MetaCAPTCHA: A Metamorphic Throttling Service for the Web

Akshay Dua*, Thai Bui*, Tien Le*, Nhan Huynh* and Wu-chang Feng*
* {akshay, buithai, letien, nhhuyng, wuchang}@cs.pdx.edu
Department of Computer Science
Portland State University

***Abstract*—Spam is a problem that refuses to go away. An immense amount of time and money is currently devoted to hiding spam, but not enough is devoted to effectively preventing it. CAPTCHAs are a prevalent spam prevention mechanism, but are getting harder for humans to solve and easier for programs to "break". CAPTCHAs also cannot prevent spam from hijacked accounts since they are mostly used during account creation. Proof-of-work approaches are gaining popularity, but current implementations are not effective enough and cannot be used by generic web applications. We present MetaCAPTCHA, an application-agnostic spam prevention service for web applications. It dynamically issues CAPTCHAs and proof-of-work "puzzles" while ensuring that more malicious users solve "harder" puzzles. In order to support its operation, MetaCAPTCHA implements a novel secure protocol — for authenticating web sites and validating proof-of-work solutions — that allows for seamless integration across a variety of web applications.**

I. Introduction

Internet spammers are relentless. Although, email spam is reducing ($\approx$ 70.5% in Jan 2012 from 92.2% in Aug 2010), the spam on social network sites is edging up [1]. Approximately 4 million Facebook users receive spam from around 600,000 new or hijacked accounts each day [1], [2]. What's worse is the success rates of social spam: in Jan 2010, 0.13% of all spam URLs on twitter were visited by around 1.6 million unsuspecting users [3]. This "clickthrough" rate is almost two orders of magnitude larger than for email spam. Spammers cost businesses $20.5 billion annually in decreased productivity and technical expenses, and this cost is projected to rise to $198 billion in the next four years [4]. There are two prevalent methods to prevent this deluge of spam: *CAPTCHA* and *proof-of-work*; both methods have benefits and drawbacks [5].

A CAPTCHA can effectively protect an online transaction so long as there aren't OCR algorithms that can automatically "solve" or "break" it [6]. Once a class of CAPTCHAs is broken, the corresponding application becomes defenseless against spam bots. CAPTCHAs are also prone to outsourcing attacks where humans are employed to solve CAPTCHAs *en masse*. A major cause of success for these attacks is that CAPTCHAs don't provide a way to change the cost of solving them [7], [8]. Additionally, the usability burden imposed by CAPTCHAs [9] limits their use to only protecting infrequent transactions like creating accounts. This leaves frequent transactions, like message posting, open to abuse. Attackers exploit this loophole by hijacking accounts and using them to send spam.

Proof-of-work does not have CAPTCHA's usability issues and can therefore be used in frequent transactions. This is because the assigned "work" can be done automatically by the device without user intervention. Additionally, this paradigm enables an application to price a transaction by varying the amount of work that needs to be done as payment. However, proof-of-work systems are only effective if the price of the transaction is based on the corresponding user's reputation [10]. Unfortunately, many proposed proof-of-work systems do not accurately characterize user reputation [11] and those that attempt to do so, are too tightly integrated with a given application [12], [13], [11], [14].

In this work, we describe MetaCAPTCHA, an application-agnostic spam prevention service for the web. In order to support its operation, MetaCAPTCHA implements a novel secure protocol — for authenticating web sites and validating proof-of-work solutions — that allows for seamless integration across a variety of web applications. Specifically, MetaCAPTCHA makes the following contributions:

- It integrates the CAPTCHA and proof-of-work approaches while augmenting each: it can dynamically issue proof-of-work or CAPTCHA puzzles while ensuring that malicious users solve much "harder" puzzles than honest users. More specifically, our results show that honest users were never issued a puzzle during 95% of their transactions.
- Puzzles are randomly picked and delivered within a generic solver that eventually executes those puzzles in the user's web browser. Thus, the solver code is metamorphic: changing randomly in each transaction. This *turns the reverse engineering problem around on the adversary* who must now attach a debugger to discover the solver's execution steps.
- It uses a Bayesian reputation system that can accurately predict a user's reputation score based on features configured by the web application. Since multiple web applications can be protected by MetaCAPTCHA, its reputation system provides global visibility on attacks across all those applications.
- It contains a modular puzzle library that can be configured with new classes of CAPTCHAs or proof-of-work puzzles while allowing the removal of those classes that are

known to be "broken". These puzzle library modifications can be made by the web application without *any change to its source code*. Furthermore, the variety of puzzles in the library ensures that breaking one class of puzzles won't compromise MetaCAPTCHA as a whole.
- Web applications can easily install the MetaCAPTCHA API by making changes similar to those required by existing CAPTCHA implementations [15], [16].

II. Background

MetaCAPTCHA dynamically issues CAPTCHA and proof-of-work puzzles. We now provide a brief background on each kind of puzzle.

A. CAPTCHA

CAPTCHA stands for "Completely Automated Public Turing-test to tell Computers and Humans Apart". CAPTCHAs usually consist of images containing squiggly characters that are easy for humans to read, but hard for programs to parse. The idea is to allow humans to access the web application's services while deterring automated adversaries like bots. A popular implementation of the CAPTCHA is the reCAPTCHA [17].

B. Proof-of-work

The proof-of-work approach was first proposed by Dwork and Naor [18] to combat email spam. The idea was to impose a per-email cost on senders, where, the cost was in terms of computational resources devoted by the sender to compute the pricing function. Once a sender proved that it correctly computed the pricing function, the server would send the email. Effectively, sending bulk spam would become "expensive" because computational resources are finite. The characteristics of such a pricing function f was then described as follows:

1) "moderately" easy to compute
2) not amenable to amortization: given any l values $m_1, \ldots, m_l$, the cost of computing $f(m_i)$ is similar to the cost of computing $f(m_j)$ where $i \neq j$. In other words, no amount of pre-processing should make it easier to compute f on any input.
3) Given x and y, it is easy to check if $y = f(x)$

An example of a pricing function is one that finds partial hash collisions [19]. A function $f_k : x \rightarrow y$ is said to compute a k-bit partial hash collision on string x, if given a hash function H, the first k bits of $H(x)$ are equal to the first k bits of $H(y)$. Notice that $f_k(\cdot)$ has all the properties required of a pricing function.

Although the proof-of-work approach seemed promising, Laurie and Clayton [10] demonstrated in 2004 that reducing spam to 1% of normal email would require delaying each message — including one that an honest user sends — by $\approx$ 6 minutes; a high price to pay for innocent users. This delay was computed based on then current rates of spam, number of email users, and under the assumption that 1 million compromised machines were spewing spam. Since then, spam has increased by 18% to 74.2%, so we expect the aforementioned delay to be much larger now.

To reduce this delay, Liu and Camp [20] proposed basing puzzle difficulties on user reputation. The idea was that users with lower reputations would receive harder puzzles than those with higher reputations. Since easier puzzles would be much quicker to solve, honest users would experience a nominal delay when sending messages where as malicious users may be significantly delayed. Thus, with an accurate reputation system, the proof-of-work approach can be a practical, fair, and effective technique for combating spam.

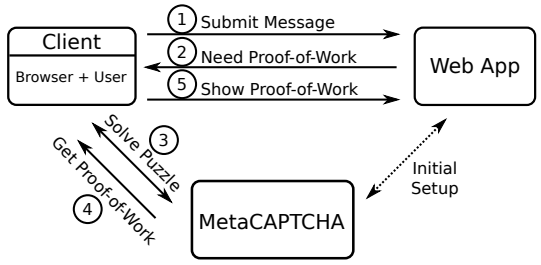


Fig. 1. System model: user's browser must show proof-of-work before the web application accepts the user's message. The dotted line indicates initial setup performed by the web application to use the MetaCAPTCHA service.

III. System Model

This section describes the system model in which MetaCAPTCHA is applicable. In general, interactive web applications where online transactions can be exploited by spammers, such as message forums, webmail, social applications, and event-ticket purchasing can employ MetaCAPTCHA for spam prevention. Heymann et al. [5] provide an exhaustive discussion on the common characteristics of such web applications.

An overview of the system model and high-level interactions between the MetaCAPTCHA service, the web client, and the corresponding web application is shown in Figure 1. The interactions begin when a user attempts to perform an online transaction. The web application allows the transaction to proceed only when it has sufficient proof that the client completed the work it was assigned by MetaCAPTCHA.

As shown in Figure 1, we treat the *user* separate from the *browser* while collectively referring to them both as the *client*. The next section discusses the details of how a user interacts with a web application protected by MetaCAPTCHA.

IV. Communication Protocol

This section discusses the MetaCAPTCHA communication protocol. For simplicity, we assume the scenario where a client is attempting to post a message. Note, however, that MetaCAPTCHA can protect more general web transactions like purchasing event tickets, creating accounts, etc.

A web client begins communicating with MetaCAPTCHA after being referred by the corresponding web application. In this case, the application will refer a client attempting to post a message to MetaCAPTCHA. The client will then

need to obtain and solve a puzzle. The idea is that the web application will allow messages from only those clients that have successfully solved a puzzle issued by MetaCAPTCHA. The communication protocol for obtaining and solving a puzzle begins with authentication as explained in the next section.

A. Authentication

MetaCAPTCHA only issues puzzles to clients of participating web applications. This requires MetaCAPTCHA to authenticate two things, (i) the identity of the web application, and (ii) the client is an authorized user of the web application. MetaCAPTCHA provides each web application with an API key K during a registration phase. The web application must keep K secret as it will later be used for authenticating both the application itself and all its clients.

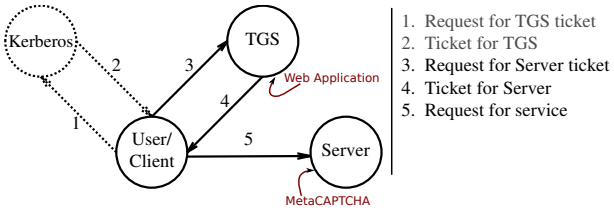


Fig. 2. Kerberos authentication overview and how it relates to MetaCAPTCHA authentication. Figure adapted from Steiner et al. [21]

As implied by the system model in Section III, a client is not given access to the services provided by the web application until it shows proof of a correctly solved puzzle. The only way to be issued a puzzle is to first show that the client is an authorized user of a registered web application. A client does so by presenting to MetaCAPTCHA a "server-ticket" issued by the web application. The authentication protocol used is modeled around Kerberos [21], wherein the web application acts as the Ticket-Granting-Server (TGS) for the MetaCAPTCHA service as shown in Figure 2 [cite steiner]. Notice that steps 1 and 2 of the Kerberos protocol — where a client authenticates itself to Kerberos — are not required because MetaCAPTCHA assumes that it will be replaced by the web application's existing authentication mechanism (e.g password).

After a client submits a message, the web application returns a server-ticket $S_1 = C||ID||\mathbf{HMAC}(K, C||ID)$ containing client-specific information C, a web application ID issued by MetaCAPTCHA during the registration phase, and a Hash-based Message Authentication Code (HMAC) for C created using the web application's secret key K (See Figure 3). The server-ticket S_1 is called the *puzzle-request ticket* and is sent by clients to MetaCAPTCHA for requesting puzzles.

When MetaCAPTCHA receives the puzzle-request-ticket S_1, it verifies that the client is indeed a user of a registered web application. MetaCAPTCHA performs this verification by checking the integrity of the HMAC included in the ticket. Notice that the correct HMAC can only be generated by a registered web application because it includes that application's unique API key.

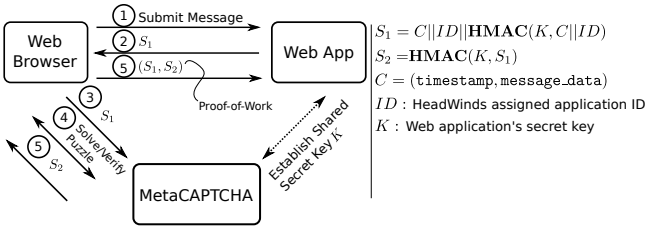


Fig. 3. MetaCAPTCHA authentication and puzzle solution verification

Once the integrity of the HMAC is ascertained by MetaCAPTCHA, the client is issued a puzzle to solve. Details of client-specific information C are presented in Section IV-B.

B. Puzzle Delivery and Verification

MetaCAPTCHA only issues puzzles to authenticated clients as previously shown. The hardness of the issued puzzle depends on the client-specific information $C = (\texttt{timestamp}, \texttt{message\_data})$ sent by the client to MetaCAPTCHA during the authentication phase. Here, `timestamp` indicates when the message was created (this assumes the web application and MetaCAPTCHA are loosely time-synchronized); `message_data` contains the message text submitted by the client and any other information related to it. MetaCAPTCHA uses the information in C to compute a *reputation score*, which in turn is used to determine the puzzle *difficulty level*: the amount of time a user's browser must compute to provide sufficient proof-of-work to the web application. Higher the reputation score, more malicious the client, and longer their browser spends solving puzzles. Its important to note here that MetaCAPTCHA may issue *multiple* puzzles to ensure that the browser stays busy for the determined amount of time (i.e. the difficulty level)

Once the user's browser has solved all puzzles, it must send back the final solution to MetaCAPTCHA. If the solutions are correct, MetaCAPTCHA will issue the client a *proof-of-work-ticket* $S_2 = T_s||T_e||HMAC(K, T_s||T_e||S_1)$, where T_s and T_e are the start and end time stamps of the puzzle solving session. The client must present this ticket to the web application (see Figure 3), which will then verify its integrity before allowing the client to complete posting the message. Additionally, if the difference between the current time and T_e is greater than some threshold t_{diff}, the client's proof-of-work ticket is rejected.

C. Calculating Reputation

The *reputation score* is the probability that a given message is spam as determined by a Naive Bayes classifier. A client's reputation score is calculated each time she posts a message to the web application and is dependent on the features of the message and the client that sent it. A *feature* is any metric with a finite set of values. For example, blacklist status of the message's source IP address, SpamAssasin score of the message, or number of times the poster was "thanked". Given such message features and any other client-related features provided by the web application, MetaCAPTCHA's *reputation*

service can generate the client's reputation score. For the list of features used in this work, refer to Figure 4.

The reputation service is initialized by training the classifier using ground-truth feature values for messages that have already been posted. Training information about each message must include the values for each feature and its classification as spam or ham (not spam). The information about all messages is then fed to the classifier, which builds a probability model to determine how likely a given new message is spam. This likelihood or probability is called the reputation score and its value ranges from 0 to 1 with higher scores implying more malicious users.

D. Reputation Score to Puzzle Difficulty

Puzzle difficulty is the amount of time a client must be kept busy solving puzzles. MetaCAPTCHA's approach is to first, determine the puzzle difficulty based on the reputation score, and then, continuously issue puzzles until the client has computed for the amount of time indicated by the puzzle difficulty. The advantage of this approach is that it gets rid of an adversary's incentive to solve puzzles quicker (e.g by offloading, or parallelizing the computation). The formula used for determining puzzle difficulty is inspired by Laurie and Clayton's work on proof-of-work systems [10] and can be stated as:

$$t = (t_{max} + 1)^r - 1$$

Here, t is the puzzle difficulty we want to calculate, r is a reputation score between 0 and 1, and t_{max} is:

$$t_{max} = \frac{t_p}{s_p(1-\delta)}$$

where, δ is the reduction in spam the web application is seeking (e.g. 10%) from an average s_p of spam messages received in time period t_p. Notice that difficulty t increases exponentially with reputation r.

E. Issuing Puzzles

Once the puzzle difficulty t is determined, the puzzle service randomly generates a puzzle based on the list that is configured. The puzzle is then issued to the client who must solve it and return a solution. If the solution is returned in time $t' < t$, then a new puzzle is chosen and issued. This process is repeated until the client has computed for *at least* t amount of time. The idea behind issuing several puzzles is to ensure that no user can complete an online transaction unless they have computed for a length of time $\geq t$.

1) Puzzle Types: Essentially, a *puzzle type* is a parameterized function. A puzzle-type with an instantiated set of parameters is called a *puzzle*. Puzzles that require human interaction to solve (e.g. CAPTCHA) are called *interactive* puzzles, while those that don't (e.g. proof-of-work), are called *non-interactive* puzzles. MetaCAPTCHA additionally supports *hybrid* puzzles that have both an interactive and a proof-of-work component. The following puzzles are currently supported, however, this set can be dynamically updated:

- Targeted Hash-Reversal (non-interactive): forces a client to compute d hashes before finding the right answer [22].
- Modified Time-Lock (non-interactive): forces the client to compute in a tight loop for an amount of time that can be precisely controlled [11].
- CAPTCHA (interactive): forces the client to solve a traditional CAPTCHA [17], [16]
- CAPTCHA+ (hybrid): traditional CAPTCHA with a modified time-lock puzzle in the background

V. Results

We now evaluate MetaCAPTCHA and show that its defense-in-depth approach improves spammer identification, that this identification is accurate, and that it is an efficient spam prevention service.

A. Experimental Setup

The experimental setup used in our evaluation includes a MetaCAPTCHA server with a 2.4 GHz Intel Xeon quad-core processor running Red Hat Linux on a 2.6.18 kernel. A live discussion forum active from Sep 1 to Oct 19^{th} 2012 employed MetaCAPTCHA as its spam prevention service. MetaCAPTCHA's effectiveness and performance has been evaluated in the context of this forum. At the time, the forum had 2282 messages from 485 users in 112 sub-forums containing 997 conversation threads. Upon registration, the forum provided most of this historical user and message data to help train MetaCAPTCHA's Naive Bayes classifier in identifying spam. Since the provided data was considered ground-truth, a part of it was used to train the classifier and the rest to evaluate it. The classification (spam or ham) was then compared with ground-truth to judge the classifier's effectiveness. The data consisted of values for all features shown in Figure 4 for each of 1442 messages posted to the forum. We now describe the experiments used to evaluate MetaCAPTCHA.

B. Defense-in-Depth

Defense-in-depth implies the use of multiple features to determine user reputation as opposed to only one or a few. Recall that a user's reputation score is the probability that the message she is posting is spam. This probability is determined by the Naive Bayes classifier. If the probability that a message is spam is higher than the probability that it is not, the classifier tags the message as spam. Therefore, the better the classifier is at identifying spam, the better it is at identifying spammers and assigning appropriate reputation scores.

We evaluated the spam identification accuracy of the classifier by using standard machine learning techniques. The idea was to measure the classifier's *precision* and *recall*; *precision* is the fraction of messages that are actually spam (or ham) among those classified as spam (or ham); *recall* is the fraction of actual spam (or ham) that gets classified correctly. A commonly used combined metric is the harmonic mean of precision and recall, called the *F-measure*. Higher the F-measure, better the classifier is at identifying spam.

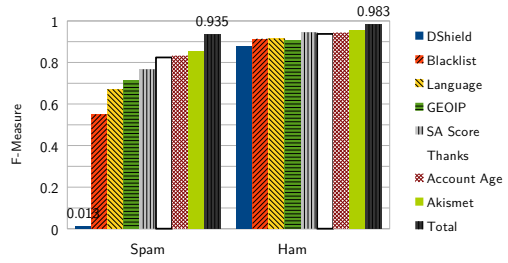
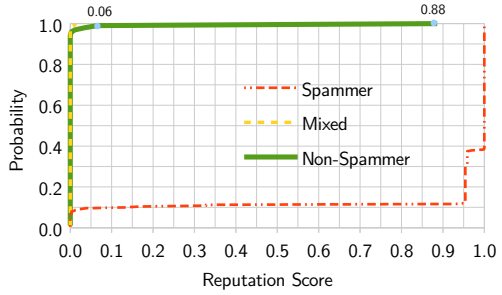
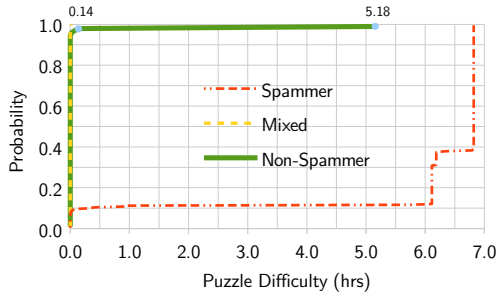


Fig. 4. Defense-in-depth: using multiple features for spam classification is better than using one or a few. "Total" implies that all-of-the above features were used for training the classifier.

We used 10-fold cross-validation to train and test the classifier on feature data for 1442 messages. During each train-and-test run we limited the set of features that the classifier could use. More specifically, in all but the last run, the classifier was trained on one distinct feature. However, in the last run, it was trained on all features together. The F-measure was then computed and plotted for each of the runs. We can see in Figure 4 that the classifier's F-measure is largest when using all features together than when using any single one.



(a) Reputation score accuracy



(b) Puzzle difficulty accuracy

Fig. 5. Reputation Accuracy: CDF of reputation scores and puzzle difficulties assigned to spammers, non-spammers, and mixed users (those that sent at least 1 spam and 1 ham)

C. Reputation Accuracy

We evaluated the accuracy with which MetaCAPTCHA's reputation service distinguished between spammers and honest users. To do this, we first divided forum users into one of three categories, (i) *spammers*: those how sent only spam, (ii) *non-spammers*: those who sent no spam, and (ii) *mixed*: those who sent both spam and ham. Here, 'users' implies the senders of messages included in ground-truth information provided by the forum. After the categorization, there were 99 messages sent by non-spammers, 240 messages sent by spammers, and 151 messages sent by mixed users in the test set (34% of ground-truth data picked uniformly at random). We then fed these messages to MetaCAPTCHA's classifier and extracted the reputation scores from the output (note that reputation scores range from 0 to 1 and higher scores imply more malicious users). Finally, we plotted a CDF of reputation scores for each category of users.

Figure 5(a) shows that $\approx 90\%$ of *spammers* have reputation scores over 0.95, whereas $\approx 99\%$ of non-spammers got a reputation of 0.065 or less. Among the honest users, only one suffered the ill fate of being assigned a reputation of 0.88, whereas 94% were assigned a reputation of zero — implying that they did not solve a puzzle at all!

Interestingly, mixed users were treated largely as non-spammers. To understand why, we further analyzed messages sent by these users and realized that a majority of the users had posted vastly more ham than spam (see Figure 6). Thus, justifying their lower reputation scores.

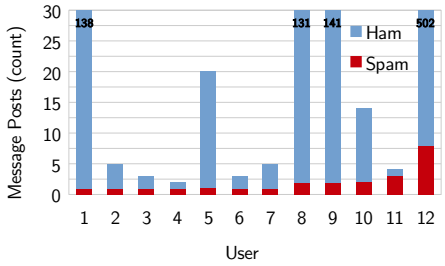


Fig. 6. Distribution of spam and ham sent by mixed users. Mixed users sent very little spam (between 1 and 8) when compared to the total messages they posted. Note: columns that exceeded the y-scale have explicitly marked y-values

Although reputation scores have accurately identified spammers from non-spammers, MetaCAPTCHA's success depends on issuing harder puzzles to more malicious users. This requires evaluating the function that converts reputation score to puzzle difficulty (see Section IV-D). We first computed the maximum puzzle difficulty $t_{max} = 6.82$ hrs based on time period $t_p = 1$ month, number of spam messages s_p seen in that month, and a spam reduction factor $\delta = 0.6$. We then plotted a CDF, shown in Figure 5(b), of the difficulty of puzzles issued to spammers, non-spammers, and mixed users for each message they sent. We can see that in this scenario, $\approx 90\%$ of spammers solved a puzzle over 6 hrs long, $\approx 5\%$ of non-spammers solved a puzzle between 7.2 secs and 8.4 minutes long, and $\approx 95\%$ of non-spammers solved *no puzzles at all*.

VI. Related Work

This section discusses relevant *spam prevention* schemes and how they relate to MetaCAPTCHA: the two prevalent ones are CAPTCHAs and proof-of-work [5].

CAPTCHAs come in many shapes and forms: textual CAPTCHAs require users to identify distorted letters [17], [16], visual CAPTCHAs require users to identify the content or characteristics of an image (e.g. orientation [23]), and audio CAPTCHAs usually require users to identify words in a noisy environment [24]. However, CAPTCHAs are not always fun to solve, so systems like Mollom [25] selectively issue them to only those users that appear to be posting spam. MetaCAPTCHA can incorporate the above CAPTCHAs with the added benefit of a difficulty setting.

Proof-of-work systems that discourage spam include Hashcash, a system that requires senders to attach "postage" to e-mail [12]. The postage is a partial hash collision on a string derived from the recipient's email address. Another proof-of-work solution for throttling email spam was presented by Zhong et al. [13]. However, unlike Hashcash, their system based puzzle difficulty on the "spamminess" of the message. Feng et al. proposed kaPoW [11], a reputation-based proof-of-work system to discourage spam in webmail. MetaCAPTCHA incorporates the features of above proof-of-work systems while augmenting them with a generic puzzle issuing mechanism and a comprehensive reputation service. Furthermore, it can be easily configured and used by generic web applications.

VII. Conclusion

We presented MetaCAPTCHA, an application-agnostic spam prevention service for the web. MetaCAPTCHA seamlessly integrates the CAPTCHA and proof-of-work approaches allowing web application to issue "harder" CAPTCHAs to malicious users. Web applications can configure the puzzles MetaCAPTCHA issues; a configurable library of puzzles also ensures that weaknesses in one class of puzzles won't compromise MetaCAPTCHA as a whole. We evaluated MetaCAPTCHA in the context of a reference web application and showed that 95% of honest users hardly notice MetaCAPTCHA's presence, whereas the remaining 5% were required to solve very "easy" puzzles before accessing the application's services.

References

[1] Geoffrey A. Fowler, Shayndi Raice, Amir Efrati, "Facebook, Twitter battle 'social' spam," http://www.theaustralian.com.au/business/wall-street-journal/facebook-twitter-battle-social-spam/story-fnay3ubk-1226237108998, Jan 2012.

[2] Mark Risher, "Social Spam and Abuse — Annual Trend Review," http://blog.impermium.com/2012/01/13/social-spam-and-abuse-the-year-in-review/, Jan 2012.

[3] C. Grier, K. Thomas, V. Paxson, and M. Zhang, "@spam: the underground on 140 characters or less," in *Proceedings of the 17th ACM conference on Computer and communications security*, ser. CCS '10. New York, NY, USA: ACM, 2010, pp. 27–37. [Online]. Available: http://doi.acm.org/10.1145/1866307.1866311

[4] SPAM LAWS, "Spam Statistics and Facts," http://www.spamlaws.com/spam-stats.html, 2011.

[5] P. Heymann, G. Koutrika, and H. Garcia-Molina, "Fighting spam on social web sites: A survey of approaches and future challenges," *Internet Computing, IEEE*, vol. 11, no. 6, pp. 36–45, 2007.

[6] O. R. Team, "List of weaknesses," http://ocr-research.org.ua/list.html.

[7] M. Motoyama, K. Levchenko, C. Kanich, D. McCoy, G. M. Voelker, and S. Savage, "Re: Captchas–understanding captcha-solving services in an economic context," in *USENIX Security Symposium*, vol. 10, 2010.

[8] M. Motoyama, D. McCoy, K. Levchenko, S. Savage, and G. M. Voelker, "Dirty jobs: The role of freelance labor in web service abuse," in *Proceedings of the 20th USENIX conference on Security*. USENIX Association, 2011, pp. 14–14.

[9] J. Yan and A. El Ahmad, "Usability of captchas or usability issues in captcha design," in *Proceedings of the 4th symposium on Usable privacy and security*. ACM, 2008, pp. 44–52.

[10] B. Laurie and R. Clayton, "Proof-of-work proves not to work," in *The Third Annual Workshop on Economics and Information Security*, 2004.

[11] W. Feng and E. Kaiser, "kapow webmail: Effective disincentives against spam," *Proc. of 7th CEAS*, 2010.

[12] A. Back *et al.*, "Hashcash-a denial of service counter-measure," *URL: http://www. hashcash. org/papers/hashcash. pdf*, 2002.

[13] Z. Zhong, K. Huang, and K. Li, "Throttling outgoing spam for webmail services," in *Conference on Email and Anti-Spam*, 2005.

[14] E. Kaiser and W. Feng, "Helping ticketmaster: Changing the economics of ticket robots with geographic proof-of-work," in *INFOCOM IEEE Conference on Computer Communications Workshops, 2010*. IEEE, 2010, pp. 1–6.

[15] Google, "recaptcha: Stop spam, read books," http://www.google.com/recaptcha.

[16] D. Phillips, "Securimage php captcha — free captcha script," http://www.phpcaptcha.org/.

[17] L. Von Ahn, B. Maurer, C. McMillen, D. Abraham, and M. Blum, "recaptcha: Human-based character recognition via web security measures," *Science*, vol. 321, no. 5895, p. 1465, 2008.

[18] C. Dwork and M. Naor, "Pricing via processing or combatting junk mail," in *Advances in CryptologyCRYPTO92*. Springer, 1993, pp. 139–147.

[19] A. Back, "Hashcash faq," http://www.hashcash.org/faq/.

[20] D. Liu and L. Camp, "Proof of work can work," in *Fifth Workshop on the Economics of Information Security*, 2006.

[21] J. G. Steiner, C. Neuman, and J. I. Schiller, "Kerberos: An authentication service for open network systems," in *USENIX conference proceedings*, vol. 191, 1988, p. 202.

[22] W.-c. Feng and E. Kaiser, "The case for public work," in *IEEE Global Internet Symposium, 2007*. IEEE, 2007, pp. 43–48.

[23] R. Gossweiler, M. Kamvar, and S. Baluja, "What's up captcha?: a captcha based on image orientation," in *Proceedings of the 18th international conference on World wide web*, ser. WWW '09. New York, NY, USA: ACM, 2009, pp. 841–850. [Online]. Available: http://doi.acm.org/10.1145/1526709.1526822

[24] Y. Soupionis and D. Gritzalis, "Audio captcha: Existing solutions assessment and a new implementation for voip telephony," *Computers & Security*, vol. 29, no. 5, pp. 603–618, 2010.

[25] Mollom, "How mollom works — mollom," http://mollom.com/how-mollom-works.