kaPoW Webmail: Effective Disincentives Against Spam

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

Ed Kaiser
Portland State University
edkaiser@cs.pdx.edu

ABSTRACT

Webmail spam poses a significant threat to major webmail providers such as Google GMail, Yahoo! Mail, and Microsoft Live Mail, as well as to individual companies and universities that provide web-based interfaces to their email. Whether spammers create new accounts or hijack existing accounts to send spam, the transmission of spam drives up server operating costs as well as the human costs required to identify and disable spamming accounts.

This paper presents kaPoW Webmail, a system for slowing down and disincentivizing webmail spammers using transparent, web-based proof-of-work (also known as client puzzles). The approach requires that clients must solve a computational puzzle for each email sent. The system employs a novel puzzle algorithm that efficiently generates and verifies fine-grained computational puzzles that have deterministic solution-times. Unlike prior proof-of-work systems, kaPoW Webmail also adaptively issues puzzles of varying difficulties based on a comprehensive set of client-specific and content-specific measurements. The evaluation shows that this system thwarts spammers while preserving service to legitimate webmail clients.

Keywords: Webmail, Spam, Proof-of-Work

1. INTRODUCTION

As sender reputation systems for email transmission continue to improve, spammers are increasingly turning towards web-based email to continue their operations. As a result, in 2008, the amount of webmail spam approached close to 5% of all spam sent [18]. To execute attacks using webmail services, spammers attempt to automate the creation of new accounts at free webmail sites such as Google GMail, Yahoo! Mail, and Microsoft's Live Mail or they perform reputation hijacking by obtaining the login credentials for existing legitimate webmail accounts via methods such as spear phishing [6].

CEAS 2010 - Seventh Annual Collaboration, Electronic messaging, Anti-abuse and Spam Conference, July 13-14, 2010, Redmond, Washington, US.
.

Aside from disabling webmail accounts that send spam, most webmail services combat spam transmission through the use of CAPTCHAs [41]: automated Turing tests that typically consist of skewed representations of letters and numbers. A user must correctly interpret the characters before being granted service. Unfortunately, there are several problems with using CAPTCHAs. The tests create a terrible user interface experience especially to users that are visually impaired [22]. Another problem is the cost to employ humans to solve them is fixed and small [14], making them unsuitable for protecting valuable resources [23, 37]. Finally, increasingly more sophisticated optical character recognition algorithms are becoming available making it hard to generate CAPTCHAs that are easy for humans yet difficult for computers to solve [7, 17, 29].

The proof-of-work (or "*client puzzles*") approach provides a complementary alternative to CAPTCHAs. The approach forces clients to solve computational puzzles of client-specific difficulty before granting them service, acting as a filter for users based on their willingness to commit their own resources. Proof-of-work does not impose user interface problems and is based on cryptographic primitives that are provably hard to bypass. In addition, the difficulty of the puzzle is adaptable on a per-user or per-request basis. A number of proof-of-work systems have been proposed to protect network protocols [11, 12, 13, 27, 43], transport protocols [8, 19, 42], authentication protocols [2], web protocols [20], and email [3, 10, 44]. Unfortunately, while the promise of proof-of-work has been touted for over a decade, proposed proof-of-work approaches have met resistance to deployment because they suffer from the following shortcomings:

- **Hash-based puzzles**: Most proof-of-work systems are based on puzzles that require a client to reverse a weakened cryptographic hash function. While hash-based puzzles are very efficient to implement, they have several drawbacks. Specifically, such puzzles are easily parallelizable across multiple machines and have probabilistic solution-times that are not predictable. In addition, the difficulty settings on many hash-based puzzles [8, 19, 27, 42, 44] are coarse, making it hard to control the amount of work assigned to a client.

- **Simplistic difficulty setting**: Proof-of-work systems that do not differentiate adversaries from legitimate clients are easily defeated [23]. Most proof-of-work systems set the difficulty using a single metric such as the load on the system [8, 19, 27], the request rate of the client [12, 13, 20], the demand for the service [42, 43], or the content of the request [44]. Without sufficient defense-in-depth, it is unlikely such systems will deter all automated adversaries.

- **Client software modifications**: Most proof-of-work systems require adoption of special client software to receive proof-of-work challenges and solve them on behalf of the client. An exception is [20] which requires no software modifications to the client.

This paper presents kaPoW Webmail, a prototype webmail service that protects itself from sending spam using proof-of-work. In order to address the above limitations, kaPoW Webmail uses a more-efficient construction of the time-lock algorithm [32] to issue non-parallelizable, fine-grained puzzles that have deterministic solution-times. In addition, the system uses a comprehensive set of metrics for determining puzzle difficulties that provide significant disincentives for spammers. Finally, the system is implemented using standard web scripting environments allowing it to be deployed without modifications to either the client or server software.

2. PROOF-OF-WORK

Proof-of-work or client puzzle systems consist of three distinct parts. The *issuer* generates and delivers a puzzle to the client on behalf of the server. The *solver* generates solutions to puzzles received by the client. The *verifier* denies or accepts solutions sent to the server based on their freshness and validity. In the proof-of-work model, all clients are considered adversaries, but of varied maliciousness. Based on their current and past behavior, they are then issued puzzles of appropriate difficulty. The puzzle difficulty is expressed in terms of units of work, which are uniform-length computations such as the execution of a hash function. A proof-of-work scheme alters the operation of a network protocol so that a client must return their puzzle along with a correct answer before being granted service. If the server receives a request without a valid puzzle or an incorrect answer, the request is ignored and a valid puzzle is sent to the client. The puzzle given to the client has a difficulty setting that determines how much computation it must perform before generating an answer. After receiving and solving the puzzle, the client attaches both the puzzle and answer when resending the request. The server, upon receiving the answer, verifies its correctness before allowing the client access.

3. KAPOW

3.1 Modified Time-Lock Puzzle Algorithm

One of the key components of a proof-of-work system is the algorithm that issues and verifies the puzzles. Specifically, this algorithm must meet the following characteristics to be useful:

- **Fast generation and verification**: Issuing the puzzle and verifying the correctness of subsequent answers must add minimal computation and memory overhead to the system in order to prevent the proof-of-work mechanism from becoming a target for attack.

- **Non-parallelizable**: The puzzle must be not be parallelizable, that is, it should not be possible to break up the work into smaller components that can be solved across many machines simultaneously.

- **Deterministic run-time**: The amount of computation a client is required to consume should be predictable and deterministic in order to ensure consistent client execution.

- **Fine granularity**: The puzzle must support difficulties that can be finely controlled in order to match the amount of work a client performs with the level of protection a server might require.

While kaPoW can support several puzzle algorithms, in order to address the above characteristics, its default puzzle algorithm is based on a novel construction of time-lock puzzles [32]. Time-lock puzzles are 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. Time-lock puzzles are an attractive alternative to hash-based puzzles in that they are non-parallelizable and have deterministic run-times. However, the cost of generating time-lock puzzles is prohibitively expensive for use in high-speed network protocols and services [13]. Specifically, the issuer generates p and q, two large prime numbers as well as a difficulty t that determines the amount of work a client must perform. It then calculates the modulus $n = p \times q$, randomly selects a number a, and sends the client a, t, and n. The client must then return an answer A such that $A = a^{2^t} \bmod n$. The server can check that A is correct by performing a short-cut computation $\phi = (p-1) \times (q-1)$, $r = 2^t \bmod \phi$, and $A' = a^r \bmod n$. If A matches A', then the client has performed the computation accurately.

Unfortunately, the generation of the two large prime numbers p and q (akin to those in the generation of an RSA key pair) takes on the order of tens of milliseconds on modern processors making time-lock puzzles unsuitable for high-speed networks and applications. Furthermore, the time-lock algorithm requires the issuer to

PUZZLE TYPE	ISSUING METHOD	VERIFICATION METHOD
Time-Lock [32]	RSA key generation	modular exponentiation
Hash-Reversal [19]	cryptographic hash	cryptographic hash
Hint-Based Hash-Reversal [13]	cryptographic hash	cryptographic hash
Targeted Hash-Reversal [12]	minimal	cryptographic hash
kaPoW Modified Time-Lock	**cryptographic hash**	**modular exponentiation**

Table 1: Puzzle algorithm characteristics for server.

PUZZLE TYPE	NON-PARALLELIZABLE	DETERMINISTIC	WORST-CASE GRANULARITY
Time-Lock [32]	Yes	Yes	1 modular squaring
Hash-Reversal [19]	No	No	2^{n-1} cryptographic hashes
Hint-Based Hash-Reversal [13]	No	No	1 cryptographic hash
Targeted Hash-Reversal [12]	No	No	1 cryptographic hash
kaPoW Modified Time-Lock	**Yes**	**Yes**	**1 modular squaring**

Table 2: Puzzle algorithm characteristics for client.

keep track of the puzzle parameters (a,t,n) issued to each client. To address these problems, kaPoW modifies the time-lock puzzle generation component so that a single pair of prime numbers can be used to generate multiple client puzzles in a consistent fashion. This allows the system to operate with constant state and amortize the cost of generating the prime numbers across many issued puzzles.

kaPoW modifies time-lock puzzles by setting t based on the maliciousness of the client and by modifying the generation of a. Instead of selecting a randomly, kaPoW generates a as a cryptographic hash of client characteristics $f_c()$ and a periodically updated random server nonce K (e.g. $a = SHA1(K||f_c())$) where $f_c()$ can consist of any number of client parameters including the URL being requested, the IP address of the client, and the difficulty of the puzzle given to the client. Rather than incur the overhead of generating large prime numbers for each puzzle, a new puzzle can be issued by performing a single cryptographic hash. In addition, the verifier only needs to keep track of K, p, and q in order to properly validate subsequent puzzle answers from the client since it is able to regenerate t and $f_c()$ from the client's request.

Note that the cryptographic strength of kaPoW's modified time-lock algorithm is configurable to match their use in this context. Because the cryptographic mechanism is *expected* to be broken on the order of several seconds to minutes and because the keys themselves can be easily regenerated during operation, it is possible and desirable to use "weak" cryptographic keys for efficiency. The two main parameters that drive the modified algorithm are the size of the prime numbers used to generate subsequent time-lock puzzles and the frequency in which those keys are regenerated. The size of the prime numbers determines the scheme's resistance to a brute-force attack that seeks to factor n into the prime numbers p and q. Currently, keys below 300 bits can be factored by a single computer on the order of several hours while 512-bit keys can be factored on the order of several months [5]. Key regeneration must be done periodically to address brute force cracking, but also to address the fact that *multiple* puzzles are issued from a single pair of prime numbers. A well-known attack against this revolves around the birthday paradox in which among a set of random people, some pair of them will share identical birthdays with increasing probability as the number of people in the set grows. In the modified time-lock algorithm, the squaring is done with the same modulus across all of the puzzles. Thus, if two puzzles happen to share an intermediate result that is the same in their computation, the computation can be shortcut via the giant-step, baby-step algorithm [33]. Note that the smaller the modulus (i.e. the key size) is, the more quickly the key must be regenerated before enough intermediate results are stored to create effective shortcuts using giant-step, baby-step. As a result, care must be taken to refresh the modulus used to generate the time-lock puzzles before such an attack is feasible.

Table 1 and Table 2 summarize the characteristics of a range of puzzle algorithms in the literature. As Table 1 shows, for puzzle issuing, hash-reversal puzzles have a significant advantage over unmodified time-lock puzzles in their issuing overheads (cryptographic hash versus RSA key generation). However, with the modified time-lock algorithm, this advantage is erased. As Table 2 shows, hash-reversal puzzles have a significant disadvantage compared to time-lock puzzles in that their solution times are probabilistic and their work can be parallelized easily across multiple machines. Thus, kaPoW's modified time-lock algorithm provides the efficiency at the issuer and verifier that hash-reversal puzzles provide while supporting the deterministic run-times and non-parallelizability of time-lock puzzles at the solver.

3.2 Comprehensive Disincentives

Besides a robust puzzle algorithm, another key component to a proof-of-work system is a robust means for configuring the difficulty of the puzzle issued to each client. This difficulty algorithm must be based on a comprehensive set of measures to single out and disincentivize all types of misbehavior from adversaries. Unfortunately, current proof-of-work systems take a simplistic approach for setting the difficulties of the puzzles they issue, making them ineffective.

One policy used by many proof-of-work systems is to have the server issue puzzles with uniform difficulty across all clients whenever it becomes overloaded [2, 8, 19]. Another policy used is a market-based one where clients "bid" on the service by solving computational challenges that are based on how much they value the service [3, 42]. The service then processes requests in a priority order based on the amount of work committed by each client. Unfortunately, policies that treat clients uniformly have been shown to be ineffective [23]. Such systems unfairly penalize legitimate clients while having minimal impact on adversaries that control a significant amount of resources such as a botnet.

More sophisticated proof-of-work systems tailor the difficulties of puzzles to individual clients to incentivize good behavior. For example, in [20], a counting Bloom filter is used to track the usage of individual clients over time. When the server is overloaded, harder puzzles are delivered to clients that have sent a large number of requests to the server recently. In [44], the mail server determines the difficulty of the puzzle based on how "spammy" the message a client is attempting to send appears. Unfortunately, both systems provide disincentives only for specific misbehavior and are vulnerable to alternative attacks. Specifically, the request-based approach does not provide disincentives to an adversary posting web comment spam at a reasonable rate while the content-based approach does not provide disincentives against an adversary attempting to take down the service with a flood of requests. To address the shortcomings of previous approaches, kaPoW provides a comprehensive framework that adaptively delivers puzzles with difficulties that are based on a range of characteristics about the client and the request. These characteristics include:

TIME-BASED COMPONENTS. Spammers typically send messages non-stop throughout the day. Thus, the time elapsed since an account's last message was sent, the time of day the message is sent [31], and the difference in time the message is sent and the typical time of day the account's owner sends messages can be used to indicate anomalous behavior and to issue more difficult puzzles. Another useful time-based component is the time elapsed since the creation of the user's account on a webmail service. Spammers often create thousands of e-mail addresses on free web-based e-mail services using automated bots. There is then a small window of time for spammers to use the account before a complaint is eventually received and the account is terminated. Thus, it has been shown previously that the optimal strategy for spammers to use these accounts is to send as much as possible as quickly as possible [16]. Using puzzles to slow the sending of email on a newly created account forces the spammer to create many more accounts to send the same amount of messages, thus increasing the spammers' cost. Accounts that are older and established are less likely to be sources of spam and can receive progressively easier puzzles compared to newly created accounts.

USAGE-BASED COMPONENTS. Denial-of-service and spam attacks on webmail are volume operations. Thus, using past and current usage of a client to drive puzzle difficulties can help disincentivize misbehavior. Specifically, difficulties can be based on the total number of messages a client has sent in the past, the number of messages a client has sent in the past that have not been classified as spam, the number of messages a client has sent in the past that have been classified as spam, and the total number of recipients the message will be sent to. In addition, as with prior proof-of-work systems, the current load on the webmail system can also be used to drive puzzle difficulties in order to give the server an ability to throttle clients when overloaded.

LOCATION-BASED COMPONENTS. The geographic location of a client obtained via geographic databases [24] can often be used to determine whether or not the source is sending spam or not. For example, some spam is sent with specific geographic patterns [25] while spam sent from accounts that have been spear phished will often originate from machines that have different geographic locations than the victim's typical location. Furthermore, for webmail services that serve local communities such as a university's student population [28], the geographic distance the client is from the server can roughly differentiate legitimate versus adversarial behavior.

REPUTATION-BASED COMPONENTS. One of the reasons spammers have turned to webmail is the widespread use of blocklists on mail servers. Since the IP addresses of many compromised machines are well-known, mail servers can be easily configured to block mail from them. In order to leverage this protection, network services can query a number of distributed IP address blocklists to determine the reputation of a client based on its address [9, 30, 35, 36]. Specifically, the presence of a client machine in any of these databases can be used to substantially increase the difficulty of the puzzle the client must solve before

allowing access to a service.

CONTENT-BASED COMPONENTS. Prior work on stopping webmail spam has applied a Bayesian filter to the content of a message to generate a score that is used to configure the difficulty of a puzzle [44]. While this has been shown to be effective, another content-based check that can be used to identify spam is to see if the URLs embedded in messages are part of spam campaigns. Spam campaigns often point users to the same ephemeral web sites. As a result, several distributed blocklists have been developed to collect such URLs in a database that can be queried in real-time [38, 40]. By querying such sources and automatically increasing the difficulty of puzzles given to clients attempting to send messages with such URLs embedded, one can thwart the ability of spammers to sustain spam campaigns.

SOCIAL NETWORK BASED COMPONENTS. Most spam is sent using email addresses that the recipient has never communicated with in the past or e-mail addresses that are not within the recipients social network [15]. Using social network connectivity and prior communication history to determine puzzle difficulty can reduce unnecessary computation for legitimate webmail clients.

4. EVALUATION

4.1 Prototype

To demonstrate our approaches for generating time-lock puzzles and for setting puzzle difficulties, we implemented a prototype that is publicly accessible [1]. The prototype implementation is a web-based email transmission service written completely in PHP that delivers a JavaScript solver to the client for solving the modified time-lock puzzles. The solver is only 9KB and can be cached at the client since all puzzle parameters are simply passed as arguments to the solver via standard HTML tags. The system does not require modifications to either the web server or the web client software. kaPoW leverages OpenSSL [26] at the server to efficiently generate the modulus used in the modified time-lock algorithm and employs a geographic database when the location component is enabled [24]. By default, the prototype system runs a user's message through SpamAssassin [34], checks the URLs and domain names within the message against two blacklists [1, 40], checks the user's IP address against three blacklists (Spamhaus [36], SpamCop [35], Project HoneyPot [30]) and computes the geographic distance the user's IP address is away from the server's. Based on these checks, an overall score is generated to determine puzzle difficulty. Since the kaPoW prototype does not provide the complete functionality that an email service provider would, it emulates the rest of the components described previously. For simplicity, the age of the account, the recent message volume of the account, and whether the recipient has emailed the sender before are emulated as an aggregate measure that modifies the difficulty up or down. Figure 1 shows a screenshot of the kaPoW Webmail prototype.

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

KEY SIZE (bits)	MEAN TIME TO ISSUE (seconds)
400	0.165
600	0.471
800	1.13
1,000	2.04
1,200	3.90
1,400	6.10
1,600	9.88
1,800	15.3
2,000	20.6

Table 3: Issuing overhead (modulus generation) for the modified time-lock construction averaged over 1,000 trials.

4.2 Server Performance

One of the key components of kaPoW's webmail system is the modified time-lock algorithm that issues multiple puzzles using a single modulus n. The modulus is computed via the generation of two large prime numbers. Table 3 shows the baseline performance of the computational cost involved in generating this modulus as a function of the size of the modulus. The measurements were conducted on an Intel Core 2 Quad system (Q6600/2.4GHz) using OpenSSL 0.9.8 on Fedora Linux and the table lists the average generation time across 1,000 trials. As the table shows, the minimum cost for generating a new modulus is on the order of hundreds of milliseconds. Thus, for high-performance web applications, it prohibitively expensive to generate a new modulus for each puzzle issued to a client. The table also shows that the cost for generating a reasonably-sized modulus is several seconds. This indicates that it is feasible to periodically refresh the modulus to avoid giant-step, baby-step attacks that can shortcut the time-lock computation [33].

The modified time-lock algorithm amortizes the overhead of generating two large prime numbers by issuing multiple time-lock puzzles using a single modulus. This is done by generating the puzzle parameter a as a cryptographic hash of a periodically updated random server nonce K and client parameters such as the URL being requested, the client's IP address, and the difficulty of the puzzle issued. Creation of a new puzzle is thus limited by the speed the cryptographic hash can be done in PHP. For the prototype system, the standard $SHA1()$ function

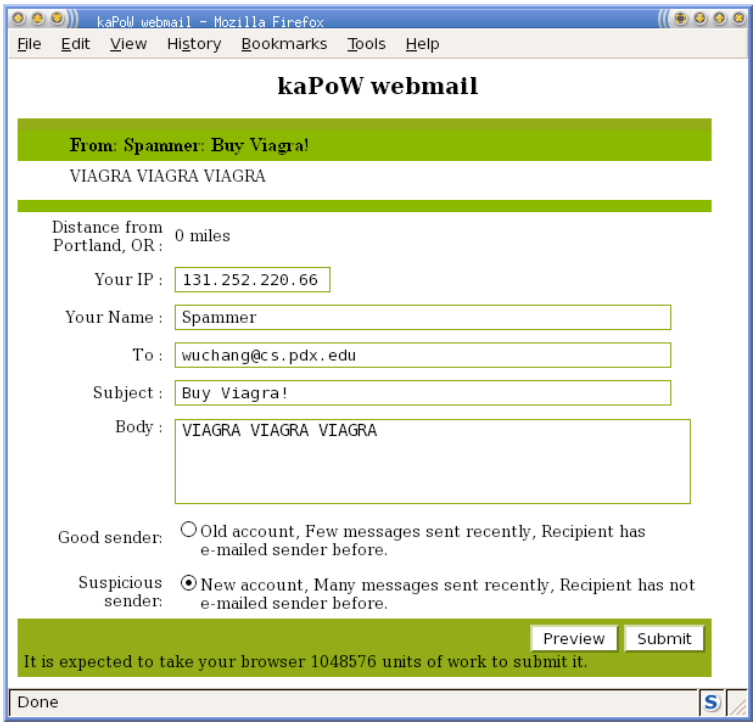

Figure 1: Screenshot of kaPoW Webmail prototype.

KEY SIZE (bits)	MEAN TIME TO VERIFY (milliseconds)
400	0.184
600	0.580
800	1.16
1,000	1.99
1,200	2.35
1,400	2.98
1,600	4.01
1,800	4.76
2,000	5.66

Table 4: Verification overhead in PHP: Calculating $r = 2^t \bmod \phi$ and $A' = a^r \bmod n$ averaged over 10,000 trials.

KEY SIZE (bits)	UNIT OF WORK (milliseconds)
400	10.4
600	11.8
800	15.0
1,000	21.6
1,200	22.6
1,400	25.4
1,600	27.2
1,800	28.6
2,000	30.6

Table 5: Unit work effort (modular squaring) in client puzzle solution averaged over 5,000 trials.

is used to generate a. Using the same machine as above, the overhead of issuing modified time-lock puzzles was measured over 1 million trials and the mean execution time for this operation was $5.32\mu s$. Thus, issuing a modified time-lock puzzle is many orders of magnitude faster than using the unmodified time-lock puzzle algorithm.

The final piece of the modified time-lock algorithm is the verification of answers. The verification procedure is the same as the original time-lock algorithm with one addition. The verifier must validate that the parameter a matches the client's request by recalculating the $SHA1()$ function on K and the client parameters. The main overhead in verification is performing the shortcut computation by calculating $r = 2^t \bmod \phi$ and $A' = a^r \bmod n$. Table 4 shows the overhead in performing the verification step across a range of key sizes. As the figure shows, the overhead in verifying a client's answer is only a handful of milliseconds across all key sizes.

4.3 Client Performance

The client solver is written in JavaScript and leverages a Big Integer Library [4] to perform the modular squaring with arbitrarily large integers. The key component for the solver is the amount of time a client consumes to perform an operation. Table 5 shows the average time a modular squaring operation takes on our test machine using Firefox 3.6.2 with TraceMonkey (the JavaScript just-in-time compiler) enabled. The measurements are taken across a

range of modulus sizes and the average is taken across 5,000 squarings. As the table shows, each operation consumes tens of milliseconds.

4.4 Difficulty Setting

kaPoW applies a defense-in-depth approach against the problem of webmail spam. Rather than use a single detector such as the content of the message or the recent request rate of the client, it uses a comprehensive set of metrics for determining the difficulty of puzzles that clients must solve. This is important for properly identifying and penalizing misbehavior while allowing legitimate use to go through. Our approach for setting difficulties is to apply individual tests against the message being sent and the client sending it. These tests are aggregated into a single score that is then used to generate the difficulty.

It is an open question *how* to set the difficulties appropriately. However, the benefits of our approach can be demonstrated even with the simplest of algorithms. Consider the scenario of a webmail interface for a university [28]. Such systems are under constant threat of spear phishing attacks where adversaries obtain legitimate account credentials and use them to send large amounts of spam via bots. To address these attacks, the algorithm we consider uses a scoring algorithm across all components: time, usage, location, reputation, content, and social network characteristics. For each component, we use a binary test to indicate whether the activity is suspicious or not. The individual tests we use for each component are:

- *Time (S_t)*: Does the current time of day fall within an 8-hour window during the day that users typically send email?
- *Usage (S_u)*: Has the user account sent a message within the last 5 minutes?
- *Location (S_l)*: Is the geographic location of the IP address of the client within 500 miles of the institution?
- *Reputation (S_r)*: Does the IP address of the client appear on any blacklists?
- *Content (S_c)*: Does SpamAssassin consider the message spam?
- *Social network (S_s)*: Does the recipient of the message appear in the user account's address book?

Using these metrics, the algorithm generates an overall score by summing the individual tests up resulting in a score from 0 to 6:

$score = S_t + S_u + S_l + S_r + S_c + S_s$

From this score, the difficulty of the modified time-lock puzzle issued to a client is set as:

$t = 20 \times score^6$

Thus, the range of t goes from 0 to 933,120 which corresponds to client solution times of 0 seconds to 18,662 seconds as measured on our test system. Given this, we then simulate a range of bots attempting to send as much spam as possible through the webmail interface using the compromised account. We assume that bots send messages that are classified correctly by SpamAssassin with 80% success (i.e. $S_c = 1$ for 80% of the messages). They also send messages to recipients that are not in the user's address book ($S_s = 1$). The experiment also simulates a legitimate user attempting to send a message that is not classified as spam ($S_c = 0$), to someone in his/her social network ($S_s = 0$), during regular hours ($S_t = 0$), at a local location ($S_l = 0$), on a machine whose IP address does not appear on a blacklist ($S_r = 0$). With this setup, the only potential penalty against the legitimate user is the usage component S_u, as the adversary has hijacked the account and has been sending messages throughout the day on it.

Table 6 shows the average number of messages different bots are able to send using the hijacked account over an entire day across 1,000 simulations. The table also lists the average delay the bot experiences in sending each message due to solving the modified time-lock puzzle. As the table shows, bots that are local and have IP addresses with good reputations are able to send the most messages through the service. However, since they are sending messages that are likely to be classified as spam, to recipients that are not in the user's social network, at a rate that will trigger the usage component, and during times of day that are abnormal, they are eventually given puzzles with significantly higher difficulty and are forced to slow down. For bots that are not local or that have IP addresses that appear on blacklists, the penalty is even steeper and they send substantially fewer messages. Finally, the table lists the average delay the legitimate user experiences when attempting to send a message. As the table shows, while the adversary is impacted significantly, the legitimate user experiences a nominal delay in sending a message.

5. CONCLUSION

Current proof-of-work systems have several limitations in their use of hash-based puzzles and simplistic difficulty settings. kaPoW webmail addresses these limitations through a novel and efficient construction of time-lock puzzles and the use of a comprehensive set of metrics to drive puzzle difficulties. While our initial prototype system and difficulty algorithms show promise, it is an open question whether such a system can effectively turn back webmail spam. As part of future work, we plan to rigorously evaluate a wider range of difficulty algorithms across a wider range of adversaries. We also plan

BOT TYPE	AVERAGE MESSAGES SENT BY BOT IN ONE DAY	AVERAGE PER-MESSAGE DELAY FOR BOT (S)	AVERAGE MESSAGE DELAY FOR LEGITIMATE CLIENT (S)
Local Bot with Good Reputation ($S_l = 0$, $S_r = 0$)	159.7 ± 5.6	540.4 ± 24.1	0.400 ± 0.000
Local Bot with Bad Reputation ($S_l = 0$, $S_r = 1$)	30.3 ± 2.2	2,860 ± 47.9	0.116 ± 0.040
Remote Bot with Good Reputation ($S_l = 1$, $S_r = 0$)	30.4 ± 2.3	2,851 ± 47.6	0.104 ± 0.041
Remote Bot with Bad Reputation ($S_l = 1$, $S_r = 1$)	8.4 ± 1.2	10,309 ± 83.8	0.000 ± 0.000

Table 6: Performance of difficulty algorithm across several bots.

on applying our approach to popular open-source webmail packages in order to validate its utility in a real deployment [39]. Finally, similar to our previous work applying proof-of-work against web-based denial-of-service attacks [12, 20] and ticket purchasing robots [21], we plan to apply our approach to additional applications such as multi-factor web authentication and web comment spam.

6. REFERENCES

[1] AnonWhois.org. The Anonymous Whois List. http://anonwhois.org.

[2] T. Aura, P. Nikander, and J. Leiwo. DoS-Resistant Authentication with Client Puzzles. In *Workshop on Security Protocols*, April 2000.

[3] A. Back. Hashcash: A Denial of Service Counter-Measure. Technical report, Cypherspace, August 2002. http://www.hashcash.org/papers/hashcash.pdf.

[4] L. Baird. Big Integer Library v. 5.1. http://www.leemon.com.

[5] D. Boneh. Twenty Years of Attacks on the RSA Cryptosystem. *Notices of the American Mathematical Society*, 46(2), 1999.

[6] Cisco Systems. Cisco Annual Security Report, December 2008. http://newsroom.cisco.com/dlls/2008/prod_121508.html.

[7] D. Danchev. Gmail, Yahoo and Hotmail's CAPTCHA Broken by Spammers, July 2008. ZDNet News.

[8] D. Dean and A. Stubblefield. Using Client Puzzles to Protect TLS. In *USENIX Security Symposium*, August 2001.

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

[10] C. Dwork and M. Naor. Pricing via Processing or Combatting Junk Mail. In *CRYPTO*, August 1992.

[11] W. Feng. The Case for TCP/IP Puzzles. In *ACM SIGCOMM Workshop on Future Directions in Network Architecture (FDNA)*, August 2003.

[12] W. Feng and E. Kaiser. The Case for Public Work. In *IEEE Global Internet Symposium*, May 2007.

[13] W. Feng, E. Kaiser, W. Feng, and A. Luu. The Design and Implementation of Network Puzzles. In *IEEE INFOCOM*, March 2005.

[14] GetAFreelancer.com. Captcha Entry Projects. http://www.getafreelancer.com/projects/by-tag/captcha-entry.html.

[15] J. Golbeck and J. Hendler. Reputation Network Analysis for Email Filtering. In *CEAS*, July 2004.

[16] J. Goodman and R. Rounthwaite. Stopping Outgoing Spam. In *ACM Conference on Electronic Commerce*, May 2004.

[17] S. Hocevar. PWNtcha - Captcha Decoder. http://sam.zoy.org/pwntcha.

[18] IronPort Systems. IronPort Research Discovers Links Between Malware Originators and Illegal Online Pharmaceutical Supply Chain, June 2008. http://www.ironport.com/company/ironport_pr_2008-06-11.html.

[19] A. Juels and J. Brainard. Client Puzzles: A Cryptographic Defense Against Connection Depletion. In *NDSS*, February 1999.

[20] E. Kaiser and W. Feng. mod_kaPoW: Protecting the Web with Transparent Proof-of-Work. In *IEEE Global Internet Symposium*, April 2008.

[21] E. Kaiser and W. Feng. Helping TicketMaster: Changing the Economics of Ticket Robots with Geographic Proof-of-Work. In *IEEE Global Internet Symposium*, March 2010.

[22] D. Kesmodel. Codes on Sites 'Captcha' Anger of Web Users, May 2006. Wall Street Journal.

[23] B. Laurie and R. Clayton. 'Proof-of-Work' Proves

Not to Work'. In *Workshop on Economics and Information Security*, May 2004.

[24] MaxMind, Inc. Geolocation and Online Fraud Prevention from MaxMind. http://www.maxmind.com.

[25] MessageLabs. MessageLabs Intelligence Report: May 2009. http://www.messagelabs.com/intelligence.aspx.

[26] OpenSSL Project. OpenSSL: The Open Source Toolkit for SSL/TLS. http://www.openssl.org.

[27] B. Parno, D. Wendlandt, E. Shi, A. Perrig, B. Maggs, and Y. Hu. Portcullis: Protecting Connection Setup from Denial-of-Capability Attacks. In *ACM SIGCOMM*, August 2007.

[28] Portland State University. Portland State University Webmail. http://webmail.pdx.edu.

[29] S. Prasad. Google's CAPTCHA Busted in Recent Spammer Tactics, February 2008. http://securitylabs.websense.com/content/Blogs/2919.aspx.

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

[31] R. Malmgren and J. Hofman and L. Amaral and D. Watts. Characterizing Individual Communication Patterns. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, July 2009.

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

[33] D. Shanks. Class Number, A Theory of Factorization and Genera. In *Symposia in Pure Mathematics*, 1971.

[34] SpamAssassin. The Apache SpamAssassin Project. http://spamassassin.apache.org.

[35] spamcop.net. SpamCop. http://www.spamcop.net/.

[36] Spamhaus Project Ltd. Spamhaus Project. http://spamhaus.org/.

[37] R. Stross. Hannah Montana Tickets on Sale! Oops, They're Gone, December 2007. New York Times.

[38] SURBL Community. SURBL. http://www.surbl.org.

[39] The Horde Project. The Horde Project. http://www.horde.org.

[40] URIBL. Realtime URI Blacklist. http://www.uribl.com.

[41] L. von Ahn, M. Blum, N. Hopper, and J. Langford. CAPTCHA: Using Hard AI Problems for Security. In *CRYPTO*, August 2003.

[42] X. Wang and M. Reiter. Defending Against Denial-of-Service Attacks with Puzzle Auctions. In *IEEE Symposium on Security and Privacy (S&P)*, May 2003.

[43] X. Wang and M. Reiter. Mitigating Bandwidth-Exhaustion Attacks Using Congestion Puzzles. In *ACM CCS*, October 2004.

[44] Z. Zhong, K. Huang, and K. Li. Throttling Outgoing SPAM for Webmail Services. In *CEAS*, July 2005.