishing Scam Analysis

Edward Ferguson, Joseph Weber, and Ragib Hasan
Department of Computer and Information Sciences
University of Alabama at Birmingham
Birmingham, AL 35294, USA
{edf, weberja, ragib}@uab.edu

***Abstract*—Phishing has become a crippling problem for many of today's internet users. Despite innovations in preventative measures, phishing has evolved to become very hard to detect. Determining whether a particular site is a phishing site or not is difficult, causing many inexperienced users to fall victim. Phishing site operators often actively prevent anti-phishing measures by detecting and blacklisting the IP addresses that try to probe them. In this paper, we propose using a cloud infrastructure to mask our identity from assumed phishing sites through the use of virtual machines spread across many geographic regions. Our system presents different personas and user behavior to the phishing sites by using different IP addresses and different browsing configurations. By running a 10-day probe experiment against real phishing site, we show the effectiveness of this approach in preventing detection and blocking of anti-phishing probes by the phishing site operators.**

***Keywords*-cloud computing; security; forensics;**

I. INTRODUCTION

Although most people today associate phishing with emails or fraudulent websites, it actually applies to any attempt of acquiring personal information from a user through some type of electronic communication [1]. Today, however, the most common form of a phishing attack begins with an email sent to a large amount of people with a containing link to their own website. The website will usually have a very convincing look as many are designed to look as closely as possible to the legitimate site they are trying to spoof [2].

While researchers have developed tools to detect and probe phishing websites, the phishing site operators, or phisers, have also become smarter, and have developed evasive maneuvers to maximize the time for phishing before the spoofed site is shut down. One of the most common ways for phishing sites to avoid to detection is to block IP addresses known to be associated with phish detection groups. Additionally, they may block addresses which they have seen multiple times, assuming that they are not legitimate targets and may be attempting to discover them or shut them down [3].

To prevent such evasive maneuvers, we propose using the distributed and black box nature of clouds to create a cloaking layer between the phishing sites and the anti-phishing probes. Our cloud based content fetching system allows anti-phishing researchers to probe phishing websites without getting detected or blacklisted by the phishers. The contributions of this paper are as follows: (1) we designed a cloud-based architecture for anti-phishing probing and content fetching; (2) we explored techniques for creating a set of diverse user personas to be used by the anti-phishing probes; and (3) we performed experiments on real phishing sites from our extensive phish database to evaluate the effectiveness of our tool in preventing detection and blacklisting.

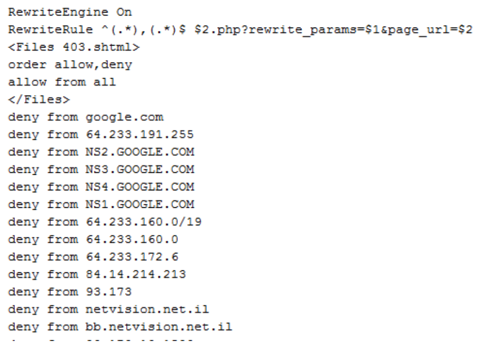
The rest of the paper is organized as follows: we provide background information and discuss related work in Section II. The methodology used in our anti-phishing cloaked probe system is presented in Section III. We discuss our findings in Section IV, comment on possible future directions in Section V, and conclude in Section VI.

II. BACKGROUND AND RELATED WORK

A. Phishing Overview

Phishing scams are social engineering scams that attempt to trick unsuspecting victims into handing over valuable information such as credit card or personal identity data [4]. Typically, this is done through the use of websites designed to mimic the look and feel of a particular bank or company. Phishers will often trick users into visiting their fake website by playing on the emotions of fear or greed. Fear based tactics are often used, such as sending an email that states a victim's account will be deactivated if they do not follow the link and fill in their information to 'verify' their account. Greed is also used to attract victims by offering something to users, such as offering a gift card if they fill out a survey with their personal information. Once the victims enter their sensitive information, the data is sent to the criminal either through the use of emails or by a text file left on the website.

While these scams vary in looks, complexity and functionality, some have been discovered to contain advanced features. Some will request that the user download a piece of software to protect them while online, which may actually contain malware. Others will check the users credentials



```
RewriteEngine On
RewriteRule ^(.*),(.*)$ $2.php?rewrite_params=$1&page_url=$2
<Files 403.shtml>
order allow,deny
allow from all
</Files>
deny from google.com
deny from 64.233.191.255
deny from NS2.GOOGLE.COM
deny from NS3.GOOGLE.COM
deny from NS4.GOOGLE.COM
deny from NS1.GOOGLE.COM
deny from 64.233.160.0/19
deny from 64.233.160.0
deny from 64.233.172.6
deny from 84.14.214.213
deny from 93.173
deny from netvision.net.il
deny from bb.netvision.net.il
```

Figure 1. Example of an .htaccess file used by a phishing website. It restricts certain IP ranges from accessing a particular site.

against the main company website to ensure the data is valid. Some instances of websites have been discovered that contain an .htaccess file which controls who and what can access the website content. We show one example .htaccess file which we recovered from a phishing site in Figure 1. The file contained a blacklist of IP addresses and ranges belonging to Google and Netvision.net.il. If a visitor from these IP addresses should attempt to download the content, they would receive a 403.shtml page. The .htaccess file also contains code to block attempts for standard WGET user agents from downloading content.

B. *Related Work*

A fair amount of work has gone into the detection and analysis of phishing sites [4], [5]. Content-based phishing website detection has proven to be an effective technique in identifying phishing websites [6], [7]. Some authors have implemented ways of analyzing the source code of a phishing scam. The files of a phishing website are typically based on a prepackaged set of files known as a phishing kit. These kits are given away freely or sold on the black market. Cova et al. developed a technique of downloading phishing kits, and automatically analyzing them for criminal dropmail addresses by detecting any time the kit attempted to send an email by calling the PHP mail() function [8].

Whether or not a particular website is a phishing site is not always apparent at first glance. For example, not all of the websites used over the course of this experiment were categorized as phishing sites before they were used. They did meet a set of criteria that listed them as potential phishing sites, however, which limits significantly the pool of websites to verify. CANTINA, a content based solution for example [9], can accurately identify roughly 95% of phishing sites with a false positive rate of only 6%. This approach, however, must use the content from the webpage, not only the URL, which is what these particular sites are initially classified with.

Unfortunately, from our experience from the UAB Phishing Database [10], content-based anti-phishing probes often face reverse-blacklisting by phishing websites, as soon as the phishers discover multiple accesses from same IP address or similar user configurations. Such blocks prevent the continuous operation of a anti-phishing system and can render continuous phishing site content-fetching almost impossible. In our system, we aim at preventing such evasive techniques by cloaking our probes using a cloud and a set of diverse user personas.

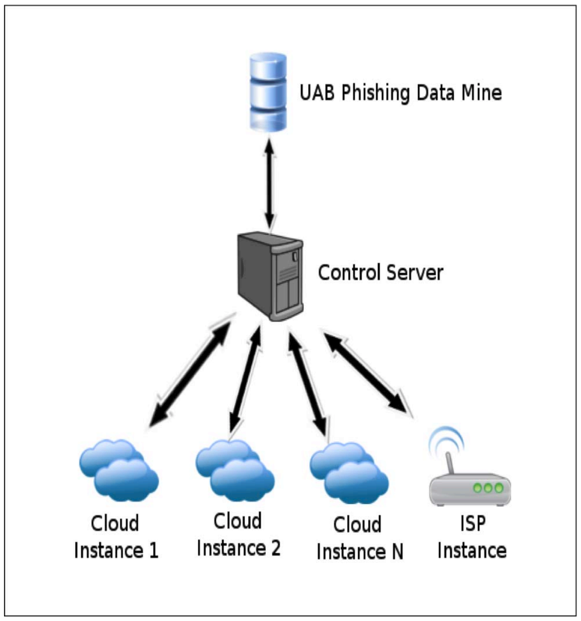


Figure 2. Diagram demonstrating the topology of our anti-phishing system. Each cloud instance would retrieve the suspected URLs from the control server and then return the compressed logs. The control server would retrieve these logs from the UAB Phishing Data Mine.

III. Methodology and System Setup

To build a cloaked anti-phishing probe system, we used a layered architecture, as shown in Figure 2. Our model uses multiple cloud providers, such as Amazon EC2, Rackspace, and GoGrid, to initiate cloud instances and fetch website content. A traditional Internet Service Provider (ISP) was also utilized as a measure of control. The Amazon instances contained a micro Ubuntu operating system in both their East coast and West coast data center, as well as a similar instance in both their European and Asia centers. Rackspace and GoGrid each used a small Ubuntu instance in their U.S. Datacenters.

A. *Topology*

The experiment was run using three separate layers to perform the necessary tasks. The pieces used were a database containing potential phishing URLs, a mediating control server and multiple clients that fetched potential phishing content.

The first entity is the UAB Phishing Database [10]. The UAB Phishing Data Mine (http://www.cis.uab.edu/PhishOps) takes potential phishing websites from a variety of sources and uses both automated and manual analysis to determine whether a particular page is a phish. The database receives thousands of potential phishing URLs every day from various sources. The content for these URLs is then examined using a variety of methods to determine if it is or is not a phishing scam. For this experiment any URL that came into UAB's data mine whether it was deemed a phish or not was fetched by the clients.

The second entity involved is the controlling server. The server's task is to act as a bridge between the phishing database and the clients that are constantly fetching the content. Twisted Python was used as the framework for the server/client connections, to handle the multiple asynchronous connections simultaneously as well as control the organization of website fetching.

The workhorse of this experiment is the cloud based clients. The purpose of the cloud clients was to fetch website content and send any data back to the controlling server to be stored.

Whenever a client was first activated or ran out of content to fetch, it requested new URLs from the controlling server. The server in turn makes the request to UAB's database to send new a list of fresh data. If any new URLs are available, the database will send them to the server, which will in turn forward them to the appropriate client. If no new data is available, the client waits 60 seconds and then asks again. After a client fetched its required data, it compressed the files in the folder into a zip file and forwarded the file to the server for storage.

The reason for this topology is centered on the idea that the clients should have no direct contact to the database, for multiple reasons. First off is security. If a client should become compromised, it would not have a direct access to UAB's data or any knowledge of where it resides on the internet. Additionally, having a control server reduces processing overhead and storage on a production environment.

B. Fetching Content

We used a 'pull' model similar to traditional email, where the client requests for additional data once they are done processing their previous batch of data. That ensured that each client could run asynchronously from the others and should a client crash it would not impact anything else.

We found that there are some differences between identical content fetched from different providers. Occasionally different providers will inject parts of their own code before it reaches users, such as a 'Hosted by x' bar at the bottom, particularly on free hosting sites. Sometimes two identical pages will not match forensically because each has references to their respective domain name in them.

The WGET program that was installed with Ubuntu was utilized to fetch website content. For every URL that was fetched, the custom Twisted Python program would call WGET to fetch the files. There were five variations of WGET used when downloading the pages. The first was a generic WGET with basic timeouts and downloading limits to prevent endless content downloads. The second variation is similar to the first, but ignores robots.txt and recursively finds multiple levels of website content in addition to the URL it had been passed to fetch. The other three instances were identical to the second but used WGET's user agent option to spoof that the user requesting the content was coming from a Windows machine, a Mac machine and an Android Droid x2 phone. The reason for this is that certain websites behave differently depending on type of operating system is accessing the content. Additionally if no custom user agent is specified, the operating system appears in the website access logs as 'wget'.

As clients are instantiated, they would connect to the server to request URLs to fetch. For each potential phishing URL, it would fetch website content using the five WGET instances, and zip up the data and forward it back to the controlling server. This was done to minimize data storage on the cloud instances, and therefore minimize operational costs as well as provide a more centralized area to store data.

C. Parsing Content

After a significant amount of time had been devoted to collecting data, the instances were stopped and the parsing of data commenced. The data was stored locally and the parsing postponed until after the fetching was complete to leave room to retrieve additional statistics from the data if need be at a future time. A simple java program was constructed to parse the various logs produced by each version of the wget and store the desired statistics into a local database. This database was then queried to provide the results seen in the following section. A different method of storing and parsing the data using less storage space and bandwidth is explored in Section 5.

IV. Results

Between November 7th and 17th of 2011, a total of 1638 sites were captured where all of the fetching instances collected data. Much more information was collected, but was not included in the results unless all of the available instances attempted to fetch something. We felt that to accurately compare results across all platforms, only results that had at least one fetch attempt per cloud instance would be used for analysis. Additionally, all of these results were then refined to only include the results from URLs that were confirmed as phishing sites by UAB's computer forensics lab. These results are then compared based on their fetch type, cloud instance, and connection code

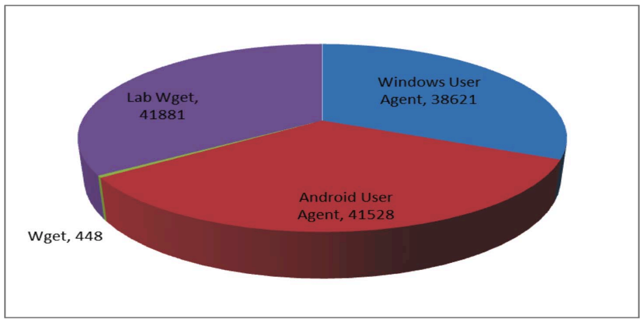


Figure 3. Total phishing results based on the user agent. The graph excludes Mac User agents due to the fact that a few instances did not fetch these.

A. Results by Fetch Type

As stated earlier, each instance attempted to fetch a given site using five different methods of WGET, with various user options. The results were then sorted and counted for each particular fetch type, as shown in Figure 3.

The main statistic to make note of is for the generic WGET results. The number of results coming from this variation returned very few results compared to the other, only 1% of the total. This is most certainly due to the large number of websites blocking wget through robots.txt. The other methods saw similar results to one another with the windows agent showing a slightly smaller number but within an error margin of 2%.

Additionally, the fetch types were sorted by their connection codes to determine which percentage of each respective WGET received connection codes of 200 (A successful request). This was to determine if any particular fetching type had a higher rate of retrieving successful results, not just results themselves. As can be seen in Figure 4, all of the variations (again, with the exception of the generic WGET) returned an almost identical amount of successful requests of their respective total results within one percent. These results confirm that outside of ignoring robots.txt, nearly all proposed phishing websites do not prefer one type of user agent to another when accessing their data.

B. Results by Cloud Location

One of the more important statistics gathered is the proportion of results collected by each respective cloud instance. Since the cloud instances were across not only multiple cloud providers but also physical locations (US, EU, Asia), this result alone can help determine the effectiveness of distributed fetching through a cloud. The number of results fetched by their respective dates can be seen in Figure 5. The results of this graph show an interesting trend that can be misleading upon first glance. While the results fetched from instances outside of the US show a significant drop in the number of items fetched per day, this can be mainly

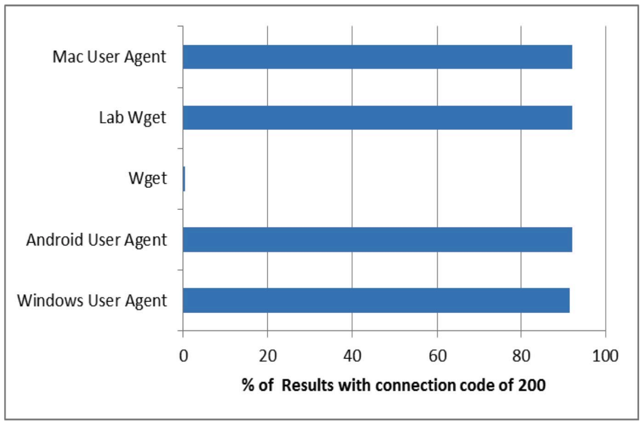


Figure 4. Percentage of each user agents results that resulted in a connection code of 200. Note that each user agent, with the exception of wget, received a similar amount of successful attempts.

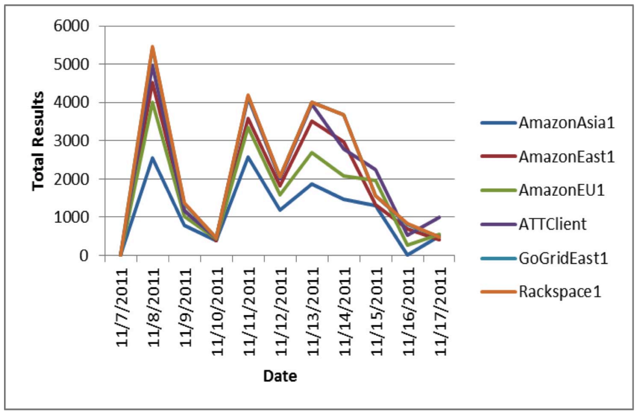


Figure 5. Total Results fetched per day on each cloud or ISP instance. The days with significant drops were days where an error occurred and the instances were left without a connection to the control server. The lower fetch total of the Asia and EU instance can be clearly seen.

attributed to when the phishing sites were attempted, not the phishing sites rejecting these instances. As a result of most of the phishing sites that were collected being located in the United States, the two instances that were outside of the US would naturally experience a longer wait time for data. This caused the two instances to sometimes lag behind the other instances when there was a large queue of URLs to fetch at that time as there was more delay between completing their fetching and beginning the next URL. Within the few hours that the other instances had fetched data from the phishing site, many of these sites had been taken down and as a result, very little (if anything) was fetched. This can be observed by looking at Figure 6. The proportion of 404 errors, which are the result of the server not being found, are clearly higher for the Asia instance. This is most likely due to those particular servers being taken down shortly after

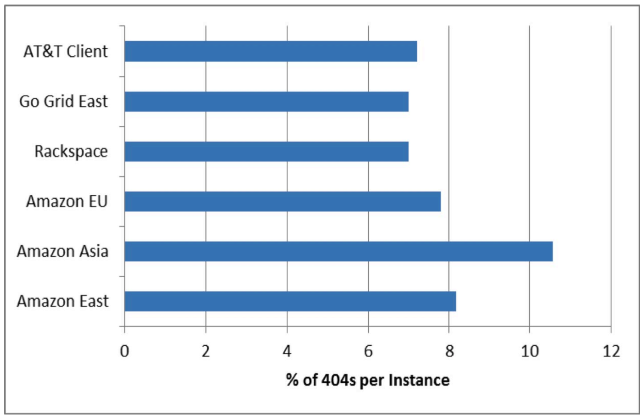


Figure 6. Percentage of 404 errors encountered of total results from each instance. Notice that Amazon Asia experienced a much higher result due to the delay in fetch time.

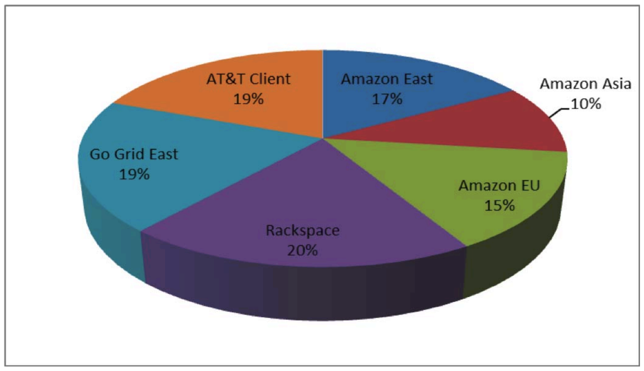


Figure 7. Percentage of the total results based on the instance type. The lack of results from the Asia and EU instance are explained in detail within this section. All other instances retrieved a similar amount of results.

their discovery. As far as within the US is concerned, the results are very similar across all cloud providers and the ISP instance. Each instance is within 3% of one another when total results are compared. These results are shown in Figure 7 and verify this similar range. This confirms that very few, if any, phishing sites are blacklisting any of the cloud providers used during the course of this experiment.

C. Cross Comparisons

Aside from looking at the overall results for the various aspects of the fetching, we wanted to investigate whether or not any of the results for a particular URL differed between each instance. The easiest and most time efficient way of doing this was to compare the various file sizes for each file URL. Although not always true, it can be assumed that if two particular instances both retrieved the same file from the same URL and the file size is equal for both, the same exact file was retrieved in each case. To automate this search, a python script was created to calculate the average file size

Service	Rate	Total
Micro-Instance (East-Linux)	$0.02/hour	$14.40
I/O Throughput (100 GB/month)	$0.120 per GB	$12.00
Total		$26.40

Table I
COST ANALYSIS FOR AN AMAZON MICRO INSTANCE IN THE EASTERN AVAILABILITY ZONE USING LINUX AS AN OPERATING SYSTEM. THE PROJECTED THROUGHPUT IS BASED ON THE AMOUNT FETCHED DURING THIS EXPERIMENT PLUS PROJECTIONS FOR AN ADDITIONAL 20 DAYS. PROJECTED TOTAL COST IS OVER THE COURSE OF 30 DAYS.

for each file URL, compare the size of each file across all cloud instances and fetch types with the calculated average and return any results that deviated from this within a certain margin.

One anomaly observed through such results was for the android user agents. Since many sites today are geared towards not only a typical browser on a PC or laptop but also mobile clients such as cellular phones, many websites have different layouts for viewing according to the browsers device. Since the android agent identified itself as a mobile phone, some sites would return smaller images, likely to cut down on bandwidth and load time for mobile users. Additionally, some of the WGETs that passed in no identifying information returned smaller file sizes or none at all, likely due to the fact that the web page shown to the user is dependent upon the browser type and if that is not given or known, nothing is shown to the user.

Another noticed anomaly is due to different files being generated randomly for the same web page. In these cases a particular item may be fetched randomly, such as an ad, and no two instances will see the same item size since each is difference. One example of this seen was on a particular site where the log-in page also required the use of a CAPTCHA. Since each CAPTCHA is unique and randomly given to the user, the file sizes were different. This came up during cross referencing as a false positive and can only be identified through manually inspecting the file or through the name of the file URL, although this cant always be trusted to be valid.

The connection code was also compared between file URLs to determine if possibly one result may have gotten a similar file size but due to different reasons. To cross reference the connection code received between the same file URLs across the multiple instances, the same python script mentioned previously was used again but compared connection codes instead of file size. Of the 156, 325 total attempts to fetch a file, 213 had a difference between file sizes and only 4 had a difference between the connection codes. Of these differences, almost all could be attributed to any of the previously mentioned anomalies.

D. Cost Analysis

The costs of maintaining a full time cloud based fetching instance was calculated to determine the feasibility of its use in a real world application. As shown in Table I, the smallest instance provided by Amazon, the micro instance, has sufficient computing resources for this type of fetching client. The I/O used for this experiment was averaged for a per day usage and projected over a 30 day period. This price could be lowered by as much as 40% if the parsing was done as the data was retrieved and only the parsed results sent to the control server. Also, if cloud storage were more cost efficient than sending the parsed results to an outside control server, the results could be directly stored in a cloud database, further reducing I/O costs.

V. Future Work

Our proof-of–concept system leaves additional room for future work in both verifying this concept over a longer time scale and also as a means for determining many phishing sites blacklisting rules. As stated previously, the results of this experiment were collected roughly over the period of ten days. Within that amount of time, we detected a few anomalies but no significant preference by phishing sites concerning where the user is or if the user is coming from a cloud provider. Longer term experiments, perhaps over several months, may allow us to determine certain trends of phishing sites and if new preventative measures are being taken to avoid detection by comparing the successful fetch rate over time. Additionally, we only considered those fetch collections that all fetching instances shared in common. It is possible that certain valuable data could be extracted that is of use, yet since this concept was mainly focused on determining the viability of any cloud provider or location over a traditional US-based user, this part of the dataset was never explored. A useful metric would be to determine blacklisting rules used by phishers through means of a cloud infrastructure such as one that was used throughout this experiment. This could be done by using multiple instances either in the same provider/location or multiple. Multiple instances would attempt to fetch information from each phishing websites, but in this scenario, the instances will continuously fetch from the same client.

After some time, it would be expected that the particular site would eventually catch on that certain users accessing the site are obviously probing them as they are continuously fetching from the site. Another instance, either one which is also fetching from the particular site or another instance whose purpose is to verify a blacklist should attempt to fetch the site and confirm that it is still up. If so, the logs for the particular instance being blacklisted should be checked to see how many times that instance fetched from the site and over what type of interval. That instance should then be given a new IP address or migrated to a new host to rerun under the same conditions and confirm that the previous procedure caused the instance to be blacklisted. This will give vital information to those phishing researchers who may need to know how much probing they can do for most phishing sites without worrying about being detected and subsequently blacklisted.

VI. Conclusion

Cloud based content fetching has proven to be a very viable concept for accessing data from known or suspected phishing sites. Based on the observed results, it is evident that fetching through a cloud infrastructure can be useful as a means of both obfuscation from phishing sites and also as a means to avoid being blacklisted. No particular cloud provider or location provided any significant difference in results gathered compared to a regular ISP. The ability to quickly switch IP addresses or availability zones can give greater mobility to a fetching agent if the particular agent is currently being blacklisted or to better obfuscate itself from the various phishing sites. By using multiple user agent setups (passed in through the WGET parameters) it can be determined what kind of precautions the phishing sites are taking towards various types of users.

Phishing has proven to be a problem that is going to be around for quite some time. It will never be completely unavoidable. The only hope to combat such a problem is to know what types of conditions are used presently and deal with them accordingly. As the phishing sites become more undetectable, so must the efforts to detect and combat them. In future work, we plan to perform longer term experiments that will allow us to determine evasive trends of phishing sites and whether new preventative measures are being taken to avoid detection by comparing the successful fetch rate over time.

Acknowledgments

The authors would like to thank Gary Warner of the UAB Forensics Lab for access to the UAB Phishing Database. Hasan was supported by the Office of Naval Research through grant N000141210217, a 2011 Google Faculty Research Award, and a 2011 Amazon Research Grant.

References

[1] T. Jagatic, N. Johnson, M. Jakobsson, and F. Menczer, "Social phishing," *Communications of the ACM*, vol. 50, no. 10, pp. 94–100, 2007.

[2] C. Drake, J. Oliver, and E. Koontz, "Anatomy of a phishing email," in *Conference on Email and Anti-Spam*, vol. 11, 2004.

[3] R. Weaver and M. Collins, "Fishing for phishes: applying capture-recapture methods to estimate phishing populations," in *Proc. of the Anti-phishing Working Groups, 2nd Annual eCrime Researchers Summit*. ACM, 2007, pp. 14–25.

[4] M. Jakobsson and S. Myers, *Phishing and countermeasures: understanding the increasing problem of electronic identity theft*. Wiley-Interscience, 2006.

[5] S. Sheng, B. Wardman, G. Warner, L. Cranor, J. Hong, and C. Zhang, “An empirical analysis of phishing blacklists,” in *Proc. of CEAS*, 2009.

[6] B. Wardman, T. Stallings, G. Warner, and A. Skjellum, “High-performance content-based phishing attack detection,” in *Proc. of eCrime*. IEEE, 2011, pp. 1–9.

[7] C. Whittaker, B. Ryner, and M. Nazif, “Large-scale automatic classification of phishing pages,” in *Proc. of NDSS*, 2010.

[8] M. Cova, C. Kruegel, and G. Vigna, “There is no free phish: an analysis of free and live phishing kits,” in *Proc. of USENIX WOOT*. USENIX Association, 2008, p. 4.

[9] Y. Zhang, J. Hong, and L. Cranor, “Cantina: a content-based approach to detecting phishing web sites,” in *Prof. of WWW*. ACM, 2007, pp. 639–648.

[10] “UAB phishing database,” Online at http://www.cis.uab.edu/PhishOps.