PhisherMan

John Banya and Brandon Botsch
Stevens Institute of Technology 2018

"Something weird is going on…"

1.1 Introduction

Phishing is a social engineering attack that exploits Internet users and aims to steal sensitive data by pretending to be another website. According to Cofense[1], 91% of cyberattacks in 2016 began with a phishing/spear phishing email, with average cost of the resulting data breach being 4 million USD. Spear phishing is also on the rise and poses a serious risk to both the average user and to corporations, potentially leading to serious data leaks and data loss. The problem of phishing attacks is one that has lead to much research over the years, but no definitive solution. Many modern web browsers now include built-in phishing detection to protect their users, and many networks use some variation of an application-level proxy to filter traffic based on public lists of malicious websites. Despite these efforts, phishing attacks are still a major problem as many techniques used by modern defenses are unable to detect zero-day phishing sites. In our project, we aim to demonstrate a novel approach to detecting these zero-day phishing sites that may otherwise bypass existing defenses.

1.2 Previous Approaches

During our research, we have found a number of approaches that are similar in their logic to ours. One patent[2] for a phishing detection system uses information such as URL, IP address, and HTML tags among other things to identify and alert the user of potential phishing sites via the Windows system tray. A paper[3] from Drexel University outlines a phishing detection method called PhishZoo that utilizes profiles of trusted sites and the ssdeep fuzzy hashing algorithm on the content of web pages in order to detect potential phishing sites. Using this method, they were able to achieve a 97% percent accuracy in detection and less than 4 second performance penalty comparing againsts 50 trusted profiles. In our approach, we hope to combine the ideas and knowledge of these approaches to create a real-world proof-of-concept in addition to exploring

other features. Combining the techniques of fuzzy hashing on page content, page appearance, and page shape to perform similarity detection, PhisherMan aims to reproduce the successes of PhishZoo in detecting zero-day attacks.

2.1 Implementation Overview

To address the problem of phishing attacks, we developed a firewall-like application-level proxy written in Go that observes and investigates outgoing connections, and, based on our internal heuristics and real-time analysis, makes decisions about the possibility of a phishing attack. PhisherMan is comprised of two fundamental parts: the active Connection Monitoring System (CMS), and the internal Heuristics Engine (HE). The Connection Monitoring System acts as a fully featured man-in-the-middle proxy for the browser's connection. The CMS intercepts all outgoing connections over both HTTP and HTTPs and, based on the content of the connection (i.e. HTML), forwards the request to be scanned by the Heuristics Engine. Some features of the CMS are a built-in cache that stores recently used URLs and their boolean status as a phishing site in order to improve connection speed for recently scanned sites, and multithreading to allow multiple users to use the proxy at once. The Heuristics Engine uses the ssdeep fuzzy hashing algorithm as well as the pHash (perceptual hash) algorithm to detect similarities between pages, and uses a relational database (sqlite3) to store the hashes for each page. When a new site is visited, multiple data points are gathered about the site, such as the site's HTML and image, and then hashed using both ssdeep and phash (phash is only applicable to images, not text). The fuzzy hash results of these computations are compared to the contents of the database, and if a close match is found the site is deemed to be a phishing site.

2.2 Proxy Implementation

In order for PhisherMan to perform analysis on websites a user connects to, it needs a way to capture all the traffic between a user and the internet. The method we used to accomplish this was a proxy server listening for HTTP Connect requests from clients. The proxy server could be installed on a single host or at the network layer depending on the use case. All client browsers must be configured to go through the proxy server (done by the system administrator). In addition to the raw TCP traffic being passed from client to server, the proxy server is able to get some information from the HTTP Connect request. An HTTP Connect request will reveal the connection destination host (but not any other information such as path or query parameters). Our first idea to be able to scan HTTPS sites was for the proxy server to make its own separate connection to the site, but this will not work since the path and query parameters are hidden inside the encrypted HTTP header. Our solution to this was to have the proxy perform a man-in-the-middle between the client and the destination. This is done this by dynamically generating a new certificate for the domain if it does not already exist in PhisherMan's system. The client will use PhisherMan's generated certificate and PhisherMan will use the real certificate when connecting the destination. Since PhisherMan is acting as a certificate authority, it requires a self-signed root certificate to be installed in any browser wishing to use PhisherMan. This is also

intended to be done by the system administrator.

When a client makes a new connection to a site, PhisherMan will receive the HTTP Connect request, read the HTTP header from the request, take note of a few things, and then hold the connection hostage while the detection takes place. We take note of the HTTP request method, path, host, and content-encoding of the header. First, we only care about GET requests since these are the first requests made from a web browser when connecting to a website and no POST requests can occur until after the user has loaded the site. The content-encoding is important in case the site is gzipped so that we can still read the compressed data. Finally, the host and path information is used to determine the full URL for the site. Once we have the full URL, we determine based on the status of the cache and the database whether or not the URL must be passed the Heuristics Engine.

If a site was passed to the Heuristics Engine and it was detected as a phishing site, some information about the match will be returned to the proxy server such as URL, hash type, and fuzzy hash difference score. At this point, the proxy server will block the connection to the destination and instead send back its own customized page informing the user that the site was blocked, which site it was matched against, the match type, and the match score.

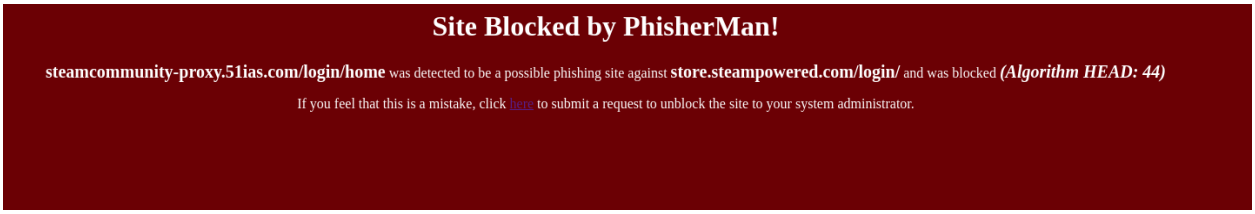

Figure 1.1 Example warning page

2.3 Detection Methods

When a URL is sent from the CMS to the Heuristics Engine to analyzed, several algorithms are run in order to try to determine whether the site appears to be legitimate. Several assumptions are made during this process. First, it is assumed that phishing sites will attempt to appear like their target site. For instance, it is in the best interest of a phishing site targeting PayPal to appear like PayPal's website. In cases where the site does not appear like the target site, we assume that users are able to quickly determine themselves that the site is not legitimate. Secondly, we assume that users will have visited the target site at some point in the past prior to visiting the phishing site. If a user has not ever visited the target site before, we assume it is unlikely that the phishing site will have any effect on the user.

When a site is passed through the Heuristics Engine (HE), the first step taken is to check whether the site has been detected as malicious in the past. If so, the HE simply states that the site is malicious and does not continue. If the site has never been visited before, however, more complex analysis must be done. A connection is made to the site and the content of the site is read and passed through a quick content-type heuristics function to detect whether the data

returned is HTML or not. If the content is not HTML, then it is allowed through without further investigation. This is used to easily weed out any obvious auxiliary requests (such as CSS and JavaScript files that are irrelevant to the main connection).

If the content of the site is believed to be HTML, then the first data point used by the HE is the ssdeep fuzzy hash of the HTML. Because we have assumed that phishing sites attempt to appear like their target sites, it is with non-negligible chance that the HTML of the malicious site will appear the like target HTML. However, if the phishing site is written from scratch without using any HTML from the target site while still visibly appearing like the target site, then this data point will result in a false negative. For this reason, we do not rely solely on the HTML but use additional data points.

One such additional pieces of data collected is the image of the webpage standardized to a height of 1080 pixels (to prevent trivial bypassing of the system). This image is generated by a tool known as ‘wkhtmltoimage’, which given the URL to a site, generates a screenshot-like JPEG file of the web page (with applied CSS). Provided that the site is not dynamically generated by JavaScript, this provides us with a screenshot of the site, which is then passed through both ssdeep and phash to generate the second and third data points (Figure 2.1). We use similar logic to generate the fourth and fifth data points, this time with the goal of detecting similarities in website shape. By taking the original screenshot of the web page output by ‘wkhtmltoimage’ and using the Canny edge detection algorithm on it, we produce a version of the page that takes into account only the shapes and sizes of the content (Figure 2.2). This is passed into ssdeep and phash as the fourth and fifth data points. Finally, the sixth and seventh data points are generated by cropping the original image of the page to the first 100 pixels of height in order to capture just the header of the site (Figure 2.3). This works under the assumption that many web pages have a logo and some uniquely identifying information at the top of the page. To account for the cases where the head of the page does not contain any information (i.e. solid color), we only processes the head image if it passes a complexity check of our own design. If the head is deemed to contain useful information, we pass it into ssdeep and phash for the sixth and seventh data points.

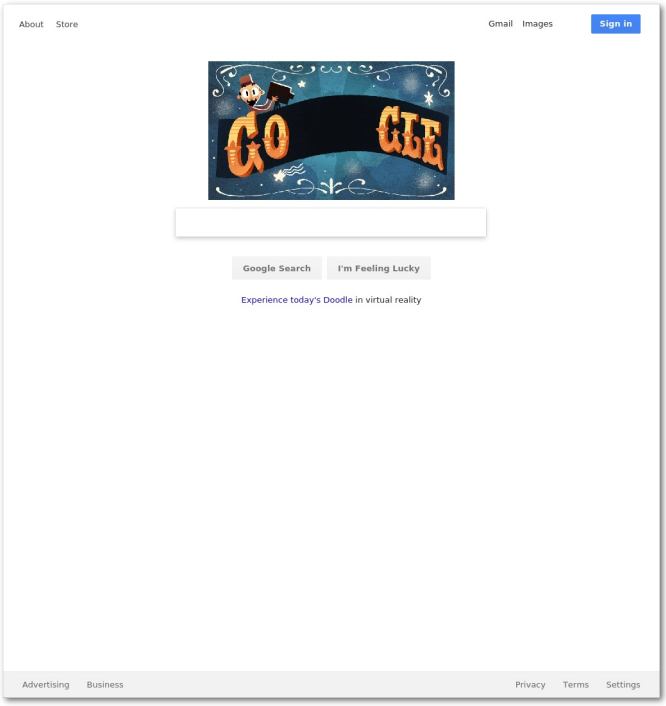

Figure 2.1 wkhtmltoimage result for www.google.com

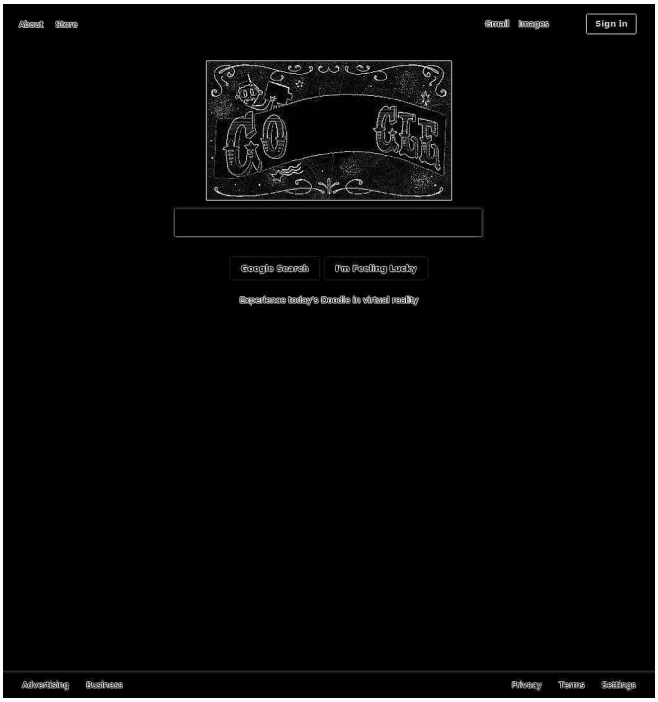

Figure 2.2 Canny edge detection of www.google.com

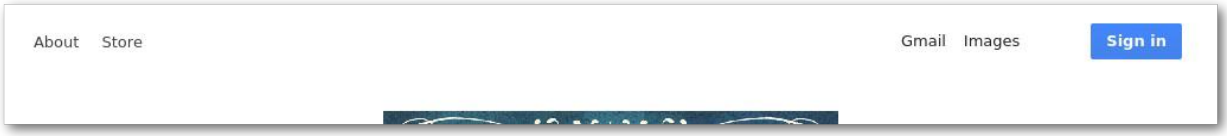

Figure 2.3 Head image for www.google.com

The resulting hashes of these pieces of information are gathered and compared to hashes in their respective category stored in the database. If a match is found with certainty beyond a predefined threshold, we mark the domain as phishing in the database. On the other hand, if no match is found, the URL is marked as legitimate and all data points are stored in the database for future comparisons.

2.4 Fuzzy Hash Database

After a connection has been through the HE and either blocked or allowed, the fuzzy hashes for the site must be logged to be used in future detections. This is done through the use of a lightweight SQLite database backed by a simple file. Each record in the database stores a single hash along with its hash type (HTML, image, edges, or header), the subdomain, domain, and path associated with the hash, and whether or not the domain is marked as safe. The domain for the hash is used during fuzzy hash comparison for phishing detection because we only want to test for hash collisions between sites on different domains. This is because it is quite likely that sites on the same domain will share some characteristics resulting in many false positives. In order for this to work correctly, we must be very careful about what we consider to be the domain and subdomain because we do not want google.co.uk and bbc.co.uk to be considered the same domain since they are both “co.uk” sites. Luckily for us, Mozilla has graciously provided the Public Suffix List (https://publicsuffixlist.org) which contains “effective top level domains (eTLD)”. This list enumerates all the domains under which one can obtain a domain name such as “com” and “co.uk”. For the purposes of the database, we store the eTLD plus one level beneath that in the domain field. Any further subdomains are put in the subdomain field. The subdomain and path information is only used to inform the user about which site was matched when a phishing site is detected. If a site was not detected as a phishing site, the domain remains marked as safe. However, as soon as even one site on a domain is detected as phishing, all hashes on all subdomains for that domain are marked as unsafe. There is no way for an unsafe site to ever be marked as safe again (unless the system administrator removes/edits the database entries). Also, note that there may be multiple hashes in the database for the same site. Safe values for sites are cached within PhisherMan for a certain period of time (currently 1 day) or until the cache is filled. If the site is not cached, new hash entries will be stored in the database even if it has been previously visited. This is so that phishing sites targeting a slightly outdated version of a site will still be detected. The database can be pruned by the system administrator if desired.

3.1 Measuring Success

To test the final software, we will perform both benchmark tests and accuracy tests. The benchmark tests will compare the speed of a system without the software against a system with the software. The benchmark will include both tests for repeated connections to a single website and connections to new websites. After performing the same experiments for both systems, we will compare the speeds to obtain a final benchmark.

In order to test the accuracy of the software, we will gather known phishing sites from various sources such as MalwareDomainList and PhishTank. We will test the software against these sites and then observe the accuracy of the software in blocking them, and obtain a final true positive percentage. We will determine a false positive percentage by browsing through common sites taken from lists such as Alexa Top 500 and track how many sites are get falsely blocked.

3.2 Test Results

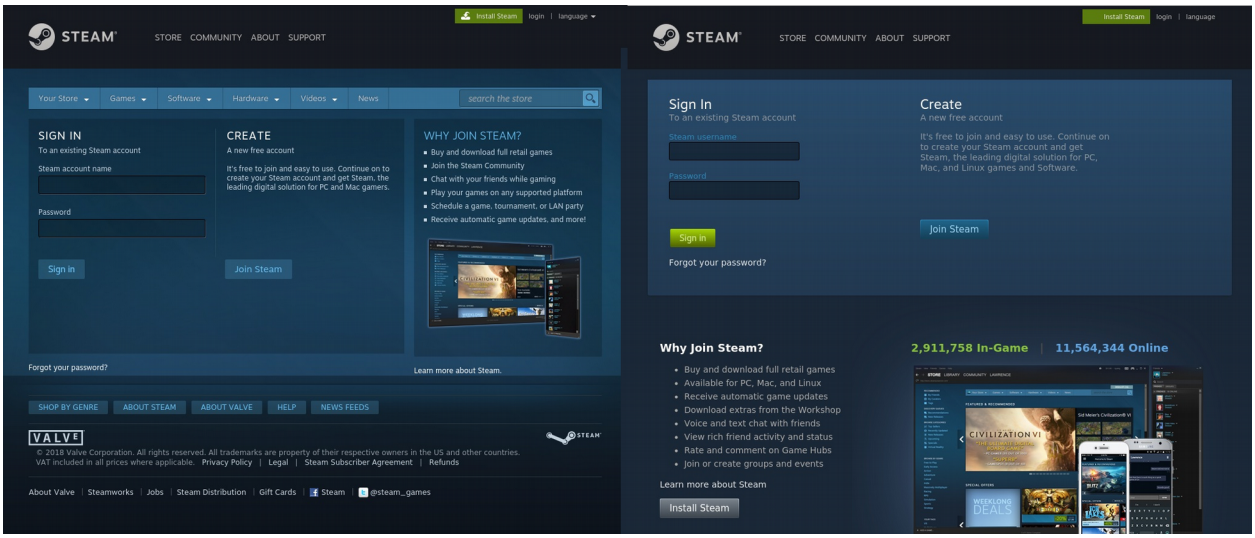

Figure 3.3 Real site (left) vs phishing site (right)

We tested 25 benign sites and 18 known phishing sites. The sites cover all the popular services like google, facebook, paypal, steam, etc. Here are the confusion matrix from the data:

True Positive Rate: 10 / (10+8) = 0.56 = 56%
True Negative Rate: 24 / (24+1) = 0.96 = 96%

	Not Detected	Detected
Non-Phishing	24	1
Phishing	8	10

* Some miscellaneous background ad sites were detected that are not factored into these results
** Does not include trivial matches such as “google.com” and “google.ca”

Example sites detected (true positive):
https://steamcommunity-proxy.51ias.com/login/home/
(vs https://store.steampowered.com/login)

https://steam.zombieden.cn/login/home/?goto=id%2Fcwh19940311
(vs https://store.steampowered.com/login)

http://italy-square.com/shop/shop/supportwells1/Wellsfargo-0nline/security/auth/1/inner_page.html
(vs https://www.wellsfargo.com/)

Example sites not detected (false negative):
http://itrade.hk/z/secure/doc/d/Docusign/pass.php#
http://www.msmetal.com/wp-content/themes/responsive/includes/images/_notes/en/mpp/Login/?7777772e6d736d6574616c2e636f6d=
http://innvation.usa.cc/file/

Here are some statistics on loading times with and without PhisherMan as well as with and without the websites being cached:

Website	Without PhisherMan	With PhisherMan	With PhisherMan (Cached)
Google	1.31	8.96	1.99
Facebook	1.41	13.72	1.94
Paypal	1.84	13.22	2.05
Steam	2.45	12.67	2.83
Outlook Live	1.70	14.94	1.34

3.3 Conclusion

Our finding show that we can successfully use fuzzy hashing techniques to detect zero day phishing sites based on their appearance, while keeping the false positive rate low enough to be used to success on most well maintained networks. Furthermore, while the detection rate was not overwhelmingly successful, we were still able to detect over half of tested phishing sites meeting our test criteria. We also note that while the impact of the heuristics scanning has a

noticeable impact on performance (sometimes increasing load times by over 10 seconds), the cache feature improves usability considerable and makes PhisherMan un-intrusive in most use cases (as it is likely that common websites used on a network will be cached). We believe that techniques similar to the ones currently used by PhisherMan can be a focus for additional work, as the detection rate will only improve as image similarity and image recognition algorithms improve. Furthermore, additional work can be done to combat techniques such as dynamically generating HTML from javascript, as the 'wkhtmltoimage' tool is currently unable to run javascript to get the generated page.

Citations

1. "Anti-Phishing Solution | Threat Management." *Cofense*, cofense.com/product-services/pm-solution/
2. Brinson, Duane, et al. "US20060080735A1 - Methods and Systems for Phishing Detection and Notification." *Google Patents*, Google, patents.google.com/patent/US20060080735A1/en.
3. Afro, Sadia. "PhishZoo: An Automated Web Phishing Detection Approach Based on Profiling and Fuzzy Matchin." *Drexel University*, www.cs.drexel.edu/tech-reports/phishzoo-tr2009.pdf.