Smartening the Crowds: Computational Techniques for Improving Human Verification to Fight Phishing Scams

Gang Liu[1,2], Guang Xiang[2], Bryan A. Pendleton[2], Jason I. Hong[2,3], Wenyin Liu[1]
[1]Department of Computer Science, City University of Hong Kong, Tat Chee Avenue, Kowloon, Hong Kong.
[2]School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 15213
[3]Wombat Security Technologies, 4620 Henry Street, Pittsburgh, PA 15213
gangliu@student.cityu.edu.hk, {guangx, jasonh}@cs.cmu.edu, csliuwy@cityu.edu.hk

ABSTRACT
Phishing is an ongoing kind of semantic attack that tricks victims into inadvertently sharing sensitive information. In this paper, we explore novel techniques for combating the phishing problem using computational techniques to improve human effort. Using tasks posted to the Amazon Mechanical Turk human effort market, we measure the accuracy of minimally trained humans in identifying potential phish, and consider methods for best taking advantage of individual contributions. Furthermore, we present our experiments using clustering techniques and vote weighting to improve the results of human effort in fighting phishing. We found that these techniques could increase coverage over and were significantly faster than existing blacklists used today.

Categories and Subject Descriptors
D.4.6 [**Security and Protection**]; K.4.4 **[Electronic Commerce**]; H.5.2 [**User Interfaces**]

General Terms
Algorithms, Security, Human Factors

Keywords
Phishing, wisdom of crowds, crowdsourcing, clustering, voting

1. INTRODUCTION
Many problems still require human intelligence to solve. Some require human intelligence as an intrinsic part of the process, such as in a democratic election. Others have no known technical solutions which match human performance, such as image labeling [1]. In the case of certain kinds of computer security tasks, it has been suggested that it is too risky to take the human entirely out of the loop [7]. No matter what the problem, it is important to consider how, when, and how much human effort is necessary to determine an appropriate and sufficient solution.

One particularly difficult to automate problem that currently requires human intelligence is identifying phishing scams (though primarily for reasons of liability, as we discuss below). The most common form of phishing is where attackers build convincing imitations of legitimate websites and lure unsuspecting victims to divulge sensitive personal information. Phishing attacks are expensive to society. Moore and Clayton estimated that the minimum loss to consumers was $320 million annually [25]. Note that this number does not include loss of productivity, cost of maintaining a helpdesk to field calls, recovery costs, or damage to an organization's reputation.

Many feature-based algorithms have been developed to automatically detect phishing sites, for example [9][21][42]. The advantage of heuristics and machine learning approaches is that they can rapidly identify attacks with no human involvement. However, these methods are prone to false positives (incorrectly labeling legitimate sites as phish) as well as false negatives (incorrectly labeling phishing sites as safe). False positives are particularly of concern. Sheng et al have observed that industry has been slow to adopt heuristics primarily from concerns over liability due to false positives [30].

An alternative that has been widely adopted by industry is human-verified blacklists. These blacklists contain URLs of sites that have been manually verified as phish. Three well-known phishing blacklists are operated by Microsoft, Google, and PhishTank. The main advantage of blacklists is that there are very few, if any, false positives, thus reducing the liability risk of incorrectly labeling a legitimate site as a phishing attack. Another advantage is the ability to detect new kinds of phishing attacks without explicit retraining. However, human verification is inherently more labor intensive and can be much slower in detecting attacks. Finally, human verification can also be overwhelmed by simply generating more phishing sites and/or URLs for phishing sites, as has been done with automated phishing attack toolkits and "fast flux" techniques that hide a phishing site behind a large number of compromised hosts to make detection more difficult [35].

	January 2010	January 2011
Submissions	18,836	16,019
Total Votes	54,847	69,648
Valid Phish	5,751	12,789
Invalid Phish	518	549
Median Time	12hrs 10min	2hrs 23min

Table 1. PhishTank self-reported statistics. Submissions require a minimum of 4 votes before labeling, with at least 70% agreement (some votes weighted differently). Median time has improved significantly.

Of particular interest to us here is the blacklist maintained by PhishTank, which uses a wisdom of crowds approach. Volunteers submit potential phish and also vote on submitted URLs, identifying them as phish or legitimate. According to PhishTank's own statistics [33], out of 1.1M URL submissions from volunteers, there were 4.3M votes, resulting in about 646k identified phish between October 2006 and February 2011.

Symposium On Usable Privacy and Security (SOUPS) 2011, July 20-22, 2011, Pittsburgh, PA, USA.

PhishTank has improved in performance as shown in Table 1. From Jan 2010 to Jan 2011, the median time to identify a phish has dropped from 12 hours to about 2.4 hours. The percentage of valid phish identified has also increased, going from 5,751 out of 18,836 (30.5%) in January 2010 to 12,789 out of 16,019 (79.8%).

We have two observations. First, for January 2011, there are still 2,681 URLs not identified as phish or legitimate. Most of these URLs represent "wasted" votes which did not reach the required number of votes for verification. Optimally, with 4 votes required to identify a phish, 69,648 votes could have identified a maximum of 17,412 labels rather than the 12,789 phish and 549 legitimate sites actually identified. Second, a median delay of 2.4 hours still represents a significant gap in protection, as most victims of a phishing scam fall for it within 8 hours of the start of the attack [19]. Furthermore, 2.4 hours only represents the delay from when the URL was first submitted to PhishTank, meaning that the phish was in the wild longer. Lastly, 2.4 hours represents the median, with past work suggesting that there is a power-law distribution in identifying and taking down phish [26].

We believe a promising solution is to improve the wisdom of crowds by combining manually verified blacklists with computational techniques, to keep false positives extremely low while also reducing the time to verify attacks. Such an approach would benefit not only sites like PhishTank, but also other manually-verified blacklists such as Google and Microsoft. A hybrid approach could also help with forensic analysis (such as identifying trends in phishing attacks and attacked brands), as well as help reduce the labor in maintaining the many databases that store data about current and past phishing attacks.

In this paper, we present the results of a study that we conducted with Aquarium, an experimental system we developed on top of Amazon's Mechanical Turk system for gathering human-verified labels on potential phishing sites. From a broad perspective, this paper looks at how to apply crowdsourcing techniques to a security task, and how to use computational techniques to improve the performance of a crowd. More specifically, this paper makes the following research contributions:

1. We present the design of Aquarium, a novel phish detection approach that makes use of two points in this design space, namely (a) clustering similar phish together and having minimally trained participants vote on clusters rather than individual phish, and (b) developing a vote weighting mechanism based on a participant's historical performance.
2. We present an evaluation of our two approaches, examining time to label a URL, accuracy, coverage, and monetary cost. Through a two-week study of verification of suspicious URLs, we show that our approach achieves a TP of 95.4% with a FP of 0%, with a median time to label of 0.7 hours.
3. We present our voteweight formula and the results of our parameter tuning, which can reduce the median time to label a URLs down to 0.5 hours.

2. RELATED WORK

Work on combating phishing can be categorized into four major approaches: making the problem invisible to end users, improving the design of user interfaces to help end-users make better decisions, improving end-user training, and leveraging wisdom of crowds.

2.1 Making it Invisible to End-Users

2.1.1 Algorithms for Detecting Phish

The main goal of this anti-phishing strategy is to keep users from ever seeing potential phishing attacks. Example past work in this category includes phishing email filters and phishing web page detection (which is complemented by taking down the offending web site). Here, we will focus the related work on algorithms for automatically detecting phishing web pages.

One class of algorithms uses URL features to detect phishing web pages. For example, Garera et al [9] categorized phishing URLs into four groups, each capturing a common phishing pattern, and used a set of fine-grained features from the phishing URLs together with other features to detect phish.

Another complementary class of algorithms makes use of features based on the HTML content to detect phish. For example, in [21], Ludl et al applied a J48 decision tree algorithm on 18 features solely based on the HTML and URL. Another feature-based work exploring the HTML content is CANTINA [42], in which Zhang et al proposed a content-based method using a simple linear classifier on top of eight features.

A similar area of work examines the visual and image elements to protect users from phishing attacks. To exploit visual similarity between web pages, Liu et al [20] proposed a method using three similarity metrics, i.e., block level similarity, layout similarity and overall style similarity, based upon web page segmentation. A page is reported as phishing if any metric has a value higher than a threshold. SpoofGuard [4] used image check as one feature, examining the domain name and the existence of popular target site logos on a given web page. Medvet et al [23] computed a signature using the visible text, visible images, and overall visual look-and-feel to compare the suspected pages with their legitimate counterparts. Recently, Chen et al [3] took a holistic view of the visual similarity between web pages, and applied compression algorithms on the pages as indivisible entities to detect phish.

Other techniques have been proposed that detect phish by inferring the target brand being phished. For example, Pan et al [27] proposed a method that extracts the web page identity from key parts of the HTML via the x^2 test, and compiled a list of features based on the extracted identity. Xiang et al [37] proposed a hybrid detection model that recognizes phish by discovering the inconsistency between a web page's true identity and its claimed identity via search engine and information extraction techniques.

Our work with Aquarium uses minimal algorithms to detect phish. Instead, it relies on optimizing human verification by weighting more effective participants more highly, and by clustering similar web pages together. Perhaps the closest work to Aquarium is our past work in augmenting existing human-verified blacklists by using shingling (a popular near-duplicate detection algorithm used by search engines) to compare a given page to known phish [39]. In contrast, Aquarium looks at techniques for improving the verification of blacklists in the first place.

2.1.2 Algorithms for Managing Information Flow

Researchers have also proposed approaches to guard users against phishing attacks by monitoring the flow of information such as passwords. For example, AntiPhish [17] watches the password field of HTML forms and searches the domain of the site being visited among a list of previous logins when an identical password

was used, warning users of potential attacks if a domain match is not found. Rosiello et al proposed an algorithm to address weaknesses in AntiPhish [28]. When the user enters in a password, their system checks the similarity of the HTML between the web page currently being visited and one previously visited before that used the same password. PwdHash [29] uses a hash value computed from the user's password and the website domain when authenticating, rather than the plain text password. This approach makes password stealing through phishing much harder. Yue et al [40] designed a client-side tool called BogusBiter, which sends a large number of bogus credentials to suspected phishing sites, hiding the real credential among the bogus ones.

2.1.3 PhishTank and Manually-Verified Blacklists

There are a number of phishing blacklists available. Perhaps the most popular are offered by Microsoft, Google, and PhishTank. Zhang et al presented an evaluation of these and other automated detection tools [41], showing that there was still many phish not detected by any tools even after 24 hours.

We will focus our description on PhishTank. PhishTank [33] is an open anti-phishing site launched in October 2006 to provide parent company OpenDNS with a reliable phishing dataset. Anyone who creates an account can submit potential phish and vote on submitted phish. Phish with enough verification votes are added to a blacklist. Submitted phish that do not gather a sufficient number of votes may never get a final label, and PhishTank does not publish this list of unknown results. Submissions that either do not have enough votes to verify as phish or are labeled as legitimate are still findable on the site but are not included in any further processing.

PhishTank automatically removes sites that are down or not responding from the verification queue. PhishTank attempts to generate a thumbnail for each submitted site, as well as collecting other technical details about the hosting company and hosting network, to help users determine the nature of a submitted site.

There have been studies in the past examining PhishTank. The closest work to ours is by Moore and Clayton, who found that PhishTank's participation follows the common power-law pattern seen in many online sites, and discovered that users who only periodically participate are more prone to making errors in labeling [26]. Moore and Clayton offer three lessons for improving PhishTank: (1) addressing power-law issues of participation, (2) having crowd-source decisions be hard to guess, and (3) not having users work harder than necessary. Our work somewhat addresses the first issue, and tackles the third issue directly. Our work explores individual accuracy in a similar setting, while also considering approaches to further improve the performance and reliability of a wisdom of crowds approach. We examine ways of managing the second issue in our discussion.

2.2 Improving the User Interface

Another primary strategy for anti-phishing is to improve user interfaces and help users make better decisions. Examples of past work here include Dhamija et al's work in dynamic security skins [5], Wu et al's Web Wallet [36], and Egelman et al's study on browser anti-phishing warnings. Given that our work focuses on combining human verification with computational techniques, we will not discuss these past projects in detail.

2.3 Training End-Users

The third primary strategy for anti-phishing is to train end-users. Examples of past work here include Anti-Phishing Phil, a game designed to engage the participant while progressively exposing them to more sophisticated phish-identification training [31], and PhishGuru, which uses simulated phishing attacks to train end-users [19]. Our work with Aquarium made use of Anti-Phishing Phil to train participants, and focuses on using minimally trained participants to help identify phishing web sites.

2.4 Leveraging Wisdom of Crowds

There has been a substantial amount of work looking at how to organize people online in an effective manner. In particular, in recent years, there has been rapid growth in research investigating how to build systems that leverage human effort for tasks that are too difficult for computers to do today.

Some research has examined specific domains, for example using games for image labeling tasks [1] or tagging shared documents [11][24]. Other research has investigated how to improve people's contribution to a group, for example by assigning work to users in a way that makes the user believe their work is uniquely matched to his or her capabilities [16]. SuggestBot generated suggestions for articles to edit in Wikipedia based on machine learning techniques, to increase participation [2].

Our work with Aquarium does not examine motivation. Instead, our work looks at how to improve the wisdom of crowds for a computer security task, to improve the results of human effort by applying computational techniques. Our work focuses on effective use of participants rather than increasing participation, by applying computational techniques such as clustering and vote weight.

There have also been several papers that have either used or have examined the use of Mechanical Turk for user studies. For example, Heer and Bostock [12] showed that MTurk was effective for crowdsourcing evaluations of visualizations. Kittur et al [18] used MTurk to collect ratings on the quality of Wikipedia articles, and offered guidelines for improving worker performance. Mason and Watts [22] investigated the effects of compensation for simple tasks, finding that increasing compensation increased the quantity of responses but not quality. Ipeirotis [15] examined the distribution of compensation for tasks, completion rates of tasks on different days, and the distribution of time to complete tasks. Relevant to our work here, Ipeirotis found that the distribution of completion times follows a power-law, where most tasks are finished quickly but a few tasks take very long. Partly for this reason, in our experiment we posted new tasks every day. Our work in this paper looks at how to apply crowdsourcing techniques to a security task, in this case, phishing.

3. IMPROVING HUMAN EFFORT WITH COMPUTATIONAL TECHNIQUES

3.1 Improving Human Effort

Here, we outline a design space for improving human effort in phish identification. This design space is not comprehensive, but rather sketches out some of the opportunities at hand.

One area for improvement is modifying *the order in which suspicious URLs are shown* to participants. For example, one could show a submission that is closest to completion, newest

submissions, oldest submissions, or even random. One could also tailor what phish a participants sees based on their presumed knowledge of that brand or past votes. PhishTank's ordering has not been formally published; however, it does not seem to be by recency only. With Aquarium, we order submissions first by closest to completion and then by newest.

Another area for improvement is modifying *how submissions are shown*, for example showing them one-by-one or showing similar submissions together. In Aquarium, we compare the effectiveness of both of these approaches. We believe showing groups of suspicious URLs should help in two ways. First, one can apply a vote to multiple suspicious URLs simultaneously rather than going through them individually, mitigating the effect of attackers trying to overwhelm the people verifying these phishing sites. Second, for unfamiliar brands, seeing multiple copies of the same page, each of which have unusual URLs, can help participants in inferring whether or not the cluster is a phish.

A third possible intervention is to *adjust the threshold for when a submission is labeled*. PhishTank's threshold has not been formally published, but appears to require at least 4 votes minimum and at least 70% agreement between voters (with some votes weighted more than others). One could imagine many variants of this, including for example changing the minimum number of votes, changing the level of agreement needed (e.g. from 70% to 80%), changing how votes are weighted, and even having an automated algorithm provide a vote. Changing this threshold could affect accuracy, the time it takes to successfully label a submission, and breadth of coverage. In this paper, we experimented with changing how participants' votes are weighted.

A fourth kind of intervention is to find *better ways of motivating people* to submit more votes or more accurate votes. As we noted in the related work section, there have been several papers looking at how to motivate people to contribute more work and higher-quality work. In the domain of phishing, some possibilities include showing specific brands to people who either care a lot or know a lot about that brand, having competitions, organizing people into teams of voters with specific goals, and virtual rewards such as achievements or leaderboards. We do not investigate these issues in this current paper, and instead use MTurk's payment system.

3.2 Aquarium System Architecture

Our system architecture is shown in Figure 1. We first crawl the web pages of URLs submitted to PhishTank that have not yet been verified as phish. These URLs may or may not have any votes on them. PhishTank's API and web page do not show how many people have voted on unverified URLs. We submit these URLs as tasks to Amazon's Mechanical Turk, where qualified participants are paid to label them as phish or legitimate. Aquarium then clusters web pages by similarity before they are presented to users. We currently use DBSCAN and shingling, a common algorithm often used by search engines for detecting duplicate pages. To be qualified on Mechanical Turk, we required participants to achieve a certain score on the Anti-Phishing Phil micro game [31]. As participants cast votes, we weight those votes based on their history of votes.

In the first step, we collect URLs submitted to PhishTank as our test dataset. We use a small whitelist to filter legitimate web pages, to reduce effort by users. In February 2011, we collected 2,784 domains to whitelist from Google safe browsing [13] and 424 from millersmiles [14]. In our past research, we found that this combination of whitelist works reasonably well with minimal false positives [37][39].

Next, our system clusters similar phish together. We set the shingling similarity threshold to 0.65, a figure that worked well in our past work [39]. To demonstrate the potential of clustering, using all of the data crawled from PhishTank, we found 3,180 out of 3,973 web pages could be grouped into 392 clusters, with cluster size ranging from 2 to 153 URLs. Note that these clusters do not take into account time. For Aquarium, we cluster similar URLs currently available at that time. We also made the maximum size of clusters 25, high enough that clusters would be useful but low enough so that mistakes (or malicious votes) would have limited damage. The distribution of clusters after capping at 25 is shown in Figure 2.

Submissions are then submitted to Amazon's Mechanical Turk (MTurk) system as Human Intelligence Tasks (HITs) for verification. We submit two kinds of HITs. The first lets participants verify submissions one-by-one. The second one lets participants verify clusters of phish (see Figure 3). Participants saw a given URL at most once regardless of HIT condition.

Ideally, as participants vote on submissions, we can apply our *voteweight model* to modify the impact of a user's vote. Currently, we do not do this, and in this paper only examined the effects of voteweight after the fact. In the voteweight model, we consider two factors, namely a user's performance on verification and the time when a user casts a vote. Briefly, people who vote early and have a high accuracy in voting correctly are weighted more. We factor in time because an old vote does not tell us as much about a user's current performance as a more recent vote. The exact formula used is described in Section 6.1.

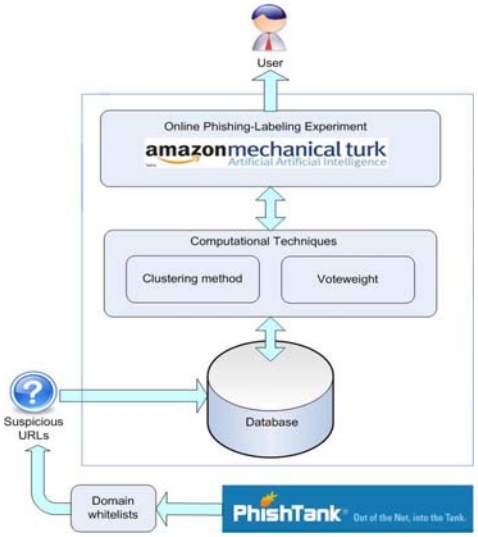


Figure 1. System architecture for Aquarium. We first crawl unverified URLs from PhishTank and check them against a whitelist. We download the web pages of URLs not on the whitelist. We use DBSCAN and shingling to cluster similar pages. We submit these clusters to Amazon's Mechanical Turk for verification by participants. Finally, each participant's vote weight is adjusted based on past performance.

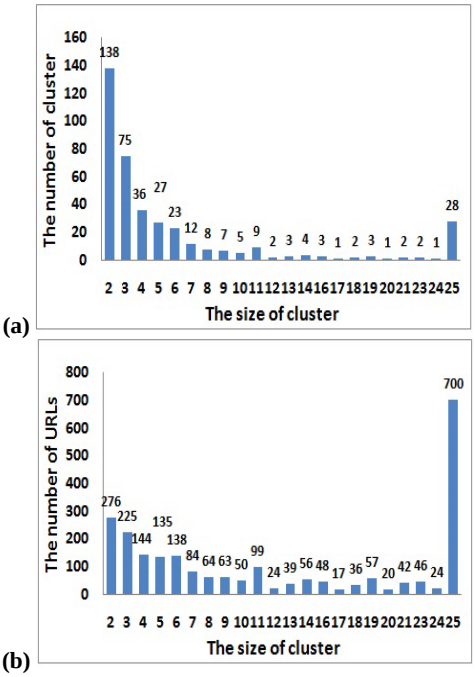


Figure 2. The distribution of clusters in our time-based approach to grouping. The top figure (a) shows that there are many small clusters of size 2 which quickly tail off. The bottom (b) shows the total number of URLs in different size of clusters. For example, we have 28 clusters of size 25, meaning that these clusters represent 28 x 25 = 700 URLs.

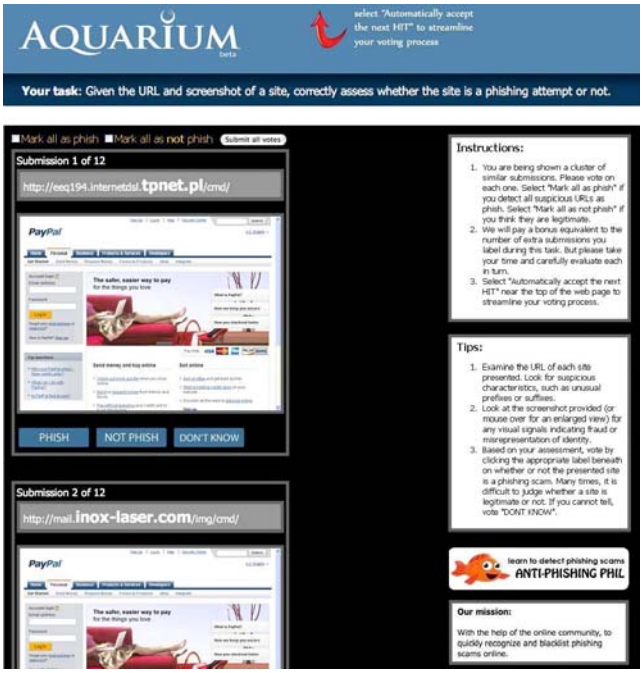


Figure 3. A sample task on Aquarium. Users can see the URL and screenshot of a suspicious web page and then label it as *phish, not phish,* or *don't know*. Users in the Cluster Condition (as shown above) could see up to 25 similar sites all at once. Participants in the Cluster Condition could "mark all as phish" or "mark all as not phish."

Like PhishTank, Aquarium requires a minimum of 4 votes. If the majority of votes for that URL identify it as phish, then we label that URL as phish (this mimics PhishTank's threshold of 70% with 4 votes). The same is true with legitimate URLs. However, if a URL has equal votes both for phish and legitimate, we label it as unidentified. In the Control Condition, there were 153 URLs (3.9%) not labeled due to tie votes. In the Cluster Condition, there were 127 (3.2%). Unlike PhishTank, we do not continue to gather more votes from people. This is primarily due to limitations with MTurk, which make it very difficult to have a variable number of workers per HIT. Although this does place caveats on our results, we argue that our results are very strong and should still generalize despite this weakness.

3.3 Measuring Page Similarity with Shingling

To cluster effectively, we need a way of measuring similarity. We could easily do exact page comparisons or use hash functions. Given that many phishing web pages are created using toolkits, this simple approach should work reasonably well today. In fact, in our early evaluations, we found that hash codes worked reasonably well for clustering. However, exact matching is very brittle, in that changing a single byte would lead to a non-match.

As such, we opted to use an approximate matching algorithm. Shingling is a popular page duplication algorithm invented for search engines. The core idea behind shingling is to break up web pages into n-grams and then compare how many n-grams two pages have in common. Here, *n-grams* are a term from natural language processing, and are subsequences of n contiguous tokens from the text. For example, sample text "shop without exposing your financial information" has the following 3-grams: {shop without exposing, without exposing your, exposing your financial, your financial information}.

Shingling employs a metric called *resemblance* to calculate the percent of common n-grams between two web pages. Let *S (p)* denote the set of unique n-grams in page p and the similarity metric resemblance *r(q, d)* for pages *q* and *d* is then defined as:

$$r(q,d) = \frac{|S(q) \cap S(d)|}{|S(q) \cup S(d)|} \quad (1)$$

This approximate matching approach first breaks each page into a set of unique n-grams, and saves them in memory to speedup runtime performance. After excluding good pages whose domains appear in the whitelist, we compute resemblance *r(q, d)* for a query page *q*, and fire an alarm whenever *r(q, d)* exceeds a threshold *t*. We used the same threshold as in our past work [39], namely 0.65. The average time cost of calculating similarity of two web pages on a laptop with 2GHz dual core CPU with 1 GB of RAM is 0.063 microseconds (SD=0.05).

3.4 Clustering Algorithm

Shingling is only a page similarity algorithm, so we still need a way of clustering similar pages together. In Aquarium, we used the well-known density-based DBSCAN algorithm. We chose this clustering method for two reasons. First, it can select any data point as the start point for clustering. Second, the algorithm only needs one scan of the database to finish clustering. The concepts used in our approach are described as follows:

Eps: Minimum similarity of the neighborhood of the cluster.

MinPts: Minimum number of points in an Eps-neighborhood of that point.

core point (CO): Point is in the interior of a density-based cluster.

border point: A border point is not a core point, but falls within the neighborhood of a core point.

directly-density-reachable (DDR): If point *x* is *CO*, point *y* is in x's Eps-neighborhood.

density-reachable: There exists a chain of DDR objects from point x to point y.

Based on the above, we present the clustering method as follows:
1) Given a submission *P*, quantify the similarity from *P* to each submission in the set through shingling.

2) Select *P* as the start point and retrieve all points density-reachable from *P* with respect to ***Eps*** and ***MinPts***.

3) If *P* is a core point, a cluster is formed.

4) If *P* is a border point, no points are density-reachable from *P* and DBSCAN visits the next submission.

5) Continue until all of submissions have been processed.

We tested on our data with different values of *Eps* from 0.6 to 1 by steps of 0.5 and *MinPts* of 2. With *Eps* at 0.60, the accuracy is 98.8% (we visually scanned all of the generated clusters). However, accuracy was 100% with *Eps* from 0.65 to 1. Hence, for our clustering, we chose *Eps*=0.65 and *MinPts*=2. The time cost of clustering over all 3,973 pages collected was about 1 second.

3.5 Incremental Update of the Data

Since there is a stream of suspicious URLs, the clusters discovered by our method need to be periodically updated. Clustering can be expensive in terms of time. However, it is not necessary to re-cluster the whole database each time. We use following method to assign a new URL to a cluster. We first compare the content similarity of each new submission with those of the dataset.

- If there is no similar web page, we create a new cluster for the new submission.
- If the similarity is above the given threshold and all similar web pages are in the same cluster, we assign the new submission to this cluster (unless the cluster is at its maximum size).
- If there are many similar webpapes in different clusters, we choose the largest cluster that is not at its maximum size.

When a new submission is grouped in a cluster, it has zero votes and does not inherit the votes of any other submissions in the same cluster. It is simply presented with other available similar submissions of the cluster for verification.

4. ONLINE PHISH-LABELING EXPERIMENT

We conducted an experiment to evaluate the effectiveness of our ideas. More specifically, we wanted to (a) assess how well clustering worked versus labeling each submission individually, (b) determine the effectiveness of various approaches for weighting votes, and (c) compare the effectiveness of Aquarium to existing blacklists in terms of time, accuracy, and coverage.

In an early pilot test of this work before clustering was implemented, we found that people often did not know certain brands and had a hard time labeling a site as phish or legitimate the first time they saw that brand. However, we also saw that people realized a site was phish after seeing the same site for the third or fourth time. Furthermore, we saw a large number of visually duplicate sites in our pool of URLs. This insight led us to add clustering as a possible way of improving accuracy as well as reducing time and overall effort.

4.1 Gathering Data with Mechanical Turk

Since PhishTank does not make its raw voting data easily available, and since we could not directly modify the PhishTank site, we created Aquarium to mimic the functionality of PhishTank. We used PhishTank's API to sample live data from the stream of sites being submitted. We then submitted both individual submissions as well as clusters of submissions as Human Intelligence Tasks (HITs) to Amazon's Mechanical Turk (MTurk), an online service designed to allow work requesters to quickly hire web-based workers by posting tasks for a set price. Workers were paid $0.01 for each HIT.

Normally, having MTurkers simply label data does not require an IRB at our university. However, since we had designed an intervention which was the subject of an experiment, we submitted an IRB, which was approved as a minimal risk study.

To compare Aquarium with PhishTank, we first collected unverified URLs submitted to PhishTank from Jan. 1, 2011 to Jan. 14, 2011. Unverified URLs are those that do not have enough votes to be verified as legitimate or phish. We also captured a screenshot of each submission when they were alive. We replayed this data as HITs over a different period of 14 days from Feb. 11 to Feb. 24 and mapped them to the submissions we downloaded from PhishTank from Jan. 1 to Jan. 14. Tasks were presented to users for verification only after the same time corresponding to when they were previously submitted to PhishTank. For example, suppose a suspicious URL was submitted to PhishTank at 2:51 am, Jan. 3, 2011. In our study, the task of such URL could be viewed by our participants only after 2:51 am, Feb. 13, 2011.

The Control Condition and the Cluster Condition were listed as separate HITs. Both conditions had the same exact data. Participants could move back and forth between the conditions. However, a participant only saw a given URL at most once. We chose this experimental design primarily because Mechanical Turk offers no facilities for enforcing between-subjects designs. Furthermore, we felt that there would be minimal learning effects if people switched between conditions.

To avoid having few votes at the beginning of the HIT and too many rushed votes at the end (which we saw in an earlier iteration of the experiment), we added a new HIT each day rather than having a single HIT last two weeks.

Since our task is one of identifying intentionally misleading websites, sites which criminals have deliberately built to deceive, we required our participants to complete a short training task using the first two rounds of Anti-Phishing Phil, which has been shown to increase phishing recognition in those who play it [31].

Though our model site, PhishTank, does not explicitly train users, we assume that users who participate there are more likely to be familiar with how to identify phish than our users recruited through MTurk, because of the selection bias of a volunteer opting

to donate time to participate. We also chose to train users to decrease the likelihood of low identification performance that lower participation users exhibit on PhishTank [26]. Once users completed both rounds of Anti-Phishing Phil, they were then eligible to complete our HITs. Participants who completed the game spent an average of 5.2 minutes (SD=6.5 minutes).

4.2 Task Design for Mechanical Turk

Figure 3 shows the Aquarium user interface that was presented to MTurk users. Our interface was modeled to be functionally similar to PhishTank's site, with the primary difference being that our interface did not display any voting progress indicators, unlike PhishTank which displays a breakdown of the voting percentages after a user has voted.

To complete a HIT, a participant only has to click "Phish", "Not Phish", or "Don't Know". In the Cluster Condition, participants can also select "Mark All as Phish" and "Mark All as Not Phish."

We sampled data from the PhishTank site on an ongoing basis, extracting newly-submitted potential phish, typically within minutes of being submitted to PhishTank. We crawled new submissions using an automated tool that we created that collects a screenshot as well as the raw content used in an actual browser rendering the suspicious site. This process was run in a virtual machine to protect against any content or malware attacks, and virtual machines were reset to a clean state approximately every 10 minutes. This data collection process allowed us to protect our participants from any possible malware, as well as provide a more uniform experience robust to some kinds of fast flux where phishing sites temporarily shut themselves down to interfere with detection (which we attempt to overcome by regularly re-checking sites which were previously down), network problems, or differences between participants' browsers and security settings.

The raw content extracted from each site by our tools represents all of the data that is required by a web browser to render the web page, including the final internal document representation that is used to render the web page on a normal user's screen, and is the content upon which we cluster submissions.

Once we had all the information for a submitted site, we added it to our live study site, where users were given tasks based on (a) closest to 4 minimum votes, and then (b) newest submission.

5. RESULTS OF EXPERIMENT

In this section we present the results of our study.

5.1 Summary of Participation Data

During the 2 weeks of this study, we had 267 users visit Aquarium, with 239 users participating. Of these 239, 174 cast votes in both conditions (as stated earlier, participants only saw a given URL at most once), 26 in the Control Condition only, and 39 users in the Cluster Condition only.

A total of 33,781 votes were placed, with 16,308 in the Control Condition, and 11,463 votes on clusters (yielding an equivalent of 17,473 votes on URLs without clustering) in the Cluster Condition. We paid $ 277.71 for the users for completed and approved HITs, and $198.96 to Amazon for approved HITs and bonus rewards, yielding a total cost of $476.67.

Because we only presented tasks to our participants if we were able to generate a thumbnail and download the site's content, our feed of submitted phish was not as large as PhishTank's, having 3,973 of the 5,686 submissions available from PhishTank. There were 1,713 submissions not used in the experiment since we could not obtain their screenshots.

We compared our results to four different resources. The first is the label from PhishTank identifying it as phish or not phish. We periodically checked the status of a given URL on PhishTank. If PhishTank updated their information, we would update our database accordingly. The second is the Google Safe Browsing API, which checks a given URL against their blacklist. We periodically checked the status of given URL using this API. The third is the SmartScreen Filter used by Microsoft Internet Explorer. We created a program that instantiated the MSIE browser in a virtual machine, visited the suspicious URL, and then analyzed the response of the IE browser to verify whether it is phishing or not. Fourth, when we could not obtain the status of a suspicious URL from above methods, we manually checked it. We use a queue to store the unverified URLs and repeatedly checked them following FCFS (First Come First Served) service discipline until they are verified. At worst, these URLs are checked every 10 minutes. We manually checked those URLs unverified by the first three methods during a given time (i.e. two weeks). In our study, we only manually checked 137 URLs, the majority of which were checking sites labeled as not phish. We also did not see any disagreement in the blacklists during the study period.

Using the above methods, we identified 3,877 as phishing URLs and 96 as not phish. Table 2 shows the comparison of PhishTank, Google Safe Browsing, and Microsoft's SmartScreen Filter during the study period. There are large differences between the average time and median time phish were reported, due to a power law distribution, which has also been reported in past work [26].

Also note that our reported coverage rates are different than from those in our past work [32]. This is primarily due to our source of phish, which is drawn from PhishTank rather than the UAB feed which is more comprehensive and has fresher phish. These previous results should still be considered more representative of blacklist behavior. Our results here should be viewed as a relative comparison of anti-phishing techniques on a sample of phishing attacks, rather than an absolute comparison.

Tables 3 and 4 show the results of our two conditions. In Table 3, the first row "All Votes in Control Condition" shows the TP and FP of all 16,308 votes cast in that condition. Note that we saw a FP rate of 2.6%, which is fairly high. The second row "All Labeled URLs in Control Condition" shows the results of our labels when compared to our four resources (i.e. PhishTank, Google Safe Browsing, Microsoft's SmartScreen Filter, and manual checks). Note that the labels for URLs have a reasonably good TP rate (94.8%), which is higher than individual votes (83.0%). Aggregating people's votes also led to 0% FP in our experiment. We saw no systematic errors in false positive votes.

The third row of Table 3, "All Votes on Clusters in the Cluster Condition", shows the TP and FP of all 11,463 votes on clusters. The fourth row, "All Labeled URLs in Cluster Condition", shows a comparable TP and FP rate to the Control Condition.

Table 4 shows Aquarium's performance with respect to coverage and time. Here, Aquarium does quite well compared to PhishTank, Google, and Microsoft. The coverage rate of our Control and Cluster Conditions (96.1% and 96.8% respectively) is higher than the other blacklists (89.2%, 65.7%, and 40.4%). However, it should be noted that our recorded coverage rates for PhishTank,

Google, and Microsoft do not take into account phishing pages that are taken down, since these blacklists may not bother labeling a phish that no longer exists.

To a large extent, this problem of not labeling a page that no longer exists would be less of a problem if blacklists could label pages faster, which would also provide better protection for people in the first few hours of an attack when people are most vulnerable [19]. Table 4 shows the average and median time to label a page in our two conditions. In particular, the clustered condition offers the best average time (1.8 hours, SD=2.6 hours) as well as median time (0.7 hours), outperforming all other blacklists by a wide margin.

	Coverage Rate	Avg Time (hours)	Median Time (hours)
PhishTank	89.2%	16.4 (SD=25.3)	3.98
Google Safe Browsing	65.7%	10.1 (SD=10.1)	8.47
SmartScreen Filter of MSIE	40.4%	24.5 (SD=24.0)	15.01

Table 2. Comparison of the coverage and time of 3973 URLs among PhishTank, Google Safe Browsing, and Microsoft's SmartScreen Filter. Given past work in this area, we assume that the false positive rate of Google and Microsoft are 0%.

	TP	FP
All Votes in Control Condition	83.0%	2.6%
All Labeled URLs in Control Condition	94.8%	0.0%
All Votes on Clusters in Cluster Condition	89.4%	0.05%
All Labeled URLs in Cluster Condition	95.4%	0.0%

Table 3. Comparison of True Positives and False Positives of all votes in the two conditions, as well as all labeled URLs based on those votes. The TP of votes from the Control Condition to the Cluster Condition improved by 6.4%, which was statistically significant (p=0.026). There were no other differences, however.

	Coverage Rate	Avg Time (hours)	Median Time (hours)
All Labeled URLs in Control Condition	96.1%	11.8 (SD= 22.6)	3.8
All Labeled URLs in Cluster Condition	96.8%	1.8 (SD=2.6)	0.7

Table 4. Comparison of URLs labeled in the two conditions. Due to tie votes, there are 96.1% URLs labeled in the Control Condition and 96.8% URLs in the Cluster Condition. FP was reduced to zero in both conditions.

5.2 Individual Human Accuracy

Earlier, we had hypothesized that clustering could help people identify phish better. For example, a given participant might not recognize a single instance of phish on an unknown brand (for example, a single instance of the Tibia phish shown in Figure 4), but seeing four instances of the same site but with different URLs would suggest that it is suspicious.

We calculated each individual's accuracy (true positives plus true negatives over all votes) based on votes in both conditions. Individual performance varied, with a mean accuracy of 82.7% (SD=23.3) in the Control Condition, and a mean accuracy of 86.7% (SD=18.5) in the Cluster Condition. In our experiment, 174 of these 239 users cast votes in both conditions. We compared their performance between two conditions using a paired t-test with one-tailed distribution. There was a statistically significant effect for clustering, $t(173)=2.78$, $p<0.05$ ($p=0.006$), with users' performance in the Cluster Condition obtaining higher accuracy than that in Control Condition. As such, our results support our hypothesis that clustering helps people identify phish better. We also examined if the size of a cluster helped with accuracy. There was marginal improvement, but not statistically significant.

Figure 5 shows the overall performance of all participants sorted by performance and organized into deciles. The top 50% of participants performed very well in both conditions. However, there is a large dropoff in performance in the Control Condition, with the bottom 10% of MTurkers in the Control Condition performing under 30%. We suspect this is due to lazy workers.

5.3 Reducing Effort Using Task Clustering

To determine if the Cluster Condition is more effective in determining if a submission is a phishing attack, we looked for a difference in the performance of users in evaluating submissions. Time to label is an important metric here, as it measures both how quickly a user was able to identify an attack, and is a coarse representation of the effort required to complete the task.

Participants in the Control Condition took 11.8 (SD=22.6) hours on average to label a site, whereas participants in the Cluster Condition took 1.8 (SD=2.6) hours. By comparing two conditions with one-tailed paired t-Test, there was a significant main effect on clustering, $t=23.63$, $p<0.001$, with much less time used in identifying a URL in Cluster Condition than that in Control Condition. Comparing median times (3.8 hours to 0.7 hours) yields a similar result.

Figure 4. Four phishing examples from a cluster collected during our study. In this case, all 42 submissions in the cluster were nearly or completely visually identical. In this case, it

should not be hard to identify the phishing attack, as the sites are identical, but do not share the same primary domain name.

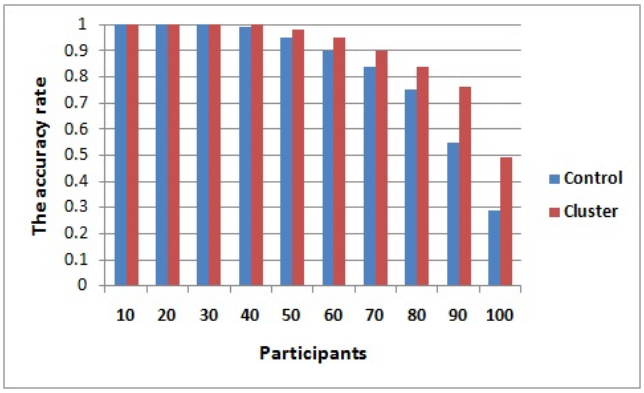


Figure 5. Average accuracy for each decile of users, sorted by accuracy. For example, the average accuracy of the top 10% of users in both conditions was 100%, whereas the average accuracy of the bottom 10% was under 30% for the Control Condition and under 50% in the Cluster Condition.

6. INVESTIGATING VOTE WEIGHT

In this section, we present our analysis and tuning of vote weight.

6.1 Voteweight

The core idea behind voteweight is that participants who are more helpful in terms of time and accuracy are weighted more than other participants. Weighting votes more accurately should also help reduce the time it takes to label a submission. Our notion of voteweight is similar to the concept of mavens in the Acumen system [10], though we examine a different domain (phishing vs cookies) and leverage time in our model. Our voteweights are also continuous, whereas mavens were chosen to be the top 20% of users in Acumen.

Intuitively, a correct vote should be rewarded and a wrong one should be penalized. In addition, recent behavior should be weighted more than past behavior, as it gives us a better sense as to a participant's current abilities (or level of malice).

Towards this end, we propose a metric called voteweight in Eq.(2) that combines these factors in one summary statistic. In our model, we use $y \in \{t, +\infty\} \cup y \in \{-t, -\infty\}$ to label the status of a URL, where y is the sum of voteweight of a given URL, t is the threshold of voteweight, and $y \geqslant t$ means a URL has been voted as a phishing URL and $y \leqslant -t$ means voted as legitimate.

$$v_i^{'} = \frac{v_i}{\sum_{k=1}^{M} v_k} \tag{2}$$

Here, $v_i^{'}$ is the normalized voteweight of user i, with a value between [0, 1]. This value will be the voteweight that we use for the user. v_i is the raw voteweight, and M is the number of users. Note that because our normalized value is less than 1 we also have to adjust our threshold accordingly.

$$v_i = \begin{cases} RV_i & if\ RV_i \geq 0 \\ 0 & otherwise \end{cases} \tag{3}$$

$$RV_i = R_i - \alpha \cdot P_i \tag{4}$$

Equations 2 and 3 show the formulas for raw voteweight (RV). Conceptually, raw voteweight for a user i is the sum of the rewards for correct votes $R_{i,}$ minus a weighting parameter α times the penalization for incorrect votes P_i. Adjusting the parameter α allows us to weight the penalty relative to the reward. For example, an α value greater than 1 means that participants are penalized more heavily for wrong votes than they are rewarded for correct votes.

$$R_i = \sum_{j=1}^{N} \frac{T_j - T_0 + 1}{T - T_0} \cdot I_{C_{ij} = L_j}(j) \tag{5}$$

$$P_i = \sum_{j=1}^{N} \frac{T_j - T_0 + 1}{T - T_0} \cdot I_{C_{ij} \neq L_j}(j) \tag{6}$$

Equations 5 and 6 show our reward and penalty formulas. The first part of both equations 5 and 6 show the weight we give to time. Here, T_0 is the timestamp of user i's first vote ever; T_j is the timestamp of user i's vote on phish candidate j; and T is the current time when computing user i's voteweight based on the historical vote information. Thus, if T_j is recent and close to current time T, then this first part is close to 1. If T_j is very old, then the first part becomes smaller, meaning that older votes have less weight. In our study, we calculated the interval of time in hours.

There are alternative variations for weighting time that also could have worked, for example, having a sliding window of the last N days of votes, taking only the last N votes, and so on. We wanted to explore how well any voteweight feature worked first before trying the many alternatives. As such, we chose one that worked well with the two weeks of data we had.

The right half of equations 5 and 6 are an indicator function with a value of 0 or 1. Essentially, for the reward formula, we want the indicator to be 1 if they voted correctly and 0 otherwise. For the penalty formula, the opposite is true. More formally, C_{ij} is the label that user i assigns to phish candidate j; L_j is the ground truth label for phish candidate j; N is the number of phish candidates that user i has voted on; $I_A(x)$ is an indicator function in mathematics which is defined as:

$$I_A(x) = \begin{cases} 1 & if\ x \in A \\ 0 & if\ x \notin A \end{cases} \tag{7}$$

With our voteweight, we can determine the label for a candidate phish based on users' vote by

$$l_t = \sum_{i=1}^{K} v_i^{'} \cdot C_{it} \tag{8}$$

where the label of phish candidate t is a weighted average of the votes by K users and the value of a vote is defined as

$$C_{it} = \begin{cases} 1 & if\ voted\ as\ phish \\ -1 & otherwise \end{cases} \tag{9}$$

6.2 Tuning the Voteweight Parameters

In this experiment, we tuned the parameter required in Eq. (4) to optimize the accuracy rate and time cost in labeling URLs.

We tested on 16,308 votes from the Control Condition on 3,973 URLs, and 11,463 votes on clusters from the Cluster Condition on 3,973 URLs using different values of α ranging from 0.5 to 9 in increments of 0.5. Again, a higher α here means that incorrect votes incur a higher penalty relative to the reward. We calculated the accuracy and time cost when a URL was identified based on different values of voteweight.

We also tested the threshold of voteweight from 0.01 to 1 by steps of 0.01. Figure 6 only shows the results under the threshold from 0.01 to 0.20, since there was no improvement above 0.20. Again, our label for a URL was determined based on the voteweight it obtained beyond the threshold. For this tuning, we recalculated voteweight after each hour.

Figure 6 shows our results for both the Control Condition and Cluster Condition. Figure 6a shows how accuracy varies as the value of α increases. Accuracy increases incrementally for a while and then plummets dramatically when α approaches 4.5 regardless of the threshold. This finding suggests that an appropriate penalty can offer a small benefit in terms of distinguishing between skilled and unskilled participants. However, excessive punishment on occasional errors of users dramatically decreases performance. Figure 6b shows that as the threshold *t* is increased, the time cost of identifying a URL also increases, as one might expect.

Overall, in the Control Condition, voteweight obtained its highest accuracy (true positives and true negatives over all votes) of 95.6% when the threshold of *t* is 0.08 and α is 2.5. At these values, the average time cost was 11.0 hours and median time cost 2.3 hours.

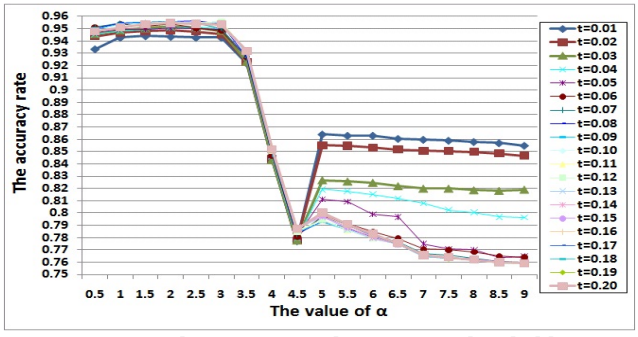
Given this tuning, how much improvement does voteweight offer over not having it? In the Control Condition (without voteweight), Table 3 shows that the true positive rate was 94.8% and false positive rate was 0%, with an average time cost of 11.8 hours (SD= 22.6) and a median of 3.8 hours. By using voteweight, we can achieve a comparable accuracy, and reduce the average time by 0.8 hours and reduce the median time by 1.5 hours.

In the Cluster Condition, we obtain the highest accuracy rate at 97.2% when threshold *t* is 0.06 and α is 1. With this accuracy rate, the average time is 0.8 hours and median time is 0.5 hours. In the Cluster Condition without voteweight, the true positive rate is 95.4%, with an average time of 1.8 hours (SD=2.6) and median time of 0.7 hours. By using voteweight, we can reduce the average time cost by about 1 hour and reduce the median time cost by about 0.2 hours, while achieving comparable accuracy.

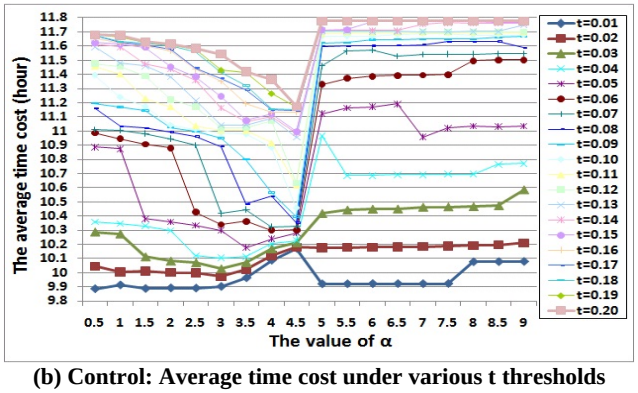
7. DISCUSSION

Overall, our results with Aquarium are quite positive. Clustering alone achieves a true positive rate of 95.4%, with an average time of 1.8 (SD=2.6) hours and a median time of 0.7 hours. Applying voteweight without clustering also yielded stronger results, reducing the average time by 0.8 hours and the median time by 1.5 hours. Combining these two techniques yielded the best results. Our total cost for both conditions was under $500 to compensate 239 participants.

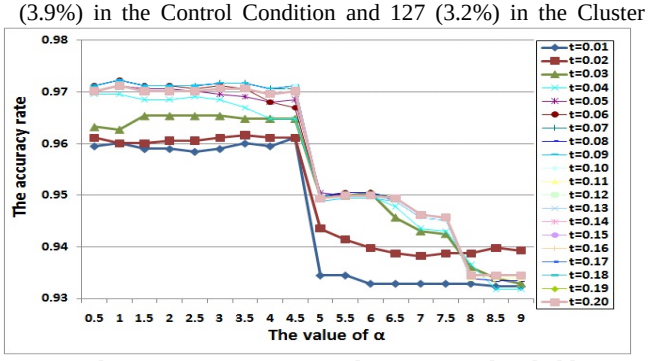
As noted earlier, one limitation of this study is that 153 URLs (3.9%) in the Control Condition and 127 (3.2%) in the Cluster



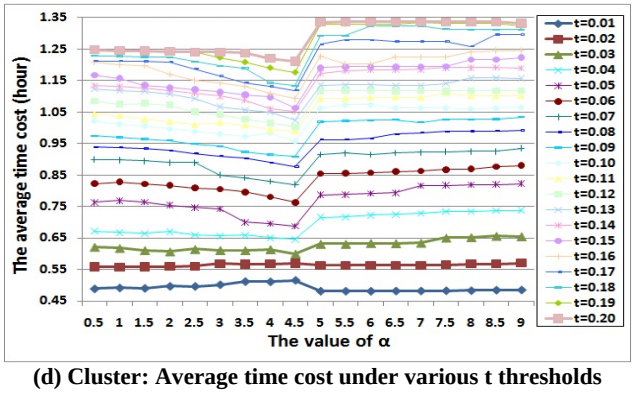
(a) Control: Accuracy under various t thresholds



(b) Control: Average time cost under various t thresholds



(c) Cluster: Accuracy Rate under various t thresholds



(d) Cluster: Average time cost under various t thresholds

Figure 6. Voteweight parameter tuning in the Control and Cluster Conditions. *t* is the threshold of votepower, α is the weight of penalty for wrong verification of URLs. As α increases, the accuracy first increases a little and then drops down quickly while the average time cost increases in a small range. When *t* increases, the average time cost increases accordingly. Voteweight achieves its best accuracy with *t*=0.08 and α=2.5 in the Control Condition and *t*=0.06 and α=1 in the Cluster Condition.

Condition were not identified due to tie votes. It is possible that these were the most difficult URLs to verify and could lead to longer tails for time or lower accuracy rates. However, we would also argue that the results that we do have are quite strong and still represent an advance over current manual verification as well as many published algorithms.

Another limitation of this study was imposed by Mechanical Turk itself. We have discussed two of these in the paper already, including the challenges in running a strict between-subjects study on MTurk, and dynamically adding more participants to URLs if there is not strong agreement.

A third limitation of this study is that there are possible learning effects, since both conditions were run simultaneously and that some people were in both conditions. Given the limitations of our collected data, it will be difficult to tease out whether there were learning effects involved. What we can say, though, is that Aquarium does not offer feedback as to whether a vote is correct or not, minimizing one potential angle for learning. Furthermore, all participants had to train on Anti-Phishing Phil, helping to level the playing field. Lastly, we do not prioritize one condition over another, so neither condition should have any substantial benefits if there are learning effects.

Based on their study on PhishTank [26], Moore and Clayton offered three lessons for improving crowdsourcing for security: (1) addressing power-law issues of participation, (2) having crowdsourced decisions be hard to guess, and (3) not having users work harder than necessary. We addressed the third issue in this paper, and so only discuss the first two issues below.

For the first issue, we do not address power-law issues of participation directly. Ipeirotis also showed that task completion times follow power-law distributions on MTurk [15]. What we have demonstrated, however, is that recruiting workers on MTurk can be a reasonably good approximation of PhishTank, as evidenced by Aquarium's coverage rates and median times in the Control Condition, which are comparable to PhishTank's. Here, Aquarium does slightly better than PhishTank, possibly due to a combination of training with Anti-Phishing Phil and the number of workers available on MTurk and their timeliness.

For the second issue, having crowdsourced decisions be hard to guess, there is a lot of room for improvement in Aquarium. We did not encounter workers actively trying to manipulate our system, unlike PhishTank. As such, we can consider this paper to be a study under optimistic conditions without active adversaries. However, previous studies have encountered the problem of lazy workers doing the minimum amount of work needed to be paid. We had a few participants that underperformed, though we believe that using Anti-Phishing Phil as a qualification task helped filter out many other workers who would have underperformed.

A system with active adversaries would have several challenges, as the risk is increased with clustering and voteweight, especially since the prior probability of phish is quite high. There are several ways to counter this problem of lazy workers and active adversaries. One is increased randomization of what URLs are presented, helping to minimize the impact of a lazy worker and making it harder for attackers to label their own phishing site or to coordinate attacks. Given the large pool of workers on MTurk and the relatively short time it takes to identify a phish on Aquarium, this approach would also give attackers only a small time window to try to manipulate the system.

Another countermeasure would be to calculate baseline performance based on known results. For example, in an earlier iteration of our experiment, we calculated a baseline performance to see if people were improving over time (they were, but marginally) and to verify that users were not merely always clicking "Phish". We introduced a controlled stream of previously resolved tasks to periodically measure participant performance. The first 20 tasks completed by each user were selected from the previously-resolved cases, as well as at least 1 of every 10 subsequent tasks, with a fixed 20% of these test cases selected from the "not-phish" resolution to provide a minimum of not-phish cases a participant would experience. This periodic testing would allow us to estimate user performance and detect low performing or malicious workers. One could also tie MTurk bonuses to baseline performance, reducing the incentives of lazy workers. Such an approach could be used in a deployed system, at the cost of more time, money, participants, and votes.

Other countermeasures include ones we previously identified in the design space, such as increasing the number of votes required or using existing automated phish-detection algorithms as a backup check or additional vote.

8. CONCLUSION

Purely computational approaches to detecting phish are popular in the research community, but have not seen adoption primarily due to concerns about false positives. In contrast, manual verification is the norm today, but has potentially long lag times and do not scale well.

In this paper, we presented the design and evaluation of Aquarium, a system that uses computational techniques to improve crowdsourcing in identifying phish. Specifically, we outlined the design space of techniques for improving the wisdom of crowds, investigated the use of clustering and weighting of votes, and presented the results of our evaluation. Through a two-week study replaying submissions on PhishTank, and using minimally trained participants from Amazon's Mechanical Turk, we demonstrated that clustering and weighting of votes can be very effective in terms of accuracy, time, and monetary cost.

Our work in this paper represents two points in the design space for improving the wisdom of crowds for phishing. Our ideas can be easily adopted by existing manually-verified blacklists such as those operated by Google, Microsoft, and PhishTank. Other applications include forensic analysis of phishing trends and maintaining databases of phishing attacks. Our work also represents one way of using crowdsourcing techniques for computer security, which may be a useful approach for other kinds of computer security problems that require a human in the loop.

9. ACKNOWLEDGMENTS

This research was supported by CyLab at Carnegie Mellon under grants DAAD19-02-1-0389 and W911NF-09-1-0273 from the Army Research Office. It was also supported by a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China [Project No. CityU 117907].

10. REFERENCES

[1] Ahn, L. and Dabbish, L. 2004. Labeling images with a computer game. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI'04), 319-326.

[2] Cosley, D., Frankowski, D., Terveen, L., and Riedl, J. 2007. Suggestbot: Using intelligent task routing to help people find work in wikipedia. In Proceedings of the 12th International Conference on Intelligent User Interfaces (IUI'07), 32-41.

[3] Chen, T, Dick, S. and Miller, J. 2010. Detecting visually similar Web pages: Application to phishing detection. In ACM Transactions on Internet Technology (TOIT), Vol. 10(2).

[4] Chou, N., Ledesma, R., Teraguchi, Y. and Mitchell, J. 2004. Client-side defense against web-based identity theft. In Proceedings of the 11th Annual Network and Distributed System Security Symposium (NDSS'04).

[5] Dhamija, R. and J. D. Tygar. 2005. The battle against phishing: dynamic security skins. In Proceedings of the 2005 Symposium on Usable Privacy and Security (SOUPS'05), 77-88.

[6] Dhamija, R., J. D. Tygar, and Hearst, M. 2006. Why phishing works. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI'06), 581-590.

[7] Edwards, W., Poole, E., and Stoll, J. 2007. Security automation considered harmful. In Proceedings of the IEEE New Security Paradigms Workshop (NSPW'07), 33-42.

[8] Egelman, S., Cranor, L., and Hong, J. 2008. You've been warned: An empirical study of the effectiveness of web browser phishing warnings. In Proceeding of the 26th Annual SIGCHI Conference on Human Factors in Computing Systems (CHI'08), 1065-1074.

[9] Garera, S., Provos, N., Chew, M., and Rubin, A. D. 2007. A framework for detection and measurement of phishing attacks. In Proceedings of the 2007 ACM Workshop on Recurring Malcode (WORM'07), 1-8.

[10] Goecks, J. and Mynatt, E. D. 2005. Supporting privacy panagement via pommunity experience and expertise. In Proceedings of the 2nd Communities and Technologies Conference, 397-418.

[11] Golder, S. A. and Huberman, B. A. 2006. Usage patterns of collaborative tagging systems. Journal of Information Science, Vol. 32(2), Apr. 2006, 198-208.

[12] Heer, J. and Bostock, M. 2010. Crowdsourcing graphical perception: using mechanical turk to assess visualization design. In Proceedings of the 28th international conference on Human factors in computing systems (CHI'10), 203-212.

[13] http://sb.google.com/safebrowsing/update?version=goog-white-domain:1:1.

[14] http://www.millersmiles.co.uk/scams.php.

[15] Ipeirotis, P. 2010. Analyzing the amazon mechanical turk marketplace, NYU Working Paper No. CEDER-10-04.

[16] Karau, S. and Willianms, K. 1993. Social loafing: A meta-analytic review and theoretical integration. Journal of Personality and Social Psychology. Vol 65(4), Oct. 1993, 681-706.

[17] Kirda, E. and Kruegel, C. 2005. Protecting users against phishing attacks with antiPhish. In Proceedings of the 29th Annual International Computer Software and Applications Conference (COMPSAC'05), 517-524.

[18] Kittur, A., Chi, E. H., Suh B. 2008. Crowdsourcing user studies with mechanical turk. In Proceedings of the 26th annual SIGCHI conference on Human factors in computing systems (CHI'08), 453-456.

[19] Kumaraguru, P., Cranshaw, J., Acquisti, A., Cranor, L., Hong, J., Blair, M., and Pham, T. 2009. School of phish: A real-word evaluation of anti-phishing training. In Proceedings of the 5th Symposium on Usable Privacy and Security (SOUPS'09)

[20] Liu, W., Huang, G., Liu, X., Zhang, M. and Deng, X. 2005. Detection of phishing webpages based on visual similarity. In Proceedings of the special interest tracks and posters of the 14th international conference on World Wide Web (WWW'05), 1060-1061.

[21] Ludl, C., McAllister, S., Kirda, E., Kruegel, C. 2007. On the effectiveness of techniques to detect phishing sites. Lecture Notes in Computer Science (LNCS). Vol. 4579/2007, 20-39.

[22] Mason, W. and Watts, D. J. 2009. Financial incentives and the "performance of crowds". In Proceedings of the ACM SIGKDD Workshop on Human Computation (HCOMP'09), 77-85.

[23] Medvet, E., Eurecom, E., and Kruegel. C. 2008. Visual-similarity-based phishing detection. In Proceedings of the 4th international conference on Security and privacy in communication networks (SecureComm'08), 30-36.

[24] Millen, D., Yang, M., Whittaker, S., and Feinberg, J. 2007. Social bookmarking and exploratory search. In Proceedings of the 2007 Tenth European Conference on Computer-Supported Cooperative Work (ECSCW'07), 21-40.

[25] Moore, T. and Clayton, R. 2007. Examining the impact of website take-down on phishing. In Proceedings of the Anti-Phishing Working Groups 2nd Annual Ecrime Researchers Summit (eCrime'07), 1-13.

[26] Moore, T. and Clayton, R. 2008. Evaluating the wisdom of crowds in assessing phishing websites. Lecture Notes in Computer Science (LNCS). Vol 5143/2008, 16-30.

[27] Pan, Y. and Ding, X. 2006. Anomaly Based Web Phishing Page Detection. In Proceedings of the 22nd Annual Computer Security Applications Conference (ACSAC'06), 381-392.

[28] Rosiello, A., Kirda, E., Kruegel, C. and Ferrandi, F. 2007. A layout-similarity-based approach for detecting phishing pages. In Proceedings of the 3rd International Conference on Security and Privacy in Communication Networks (SecureComm'07), 454-463.

[29] Ross, B., Jackson, C., Miyake, N., Boneh, D. and Mitchell, J. 2005. Stronger password authentication using browser extensions. In Proceedings of the 14th conference on USENIX Security Symposium, 17-32.

[30] Sheng, S., Kumaraguru, P., Acquisti, A., Cranor, L., Hong, J. 2009. Improving phishing countermeasures: an analysis of expert interviews. In Proceedings of the Anti-Phishing Working Groups 4th Annual Ecrime Researchers Summit (eCrime'09), 1 – 15.

[31] Sheng, S., Magnien, B., Kumaraguru, P., Acquisti, A., Cranor, L., Hong, J., and Nunge, E. 2007. Anti-Phishing phil:

The design and evaluation of a game that teaches people not to fall for phish. In Proceedings of the 3rd Symposium on Usable Privacy and Security (SOUPS'07), 88-89.

[32] Sheng, S., Wardman, B., Warner, G., Cranor, L., Hong, J., and Zhang, C. 2009. An empirical analysis of phishing blacklists. In Proceedings of the 6th Conference on Email and Anti-Spam (CEAS'09).

[33] Statistics about Phishing Activity and Phishtank Usage. (2011). Retrieved January, 2011, from http://www.phishtank.com/stats/.

[34] Surowiecki, J. 2004. The wisdom of crowds: Why the many are smarter than the few and how collective wisdom shapes business, economies, societies and nations. Doubleday.

[35] Weaver, R. and Collins, M. P. 2007. Fishing for phishes: Applying capture-recapture methods to estimate phishing populations. In Proceedings of the Anti-Phishing Working Groups 2nd Annual Ecrime Researchers Summit (eCrime'07), 14-25.

[36] Wu, M., Miller, R. C., and Little, G. 2006. Web wallet: Preventing phishing attacks by revealing user intentions. In Proceedings of the 2nd Symposium on Usable Privacy and Security (SOUPS'06), 102-113.

[37] Xiang, G. and Hong, J. 2009. A hybrid phish detection approach by identity discovery and keywords retrieval. In Proceedings of the 18th International Conference on World Wide Web (WWW'09), 571-580.

[38] Xiang, G., Pendleton, B. A., Hong, J. 2009. Modeling content from human-verified blacklists for accurate zero-hour phish detection. Technical report, CMU-LTI-09-005.

[39] Xiang, G., Pendleton, B. A., Hong, J., and Rose, C. P. 2010. A hierarchical adaptive probabilistic approach for zero hour phish detection. In Proceedings of the 15th European Symposium on Research in Computer Security (ESORICS'10), 268-285.

[40] Yue, C. and Wang, H. 2010. BogusBiter: A transparent protection against phishing attacks. In ACM Transactions on Internet Technology (TOIT), Vol. 10(2).

[41] Zhang, Y., Egelman, S., Cranor, L., and Hong, J. 2007. Phinding phish: An evaluation of anti-phishing toolbars. In Proceedings of the 14th Annual Network & Distributed System Security Symposium (NDSS 2007).

[42] Zhang, Y., Hong, J., and Cranor, L. 2007. Cantina: A content-based approach to detecting phishing web sites. In Proceedings of the 16th International Conference on World Wide Web (WWW'07), 639-648.