kaPoW Plugins: Protecting Web Applications Using Reputation-based Proof-of-Work

Tien Le
Portland State University
letien@cs.pdx.edu

Akshay Dua
Portland State University
akshay@cs.pdx.edu

Wu-chang Feng
Portland State University
wuchang@cs.pdx.edu

ABSTRACT

Comment spam is a fact of life if you have a blog or forum. Tools like Akismet and CAPTCHA help prevent spam in applications like WordPress or phpBB. However, they are not devoid of shortcomings. CAPTCHAs are getting easier to solve by automated adversaries like bots and pose usability issues. Akismet strives to detect spam, but can't do much to reduce it. This paper presents the kaPoW plugin and reputation service that can complement existing anti-spam tools. kaPoW creates disincentives for sending spam by slowing down spammers. It uses a web-based proof-of-work approach wherein a client is given a computational puzzle to solve before accessing a service (e.g. comment posting). The idea is to set puzzle difficulties based on a client's reputation, thereby, issuing "harder" puzzles to spammers. The more time spammers solve puzzles, the less time they have to send spam. Unlike CAPTCHAs, kaPoW requires no additional user interaction since all the puzzles are issued and solved in software. kaPoW can be used by any web application that supports an extension framework (e.g. plugins) and is concerned about spam.

Categories and Subject Descriptors

H.4.3 [**Communications Applications**]: Security

1. INTRODUCTION

Internet spam is a problem that refuses to go away. Although the amount of 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. In spite of Facebook's best efforts, approximately 4 million users receive spam from around 600,000 new or hijacked accounts each day [8, 14]. What's worse are the response rates from social spam: in Jan 2010, an estimated 0.13% of all twitter spam — almost two orders of magnitude higher than email spam [12] — was clicked by around 1.6 million unsuspecting users [9]. All this spam costs businesses $20.5 billion annually in decreased productivity and technical expenses, and is projected to rise to $198 billion in the next four years [19].

The most common approaches to combatting spam involve using spam filters, heuristics or user reports for identifying accounts that send spam [24], and CAPTCHAS that prevent automatic creation of malicious accounts [25]. Spam filters are effective at "hiding" spam, but don't do much to reduce it. Heuristics can be easy to circumvent [5], while investigating reported accounts can be expensive and most often too late [9]. CAPTCHAs are getting increasingly easy and cost-effective to solve due to OCR techniques, tricking infected victims into manually solving captchas, reusing session IDs of known CAPTCHAs, and farming out CAPTCHA solving to cheap human laborers [5, 7, 11, 10]. Moreover, CAPTCHAs can't prevent spam from being sent from hijacked accounts, or be used too frequently (e.g before sending each email) due to usability limitations [26].

This paper presents kaPoW plugins for web applications like Wordpress, phpBB etc. kaPoW is a Proof-of-Work based approach that integrates with several widely used web applications and can complement existing anti-spam techniques (e.g. CAPTCHAs, spam filters). The plugins use an efficient construction [7] of the cryptographic time-lock algorithm [18] to construct puzzles with client-specific difficulties. Difficulties are determined using the kaPoW reputation service. Higher the difficulty, the longer a puzzle takes to solve. The main contributions of this work are:

- A plugin-based architecture that allows generic web applications to use kaPoW
- A fine-grained (per message) approach to reducing spam without compromising usability
- An implementation of the kaPoW Plugins for Wordpress [2] and phpBB [1], as well as the kaPoW reputation service.

2. BACKGROUND

We now present a brief background on client puzzles and the kaPoW project.

2.1 Client Puzzles

Proof-of-work systems generally consist of three distinct parts: the issuer, the solver, and the verifier. The issuer issues the puzzle to the solver, which solves them and sends the solution to the verifier. kaPoW plugins use the modified time-lock puzzle algorithm introduced by Feng and Kaiser [7]. The puzzle is based on repeated squaring, a sequential process that forces the client to compute in a tight loop for an amount of time that is precisely controlled by the issuer.

At first, the issuer generates p and q, two large prime numbers and calculates the modulus $n = p \times q$. Then, it

WebQuality '12, April 16, 2012, Lyon, France

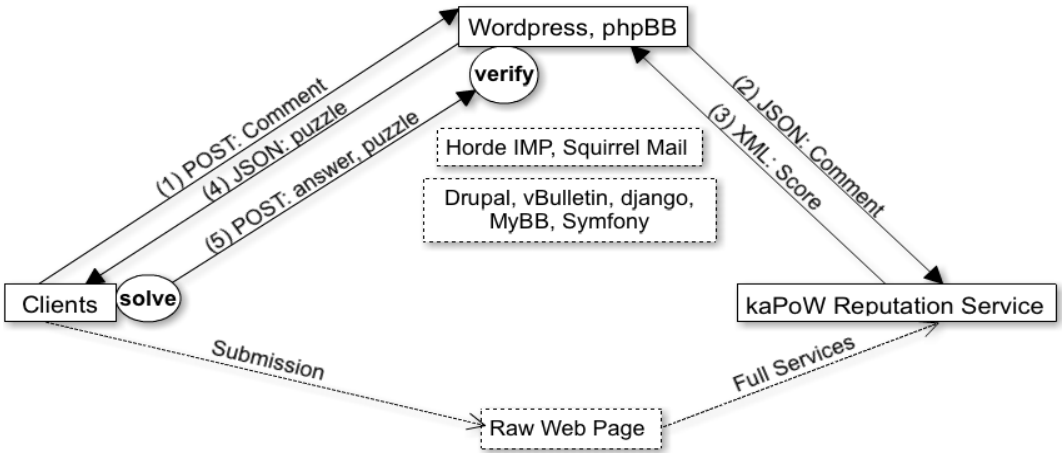

Figure 1: kaPoW model

generates a difficulty t that determines the amount of work a client potentially performs. Lastly, it efficiently generates a nonce a [7] and sends (a, t, n) to the client. The client must then return an answer A such that $A = a^{2^t} mod n$. The verifier can check that A is correct by performing a short-cut computation $\phi = (p-1) \times (q-1)$, $r = 2^t mod \phi$, and $A' = a^r \ mod \ n$. If A matches A', then the client has performed the computation accurately.

The modified time-lock puzzle was designed to be quick to generate and verify, be non-parallelizable and have deterministic run-times, and be configurable with fine-grained puzzle difficulties. The aforementioned characteristics were needed for the system to scale, to deterministically verify puzzle solutions, and appropriately match the amount of work a client performs to the level of protection the application needs.

2.2 kaPoW

The first prototype of the kaPoW project was *kaPoW Webmail* [7]. It introduced a more-efficient construction of the time-lock algorithm that is used in our system as well (Section 2.1). In addition, kaPoW Webmail used a comprehensive set of metrics for determining puzzle difficulties that provided significant disincentives for spammers. The system was designed to be deployed without modifications to either the client or server software.

3. ARCHITECTURE

kaPoW plugins attempt to address some of kaPoW Webmail's [7] drawbacks. Since kaPoW webmail is a stand-alone system, it cannot work with other web applications. Also, its own local reputation service cannot leverage adversary behaviors across multiple applications. kaPoW Plugins employ the same time-lock puzzle, but are diversely applicable and use a centralized reputation service for maintaining cross-application reputations.

As shown in Figure 1, each web application must install a corresponding plugin to use the kaPoW service. Like any proof-of-work system, the kaPoW plugin consists of an issuer, solver, and verifier. The puzzle *issuer* and *verifier* make up the server-side component of the kaPoW plugin. When a client submits a message (e.g. form, comment) to the web application, the issuer forwards that content to the centralized reputation service. The returned reputation score is then combined with another local score to compute the final puzzle difficulty (Section 4). The issuer then constructs the puzzle and sends it to the client-side component or *solver*. The solver is responsible for receiving, solving, and returning the puzzle to the verifier. If verification succeeds, the client's content is handled by the web application, otherwise it is placed in a spam queue.

The central reputation service combines intelligence on spam attacks across multiple kaPoW plugins installed on a diverse set of web applications. For example, a spammer may target multiple kaPoW-enabled web applications in a short period of time. The individual applications may not see this as an attack, but the centralized reputation service with its global view may detect the attack. It can then blacklist the spammer's IP address and flag the content as spam.

4. PUZZLE DIFFICULTY

No proof-of-work system can be effective unless it can issue puzzles of different difficulties [13]. Difficulties are set based on reputation scores from, 1) kaPoW's reputation service, and 2) a per-application *local policy*. Local scores are necessary for incorporating reputation information from application-specific characteristics. For example, age of user accounts: a web application may consider a long-time client more trustworthy than a new one. More specifically, kaPoW supports local reputation scores based on age of account, locally defined spam words in a client's message, and blacklisted IP addresses or usernames. A client's reputation score is computed as

$$score = S_1 + S_2 + \ldots + S_m \quad (1)$$

where S_i is a binary metric such as *"does the IP address or username of the client appear on any blacklists?"*.

Once a reputation score is generated, it is translated to a puzzle difficulty using the score-to-difficulty equation from kaPoW webmail [7]. Namely, the puzzle difficulty

$$t = \alpha \times score^m \quad (2)$$

where m is the number of reputation metrics, and α is a "computational power" factor. α should be set smaller for computationally limited devices like smart phones and larger

for more capable ones like PCs. Currently, the system uses an empirically determined $\alpha = 20$ for all devices. We plan to address the selection of α in the future. The metrics are described in more detail in Section 5.2. Unlike the thumbs-up/thumbs-down approach of spam detection and CAPTCHs, the proof-of-work model considers all clients to be adversaries, but with varying degrees of maliciousness. Later, based on a client's current and past behavior, a puzzle of appropriate difficulty is issued.

5. IMPLEMENTATION

This section describes the implementation of the kaPoW Plugin, and reputation service [1].

5.1 kaPoW Plugin

The kaPoW plugin is implemented in PHP and delivers a JavaScript solver to the client for solving the modified time-lock puzzle. The solver is only 9KB and is called via AJAX. AJAX allows puzzle solving to occur in the background. This minimizes changes to the original web application user interface. Figure 2 shows, how the puzzle is received and computed in the background after the user clicks the "Post Comment" button. In the unlikely event that JavaScript is not enabled on the client's browser, the submitted content is placed into the web application moderator's queue for further investigation.

Figure 2: The kaPoW plugin integrated into the Wordpress message posting interface

We have developed kaPoW plugins for Wordpress and phpBB. The Wordpress plugin currently integrates with the "New Comment" and "Account Register" pages. As mentioned before, the plugin can complement other approaches like CAPTCHA. The kaPoW phpBB plugin is implemented as an AutoMOD [16] and works on bulletin boards with jQuery support. The plugin can then integrate client puzzles into all message posting and editing pages.

[1]Available at `http://kapow.cs.pdx.edu/kapow/`

5.2 kaPoW Reputation Service

The kaPoW reputation service runs on an Intel Xeon E5620 with 24GB RAM running RedHat Enterprise Linux. Its main job is to generate scores based on client information and content received from web applications. It also offers web-based, real-time and historical reporting on identified spam attacks. The client information and content received from the application is first analyzed by several popular anti-spam services (SpamAssassin [20], kaPoW[7], Spamhaus[22], SpamCop[21], Project HoneyPot[17], Dshield[6], Akismet[4], StopForumSpam[23]). Then, the kaPoW reputation service will check the URLs within the message, the username (if one exists), and the client's IP address against multiple blacklists. Each time a check returns true, the reputation score is increased by 1. The maximum reputation score is 7. The final score is then sent back to the web applications in XML format. The scoring algorithm uses the following binary metrics to generate the spam score:

- *Spam Content:* do anti-spam tools consider the message spam?
- *Spam Words:* does the message contain any words present in the local spam-word list?
- *Blacklists:* does the IP address or username of the client appear on any blacklists?
- *Location:* is the geographic location of the IP address of the client outside 500 miles of their usual region of access? Unfortunately, this has the adverse effect of lowering the reputation of legitimate travelers. However, we hope to address this issue in the near future.
- *Time:* does the current time of day fall outside an 8-hour window during the day that users typically send messages?
- *Usage:* has the user account sent a message within the last 5 minutes?
- *Account age:* is it a new account? Spammers generally create multiple new accounts to evade IP reputation systems [7]. These accounts are usually utilized for a short period of time before being abandoned. New accounts are penalized up to a threshold number of message postings (currently set to 5) before being considered "old".

Some of the metrics above are determined locally, where as others are determined remotely at the reputation service. The respective metrics are combined to generate a *local* and *remote* score. The local ($score_{local}$) and remote score ($score_{rep}$) are then used to generate the final reputation score:

$$score = score_{local} + score_{rep} \quad (3)$$

The final score is used to set the puzzle difficulty as described by Equation 2.

6. DISCUSSION

Bots that do not execute JavaScript will be unable to post spam because they will fail to execute the issued puzzle. Bots that are capable of executing JavaScript (e.g.

Googlebot) [15] will be issued harder puzzles than "normal" users, and that is exactly the intention of the kaPoW system. The computational effort spent solving puzzles will slow the rate at which the bots can send spam. Similarly, browsers used by hired "spam" laborers are forced to solve time-lock puzzles, *even if* they can overcome CAPTCHAs easily. Thus, kaPoW can nicely complement a CAPTCHA-based approach. kaPoW also does not undermine existing anti-spam tools like Akismet or SpamAssassin, since it uses those services for computing client reputations (see Section 5.2). We believe that the combination of existing anti-spam tools and kaPoW is an effective defense.

In the current implementation, the kaPoW plugin does not issue puzzles based on the capabilities of the client platform. Thus, cell phones would unfairly be issued puzzles meant for PC-class machines. It has been shown that memory-bound puzzles [3] are more fair than CPU-bound ones (e.g. time-lock puzzle). However, we leave this investigation for future work. Additionally, we plan to rigorously evaluate the kaPoW Plugin system and improve the cross-application reputation scores.

7. CONCLUSION

Current anti-spam efforts have several limitations: Akismet and other spam filters can determine if comments are spam or not but cannot reduce spam. Additionally, filters can have false positives and negatives. CAPTCHAs are getting easier to solve by automated adversaries (e.g. bots) and pose several usability issues. The KaPoW plugin and reputation service addresses these limitations using an efficient construction of time-lock puzzles and a comprehensive set of metrics to drive puzzle difficulties. kaPoW is an open-source project under the GPLv2 License. The source code plugins for Wordpress and phpBB are available for download at the kaPoW main website.

This material is based upon work supported by the National Science Foundation under Grant No. CNS-1017034. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

8. REFERENCES

[1] phpBB: Creating Communites Worldwide. `http://www.phpbb.org`.

[2] WordPress: Blog Tool and Publishing Platform. `http://wordpress.org`.

[3] M. Abadi, M. Burrows, M. Manasse, and T. Wobber. Moderately Hard, Memory-bound Functions. In *NDSS*, February 2003.

[4] Akismet. Comment spam prevention for your blog. `http://akismet.com`.

[5] Y. Boshmaf, I. Muslukhov, K. Beznosov, and M. Ripeanu. The socialbot network: When bots socialize for fame and money. Dec 2011.

[6] DShield.org. Distributed Intrusion Detection System. `http://www.dshield.org`.

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

[8] 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.

[9] 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*, CCS '10, pages 27–37, New York, NY, USA, 2010. ACM.

[10] E. Kaiser and W. Feng. mod kapow: Protecting the web with transparent proof-of-work. In *INFOCOM Workshops 2008, IEEE*, pages 1–6. IEEE, 2008.

[11] 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*, pages 1–6. IEEE, 2010.

[12] C. Kanich, C. Kreibich, K. Levchenko, B. Enright, G. Voelker, V. Paxson, and S. Savage. Spamalytics: An empirical analysis of spam marketing conversion. In *Proceedings of the 15th ACM conference on Computer and communications security*, pages 3–14. ACM, 2008.

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

[14] 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.

[15] P. McKenzie. Detecting Bots with Javascript. `http://www.kalzumeus.com/2010/06/07/detecting-bots-in-javascrip/`.

[16] phpBB.com. phpBB AutoMOD. `http://www.phpbb.com/mods/automod/`.

[17] Project Honey Pot. Http:BL. `http://www.projecthoneypot.org/httpbl.php`.

[18] R. Rivest, A. Shamir, and D. Wagner. Time-lock puzzles and timed-release Crypto. Technical report, MIT, March 1996. MIT/LCS/TR-684.

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

[20] SpamAssassin. The spam filtering service. `http://spamassassin.apache.org`.

[21] spamcop.net. SpamCop. `http://www.spamcop.net/`.

[22] Spamhaus Project Ltd. Spamhaus Project. `http://spamhaus.org/`.

[23] Stop Forum Spam. A database of known forum and blog spam. `http://stopforumspam.com`.

[24] Twitter. The Twitter Rules. `http://support.twitter.com/articles/18311-the-twitter-rules`, Jan 2012.

[25] L. Von Ahn, B. Maurer, C. McMillen, D. Abraham, and M. Blum. recaptcha: Human-based character recognition via web security measures. *Science*, 321(5895):1465, 2008.

[26] 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*, pages 44–52. ACM, 2008.