Portland State University
PDXScholar

Dissertations and Theses Dissertations and Theses

Winter 3-13-2014

Using Spammers' Computing Resources for Volunteer Computing

Thai Le Quy Bui
Portland State University

Let us know how access to this document benefits you.

Follow this and additional works at: https://pdxscholar.library.pdx.edu/open_access_etds

Recommended Citation
Bui, Thai Le Quy, "Using Spammers' Computing Resources for Volunteer Computing" (2014). *Dissertations and Theses.* Paper 1629.

10.15760/etd.1628

This Thesis is brought to you for free and open access. It has been accepted for inclusion in Dissertations and Theses by an authorized administrator of PDXScholar. For more information, please contact pdxscholar@pdx.edu.

المنارة للاستشارات

www.manaraa.com

Using Spammers' Computing Resources for Volunteer Computing

by

Thai Le Quy Bui

A thesis submitted in partial fulfillment of the
requirements for the degree of

Master of Science
in
Computer Science

Thesis Committee:
Wu-chang Feng, Chair
Wu-chi Feng
Charles Wright

Portland State University
2014

للاستشارات المنارة

www.manaraa.com

Abstract

Spammers are continually looking to circumvent counter-measures seeking to slow them down. An immense amount of time and money is currently devoted to hiding spam, but not enough is devoted to effectively preventing it. One approach for preventing spam is to force the spammer's machine to solve a computational problem of varying difficulty before granting access. The idea is that suspicious or problematic requests are given difficult problems to solve while legitimate requests are allowed through with minimal computation. Unfortunately, most systems that employ this model waste the computing resources being used, as they are directed towards solving cryptographic problems that provide no societal benefit. While systems such as reCAPTCHA and FoldIt have allowed users to contribute solutions to useful problems interactively, an analogous solution for non-interactive proof-of-work does not exist. Towards this end, this paper describes MetaCAPTCHA and reBOINC, an infrastructure for supporting useful proof-of-work that is integrated into a web spam throttling service. The infrastructure dynamically issues CAPTCHAs and proof-of-work puzzles while ensuring that malicious users solve challenging puzzles. Additionally, it provides a framework that enables the computational resources of spammers to be redirected towards meaningful research. To validate the efficacy of our approach, prototype implementations based on OpenCV and BOINC are described that demonstrate the ability to harvest spammer's resources for beneficial purposes.

المنارة للاستشارات

www.manaraa.com

To my family and Sara.

المنارة للاستشارات

www.manaraa.com

Acknowledgments

I would like to thank my advisor Dr. Wu-chang Feng for inspiring me to begin this journey, for guiding me through meaningful research, for his willingness to share great ideas, for countless interesting discussions, advice and terrific foosball matches. I would like to thank Dr. Wu-chi Feng and Dr. Charles Wright for serving in my thesis committee, for providing an engaging, friendly learning environment through lunch, play and capture-the-flag gathering.

I would also like to thank Akshay Dua for his valuable advice and his eagerness to help. The completion of this thesis would be much more difficult without the support of all the fellow students and co-authors in my research group Akshay Dua, Nhan Huynh, Tien Le and Ryan Niebur.

I would like to pay my gratitude to my family, mom, dad and sister, who has always been understanding, believing and supporting me. I want to thank Sara for her kindness and love that keeps me moving forward until today.

Last but not least, I want to thank the National Foundation of Science for providing financial support through grant CNS-0627752 and CNS-1017034, thank Portland State University, Maseeh College of Engineering & Computer Science and its staff for an astounding graduate program.

المنارة للاستشارات
www.manaraa.com

Table of Contents

Abstract . . . . **i**

Dedication . . . . **ii**

Acknowledgments . . . . **iii**

List of Tables . . . . **vi**

List of Figures . . . . **vii**

1 Introduction **1**
- 1.1 Current Anti-Spam Strategies and Challenges 2
- 1.2 Design Goals 4
- 1.3 Our contributions 5

2 Metamorphic Proof-of-Work in Online Applications **8**
- 2.1 Background 9
  - 2.1.1 CAPTCHA 9
  - 2.1.2 Proof-of-work 9
- 2.2 System Model 10
- 2.3 Communication Protocol 12
  - 2.3.1 Authentication 12
  - 2.3.2 Puzzle Delivery and Verification 14
- 2.4 System Components 15
  - 2.4.1 Reputation Service 16
  - 2.4.2 Puzzle Service 18
  - 2.4.3 Public API 23
- 2.5 Implementation 24
  - 2.5.1 Reputation Service 24
  - 2.5.2 Puzzle Service 26
  - 2.5.3 Client API 26

المنارة للاستشارات

www.manaraa.com

2.5.4 Server API . . . 27
2.6 Results . . . 27
2.6.1 Experimental Setup . . . 27
2.6.2 Defense-in-Depth . . . 28
2.6.3 Reputation Accuracy . . . 29
2.6.4 Performance . . . 31
2.7 Security Analysis . . . 33

3 Volunteer Computing Workloads as Useful Proof-of-Work . . . 35
3.1 reBOINC Approach . . . 36
3.1.1 Challenges and Solutions . . . 36
3.1.2 System Model . . . 38
3.1.3 Threat Model . . . 40
3.2 System Components . . . 41
3.2.1 Native Client . . . 41
3.2.2 BOINC . . . 42
3.2.3 OpenCV . . . 43
3.2.4 MetaCAPTCHA . . . 43
3.3 Design . . . 45
3.3.1 reBOINC Proof-of-work Model . . . 45
3.3.2 MetaCAPTCHA Modifications . . . 48
3.3.3 Architecture and Communication . . . 51
3.4 Implementation . . . 53
3.4.1 Useful Services . . . 53
3.4.2 BOINC Wrapper . . . 54
3.4.3 NaCl Application . . . 55
3.5 Evaluation . . . 56
3.5.1 Client User Experience . . . 58
3.5.2 Client NaCl Performance . . . 59
3.5.3 Server Performance . . . 60
3.6 Limitations and Future Work . . . 61

4 Related Work . . . 63

5 Conclusion . . . 66

References . . . 68

المنارة للاستشارات
www.manaraa.com

List of Tables

3.1 The reBOINC server performance and overheads 58

المنارة للاستشارات
www.manaraa.com

List of Figures

2.1 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. 11
2.2 Kerberos authentication overview and how it relates to MetaCAPTCHA authentication. Figure adapted from Steiner et al. [91] 13
2.3 MetaCAPTCHA authentication and puzzle solution verification . . 13
2.4 The user's web browser is continuously issued puzzles until it has spent enough time computing. This amount of time is called the difficulty-level; the more malicious the client, the higher the puzzle difficulty-level. 15
2.5 Design of MetaCAPTCHA 16
2.6 MetaCAPTCHA's puzzle configuration dashboard. 22
2.7 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. 28
2.8 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) 30
2.9 MetaCAPTCHA performance 31
2.10 Breakdown (%) of the time spent in issuing the first puzzle. Notice that 68% of the time is spent in the reputation service due to all the remote queries that happen there. 32

3.1 System model: The client browser has to compute useful computations in order to gain a signed cookie (the proof-of-work) from the reBOINC system. Solid arrows represent requests, dashed ones represent responses and dotted ones are periodic communications and/or one-time setup. 39

www.manaraa.com

3.2 MetaCAPTCHA anti-spam web service overview 44
3.3 reBOINC system overview. The dotted boxes are reBOINC modifications to the MetaCAPTCHA service. 48
3.4 Computational Banking Service Architecture 49
3.5 Useful Puzzle Service Architecture 50
3.6 The reBOINC system. The solid lines represent run-time traffic while the dashed ones represent periodic communication. 52
3.7 Bonneville Dam Fish Ladder 55
3.8 Browser's resource ultilization while running NaCl fish counting application and watching Youtube videos in 360p, 720p and 1080p format. 57
3.9 Performance of fish counting application running on OS vs. NaCl vs. Javascript 58

www.manaraa.com

1

INTRODUCTION

Internet spammers are relentless. Although email spam is reducing (from 92.2% in Aug 2010 to $\approx$ 70.5% in Jan 2012), the spam on social networking sites is edging up [49]. In 2011, approximately 4 million Facebook users received spam from around 600,000 new or hijacked accounts each day [49, 71]. From there on, social spam had risen 355% just during the first six months of 2013 [109]. Particularly, the success rates of social spam are frightening: 0.13% of all spam URLs on Twitter were visited by around 1.6 million unsuspecting users [55]. This click-through rate is significantly higher than the ones of email spam which were reported at 0.003% - 0.006% [64]. Especially on Facebook, the mean spam conversion rate was a staggering number of 47% [86]. This means that nearly half the Facebook users who saw spam links or messages actually clicked on and read them. Furthermore, traditional web applications such as blogs, e-commerce and forums are also suffering from the increased amount of social spam. With the expanding popularity of *OAuth protocol*[1], it is very common for web applications to be tightly integrated with social network ecosystems. While this is convenient for the users, a compromised account in a social networking site can spam not only the account's friends, but every web application using the OAuth features of the networking site. As

[1]The OAuth - Open Authentication - protocol allows the same user account to be used in multiple websites. For example, a Wordpress blog that implements Facebook's OAuth protocol will allow anonymous users to use their Facebook's accounts to post comments.

المنارة للاستشارات
www.manaraa.com

a result, social networking sites and web applications in general have become the new attractive venue for Internet spammers.

1.1 CURRENT ANTI-SPAM STRATEGIES AND CHALLENGES

Current methods for fighting spam in web applications can be classified in two strategies: detection-based and prevention-based [59]. The detection-based methods classify a given data as spam or *ham* (not spam messages) whereas the prevention-based ones make it difficult to send spam. A spam filter is a famous example of detection-based methods. Spam filters work by employing various machine learning algorithms and statistical techniques to discard malicious content while leaving unmalicious data intact. By training the spam filters on a known set of large enough spam and ham, the filters can determine how likely a new data is spam or ham with up to 99.9% accuracy [108].

One drawback of spam detection methods is that they identify the malicious contents only after the contents have been transported. For instance, a bot using a hijacked account on Facebook to send spam to all of its friends only gets caught after the messages have been transported through the network and arrived at Facebook's servers. Moreover, spam detection methods don't punish malicious behaviors enough; they only hide the malicious contents away from the legitimate ones. Thus, a spam bot could use a strategy to attack many web applications for as long as its resources are available. Since any statistical spam detection algorithm will have a small false positive rate (known as *Bayes error rate* [38]), the strategy will maximize the amount of malicious contents that gets to the web applications undetected. As such, the Internet is clogged with endless spam traffic requiring demanding computing resources to filter. To prevent this deluge traffic of spam, several spam prevention methods are often employed, among them, *CAPTCHA* and *proof-of-work* are two most common methods [59].

CAPTCHAs are special tests designed to tell computers and humans apart.

المنارة للاستشارات
www.manaraa.com

The common types of CAPTCHAs usually use distorted texts and images that are hard for computers but easy for humans to solve. However, a CAPTCHA can effectively protect an online transaction so long as there aren't OCR algorithms that can automatically "solve" or "break" it [92]. On average, an algorithm to break CAPTCHAs only needs to have a precision rate as low as %1 to be considered successful [26]. Once a class of CAPTCHAs is broken, the corresponding application becomes defenseless against spam bots. In addition, CAPTCHAs are also prone to outsourcing attacks where humans are used 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 [76, 77]. Plus, the usability burden imposed by CAPTCHAs [106] 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 is a different kind of spam prevention method that does not have CAPTCHA's usability issues and therefore, can be used in frequent transactions. Proof-of-work often involves cryptographic algorithms whose results are challenging to find but easy to be verified. An application using the proof-of-work protocol will require user's devices to compute these algorithms as "work". The application's servers will verify the outcomes of this work in order to grant users access to certain services. This work will not require any user intervention which removes the usability burdens of CAPTCHAs; however, the users will need to wait for a small period of time in order to have their devices completing the work. On the other hand, spammers will have to solve an increased amount of work corresponding to the excessive amount of spam they sent. Furthermore, this paradigm enables an application to price a transaction by varying the amount of work that needs to be done as payment. In fact, proof-of-work systems are only effective if the price of the transaction is based on the corresponding user's reputation [67]. Such systems

المنارة للاستشارات
www.manaraa.com

limit the number of spam messages an attacker can send by arbitrarily increasing the resource cost for accepting the requests.

However, while the principle of proof-of-work seems effective in increasing the cost of spam, existing systems of this kind have two main shortcomings. First, the computing resources that the user's devices spent in solving the proof-of-work are wasted. Most of the proposed systems do not employ a *useful computation* (a computation whose result can be reused for other purposes such as volunteer computing) as a proof-of-work. It is crucial to note that while consuming some computational resources of the devices is required by the proof-of-work protocol, abandoning the results of those computations is undesirable. As a result, the proof-of-work protocol solves one problem by increasing the cost of spam, but creates another problem by wasting the user's computing power. The second shortcoming of current systems is the impracticality in constructing, supporting and deploying proof-of-work. Many proposed systems [42, 23, 60, 39, 62, 110] (i) demand changes of the network layer or the application layer to allow the proof-of-work protocol to operate, (ii) use only cryptographic functions to generate and verify the proof-of-work, and (iii) restrict the proof-of-work functions to a limited class of computations. Thus, creating a new proof-of-work for the former systems requires numerous conditions to be met, thereby, making it complicated to create.

1.2 DESIGN GOALS

Motivated by the above observations, we want our fighting spam systems to retain the strengths of CAPTCHAs and proof-of-work while addressing their weaknesses. For instance our systems undertake current drawbacks of CAPTCHAs such as no support to price a transaction dynamically based on user's reputations, no backup solution when one class of CAPTCHAs is broken and most importantly the usability issues imposed by CAPTCHAs. Likewise, the systems tackle existing problems

www.manaraa.com

of the proof-of-work model such as no support for big classes of computation, wasting computing cycles of the user's devices for meaningless cryptographic functions with no reusable results. Furthermore, we believe that a successful security model should create a more pleasant user's experience as well as protect a system against improper behaviors. Ultimately, the systems should be flexible, practical and secure that the community can adopt to solve real-world problems. Thus we setup the following requirements for our systems as design goals:

Flexibility New systems or modifications are easy to be added with relatively less effort than creating them from scratch. The systems can work with many existing web applications with little extra overheads.

Practicality The systems are concerned with current usability and impracticability issues and use well-known standards to solve those issues. The systems are fast and efficient with acceptable latency for web transactions such as message postings and account creations.

Security The systems are difficult for the adversaries to subvert. The systems are dynamic and adaptable to new adversarial tactics while ensuring that legitimate users are not encumbered.

1.3 OUR CONTRIBUTIONS

We designed and implemented the MetaCAPTCHA and reBOINC systems. MetaCAPTCHA is an application-agnostic spam prevention service for the web. MetaCAPTCHA adds metamorphism to address CAPTCHA's usability, strengthens proof-of-work's flexibility and security while augmenting each. reBOINC leverages MetaCAPTCHA's infrastructures and improves the proof-of-work model to support a green, flexible and practical computing model. The reBOINC system

المنارة للاستشارات
www.manaraa.com

is designed to be a computational analogy to reCAPTCHA [101] - an anti-spam system where half of the work being done goes towards digitizing text in scanned books. As a result, computing resources spent for solving proof-of-work in the reBOINC system directly benefit volunteer computing projects.

More specifically the MetaCAPTCHA and reBOINC systems:

- Seamlessly integrate the CAPTCHA and proof-of-work approaches. They can dynamically issue proof-of-work or CAPTCHA puzzles while ensuring that malicious users solve much "harder" puzzles — CAPTCHAs included — than honest users.

- Support a variety of computations, as well as workloads from *BOINC* (The Berkeley Open Infrastructure for Network Computing) [18, 11] - the most widely adopted infrastructure for volunteer computing projects - as proof-of-work. Therefore, the systems are not only green but also practical (require little changes and support big classes of computations).

- Connect the pool of computing resources at the spammer's disposal to the large quantity of research projects that require computational resources. By doing so, a win-win scenario is accomplished in that either a significant amount of useful work is completed, or a considerable reduction in spam is achieved.

- Randomly pick and deliver puzzles within a generic solver that eventually executes those puzzles in the user's web browser. Thus, the solver code is metamorphic where it is 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.

- Use a Bayesian reputation service that can accurately predict a user's reputation score based on features configured by the web application. Since

المنارة للاستشارات
www.manaraa.com

multiple web applications can be protected by our systems, the reputation service provides global visibility on attacks across all those applications.

- Contain a modular puzzle library that can be configured with new types of CAPTCHAs or proof-of-work puzzles while allowing the removal of those types 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 the systems as a whole.

The remainder of the paper is sectioned as follows. Chapter 2 describes the design, implementation and evaluation of MetaCAPTCHA and lays the foundation for the reBOINC system. Next, chapter 3 details the design, implementation and evaluation of reBOINC. The reBOINC system is the main contribution of this paper where spammer computing resources are harvested to use for good purposes. Chapter 4 outlines related work and Chapter 5 concludes this paper.

www.manaraa.com

2

METAMORPHIC PROOF-OF-WORK IN ONLINE APPLICATIONS

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. In addition, CAPTCHAs doesn't have a dynamic pricing function where the difficulty of solving the CAPTCHAs can be controlled. Proof-of-work approaches are gaining popularity, but current implementations are not effective enough and cannot be used by generic web applications. Towards this end, this chapter presents MetaCAPTCHA, an application-agnostic spam prevention service for web applications. It dynamically issues CAPTCHAs and proof-of-work puzzles while ensuring that malicious users solve challenging puzzles with exponential growth. Furthermore, MetaCAPTCHA strengthens the proof-of-work model by using metamorphic puzzle issuing and solving process. This metamorphic process ensures that the adversaries wanting to attack MetaCAPTCHA will have to continuously solve the reverse-engineering problem. We evaluate MetaCAPTCHA in the context of a reference web application and show that 95% of honest users hardly notice MetaCAPTCHA's presence, whereas the remaining 5% are required to solve very "easy" puzzles before accessing the application's services.

المنارة للاستشارات
www.manaraa.com

2.1 BACKGROUND

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

2.1.1 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 [101].

2.1.2 Proof-of-work

The proof-of-work model was first proposed by Dwork and Naor [42] 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 $y = f(x)$

المنارة للاستشارات
www.manaraa.com

An example of a pricing function is one that finds partial hash collisions [22]. A function $f_k : x \to y$ 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 [67] demonstrated in 2004 that reducing spam to 1% of normal email would require delaying each message — including one that an honest user sends — by ≈ 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 [68] 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.

2.2 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. [59] provide an exhaustive discussion on the common characteristics of such web applications.

المنارة للاستشارات
www.manaraa.com

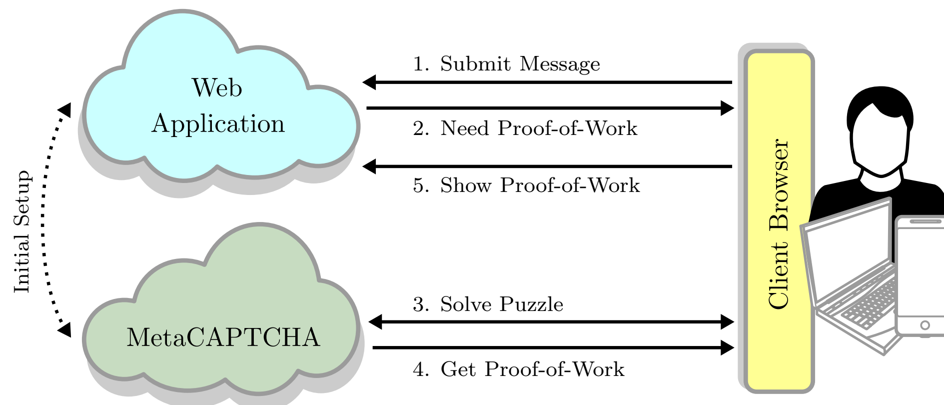


Figure 2.1: System model: user's browser must show proof-of-work before the web application accepts the user's message. The dotted line indicates initial setup performed by the web application to use the MetaCAPTCHA service.

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 2.1. 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.

The work is issued in the form of "puzzles". Puzzles can be *interactive*, *non-interactive*, or *hybrid*. Interactive puzzles are generally CAPTCHAs, whereas non-interactive puzzles are pricing functions as described in Section 2.1.2. A hybrid puzzle combines a CAPTCHA and a pricing function into one puzzle.

As shown in Figure 2.1, we treat the *user* separate from the *browser* while collectively referring to them both as the *client*. This is due to the existence of interactive puzzles that need user interaction to solve, and non-interactive puzzles that are solved automatically by the browser.

المنارة للاستشارات
www.manaraa.com

2.3 COMMUNICATION PROTOCOL

This section discusses the MetaCAPTCHA communication protocol. For simplicity, we assume a 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.

2.3.1 Authentication

MetaCAPTCHA only issues puzzles to clients of participating web applications. This requires MetaCAPTCHA to authenticate (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.

As implied by the system model in Section 2.2, 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 [91], wherein the web application acts as the Ticket-Granting-Server (TGS) for the MetaCAPTCHA service

المنارة للاستشارات
www.manaraa.com

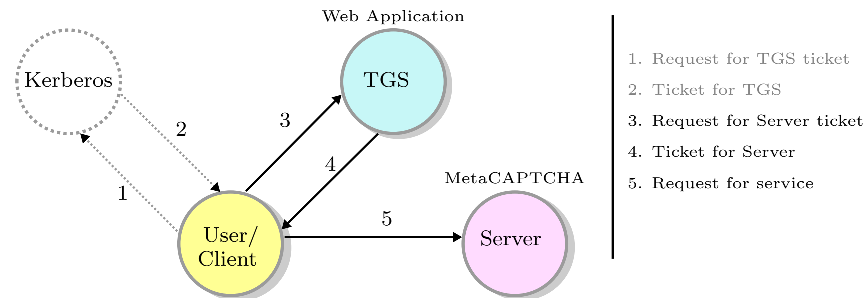


Figure 2.2: Kerberos authentication overview and how it relates to MetaCAPTCHA authentication. Figure adapted from Steiner et al. [91]

as shown in Figure 2.2 [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 2.3). The server-ticket S_1 is called the *puzzle-request ticket* and is sent by clients to MetaCAPTCHA for requesting puzzles.

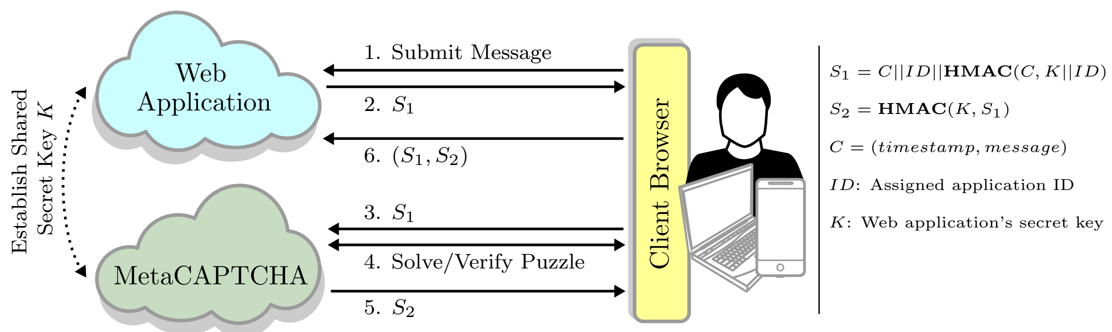


Figure 2.3: MetaCAPTCHA authentication and puzzle solution verification

When MetaCAPTCHA receives the puzzle-request-ticket S_1, it verifies that the

المنارة للاستشارات
www.manaraa.com

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.

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 2.3.2.

2.3.2 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 $C =$ (`timestamp, 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. In MetaCAPTCHA, higher reputation scores imply more malicious clients. As a result, such clients are issued puzzles with increased difficulty levels. The details of how reputation scores are computed, and how puzzles of varying difficulty are generated are presented in Sections 2.4.1 and 2.4.2 respectively.

It is important to note here that MetaCAPTCHA may issue multiple puzzles during a single puzzle solving session. Puzzles are continuously issued until the client has computed for an amount time similar to the estimated difficulty level. Pre-determining a difficulty level eliminates the usual incentive of solving puzzles faster. Furthermore, the estimated difficulty level is never directly revealed, thus

المنارة للاستشارات
www.manaraa.com

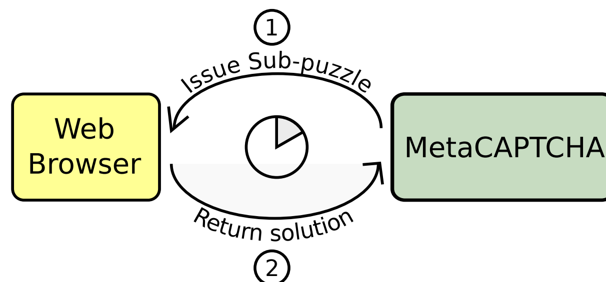


Figure 2.4: The user's web browser is continuously issued puzzles until it has spent enough time computing. This amount of time is called the difficulty-level; the more malicious the client, the higher the puzzle difficulty-level.

the client cannot make the decision to stop solving based on the amount of work it needs to do. Figure 2.4 shows the puzzle solving protocol. More details are presented in Section 2.4.2.

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 2.3), which will then verify the ticket 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.

2.4 SYSTEM COMPONENTS

In this section, we describe the main components of MetaCAPTCHA: the reputation service, the puzzle service, and the public API used by web applications and their clients to access the aforementioned services. Briefly, the interactions between these individual components begin when MetaCAPTCHA receives a client's message. This message is first handled by MetaCAPTCHA's reputation service,

www.manaraa.com

which determines the client's reputation score. The puzzle service then uses this score to generate and issue a puzzle of an appropriate difficulty. Figure 2.5 shows an overview of the various MetaCAPTCHA components and interactions, while the following sections describe each of them.

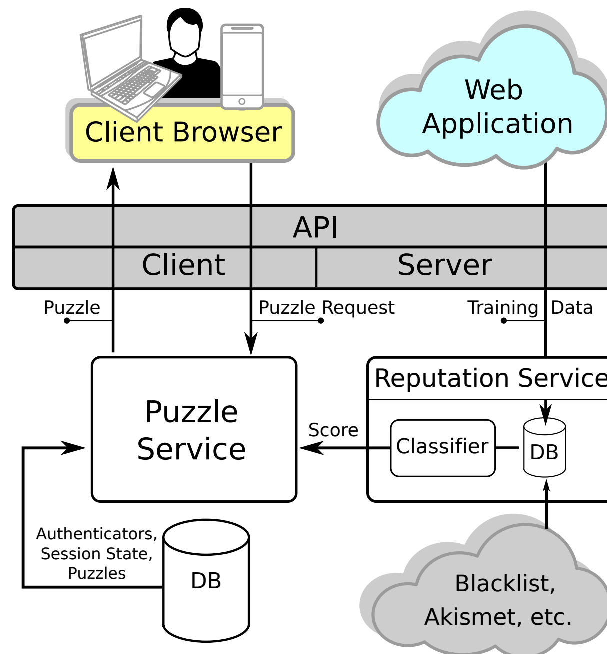


Figure 2.5: Design of MetaCAPTCHA

2.4.1 Reputation Service

Proof-of-work systems that do not assign more "work" to malicious clients than legitimate ones are easily circumvented [67]. Many existing systems do vary the amount of work, but fail to characterize maliciousness appropriately. For example, they base maliciousness on just one feature, such as system load [37, 61], a client's request rate [45, 44], contents of the request [111], or service demand [103, 104]. Without a defense-in-depth approach, it is unlikely that proof-of-work systems will be able to deter automated adversaries. Additionally, if a reputation system intends to be widely deployed, it must be capable of adapting to the needs of

المنارة للاستشارات
www.manaraa.com

individual applications [68]. For example, Twitter may associate low reputation with accounts that show aggressive following behavior [96], whereas Facebook may do the same for accounts with abnormally large amount of ignored friend requests [27]. MetaCAPTCHA's reputation service addresses these issues by allowing applications to easily configure the features that will determine a client's reputation score, and then use a Naive Bayes classifier to generically predict the score based on the values of the configured features.

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 when a message is posted to a web application and that client doesn't have an existing reputation. The score 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.

An important characteristic of a reputation system is its ability to react to a client's changing reputation. For example, if a user's account is hijacked by a spammer, her account's reputation worsens; however, once the threat from the hijacker is neutralized (say, by a password change) the reputation goes back to normal. Thus, a good reputation service must be capable of identifying these changes and assigning scores accordingly. MetaCAPTCHA's reputation service adapts to reputation changes by incorporating time-varying features in determining the reputation score. For example, relative account age, and relative number of positive votes a user's posts have received.

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

www.manaraa.com

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.

2.4.2 Puzzle Service

The puzzle service is responsible for authenticating users, using the contents of their transaction (e.g. message, IP address) to obtain a reputation score from the reputation service, converting that score to a puzzle difficulty, and finally, issuing the user a puzzle of that difficulty. Note that the authentication protocol was described in Section 2.3.1. Thus, in the following sections, we discuss the remaining responsibilities of the puzzle service.

Reputation Score to Puzzle Difficulty

As mentioned before, puzzle difficulty is the amount of time a client must be kept busy solving a puzzle. Traditionally, most puzzles have been CPU bound, causing devices with different processing speeds to solve the same puzzle for different amounts of time. Abadi et al. highlighted this issue and proposed memory-bound functions since memory access speeds vary much less across devices [16]. Memory bound puzzles, unfortunately, are expensive to create and verify [40, 34]. Furthermore, memory-bound puzzles limit the types of computation that can be performed: we envisage a future where puzzles are generic computations whose results are eventually reusable for solving larger problems like climate modeling, or curing cancer [94].

MetaCAPTCHA's approach is to first, determine the puzzle difficulty solely based on the reputation score — in units of time — and then, continuously issue puzzles until the client has computed for that pre-determined amount of time. The benefit of this approach is that it gets rid of an adversary's incentive to solve

المنارة للاستشارات
www.manaraa.com

puzzles quicker (e.g. by offloading, or parallelizing the computation). What is needed then, is a formula for converting the reputation score to a puzzle difficulty; the rest of this section derives such a formula.

Intuitively, the formula must ensure that puzzle difficulty is proportional to the reputation score since higher scores imply more malicious users. The remaining questions, then, are (i) how fast should the difficulty grow with respect to the reputation score? and (ii) for any given reputation score, what should the difficulty value be to effectively reduce the amount of spam the web application receives?

The answer to Question (ii) is inspired by work on the impact of proof-of-work systems on reducing spam by Laurie and Clayton [67]. We begin by fixing the amount of spam reduction δ the web application seeks as a fraction of the total number of spam messages s_p it receives in time period t_p. We can then determine the maximum difficulty or amount of time t_{max} a spammer must be kept busy to reduce the spam to $s_p - \delta s_p$:

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

Notice that if the desired spam reduction $\delta = 1$, the hardest puzzle a spammer may have to solve would be infinitely long; causing MetaCAPTCHA to wait forever for a solution! Since this is not feasible, the spam reduction fraction δ must be judiciously chosen. Additionally, the tighter the choice of t_p for the same s_p, the more accurate t_{max} will be. For example, assume a forum receives its first spam message of the day at 8:00 am and last spam message at 5:00 pm. Then, choosing $t_p = 9$ hours as opposed to, say 24 hours, will lead to a more accurate value of t_{max}.

Now, the answer to the question (i) depends on how accurately the reputation service can determine a user's reputation score; the more accurate it is, the less time honest clients should have to spend solving puzzles. We will see later that MetaCAPTCHA's reputation service aptly assigns ≈ 90% of spammers a score over

المنارة للاستشارات
www.manaraa.com

0.95. Since the reputation service is fairly accurate, we must fashion a function that grows slowly until large reputation scores and then steeply afterwards. In the case of the reference web application used to evaluate MetaCAPTCHA (see Section 2.6.1), we empirically settled on the exponential function with a growth constant of 5. However, another web application could choose a different growth constant based on the shape of the curve desired.

Given the aforementioned growth function and maximum puzzle difficulty t_{max}, we can compute the corresponding maximum reputation score r_{max}:

$$t_{max} = e^{5r_{max}} - 1$$
$$\implies r_{max} = \frac{\ln(t_{max} + 1)}{5}$$

We can then normalize the user's current reputation score r with respect to r_{max} and subsequently calculate puzzle difficulty t:

$$t = e^{5r \cdot r_{max}} - 1$$
$$\implies t = (t_{max} + 1)^r - 1$$

Notice that when reputation score $r = 1$ (most malicious) the puzzle difficulty $t = t_{max}$. Furthermore, when $r = 0$ (most honest), $t = 0$. This implies that an honest user may not have to solve a puzzle at all, whereas a malicious user may have to solve the hardest one.

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

المنارة للاستشارات
www.manaraa.com

unless they have computed for a length of time $\geq t$. An alternative is to first determine how long it takes to solve a puzzle, and then just generate and issue that puzzle. Unfortunately, the amount of time it takes to solve a puzzle varies on different platforms, and so clients may get issued unfairly long or short puzzles [16, 43]. For this reason, MetaCAPTCHA first computes the difficulty in units of time, then the client is required to solve the puzzles accordingly. Also, notice that the puzzle difficulty t is never revealed to the client, thus, there is no way to know how long the computation will last. This creates a disincentive for bots that would normally abandon puzzle computation altogether if t were known to be large beforehand. An additional advantage of issuing multiple puzzles is that each one can be randomly chosen, thus, preventing an attacker from being able to predict the puzzles she will be issued. This eliminates a critical advantage an adversary normally possesses: offline reverse engineering to find weaknesses. The next section discusses the various puzzle types supported by MetaCAPTCHA.

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 choice of which puzzle types to use depends on the web application's needs (See Figure 2.6).

Thus, MetaCAPTCHA is flexible and can be easily configured to support new types of puzzles. In fact, MetaCAPTCHA is so named because it is a metamorphic puzzle issuing system and because it issues meta-puzzles rather than specific puzzles. It is metamorphic because the client-side puzzle solver code is

المنارة للاستشارات

www.manaraa.com

Figure 2.6: MetaCAPTCHA's puzzle configuration dashboard.

non-deterministic. More specifically, the non-determinism results from issuing unpredictable puzzles within a generic puzzle solver — the meta-puzzle. Thus, the adversary has no way to know the client-side MetaCAPTCHA code beforehand. Furthermore, issuing a meta-puzzle ensures that finding a weakness in one puzzle type does not compromise MetaCAPTCHA as a whole. The following paragraphs discuss the currently supported puzzle types.

Hint-based Hash-Reversal (non-interactive) Hash-reversal puzzles force clients to reverse a given cryptographic hash of a random input, say x, with the k most significant bits erased. However, they lack fine-grained difficulty control because increasing k linearly, increases the solution search space exponentially. Hint-based hash-reversal puzzles address this drawback by providing an additional *hint*: the range of values to search.

Targeted Hash-Reversal (non-interactive) A targeted hash-reversal puzzle [45] with difficulty d forces a client compute an expected number d of hashes before

المنارة للاستشارات
www.manaraa.com

finding the right answer.

Modified Time-Lock (non-interactive) Time-lock puzzles [83] are based on repeated squaring, a sequential process that forces the client to compute in a tight loop for an amount of time that can be precisely controlled. *Modified* time-lock puzzles on the other hand, retain most of the original properties of time-lock puzzles, but are faster to generate and verify [46].

CAPTCHA (interactive) A reCAPTCHA [101] or Securimage [82] CAPTCHA relayed to the client. MetaCAPTCHA only acts as a proxy for these CAPTCHA services.

CAPTCHA+ (hybrid) A *CAPTCHA+* puzzle includes a reCAPTCHA or Securimage CAPTCHA along with a modified time-lock puzzle in the background. The advantage of combining the approaches, in this case, is that changing the difficulty of the time-lock puzzle changes the cost of solving the CAPTCHA. Consequently, hybrid puzzles could circumvent CAPTCHA outsourcing attacks [76, 77] since they enable the CAPTCHA solving cost to be changed.

2.4.3 Public API

The public API allows web applications and their clients to access MetaCAPTCHA services. The web applications use the server-side API to (i) register and maintain its account with MetaCAPTCHA, and (ii) as described in Section 2.3.1, use the account's unique API key to generate puzzle-request tickets for all their clients.

Web application clients implicitly use the client-side API during an online transaction, e.g. posting a message. More specifically, the API method calls are embedded in the client-side web page that accepts the online transaction (similar to a CAPTCHA setup). When the user "submits" the transaction, the client-side API is used to obtain a puzzle-request ticket from the web application and hand

المنارة للاستشارات
www.manaraa.com

it over to the MetaCAPTCHA service. Once MetaCAPTCHA returns a puzzle, the client uses the solver — also a part of the client-side API — to compute and return the puzzle solution.

2.5 IMPLEMENTATION

MetaCAPTCHA has been deployed with public APIs which any interested party can download and use. A beta version of the MetaCAPTCHA service can be found at `http://www.metacaptcha.com/`. We now discuss the implementation details of each MetaCAPTCHA component in Figure 2.5.

2.5.1 Reputation Service

The reputation service is implemented in PHP. Associated with this service is a NoSQL database, MongoDB [15], which stores the feature data required to determine user reputation. Initially, the web application provides the rows of feature data necessary to train a Naive Bayes classifier implemented in Java by the Weka [69] library. The trained classifier is then saved and used later when classifying new messages sent by the web application's users. Instead of using the binary classification — spam or ham — that usually is the output of a Naive Bayes classifier, the reputation service uses the probability distribution that the classifier determines in the step before performing the classification. That distribution represents MetaCAPTCHA's reputation score which is a percentage likelihood that a given message is spam.

To enable the reputation service to compute accurate reputation scores, a web application can provide existing message and user data for training the classifier. In the case of our reference web application, a live discussion forum that employed MetaCAPTCHA's spam prevention services from Sep 1 to Oct 19th 2012, the classifier was given the following feature values for each existing message in the

المنارة للاستشارات
www.manaraa.com

forum:

- Relative “Thanks” or “Likes”: the proportion of positive votes received by the sender of the message.
- Language: the language the forum message was written in.
- Relative account age: the proportion of time an account has been alive with respect to the age of the forum.
- Relative post count: the proportion of total posts published by a given account.
- DShield “Attacks” attribute: number of packets, from the message’s source to a distinct destination, that were blocked.
- GEOIP: an estimate of the distance between the message poster and forum server.
- Blacklist score: reputation score of the message source from Spamhaus [95]. The higher the score, the more malicious the source.
- Akismet score: Akismet [17] is a spam detection service that assigns a score of 1 to a message it thinks is spam and 0 otherwise.
- SA Score: the spam score as determined by the SpamAssassin [90] service running with only the Bayes plugin. The Bayes plugin uses a Naive Bayes classifier to determine the probability that the contents of a message resemble spam. SpamAssasin assigns a spam score between 1 and 5 to each message; 5 indicating that a message is most likely spam and 1 indicating that its not.
- isSpam: “yes” if forum moderator flagged the message as spam, “no” otherwise.

المنارة للاستشارات
www.manaraa.com

2.5.2 Puzzle Service

The puzzle service is implemented in PHP with an instance of the MongoDB database. The database stores credentials needed to authenticate a particular web application's client, to create and retrieve session details of the client solving puzzles. The authentication credentials include the web application's 96-bit API key, and an application ID; the session details store the received server ticket (see Sections 2.3.1, 2.3.2) and the amount of time the client has spent solving puzzles. This amount of time when subtracted from the puzzle difficulty level determines if the client needs a new sub-puzzle or not (see Section 2.4.2); the puzzles themselves are stored in the database as members of JSON (JavaScript Object Notation) objects and delivered in that format to clients. The parsing and execution of these JSON objects by client-side JavaScript engines can be thought of as "solving" a puzzle.

2.5.3 Client API

The client API includes JavaScript methods to request puzzles from MetaCAPTCHA, execute or "solve" them, and return the result of the execution. These methods must be embedded in an HTML form that accepts content from the users on behalf of the web application. As part of "submitting" that form, the client API will initiate the MetaCAPTCHA protocol to request a puzzle (see Section 2.3). The puzzle will be returned as a JSON object that the client must parse, evaluate, and then return the resulting value to MetaCAPTCHA.

The entire MetaCAPTCHA protocol occurs behind-the-scenes after a user clicks the "Submit" button. This behind-the-scenes behavior is enabled by the AJAX (Asynchronous JavaScript and XML) technique used to implement the MetaCAPTCHA communication protocol. Furthermore, puzzle execution is also pushed to the background by employing JavaScript worker threads [102] which are

المنارة للاستشارات
www.manaraa.com

now supported in newer versions of most popular browsers.

2.5.4 Server API

The server API consists of $\approx$ 150 lines of PHP code and requires minor modifications to the web application for its default configuration. The modifications are similar to those required by existing CAPTCHA APIs like reCAPTCHA [101]. Web applications use the server API to receive a client's message, issue the corresponding server ticket necessary to request a puzzle from MetaCAPTCHA, and verify the proof-of-work presented by clients that have solved the issued puzzle (see Section 2.3).

2.6 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.

2.6.1 Experimental Setup

MetaCAPTCHA was evaluated on a 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 19th 2012 employed MetaCAPTCHA as its spam prevention service. MetaCAPTCHA's effectiveness and performance have 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

المنارة للاستشارات
www.manaraa.com

ground-truth to judge the classifier's effectiveness. The data consisted of values for all features described in Section 2.5.1 for each of 1442 messages posted to the forum. We now describe the experiments used to evaluate MetaCAPTCHA.

2.6.2 Defense-in-Depth

Defense-in-depth strategy should yield better user reputation when using multiple features as opposed to using only one or a few. Recall that a user's reputation score is the probability that the user's message 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.

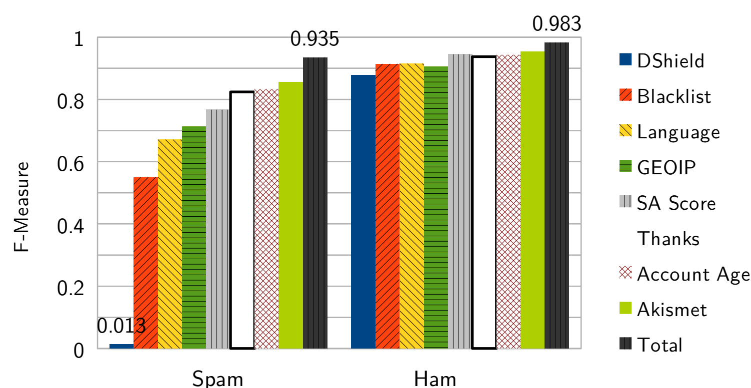


Figure 2.7: Defense-in-depth: using multiple features for spam classification is better than using one or a few. "Total" implies that all-of-the above features were used for training the classifier.

We 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

المنارة للاستشارات
www.manaraa.com

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.

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 2.7 that the classifier's F-measure is largest when using all features together than when using any single one.

2.6.3 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 who 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 2.8 *(a)* shows that ≈ 90% of *spammers* have reputation scores over 0.95, whereas ≈ 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

المنارة للاستشارات
www.manaraa.com

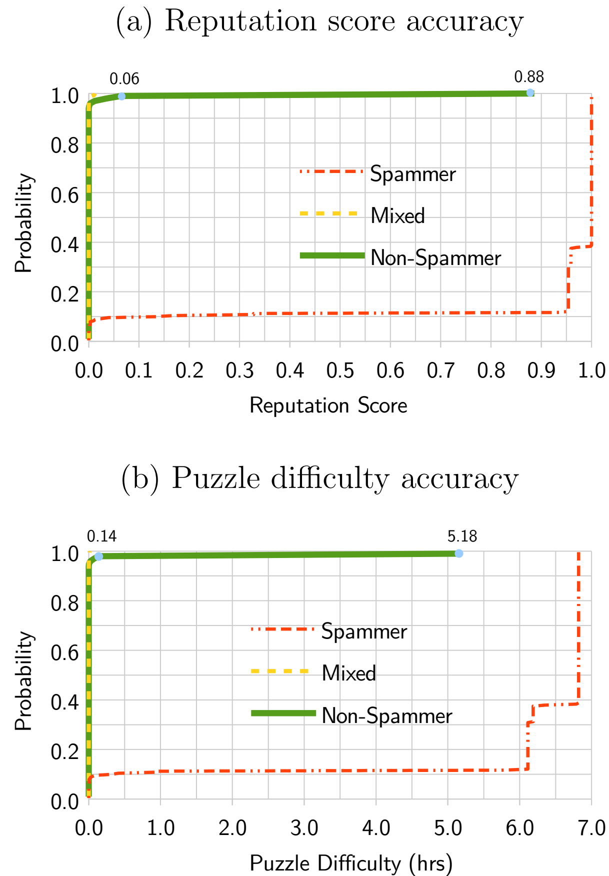


Figure 2.8: Reputation Accuracy: CDF of reputation scores and puzzle difficulties assigned to spammers, non-spammers, and mixed users (those that sent at least 1 spam and 1 ham)

not solve a puzzle at all!

Although reputation scores have accurately identified spammers from non-spammers, MetaCAPTCHA's success depends on issuing harder puzzles to malicious users. This requires evaluating the function that converts reputation score to puzzle difficulty (see Section 2.4.2). 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

المنارة للاستشارات
www.manaraa.com

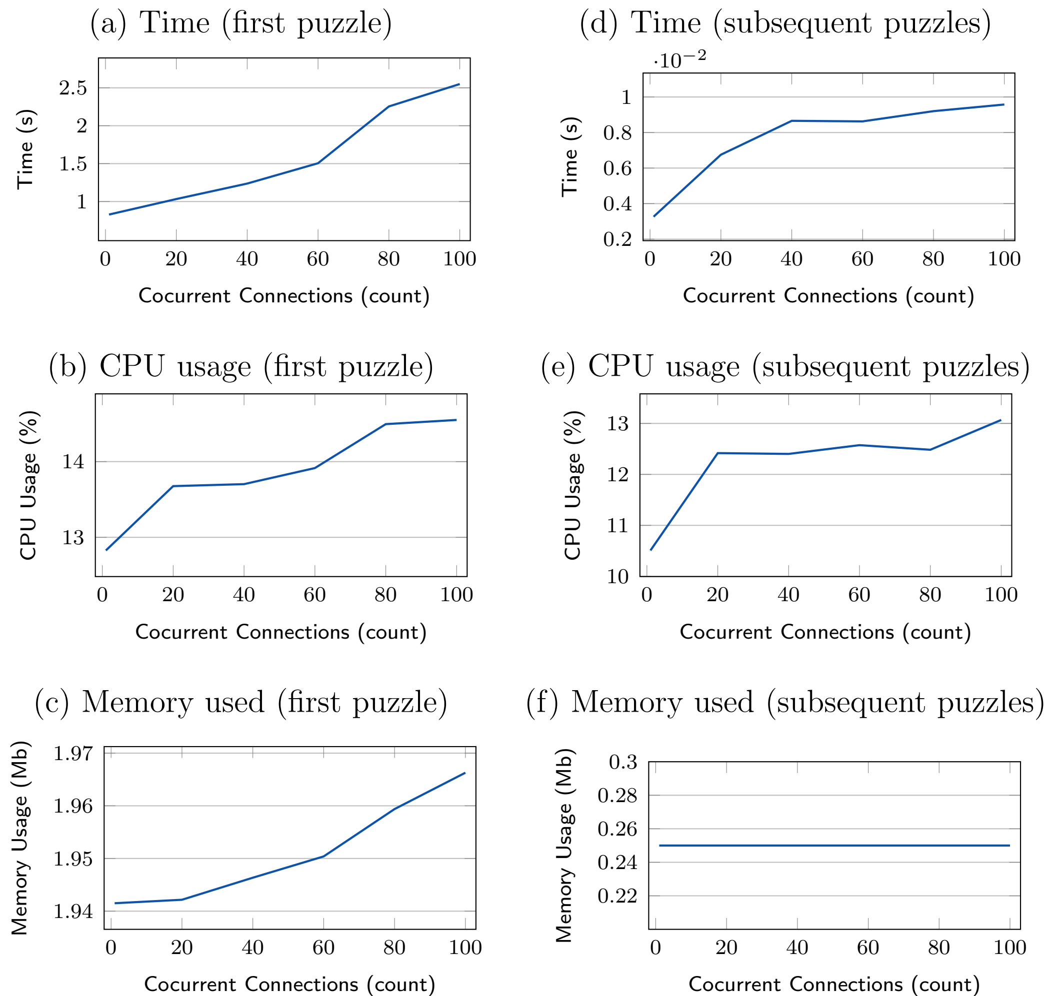


Figure 2.9: MetaCAPTCHA performance

plotted a CDF, shown in Figure 2.8 *(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*.

www.manaraa.com

2.6.4 Performance

We evaluated the performance of MetaCAPTCHA in terms of CPU usage, memory consumption, and time spent in authenticating and issuing puzzles to users. We used Apache JMeter [93], a Java application that load-tests servers, to generate 1 - 100 concurrent puzzle requests incrementing each time by 20 to MetaCAPTCHA. Each test run (e.g. 1, 20, 40, etc.) was repeated over a 100 times and the average measurement (e.g. CPU usage) was plotted against the number of concurrent connections. The 95% confidence interval for the mean of each measurement was also calculated. However, the intervals might be too small to spot in the graphs. Figures 2.9 *(a)* - *(c)* show the amount of time, CPU, and memory consumed to authenticate the user, determine reputation score and puzzle difficulty, and generate the first puzzle. Although the time consumed was not prohibitive, we were interested in determining where most of it was used. A more detailed analysis, shown in Figure 2.10, revealed that a majority of the time was spent in the reputation service when querying other remote services like Akismet [17], and blacklists like Spamhaus [95]. In the future, we hope to eliminate remote queries and mirror the applicable blacklists to significantly reduce the time required for issuing the first puzzle. Note that after the first puzzle, issuing subsequent ones only requires generating a new random puzzle (without the need for computing puzzle difficulty, or authenticating the user). Figures 2.9 *(d)* - *(f)* depict the resources consumed while issuing subsequent puzzles.

2.7 SECURITY ANALYSIS

MetaCAPTCHA's goal is to address threats from automated adversaries like spam bots. The following paragraphs discuss those threats and how MetaCAPTCHA defends against them.

www.manaraa.com

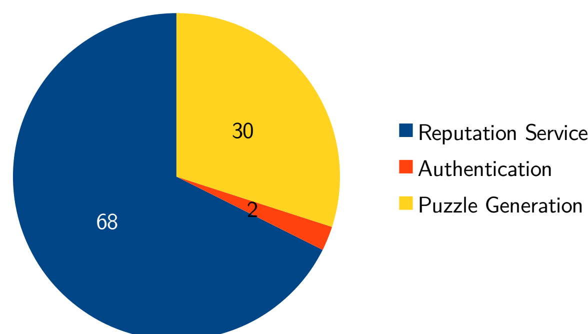


Figure 2.10: Breakdown (%) of the time spent in issuing the first puzzle. Notice that 68% of the time is spent in the reputation service due to all the remote queries that happen there.

Bots may attempt to post spam in the web application. However, with Meta-CAPTCHA protections in place, those attempts will result in a puzzle-request-ticket. Thus, preventing any efforts to directly post spam.

Bots may attempt to show proof-of-work without ever doing the work. However, they will be unable to forge a proof-of-work-ticket without the web application's secret API key.

Bots could present a proof-of-work-ticket for one message, but try to post another. However, since proof-of-work-tickets contain a digest of the original message, the ticket's verification will fail when associated with a new message.

Bots could replay old proof-of-work-tickets. However, as mentioned in Section 2.3.2, clients only have a small amount of time t_{diff} to submit the proof-of-work ticket. Thus, the same ticket cannot be replayed after t_{diff} time.

Bots could find short-cut methods to solve puzzles. However, since Meta-CAPTCHA forces adversaries to solve puzzles for a pre-determined amount of time (the puzzle difficulty), solving a puzzle faster will only result in more puzzles to solve.

Bots may attempt to reuse the solutions of puzzles solved in the past. Recall,

المنارة للاستشارات
www.manaraa.com

that puzzles are randomly selected and parameterized before being issued. Thus, bots will have to store an old puzzle in hopes of finding an exact match sometime in the future. We conjecture that this probability is negligible for the types of puzzles currently supported.

MetaCAPTCHA could be the target of a DoS attack where a flood of puzzle requests causes it to create states for an unsustainable number of puzzle sessions. However, effects of such attacks can be mitigated by using puzzle outsourcing techniques [105].

MetaCAPTCHA determines a user's reputation based on information related to messages posted by that user (e.g. contents, source IP). This may be a privacy concern for those who may trust the web application with their messages, but not MetaCAPTCHA. A possible solution to this problem is to eliminate privacy-sensitive features from being used for determining reputation. The drawback, however, would be reduced reputation score accuracy. Another way, would be to empower the application to provide a local reputation score based on privacy-sensitive features. This local score could then be combined with the one determined remotely by MetaCAPTCHA to provide an accurate characterization of reputation. We hope to explore these avenues in future research.

Currently, MetaCAPTCHA issues puzzles without considering the platform it will be solved on. Thus, some clients may solve puzzles for longer than the amount of time determined by MetaCAPTCHA. One possibility is to use browser fingerprinting techniques [78] to determine the client's CPU speed and then issue puzzles accordingly. We also hope to address this direction of research in future work.

المنارة للاستشارات

www.manaraa.com

3

VOLUNTEER COMPUTING WORKLOADS AS USEFUL PROOF-OF-WORK

Spammers are continually looking to circumvent counter-measures seeking to slow them down. One approach for dealing with spam is to force the spammer's machine to solve a computational problem of varying difficulty before granting access. The idea is that suspicious or problematic requests are given more difficult problems to solve while legitimate requests are allowed through with minimal computation. Unfortunately, most systems that employ this model waste the computing resources being used, as they are directed towards solving cryptographic problems which provide no societal benefit. While systems such as reCAPTCHA and FoldIt have allowed users to contribute solutions to useful problems interactively, an analogous solution for non-interactive proof-of-work does not exist. Towards this end, this chapter describes reBOINC, a system for supporting useful proof-of-work that is integrated into a web spam throttling service. reBOINC provides a framework that enables the computational resources of spammers to be redirected towards meaningful research. To validate the efficacy of the approach, prototype implementations based on OpenCV and BOINC are described that demonstrate the ability to harvest spammer's resources for beneficial purposes.

المنارة للاستشارات
www.manaraa.com

3.1 reBOINC APPROACH

3.1.1 Challenges and Solutions

One of the challenges in leveraging BOINC is that the workloads are specifically designed to run on the operating system (OS). In the reBOINC approach, the same applications must be adapted to operate on web browsers without compromising the utility of the whole scheme. For example, reBOINC allows binary applications delivered from a BOINC project to run on a client's web browser without additional software installed on that client's machine. As BOINC is not modeled for the web environment, making it work without extensive modification is difficult. When run natively on top of a full-fledged OS, the applications have full access to existing system libraries. In contrast, web browsers typically have limited access to such OS facilities.

One possible solution is to rewrite the binary applications in current browser scripting languages such as Javascript. Unfortunately, reimplementing BOINC scientific applications in Javascript is prone to error, requires significant development cost on a per-application basis, and results in poor performance. To address this problem, reBOINC utilizes Native Client (NaCl), an open-source technology that allows secure native execution of binary code within web browsers. NaCl enables an adapted version of BOINC applications to run on web browsers with minimal modification.

The second challenge is that the concept of useful computations is generally new to the web browser. There is very little practical support to run scientific code on a modern web browser. Perhaps the closest analogous application is the use of Javascript to mine Bitcoins [2]. While Bitcoin computations are good for generating monetary value, the software itself can not be used for arbitrary scientific computations that benefit society. To address this challenge and to support large classes of useful computations on the web, reBOINC adapts the Open Source

المنارة للاستشارات
www.manaraa.com

Computer Vision Library (OpenCV) to run within web browsers. Since OpenCV is used in a wide-range of image processing and computer vision applications, adapting the library to the NaCl environment allows reBOINC to support a large number of existing BOINC applications. As a result, reBOINC enables thousands of existing, useful, CPU-intensive algorithms in the area of object detection, object recognition, statistical machine learning and more.

Another challenge is that the coarse granularity of BOINC applications is not suitable for web applications. In order for the reBOINC approach to work, it requires client browsers to download a large amount of BOINC scientific software and data. The size of this software is large and their calculations can potentially take a long time to finish. These characteristics are unwelcome in the fast, per-request nature of the web. In addition, such computations are complete non-starters for users on smartphones and tablets. With mobile access to web services rapidly growing, any counter-measure must ensure the quality of the user experience.

To meet this challenge, reBOINC uses a credit-based approach towards the use of its challenges. Rather than requiring work to be done on a per-transaction basis, reBOINC challenges are only issued on events that are rare for normal users to encounter. Specifically, challenges are typically targeted for delivery at account creation or during account recovery. Thus, after account establishment, as long as the account has not been hijacked or used in an anti-social manner, users directly access the web application unimpeded. In contrast, because spammer accounts are shut down rather quickly, spammers are required to create accounts continuously and would thus be forced to execute the computations often. In this manner, not only is the granularity problem addressed, but also the usability problem that the reBOINC proof-of-work protocol has with mobile devices.

Lastly, the most important challenge is the fact that the solutions of the useful computations are not easily verified. Useful computations in reBOINC are general-purpose, outsourced computations from the BOINC projects. These computations

المنارة للاستشارات
www.manaraa.com

are fundamentally different from traditional proof-of-work puzzles in that the solutions are not known apriori. To address this, a significant amount of work has been done on *verifiable computations*. Research has resulted in solutions such as using trusted hardware [85, 32], assuming failures are independent and replicating the computation [70, 72, 57, 28, 19], auditing [75] or attestation [81, 87], and using probabilistically checkable proofs where the computations can be verified with small probability of being wrong [66, 50, 48, 20, 21, 88, 56].

While most of these solutions are robust, they fall under one of the categories: assuming trust in the hardware or the middleware; requiring detailed knowledge of the intermediate results; or altering the software to enforce checkable proofs and cryptographic functions. However, these conditions are unsuitable for the reBOINC approach since (a) reBOINC's goal is to keep minimal modification to the client browsers as well as the BOINC projects and (b) requiring hardware trust, detailed knowledge of the software or infusing checkable proofs and cryptographic algorithms to scientific computations would incur significant developer overhead. Towards this end, reBOINC employs a variety of simple security techniques including support for metamorphism[1] and replication[2]. These techniques allow reBOINC to securely channel useful computations as proof-of-work without adding significant additional development cost.

3.1.2 System Model

The reBOINC system is constructed on top of *MetaCAPTCHA* - a currently running proof-of-work anti-spam web service built in our previous work [41, 13]. The reBOINC system, to our knowledge, is the first system built on a working spam

[1]The useful computations are issued using many kind of proof-of-work whose answers are known and unknown.

[2]Issuing the same useful computation to multiple clients in order to catch discrepancies.

www.manaraa.com

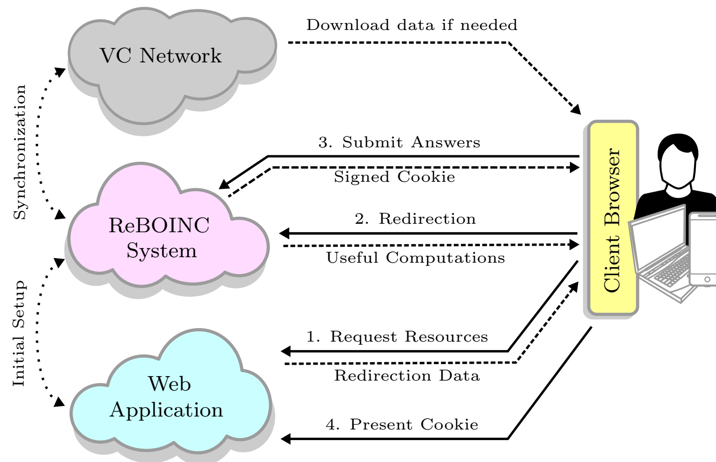


Figure 3.1: System model: The client browser has to compute useful computations in order to gain a signed cookie (the proof-of-work) from the reBOINC system. Solid arrows represent requests, dashed ones represent responses and dotted ones are periodic communications and/or one-time setup.

throttling web service, to leverage the workloads from scientific research as proof-of-work.

The reBOINC system model consists of web applications that need protection against attacks, a volunteer computing network (VC network), and clients running a modern web browser. The model forces client browsers to perform a useful computation from the VC network before releasing resources from the web applications being protected. The web applications need to perform a one-time registration with our service in order to install an anti-spam plugin. The model does not require special modification to the VC network. However, the network needs to have its binary executable ported to the NaCl platform and its workloads configured to work with the reBOINC system. The porting process is a standard procedure in supporting new platforms in most VC networks such as in BOINC. The model also requires the web browsers to support native execution of binary code. Currently, Google Chrome and Chromium are the only two browsers supporting this technology.

المنارة للاستشارات
www.manaraa.com

Figure 3.1 illustrates the high-level interactions of all the components of the reBOINC model. First a client browser requesting certain resources such as registering an account will be required to show his proof-of-work. This request will be redirected to the reBOINC system where the description of how to obtain that proof will be retrieved. At this state, instead of using traditional proof-of-work, the system describes the computations of the useful proof-of-work. Additional data may be downloaded from the VC network to compute the work according to the proof description. When the work is done, the answers are submitted to the reBOINC system and a signed cookie is created if the answers were correct. This cookie is used to release the protected resources from the web applications. Since the useful proof-of-work and answers are outsourced computations from the VC network, synchronization phases are needed periodically where the results are submitted to the network and new workloads are streamed back to the reBOINC system.

3.1.3 Threat Model

The security threats relevant to the former model mainly come from the client browsers. Although the reBOINC model is similar to the volunteer computing one in that the owners of the utilizing machines are not trusted, reBOINC faces an enormous amount of relentless spammers whereas the volunteer computing only has a fraction of cheaters. In reBOINC's security model, the client browsers are completely untrusted and assumed to be unfaithful. In contrast, the web application and the VC network are assumed to be trusted components.

The security model assumes that the communications from a client browser to a web application, and the redirecting data from a client browser to the reBOINC system are properly signed so that any tampering can be detected. In the reBOINC system, this data signing and checking is handled by the MetaCAPTCHA's APIs and web application plugins (MetaCAPTCHA is modeled around Kerberos

www.manaraa.com

[80] - an authentication granting and signing protocol). The model also assumes that the augmented anti-spam web service such as MetaCAPTCHA is functioning properly and has the ability to judge the maliciousness of requests to determine the amount of computation a client browser must perform (i.e. a reputation service).

3.2 SYSTEM COMPONENTS

The reBOINC approach relies on several essential components to operate including Native Client (NaCl), the Berkeley Open Infrastructure for Network Computing (BOINC), the Open Source Computer Vision project (OpenCV) and the Meta-CAPTCHA anti-spam web service.

3.2.1 Native Client

NaCl is a technology developed by Google to allow untrusted binaries to run safely in a sandboxed environment in the web. Experiments have shown that this technology enables C/C++ applications to run in the browsers with near native speed [107]. An NaCl application composes of a binary executable file and dependency libraries, and a manifestation file that describes the instructions to load the program for different processors. The application is compiled using the NaCl GCC toolchain and downloaded to the browser over HTTP requests.

The technology supports communication with Javascript, extending the infrastructure of the web browser. The communication is allowed through an asynchronous message posting system where each side will have a listener to interpret the *message* event on the other side. Currently, only Google Chrome and the Chromium browser support NaCl as a built-in module. The module is enabled automatically when an NaCl application is initialized through the Chrome Web Store. Otherwise, the users must manually enable it.

المنارة للاستشارات
www.manaraa.com

3.2.2 BOINC

BOINC is an open-source client-server middleware infrastructure for volunteer and grid computing [18]. To participate in volunteer computing projects using BOINC, a volunteer participant must register an account at the project webpage as well as install and log in to the software on the OS. The software will then automatically download useful computations and run them while the computer is idle.

A BOINC project consists of three main components. The first is the BOINC server which stores, schedules and distributes *workunits* (BOINC's terminology for workloads) to a volunteer's machine as well as collecting the results. A workunit is a single task with input files to be executed by the *client software* running on the volunteer's OS. The server has two CGI scripts to allow communication with its client software via HTTP. One script performs the scheduling role to distribute workunits and to receive the result, while the other handles actual result's files uploaded by the client software.

The second component is the BOINC client software which runs on the OS of the volunteer's machines. The client software authenticates with different BOINC projects before receiving any workload. After authenticating, the software fetches the scientific applications for its processor along with the workunits. These applications will be executed by the client software at a certain point on the OS. When finished, the software submits the result to the appropriate BOINC servers. This client software allows a single participant's machine to contribute to multiple volunteer computing projects.

The last component is the BOINC scientific application which is a custom program usually written in C/C++ by scientists to carry out useful computations. The application is executed with input downloaded from the BOINC server when the participant's computer is idle. In order to support multiple projects with a single client software, each project has its own configuration file set up during the first time the software authenticates with the BOINC servers.

المنارة للاستشارات
www.manaraa.com

3.2.3 OpenCV

OpenCV is an open-source, cross-platform library that facilitates real-time computer vision applications. Due to its complexity, there is currently no support to run OpenCV on web browsers. By adapting the library to the web, however, a large class of useful computations are enabled since there are a myriad applications that require some form of image processing and recognition. Adapting OpenCV to run on a web browser is challenging because OpenCV uses a synchronous, blocking I/O model whereas web browsers use an asynchronous, non-blocking one. The library also requires a file system API that supports persistent storage in order to implement its I/O operations. To overcome this challenge, reBOINC adapts the file I/O calls in OpenCV to an existing library supporting a virtual memory file system. The library allows NaCl applications which use OpenCV to run in many operating systems without user intervention.

3.2.4 MetaCAPTCHA

Many web sites use CAPTCHAs to protect themselves against automated adversaries. Previous work on metamorphic CAPTCHAs has resulted in the MetaCAPTCHA[3] web service which dynamically supports a variety of anti-spam countermeasures based on both CAPTCHAs and computational puzzles. Our implementation of reBOINC leverages MetaCAPTCHA's existing infrastructure and user-base in order to demonstrate the utility of useful computations in thwarting automated attacks.

As depicted in Figure 3.2, there are three distinct components of the MetaCAPTCHA web service: a server, a middleware for the client browser and a plugin for the web applications. The server constructs, stores and distributes proof-of-work puzzles to its clients. The server also implements a reputation service to

[3] http://www.metacaptcha.com

المنارة للاستشارات
www.manaraa.com

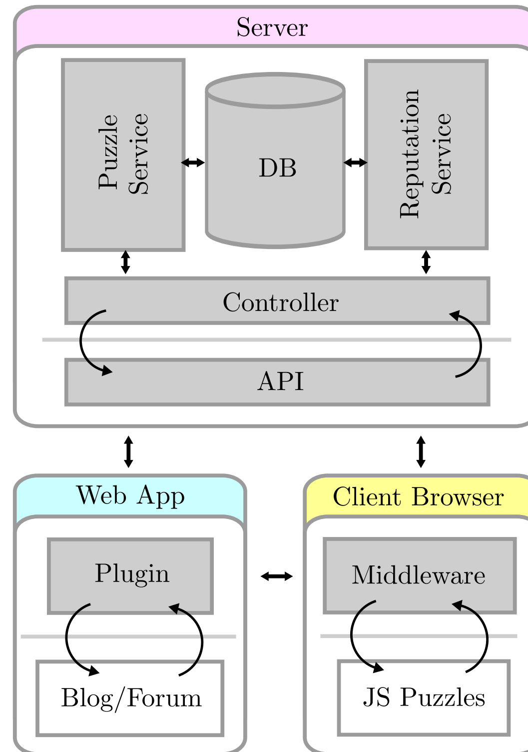


Figure 3.2: MetaCAPTCHA anti-spam web service overview

intelligently deliver more difficult puzzles to adversarial requests while allowing legitimate requests to go through.

The client is comprised of a web browser that wants to access protected resources from a participating web application. The client browser will contact the server with a signed request. This request will be routed through MetaCAPTCHA to determine a difficulty level and the proof-of-work challenges to use. The client browser then solves the different computations until the server signals completion.

The middleware that manages the execution on the client browser is a Javascript application. This application is involved whenever the client requests resources from a MetaCAPTCHA-protected web application. When invoked, the middleware dynamically instantiates the sub-puzzles the server issues the client in an iterative manner. The browser is then forced to solve each one successfully in

المنارة للاستشارات
www.manaraa.com

order to proceed.

The plugin is a module for web applications seeking to use MetaCAPTCHA to thwart spammers. Each participating web application embeds and configures this plugin to protect resources in its website. When configured, the plugin will contain an API key and a shared secret key to ensure secure communication between the web application and the MetaCAPTCHA service.

The MetaCAPTCHA server contains two services to throttle spam attacks: the *puzzle service* and the *reputation service*. The puzzle service supports the creation and execution of the proof-of-work puzzles to be carried out by the client browsers. It assembles the input data and puzzle solvers according to client's reputation and request. It then sends these back to the middleware to execute. When the results are submitted from the client, the service validates the output and authorizes the request to the protected resources assuming the answers were correct.

The reputation service is essentially a spam filter module with machine learning capability. However, instead of deciding whether a request is spam or not, the service produces an analog value to determine how likely the request is a malicious one. This analog value is then handled by the puzzle service to generate client puzzles with appropriate difficulties. This construction of MetaCAPTCHA allows flexibility in the kind of computational puzzles that are employed. For example, the service supports interactive puzzles (reCAPTCHA), non-interactive puzzles (time-lock, hash reversal) and hybrid ones which combine the other two kinds of puzzles [41].

3.3 DESIGN

3.3.1 reBOINC Proof-of-work Model

In a typical cryptographic proof-of-work model, the system is designed so that the anti-social clients are punished leaving the resources protected for legitimate

المنارة للاستشارات
www.manaraa.com

clients. This model works since cryptographic functions can be constructed to have a special property such that the solutions to the functions are hard to find but relatively easy to verify. In order for computations to be useful, the solutions to reBOINC computations from third-parties typically do not have such a property, making it difficult to detect unfaithful execution. For example, an adversary can exploit the system by finding a shortcut to skip the work being asked without computing appropriate answers.

To overcome these challenges, reBOINC leverages well-known techniques from malware and outsourced computing to strengthen the system's resilience against adversaries. These techniques include the *Metamorphic Chain of Computations* for preventing adversaries from predicting system's behaviors; the *Probabilistic Check of Duplicated Result* for ensuring that the work verification process is relatively fast; and the *Computational Banking System with Recovery* for preventing persistent abusive behavior.

Metamorphic Chain of Computations

Unlike traditional BOINC networks where users are generally volunteers with a few cheaters, the reBOINC network consists of a large population of spammers that hunt for ways to exploit the system. For this reason, finding methods to obstruct the adversaries from understanding the system is a step toward a more secure system. Inspired by the way malware changes its forms during execution, the reBOINC system randomizes the types of computations and the binary executables while the client browser is computing. An adversary who wants to launch an attack against the network will essentially have to reverse engineer the chain as it is executing using a debugger.

المنارة للاستشارات
www.manaraa.com

Probabilistic Check of Duplicated Result

In the reBOINC model, it is impossible to determine the validity of the result since the computations are from third-parties. Even when the result can be validated, there is no guarantee that verifying the result will be significantly quicker than computing them. Similar to reCAPTCHA, reBOINC addresses this challenge by precomputing a small subset of the useful computations to determine the result. These precomputed computations are randomly inserted into the chain of useful computations. When the results come back from the client, they are checked by the system to determine the validity of the whole chain. If any of the known computations have incorrect results, the entire chain is rejected and the user's IP is suspended from the system. Note that random verification checks can also be randomly performed at any point after the useful work has been submitted by the client.

Computational Banking System with Recovery

Prevention of anti-social behavior in reBOINC is a continuous process. In our system, upon performing the useful work, clients are given a computational credit that the system tracks. Credits are deducted from the client when it behaves inappropriately such as by sending spam or by submitting incorrect computations. When the client's credit reaches zero, that account is banned from further posting until the client has solved a new chain of useful computations. This way, a compromised account or an adversary who attempts to bypass the computation will not be able to abuse the system for long. In addition, we argue that this method mitigates the impact of false positives as legitimate users are asked to recover their account by contributing to research that benefits society.

المنارة للاستشارات
www.manaraa.com

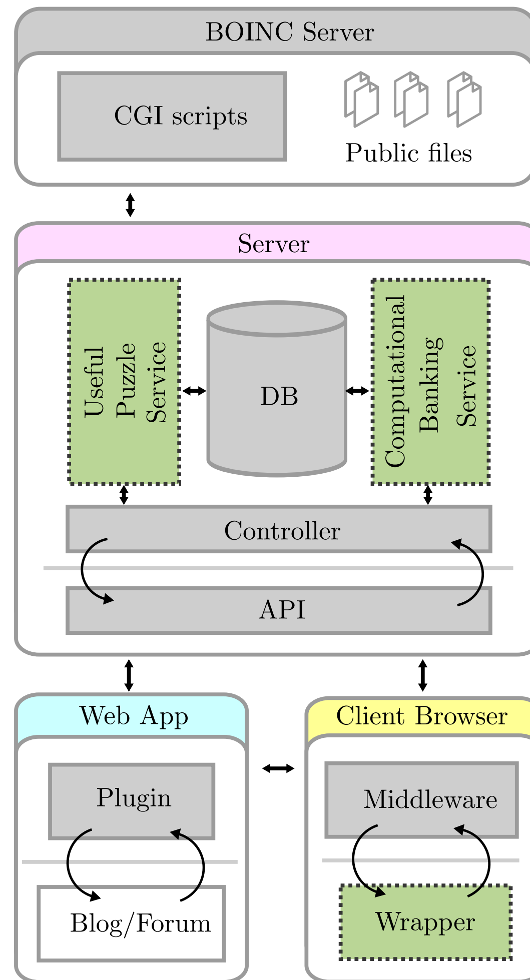


Figure 3.3: reBOINC system overview. The dotted boxes are reBOINC modifications to the MetaCAPTCHA service.

3.3.2 MetaCAPTCHA Modifications

To facilitate the creation and execution of useful computations, reBOINC augments both the puzzle service and the reputation service of MetaCAPTCHA to support useful computational capabilities. The two adapted services are the *computational banking service* and the *useful puzzle service*. These adapted services channel meaningful workloads from BOINC and convert them to proof-of-work. This adaptation retains the model and the practicality of MetaCAPTCHA. Figure

المنارة للاستشارات
www.manaraa.com

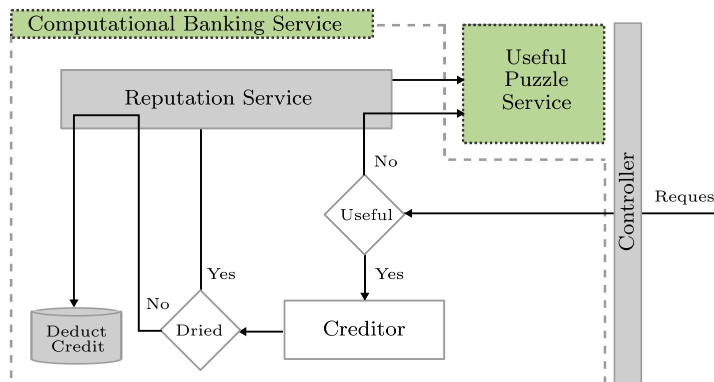


Figure 3.4: Computational Banking Service Architecture

3.3 represents the full system model of reBOINC woven into the MetaCAPTCHA infrastructure.

The useful puzzle service orchestrates the chain of useful computations for the client browsers. It dictates how many computations are enough until a registration application is complete, where to get the workload from, and what specific computation the client needs to do. By building into the existing infrastructure of MetaCAPTCHA, the service is able to randomly switch between different kinds of computations that MetaCAPTCHA supports. For example, a BOINC scientific application and a group of control functions whose results are known can be intertwined to form a chain of computations. This makes it more difficult for the adversary to predict and fake the results of useful computations.

Computational Banking Service

Figure 3.4 illustrates the architecture of the computational banking service which implements the credit-based recovery strategy. The service augments the existing reputation service of MetaCAPTCHA. Since useful computations are intended to be used at account registration, there typically isn't sufficient information for the reputation service to determine whether or not the account sign-up is fraudulent. As a result, the computational banking service can be configured to allocate a

www.manaraa.com

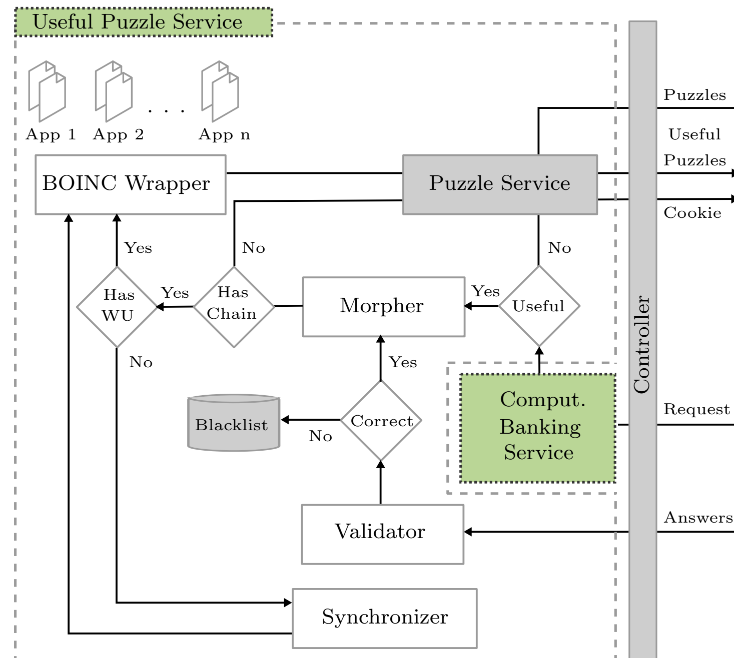


Figure 3.5: Useful Puzzle Service Architecture

fixed length computation as specified by the web application being protected. The client accounts will obtain a credit value corresponding to this computation after they have successfully computed the chain. This credit remains with the account until it is used in an anti-social manner and is only taken away when the account behaves poorly.

Useful Puzzle Service

The architecture of the useful puzzle service is depicted in Figure 3.5. The service handles two different requests, an answer to a computation and a request for new proof-of-work. The answer is checked for correctness by a *validator* which implements the probabilistic check of duplicated result strategy. The request for new proof-of-work is handled by a *morpher* that implements the metamorphic chain of computations model. The morpher decides the chain of computations to be issued to the client. The morpher also decides the workunits and the kind of useful

www.manaraa.com

computations for the client browser. Then the morpher wraps the computation as a *BOINC wrapper* computation and hands this wrapper to MetaCAPTCHA's puzzle service. This BOINC wrapper allows NaCl applications to be executed in MetaCAPTCHA's middleware.

BOINC Server Component

The BOINC server is an external but essential component of our system. It can be owned and operated by any organization over the Internet, not only by the reBOINC operators. This means that a reBOINC system can connect with as many BOINC projects as it is compatible with. The compatibility can be achieved by porting the BOINC scientific applications and configuring the projects for the Native Client platform. By having this wide support, our architecture can create a broader impact since a large number of existing BOINC projects can be executed by web browsers, making it possible to connect a large pool of useful computations with an enormous number of web clients.

3.3.3 Architecture and Communication

Since the system depends on third-parties to provide meaningful workloads, it is easy for others to support and communicate with the system. With that design goal in mind, the system requires minimal modification to the BOINC codebase. Towards this end, the system is designed so that the reBOINC servers, combined with the client browsers, act as regular BOINC client software. This combination is relatively straight-forward since NaCl on the browser and the BOINC client software support a similar programming model. The reBOINC servers effectively behave as a BOINC client, but rather than executing the workloads themselves, they manipulate the input and outsource the computational package in the NaCL version to the users' browsers. In fact, our communication protocol doesn't require any modification to BOINC source code at all. The only major code revamp is

المنارة للاستشارات
www.manaraa.com

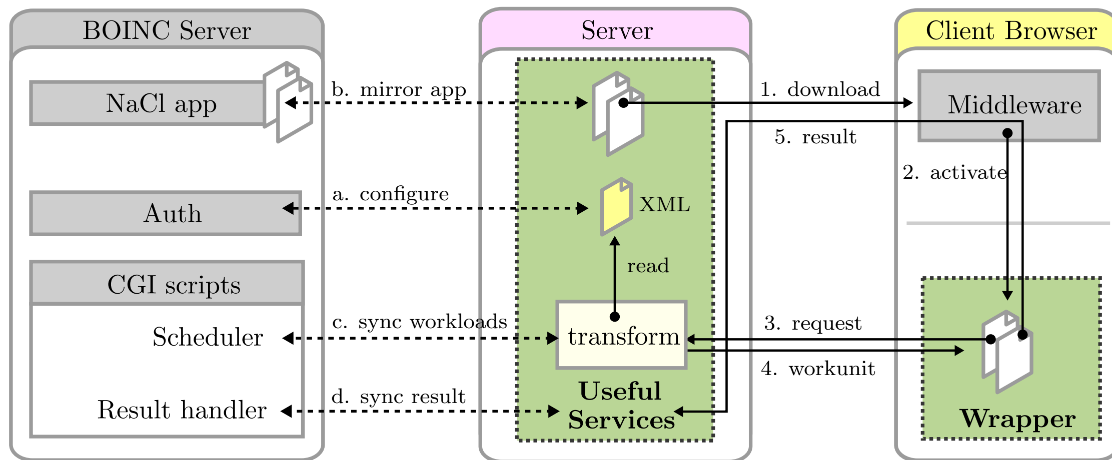


Figure 3.6: The reBOINC system. The solid lines represent run-time traffic while the dashed ones represent periodic communication.

porting the scientific application from the native system libraries to the NaCl ones. However, this is just a standard process in working with any platform the BOINC project supports.

Figure 3.6 shows how the combination mimics the communications of the BOINC client software. The communication process consists of a one-time, per BOINC project configuration stage, a real-time scheduling stage and a result handling stage.

Configuration Stage

A BOINC client needs to have a registered account and configured attributes to instruct the BOINC server to get what application and when. Similarly, the reBOINC system stores the registered accounts for many BOINC projects as well as their configured attributes and scientific applications. This configuration state (Steps *a* and *b* in Figure 3.6) will setup the XML authentication files of each BOINC project, precompute some workunits and cache the NaCl applications of BOINC projects in the system.

المنارة للاستشارات
www.manaraa.com

Scheduling Stage

The scheduling stage is effectively how the reBOINC system uses its useful services to provide meaningful workloads. Steps *1-5* in Figure 3.6 illustrate this stage. When an HTTP request is made from the client browser, the reBOINC system reads the appropriate BOINC projects from its database. The useful services transform the request into a BOINC wrapper puzzle for a specific scientific application. The system then hands the puzzle back to the client browser as a chain of computations.

Result Handling Stage

The result handling stage is controlled by the *synchronizer*. Since the results coming from the client browsers are not completely trusted, they are held in our server to validate until another client browser confirms the same result. Subsequently, the collection of confirmed results will be sent to the appropriate BOINC server and the workload cycle is completed. This stage corresponds to Steps c and d in Figure 3.6.

3.4 IMPLEMENTATION

There are three key components to the reBOINC system: the useful services, the BOINC wrapper, and a prototype NaCl application of a BOINC project that supports an environmental monitoring project in Oregon.

3.4.1 Useful Services

The useful services are written in PHP using a NoSQL database (MongoDB). The services employ a cron job to periodically fetch new workloads from various BOINC projects. These workloads are then parsed to get input file URLs and workunit information. The workunits are stored in the database in two sets, one

www.manaraa.com

with already computed answers and one without. The computed set consists of answers generated by both the server and the clients. The server generates the computed set during the configuration stage for every new BOINC project, then the set is populated with verified results submitted by the clients.

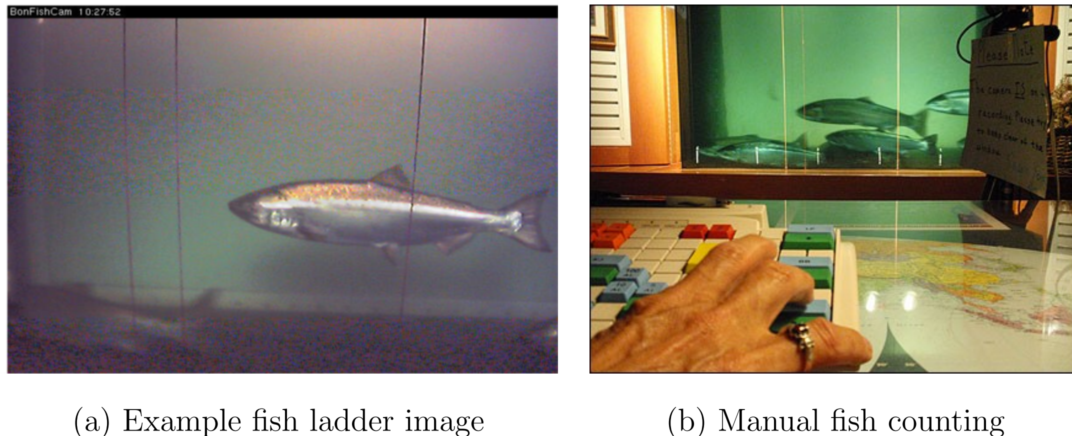
When a client browser needs to be issued a computation, the useful services check the database for the current progress of the client. If that client browser is new (e.g. has never created an account in the protected web application), the services create a new chain of computations by mixing the unanswered workunits with answered ones, then they associate the workunits with binary executables and send the client the first computation. Upon completion, the next computation in the chain is fetched and returned to the client browser until the chain is complete.

When a result is submitted to the server by a client browser, the services verify the validity of the result by checking it against the chain of computations in the database. If the result belongs to a known computation in the chain, it is compared with a stored answer. An incorrect result will get the client's IP suspended from the system whereas a correct one will have the next computation in the chain returned to the client. If the whole chain was correctly verified, the system creates a signed cookie and returns it to the middleware where it will be used to allow the client browser to access the protected web application.

3.4.2 BOINC Wrapper

The BOINC wrapper's task is to provide an easy way to execute and control NaCl scientific applications on the web browser. The wrapper is written in Javascript and delivered to a client upon being issued a useful proof-of-work computation. First, the wrapper loads an *HTML iframe* from the server into the current web page of the client. Since this iframe resides on the reBOINC server, any changes to the NaCl applications can be reflected in the iframe without intervention to the user's browsers or the web application's plugins. This iframe is also used to

المنارة للاستشارات
www.manaraa.com



(a) Example fish ladder image (b) Manual fish counting

Figure 3.7: Bonneville Dam Fish Ladder

bypass the cross-domain policy of the browser since it allows resources from the server embedded in the iframe, and the iframe to have the same origin. Next, the wrapper loads NaCl applications in the server into objects in the iframe. The iframe's script initializes the applications with workunit information and executes them. Finally, when results are returned from the applications, they are returned back to the middleware where they will be submitted for verification.

3.4.3 NaCl Application

To demonstrate the utility of the reBOINC system, we have been developed a proof-of-concept prototype that involves counting fish. To protect salmon runs in the Pacific Northwest, dams have fish ladders that salmon must climb to spawn. From an environmental policy standpoint, it is essential that an accurate measure of the number of salmon passing through is taken in order to quantify the impact the dam has on salmon populations. This measurement impacts the decision as to how much water the dam releases every day. To support this, cameras are pointed at the ladder [98] and a full-time employee is tasked with counting them for several hours a day. Figure 3.7(a) shows an example image captured from the camera pointed at the Bonneville Dam fish ladder and Figure 3.7(b) shows the set

المنارة للاستشارات
www.manaraa.com

up used to count fish. Currently, the counting is performed manually by taking samples across several hours in the day and extrapolating to determine a final daily estimate of fish passing through the ladder. This manual process is expensive to perform and, as a result, the Bonneville Power Administration has sought out researchers to automate the task. Fish counting is particularly desirable in the context of this project since it is work that is continuously generated and involves implementing an image recognition algorithm that can be easily adapted as useful proof-of-work.

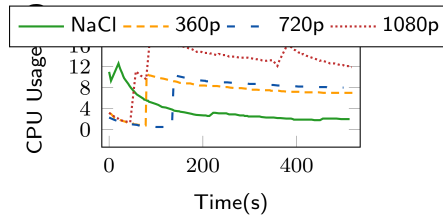
We have set up a prototype NaCl-friendly BOINC project for the fish counting application. The application is implemented using C++ and the OpenCV library. The OpenCV library itself was ported to the NaCl environment and built as a dynamic library. This library is linked with the application to allow OpenCV-enabled executables to be run on the web without any additional software installed. In addition, the application and the OpenCV library were equipped with a virtual memory file system in order to handle asynchronous I/O operations.

When invoked by the iframe's script, the browser will accept workunit information and additional data URLs from reBOINC. After that, the application downloads a series of images and a classifier cascade for object detection. Next, the images and classifier files are mapped into virtual files allowing the OpenCV library to access them. Then, the application uses OpenCV object detection functions to recognize the coordinates of the fish in each image and returns the result to the wrapper.

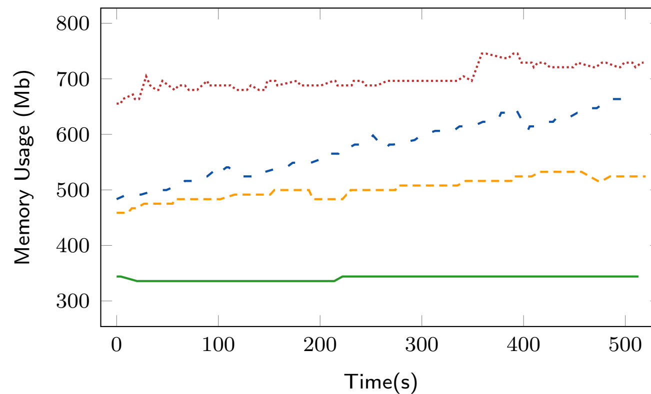
3.5 EVALUATION

To evaluate reBOINC, we examine both its client-side and the server-side performance. On the client-side, we measure the user experience while running scientific

المنارة للاستشارات
www.manaraa.com



(a) CPU usage



(b) Memory usage

Figure 3.8: Browser's resource ultilization while running NaCl fish counting application and watching Youtube videos in 360p, 720p and 1080p format.

applications and the performance of the applications running in the browser compared to native OS. We compare this performance with the performance of a similar implementation of the application in Javascript. We show that our prototype application provides significantly better performance than a Javascript implementation. On the server-side, we evaluate the server's utilization and performance in constructing useful proof-of-work, validating results and synchronizing data with the BOINC network. While there is room for improvement, our results show that the approach is feasible.

www.manaraa.com

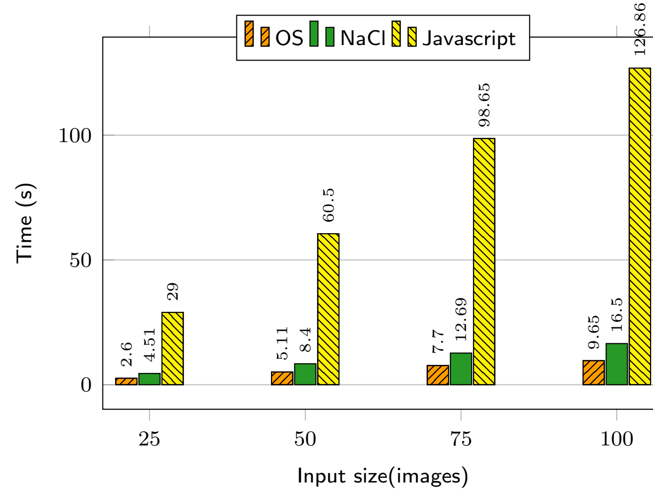


Figure 3.9: Performance of fish counting application running on OS vs. NaCl vs. Javascript

	Latency	Peak Memory	Mean Memory
New Useful PoW	$522.02 \pm 4.9 \times 10^{1}$ ms	$9.64 \pm 8.8 \times 10^{-1}$ Mb	$6.96 \pm 4.0 \times 10^{-4}$ Mb
Verify Answer	14.70 ± 2.2 ms	$5.88 \pm 3.2 \times 10^{-1}$ Mb	$5.29 \pm 1.1 \times 10^{-3}$ Mb
Stream Workloads	42.31 ± 6.3 ms	$0.79 \pm 6.0 \times 10^{-4}$ Mb	$0.75 \pm 2.8 \times 10^{-4}$ Mb
Submit Answers	43.03 ± 5.0 ms	$1.17 \pm 1.1 \times 10^{-1}$ Mb	$1.15 \pm 1.1 \times 10^{-1}$ Mb

Table 3.1: The reBOINC server performance and overheads

3.5.1 Client User Experience

We measured the resource utilization of the browser while running the fish counting application for about eight minutes on Google Chrome v31.0.1650.57 on a x86_64 testing machine. The machine was equipped with 8Gb of RAM and an Intel i7 2.4Ghz processor running on Ubuntu 12.04, Kernel v3.5.0. We compared this with the resource utilization of watching three popular Youtube videos for the same duration in 360p, 720p and 1080p video quality. The measurements were done by using the *ps* utility on Ubuntu to capture the utilization of Google Chrome processes. The tool was setup to take a sample of the browser's utilization every

المنارة للاستشارات
www.manaraa.com

one to three seconds.

Figure 3.8 (a) and (b) shows the CPU usage in percentage and memory usage in Mb of the browser on the y axis over time on the x axis respectively. The memory usage of the application was 3-6 times less than watching videos on Youtube. On the other hand, the CPU usage of the application is more similar to watching a Youtube video at 360p quality format in the first minute. However, in the long run, the average CPU usage of the application went down to $4.2 \pm 3.2\%$ of the total CPU power while the average usage of the lowest video quality was as high as $7.0 \pm 2.5\%$. This result indicates the feasibility of running scientific applications on a modern web browser as the impact of executing this particular workload is comparable to watching a video on Youtube.

3.5.2 Client NaCl Performance

We implemented an additional OS version of the fish counting application in C/C++ and OpenCV to run on the testing machine. The OS version used the same algorithms and similar code that were used in the NaCl version. Similarly, a pure Javascript version of the application using a Javascript library for object detection [4] was also implemented. The Javascript library was a port of the OpenCV Haar Feature-based Cascade Classifier algorithm [99] employed in the OS and NaCl version. To make sure that our experimental results were comparable, we used the same classifier object and image data set on all runs. The NaCl and Javascript fish counting application running on Google Chrome was benchmarked to compare with the performance of the OS one. The three versions were run 100 times to count fish on different input sizes varying from 25 to 100 images incrementing each run by 25. The time it took for each run to count all the fish in the images (e.g 25, 50, 75 etc.) was recorded.

Figure 3.9 shows the performance of the NaCl version compared to the other

www.manaraa.com

two. The result shows that the NaCl implementation of the fish counting application runs only 68% slower than the native OS version. Comparing this to the Javascript implementation which runs about 11-12 times slower than the OS version, the performance of the NaCl application is extremely fast. This is also a signficant improvement over the fastest benchmark of Javascript which runs about 200% slower than native performance [5].

3.5.3 Server Performance

The server hosting reBOINC used a 2.4 GHz Intel Xeon quadcore processor, 24Gb of memory with Red Hat Enterprise v5.10 on a 2.6.18 kernel. We set up client scripts to force the server to issue 1000 useful computations, to verify subsequent answers, and to stream workloads from and submit results to a BOINC server. The time it took for the server to handle each request type was recorded and the amount of memory consumed was measured using the XDebug extension of PHP. We benchmarked two different memory usages, a peak memory which described the maximum amount of memory reBOINC consumed at that measured time, and a mean memory usage. Table 3.1 shows average latency, peak memory and mean memory usage of four different types of requests the reBOINC server handles.

The results show that verifying answers, streaming workloads from and submitting results to the BOINC server is very fast with little impact on the server's memory. In contrast, constructing a new useful proof-of-work challenge takes longer and consumes more memory. While an average latency of 500ms per new useful proof-of-work is undesirable for a fast, real-time web application, it is more tolerable for reBOINC since such computations are infrequently issued. The request for new useful proof-of-work occurs rarely at creation and account recoveries. Thus, most of the requests handled by the server are for verifying answers which runs relatively fast.

المنارة للاستشارات
www.manaraa.com

3.6 LIMITATIONS AND FUTURE WORK

In its current state, reBOINC has several limitations. While it has been shown that a large, fixed, up-front cost to account creation can make the economics of certain spammer strategies untenable [51], in an ideal situation, one would like to give the adversarial account setups a much larger computation to perform compared to legitimate account setups [68]. This is especially the case when the adversary's resources are extensive [67]. Unfortunately, there often isn't sufficient information to tell the two apart. To address this challenge, we are looking to augment reBOINC by leveraging MetaCAPTCHA's legacy metrics such as geographic information, IP and network address blacklists to determine the per-account computation a client must perform.

In addition, while we have considered a simplistic use model, we aim to explore new modes of applying useful puzzles in which the amount of work performed determines the account capabilities given to clients. For example, a user who wants to join a web forum and immediately post an unmoderated message might be forced to compute all of the computational credit up front. However, a user who doesn't mind posts being temporarily moderated could be allowed to join with smaller up front computation. This particular user could then complete the necessary computation in the background to be granted unmoderated access. Similarly, upon initial account setup, a user could only be allowed to post messages in a "collapsed mode", while they perform the useful computations in the background. The posts would then be automatically promoted upon successful creation of sufficient computational credit.

Another issue with reBOINC is that it is difficult to ensure consistent user experience across multiple web clients. For example, a client with a fast machine will be able to complete the computations without any glitches while a slower one will take longer and might introduce lags to the browsers. The system needs to

www.manaraa.com

make sure that the performance of the issued computations it is running is not too intensive to create lags as well as not too easy allowing the adversaries to process the computations in batch. Furthermore, different scientific computations have different computational requirements. For instance, an image detection algorithm will consume mostly CPU and GPU powers of the client machine while a data mining application uses significant amount of virtual memory. To cover such cases, the system can adjust the performance of the applications by (1) benchmarking the scientific software on an average machine before publishing the work; (2) benchmarking the client hardware before receiving work from the server; (3) and monitoring client performance at real-time to reduce glitches. In addition, these benchmarks need to employ security techniques to prevent the adversaries from faking themselves as very weak machines in order to receive only easy work.

Finally, the accessibility of reBOINC is limited since only Google Chrome and Chromium browsers support native code execution. Furthermore, Chrome requires NaCL applications to be distributed through Chrome Webstore otherwise the NaCl module will not be enabled. Future directions include support for Asm.js and PNaCl. Asm.js is an open-source technology supported by both Firefox and Google Chrome to allow languages such as C/C++ and Java to be compiled into a strict subset of Javascript. This strict subset adds additional optimizations which enable Javascript to run at near native speed. PNaCl is a portable version of NaCl, this version allows NaCl applications to run on Chrome without the need to distribute them through the webstore.

www.manaraa.com

4

RELATED WORK

CAPTCHA

The term CAPTCHA is proposed by Blum et al. [25] for interactive tests that can distinguish humans and computers apart. CAPTCHAs come in many shapes and forms. The most common type is textural CAPTCHA which requires users to identify distorted and degraded texts [101, 82, 100, 30, 31, 84, 47, 33]. A different type of CAPTCHA is the image-based one where users are required to identify content or characteristics of an image such as orientation [53, 54], flipped vs. non-flipped [24], dog vs. cat [14], labels of images [36], face recognition [73]. Audio-based is another kind of CAPTCHA that usually requires users to identify words in a noisy environment [29, 89], this CAPTCHA is especially helpful for the visually impaired people. However, CAPTCHAs are not always fun to solve, so systems like Mollom [74] selectively issue them to only those users that appear to be posting spam.

Proof-of-work

Dwork and Naor first introduced the concept of a pricing function (also known as proof-of-work or client puzzle) to combat email spam. A prominent proof-of-work system that discourage spam includes Hashcash, a system that requires senders to attach "postage" to e-mail [23]. 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. [111]. However, unlike

المنارة للاستشارات
www.manaraa.com

Hashcash, their system based puzzle difficulty on the "spamminess" of the message. Kaiser and Feng [62], Feng and Kaiser [46], Kaiser and Feng [63] proposed kaPoW, a reputation-based proof-of-work system to discourage spam in webmail and ticket selling web applications. There have also been proposals to put proof-of-work to good use. MetaCAPTCHA incorporates the features of above proof-of-work systems while augmenting them with a generic puzzle issuing mechanism and a comprehensive reputation service.

Useful Proof-of-work

The idea of useful proof-of-work was first proposed by Jakobsson and Juels [60]. The idea is that a computational effort invested in solving the proof can be reused for a meaningful purpose. Diament et al. [39], Zhang et al. [110] revised the useful proof-of-work construction to offload the computations of the servers to the client side as countermeasures against denial-of-service attacks. These systems require the clients to execute specific workloads of the server in order to gain access to the service. This robust construction prevents the server resources from depleting and at the same time benefits the system with additional computing power. Similarly, da Costa Cordeiro et al. [35] developed a model for using useful puzzles for identity management and prevention of Sybil attacks in distributed systems.

Bitcoin can be regarded as useful proof-of-work system even though the system heavily depends on cryptographic functions. In a Bitcoin system, each user contributes to computations whose result can be reused as digital currency [79]. The system allows the users to find a nonce that when hashed with the network's block header will satisfy a particular condition. When found, the result will be populated through the network to prevent double-spending and therefore obtain their monetary value. In similar fashion, reCAPTCHA [101] is a spam prevention service where the result of solving the CAPTCHAs helps digitizing books. The service uses text-based interactive puzzles where a user has to correctly enter the

المنارة للاستشارات
www.manaraa.com

characters presented in an image. By combining a known and unknown text in the images, the puzzle uses the result submitted by the users to interpret scanned manuscripts.

Outsourced Computations on the Web Browsers

Outsourced computation is not a new topic; however, its advancement on browsers has been very limited due to the fast-paced, per-request characteristic of the web. Karame et al. [65] offered a unique model to this problem by executing outsourced computations on the browser as micropayments for web applications. This model is web-friendly since it enables the users to read online newspapers for free while their browsers are utilized to compute useful computations. In a different way, researchers at the University of Washington are developing a system called Lind - a secure, lightweight cloud computing environment using NaCl in the web browser [6]. The system could allow the server to issue arbitrary computations to its clients using the web browser as a computing platform. Another system called Gridbee developed by researchers at BME IK has similar capabilities [12]. The system is an adapted BOINC framework which supports scientific computations on the web browsers using Javascript and NaCl applications.

المنارة للاستشارات

www.manaraa.com

5

CONCLUSION

Spam is a perpetual problem that threatens the quality of content on the Internet. However, existing spam fighting systems are not generic and practical enough to be deployed on the web, plus their pricing functions are moderately fixed. As such, these systems are often too simple allowing the spammers to study and subvert them. Furthermore, current methods of proof-of-work spam throttling systems are effective but are not sustainable since the computational power of the users is wasted. While there is a large community that supports sustainable and meaningful use of computational power such as volunteer computing, there is not a system that leverages these characteristics to fight spam.

MetaCAPTHCA and reBOINC address these problems. MetaCAPTCHA seamlessly 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 “harder“ puzzles than honest users. In addition, the puzzles issued by MetaCAPTCHA are completely nondeterministic making it difficult to be subverted. We evaluated MetaCAPTCHA in the context of a web application and showed that 95% of honest users hardly noticed MetaCAPTCHA’s presence, whereas the remaining 5% were required to solve very “easy” puzzles before accessing the application’s services. reBOINC tackles the problem of wasting the computational resources of clients in the proof-of-work model. The reBOINC system also demonstrates that in practice, the proof-of-work model can employ various kind of useful computations whose results help to solve the world’s demanding

المنارة للاستشارات
www.manaraa.com

problems. Additionally, our evaluations have demonstrated that executing intensive scientific calculations as proof-of-work in web browsers is feasible with modest impact on the user-experience. To support this, reBOINC sets up a flexible structure so that new volunteer computing projects can plug into the network with minimal modification to both parties including large applications that depend on complex libraries such as OpenCV.

At a high-level, reBOINC and MetaCAPTCHA are a practical attempt to build an infrastructure that connects the rapidly growing demand for computing power needed by scientific research with the vast spammers supply of computing resources. As a result, the infrastructure can be part of a win-win scenario where either the scientific communities have access to greater resources to solve demanding problems or the web applications can reduce their spam load.

المنارة للاستشارات
www.manaraa.com

References

[1] Asm.js.
http://asmjs.org/.

[2] Hash me if you can.
https://code.google.com/p/hamiyoca/.

[3] Mozzila firefox.
http://www.mozilla.org/en-US/firefox.

[4] Haar.js.
https://github.com/foo123/HAAR.js/.

[5] Firefox nightly now includes odinmonkey, brings javascript closer to running at native speeds.
http://techcrunch.com/2013/03/21/firefox-nightly-now-includes-odinmonkey-brings-javascript-performance-closer-to-running-at-native-speeds/.

[6] Lind: Secure lightweight cloud computing.
https://seattle.cs.washington.edu/wiki/Lind".

[7] Naclmount.
https://github.com/austinbenson/nacl-mounts.

[8] Opencv.
http://opencv.willowgarage.com/.

المنارة للاستشارات
www.manaraa.com

[9] Pnacl - the chromium projects.
http://www.chromium.org/nativeclient/pnacl.

[10] Xdebug.
http://xdebug.org/.

[11] Boinc.
http://boinc.berkeley.edu/, 2005.

[12] Gridbee web computing framework.
http://webcomputing.iit.bme.hu/, 2012.

[13] Metacaptcha: Web-based client puzzles.
http://www.metacaptcha.com/, 2012.

[14] Microsoft asirra.
urlhttp://research.microsoft.com/en-us/um/redmond/projects/asirra/, 2014.

[15] I. 10gen. Mongodb. http://www.mongodb.org/.

[16] M. Abadi, M. Burrows, M. Manasse, and T. Wobber. Moderately hard, memory-bound functions. *ACM Transactions on Internet Technology (TOIT)*, 5(2):299–327, 2005. Presents a set of memory-bound functions that can be used as client puzzles.

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

[18] D. P. Anderson. Boinc: A system for public-resource computing and storage. In *Grid Computing, 2004. Proceedings. Fifth IEEE/ACM International Workshop on*, pages 4–10. IEEE, 2004.

المنارة للاستشارات
www.manaraa.com

[19] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer. Seti@ home: an experiment in public-resource computing. *Communications of the ACM*, 45(11):56–61, 2002.

[20] S. Arora and S. Safra. Probabilistic checking of proofs: A new characterization of np. *Journal of the ACM (JACM)*, 45(1):70–122, 1998.

[21] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and the hardness of approximation problems. *Journal of the ACM (JACM)*, 45(3):501–555, 1998.

[22] A. Back. Hashcash faq. http://www.hashcash.org/faq/.

[23] A. Back et al. Hashcash-a denial of service counter-measure, 2002.

[24] M. T. Banday and N. A. Shah. Image flip captcha. *ISeCure*, 1(2), 2009.

[25] M. Blum, L. Von Ahn, J. Langford, and N. Hopper. The captcha project (completely automatic public turing test to tell computers and humans apart). *School of Computer Science, Carnegie-Mellon University, http://www. captcha. net*, 2000.

[26] E. Bursztein, M. Martin, and J. Mitchell. Text-based captcha strengths and weaknesses. In *Proceedings of the 18th ACM conference on Computer and communications security*, pages 125–138. ACM, 2011.

[27] Caroline Ghiossi. Explaining Facebook Spam Prevention Systems. https://blog.facebook.com/blog.php?post=403200567130, June 2010.

[28] M. Castro and B. Liskov. Practical byzantine fault tolerance and proactive recovery. *ACM Transactions on Computer Systems (TOCS)*, 20(4):398–461, 2002.

المنارة للاستشارات

www.manaraa.com

[29] T.-Y. Chan. Using a test-to-speech synthesizer to generate a reverse turing test. In *Tools with Artificial Intelligence, 2003. Proceedings. 15th IEEE International Conference on*, pages 226–232. IEEE, 2003.

[30] M. Chew and H. S. Baird. Baffletext: A human interactive proof. In *Electronic Imaging 2003*, pages 305–316. International Society for Optics and Photonics, 2003.

[31] M. Chew and J. D. Tygar. *Image recognition captchas*. Springer, 2004.

[32] A. Chiesa and E. Tromer. Proof-carrying data and hearsay arguments from signature cards. In *ICS*, volume 10, pages 310–331, 2010.

[33] R. Chow, P. Golle, M. Jakobsson, L. Wang, and X. Wang. Making captchas clickable. In *Proceedings of the 9th workshop on Mobile computing systems and applications*, pages 91–94. ACM, 2008.

[34] F. Coelho. Exponential memory-bound functions for proof of work protocols. Technical report, Research Report A-370, CRI, École des mines de Paris, 2005.

[35] W. L. da Costa Cordeiro, F. R. Santos, M. P. Barcellos, and L. P. Gaspary. Make it green and useful: Reshaping puzzles for identity management in large-scale distributed systems. In *Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on*, pages 387–394. IEEE, 2013.

[36] R. Datta, J. Li, and J. Z. Wang. Imagination: a robust image-based captcha generation system. In *Proceedings of the 13th annual ACM international conference on Multimedia*, pages 331–334. ACM, 2005.

[37] D. Dean and A. Stubblefield. Using client puzzles to protect tls. In *Proceedings of the 10th USENIX Security Symposium*, pages 1–8, 2001.

المنارة للاستشارات
www.manaraa.com

[38] P. A. Devijver and J. Kittler. *Pattern recognition: A statistical approach.* Prentice/Hall International Englewood Cliffs, NJ, 1982.

[39] T. Diament, H. K. Lee, A. D. Keromytis, and M. Yung. The dual receiver cryptosystem and its applications. In *Proceedings of the 11th ACM conference on Computer and communications security*, pages 330–343. ACM, 2004.

[40] S. Doshi, F. Monrose, and A. D. Rubin. Efficient memory bound puzzles using pattern databases. In *Applied Cryptography and Network Security*, pages 98–113. Springer, 2006.

[41] A. Dua, T. Bui, T. Le, N. Huynh, and W.-C. Feng. Metacaptcha: A metamorphic throttling service for the web. Currently submitted to WWW'14, May 2013.

[42] C. Dwork and M. Naor. Pricing via processing or combatting junk mail. In *Advances in Cryptology-CRYPTO'92*, pages 139–147. Springer, 1993.

[43] C. Dwork, A. Goldberg, and M. Naor. On memory-bound functions for fighting spam. *Advances in Cryptology-Crypto 2003*, pages 426–444, 2003.

[44] W. Feng, E. Kaiser, and A. Luu. Design and implementation of network puzzles. In *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, volume 4, pages 2372–2382. IEEE, 2005.

[45] W.-c. Feng and E. Kaiser. The case for public work. In *IEEE Global Internet Symposium, 2007*, pages 43–48. IEEE, 2007.

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

المنارة للاستشارات
www.manaraa.com

[47] R. Ferzli, R. Bazzi, and L. J. Karam. A captcha based on the human visual systems masking characteristics. In *Multimedia and Expo, 2006 IEEE International Conference on*, pages 517–520. IEEE, 2006.

[48] R. Gennaro, C. Gentry, and B. Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *Advances in Cryptology–CRYPTO 2010*, pages 465–482. Springer, 2010.

[49] 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.

[50] S. Goldwasser, Y. T. Kalai, and G. N. Rothblum. Delegating computation: interactive proofs for muggles. In *Proceedings of the 40th annual ACM symposium on Theory of computing*, pages 113–122. ACM, 2008.

[51] J. T. Goodman and R. Rounthwaite. Stopping outgoing spam. In *Proceedings of the 5th ACM Conference on Electronic Commerce*, EC '04, pages 30–39, New York, NY, USA, 2004. ACM. ISBN 1-58113-771-0. doi: 10.1145/988772.988779.

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

[53] 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*, WWW '09, pages 841–850, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-487-4. doi: 10.1145/1526709.1526822. URL http://doi.acm.org/10.1145/1526709.1526822.

المنارة للاستشارات

www.manaraa.com

[54] 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*, pages 841–850. ACM, 2009.

[55] 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. ISBN 978-1-4503-0245-6. doi: http://doi.acm.org/10.1145/1866307.1866311. URL http://doi.acm.org/10.1145/1866307.1866311.

[56] J. Groth. Short pairing-based non-interactive zero-knowledge arguments. In *Advances in Cryptology-ASIACRYPT 2010*, pages 321–340. Springer, 2010.

[57] A. Haeberlen, P. Kouznetsov, and P. Druschel. Peerreview: Practical accountability for distributed systems. *ACM SIGOPS Operating Systems Review*, 41(6):175–188, 2007.

[58] Heather Arthur. Face detection for cats in javascript. https://github.com/harthur/kittydar.

[59] P. Heymann, G. Koutrika, and H. Garcia-Molina. Fighting spam on social web sites: A survey of approaches and future challenges. *Internet Computing, IEEE*, 11(6):36–45, 2007.

[60] M. Jakobsson and A. Juels. Proofs of work and bread pudding protocols. In *Secure Information Networks*, pages 258–272. Springer, 1999.

[61] A. Juels and J. Brainard. Client puzzles: A cryptographic countermeasure against connection depletion attacks. NDSS, 1999.

[62] 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.

المنارة للاستشارات

www.manaraa.com

[63] E. Kaiser and W.-c. 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.

[64] C. Kanich, C. Kreibich, K. Levchenko, B. Enright, G. M. 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*, CCS '08, pages 3–14, New York, NY, USA, 2008. ACM. ISBN 978-1-59593-810-7. doi: 10.1145/1455770.1455774. URL http://doi.acm.org/10.1145/1455770.1455774.

[65] G. O. Karame, A. Francillon, and S. Čapkun. Pay as you browse: microcomputations as micropayments in web-based services. In *Proceedings of the 20th international conference on World wide web*, pages 307–316. ACM, 2011.

[66] J. Kilian. A note on efficient zero-knowledge proofs and arguments. In *Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*, pages 723–732. ACM, 1992.

[67] B. Laurie and R. Clayton. Proof-of-work proves not to work. In *Workshop on Economics and Information Security*, volume 2004, 2004.

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

[69] Machine Learning Group, University of Waikato. Weka 3 – data mining with open source machine learning software in java. http://www.cs.waikato.ac.nz/ml/weka/.

[70] D. Malkhi and M. Reiter. Byzantine quorum systems. *Distributed Computing*, 11(4):203–213, 1998.

المنارة للاستشارات
www.manaraa.com

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

[72] N. Michalakis, R. Soulé, and R. Grimm. Ensuring content integrity for untrusted peer-to-peer content distribution networks. In *Proceedings of the 4th USENIX conference on Networked systems design & implementation*, pages 11–11. USENIX Association, 2007.

[73] D. Misra and K. Gaj. Face recognition captchas. In *Telecommunications, 2006. AICT-ICIW'06. International Conference on Internet and Web Applications and Services/Advanced International Conference on*, pages 122–122. IEEE, 2006.

[74] Mollom. How mollom works — mollom. http://mollom.com/how-mollom-works.

[75] F. Monrose, P. Wyckoff, and A. Rubin. Distributed execution with remote audit. In *Proceedings of the Network and Distributed Systems Security Symposium*, 1999.

[76] 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*, volume 10, 2010.

[77] 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*, pages 14–14. USENIX Association, 2011.

المنارة للاستشارات
www.manaraa.com

[78] K. Mowery, D. Bogenreif, S. Yilek, and H. Shacham. Fingerprinting information in javascript implementations. In *Proceedings of Web*, volume 2, 2011.

[79] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Consulted*, 1: 2012, 2008.

[80] B. C. Neuman and T. Ts'o. Kerberos: An authentication service for computer networks. *Communications Magazine, IEEE*, 32(9):33–38, 1994.

[81] B. Parno, J. M. McCune, and A. Perrig. *Bootstrapping trust in modern computers*, volume 10. Springer, 2011.

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

[83] R. L. Rivest, A. Shamir, and D. A. Wagner. Time-lock puzzles and timed-release crypto. 1996.

[84] A. Rusu and V. Govindaraju. Handwritten captcha: Using the difference in the abilities of humans and machines in reading handwritten words. In *Frontiers in Handwriting Recognition, 2004. IWFHR-9 2004. Ninth International Workshop on*, pages 226–231. IEEE, 2004.

[85] A.-R. Sadeghi, T. Schneider, and M. Winandy. Token-based cloud computing. In *Trust and Trustworthy Computing*, pages 417–429. Springer, 2010.

[86] M. Schwartz. Facebook spam conversion rate hits 47%. http://www.informationweek.com/security/client/facebook-spam-conversion-rate-hits-47/226900190, 2010.

[87] A. Seshadri, M. Luk, E. Shi, A. Perrig, L. van Doorn, and P. Khosla. Pioneer: verifying code integrity and enforcing untampered code execution on legacy

المنارة للاستشارات
www.manaraa.com

systems. In *ACM SIGOPS Operating Systems Review*, volume 39, pages 1–16. ACM, 2005.

[88] S. Setty, B. Braun, V. Vu, A. J. Blumberg, B. Parno, and M. Walfish. Resolving the conflict between generality and plausibility in verified computation. In *Proceedings of the 8th ACM European Conference on Computer Systems*, pages 71–84. ACM, 2013.

[89] Y. Soupionis and D. Gritzalis. Audio captcha: Existing solutions assessment and a new implementation for voip telephony. *Computers & Security*, 29(5): 603–618, 2010.

[90] SpamAssassin. The apache spamassassin project. http://spamassassin.apache.org/.

[91] J. G. Steiner, C. Neuman, and J. I. Schiller. Kerberos: An authentication service for open network systems. In *USENIX conference proceedings*, volume 191, page 202, 1988.

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

[93] The Apache Software Foundation. Apache jmeter. http://jmeter.apache.org/.

[94] The Economist. Spreading the Load. http://www.economist.com/node/10202635, Dec 2007.

[95] The Spamhaus Project. About the Spamhaus Project. http://www.spamhaus.org/organization/index.lasso.

[96] Twitter Help Center. How to Report Spam on Twitter. http://support.twitter.com/articles/64986-how-to-report-spam-on-twitter.

المنارة للاستشارات
www.manaraa.com

[97] US Army Corps of Engineers. Bonneville lock and dam. http://www.nwp.usace.army.mil/Locations/ColumbiaRiver/Bonneville.aspx, .

[98] US Army Corps of Engineers. Fish data. http://www.nwp.usace.army.mil/Missions/Environment/Fishdata.aspx, .

[99] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–511. IEEE, 2001.

[100] L. Von Ahn, M. Blum, and J. Langford. Telling humans and computers apart automatically. *Communications of the ACM*, 47(2):56–60, 2004.

[101] L. Von Ahn, B. Maurer, C. McMillen, D. Abraham, and M. Blum. recaptcha: Human-based character recognition via web security measures. *Science*, 321 (5895):1465–1468, 2008.

[102] w3schools.com. Html5 web workers. http://www.w3schools.com/html/html5_webworkers.asp.

[103] X. Wang and M. Reiter. Defending against denial-of-service attacks with puzzle auctions. In *Security and Privacy, 2003. Proceedings. 2003 Symposium on*, pages 78–92. IEEE, 2003.

[104] X. Wang and M. K. Reiter. Mitigating bandwidth-exhaustion attacks using congestion puzzles. In *Proceedings of the 11th ACM conference on Computer and communications security*, pages 257–267. ACM, 2004.

[105] B. Waters, A. Juels, J. A. Halderman, and E. W. Felten. New client puzzle outsourcing techniques for dos resistance. In *Proceedings of the 11th*

المنارة للاستشارات
www.manaraa.com

ACM conference on Computer and communications security, CCS '04, pages 246–256, New York, NY, USA, 2004. ACM. ISBN 1-58113-961-6. doi: http://doi.acm.org/10.1145/1030083.1030117. URL http://doi.acm.org/10.1145/1030083.1030117.

[106] 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.

[107] B. Yee, D. Sehr, G. Dardyk, J. B. Chen, R. Muth, T. Ormandy, S. Okasaka, N. Narula, and N. Fullagar. Native client: A sandbox for portable, untrusted x86 native code. In *Security and Privacy, 2009 30th IEEE Symposium on*, pages 79–93. IEEE, 2009.

[108] W. Yerazunis. The spam-filtering accuracy plateau at 99.9 percent accuracy and how to get past it. In *Proc. MIT Spam Conference 2004*, 2004.

[109] C. Zara. Facebook spam and youtube spam rampant, says 2013 social media spam report. http://www.ibtimes.com/facebook-spam-youtube-spam-rampant-says-2013-social-media-spam-report-1414506, October 2013.

[110] R. Zhang, G. Hanaoka, and H. Imai. A generic construction of useful client puzzles. In *Proceedings of the 4th International Symposium on Information, Computer, and Communications Security*, pages 70–79. ACM, 2009.

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

المنارة للاستشارات
www.manaraa.com