USING NOVEL IMAGE-BASED INTERACTIONAL PROOFS AND SOURCE RANDOMIZATION FOR PREVENTION OF WEB BOTS

A Thesis

by

SHARDUL VIKRAM

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

December 2011

Major Subject: Computer Science

USING NOVEL IMAGE-BASED INTERACTIONAL PROOFS AND SOURCE RANDOMIZATION FOR PREVENTION OF WEB BOTS

A Thesis

by

SHARDUL VIKRAM

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Approved by:

Chair of Committee,	Guofei Gu
Committee Members,	Ricardo Bettati
	Deepa Kundur
Head of Department,	Duncan M. (Hank) Walker

December 2011

Major Subject: Computer Science

ABSTRACT

Using Novel Image-based Interactional Proofs and Source Randomization for Prevention of Web Bots. (December 2011)

Shardul Vikram, B.Tech, National Institute of Technology Silchar

Chair of Advisory Committee: Dr. Guofei Gu

This work presents our efforts on preventing the web bots to illegitimately access web resources. As the first technique, we present SEMAGE (*SE*mantically *MA*tching ima*GE*s), a new image-based CAPTCHA that capitalizes on the human ability to define and comprehend image content and to establish *semantic relationships* between them. As the second technique, we present NOID - a "*NO*n-*I*ntrusive Web Bot *D*efense system" that aims at creating a three tiered defence system against web automation programs or web bots. NOID is a server side technique and prevents the web bots from accessing web resources by inherently hiding the HTML elements of interest by randomization and obfuscation in the HTML responses.

A SEMAGE challenge asks a user to select *semantically related* images from a given image set. SEMAGE has a two-factor design where in order to pass a challenge the user is required to figure out the content of each image and then understand and identify semantic relationship between a subset of them. Most of the current state-of-the-art image-based systems like Assira only require the user to solve the first level, i.e., image recognition. Utilizing the semantic correlation between images to create more secure and user-friendly challenges makes SEMAGE novel. SEMAGE does not suffer from limitations of traditional image-based approaches such as lacking customization and adaptability. SEMAGE unlike the current Text based systems is also very user friendly with a high fun factor. We conduct a first of its kind large-scale user study involving 174 users to gauge and compare accuracy and usability of SEMAGE with existing state-of-the-art CAPTCHA systems like reCAPTCHA (text-based) and Asirra (image-based). The user study further reinstates our points and shows that users achieve high accuracy using our system and consider our system to be fun and easy.

We also design a novel server-side and *non-intrusive* web bot defense system, NOID, to prevent web bots from accessing web resources by inherently hiding and randomizing HTML elements. Specifically, to prevent web bots uniquely identifying HTML elements for later automation, NOID randomizes name/id parameter values of essential HTML elements such as “input textbox”, “textarea” and “submit button” in each HTTP form page. In addition, to prevent powerful web bots from identifying special user-action HTML elements by analyzing the content of their accompanied “label text” HTML tags, we enhance NOID by adding a component, *Label Concealer*, which hides label indicators by replacing “label text” HTML tags with randomized images. To further prevent more powerful web bots identifying HTML elements by recognizing their relative positions or surrounding elements in the web pages, we enhance NOID by adding another component, *Element Trapper*, which obfuscates important HTML elements’ surroundings by adding decoy elements without compromising usability.

We evaluate NOID against five powerful state-of-the-art web bots including XRumer, SENuke, Magic Submitter, Comment Blaster, and UWCS on several popular open source web platforms including phpBB, Simple Machine Forums (SMF), and Wordpress. According to our evaluation, NOID can prevent all these web bots automatically sending spam on these web platforms with reasonable overhead.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

New web applications and services emerge everyday in all areas of life. More people are getting used to having online services, such as email services, forums and specialized interest groups. The web experience of a typical user consists of searching, browsing, social networking, web games etc. In all of these activities at some point involve information submission to the server. This happens invariably by clicking on a HTML element (normally a link or a button) to achieve the final effect. Each HTML element in a web page is has a unique name and/or id and most of the server-side logic is based on it. However, this fact is utilized by the hackers to create bots which automate the web experience for financial or personal gains. We focus on bots which automate tasks on a web page initially meant for the user to carry out manually. For the service providers, one important aspect to consider is to make sure that the services and resources are allocated to the targeted customers. Malicious usage of services, such as using 'bot' to register legal accounts [1], can take up valuable resources and distribute malicious information thereafter. Thus it is important for the service provider to be able to distinguish a bot from human users, and the use of an explicit barrier like a CAPTCHA is the primary way of preventing web bots from accessing online resources. But use of a complicated text based CAPCTHA is user unfriendly and leads to decreased usability.

CAPTCHA stands for "Completely Automated Public Tests to tell Computers and Humans Apart" [1–5]. The idea is to introduce a difficult AI problem as a CAPTCHA so that either the purpose of distinguishing bots and legitimate users is served, or that an AI breakthrough is achieved [2, 3]. The robustness of CAPTCHA systems rely not on the secrecy of the database, but on the intrinsic difficulty of the problem. The difficulty of solving a CAPTCHA problem for a bot and for a human, often increase in similar curves.

This thesis follows the style of *IEEE Transactions on Dependable and Secure Computing*.

As CAPTCHA systems are rarely stand-alone and are often integrated as an auxiliary part for applications such as online registration, it is unrealistic to ask for the user's concentration for longer than a few seconds. Hence a complicated challenge requiring the humans to devote more time would make it unrealistic to be deployed on real world systems. However with the increasing advances in the field of computer vision, bots have been known to break in Text CAPTCHAs using techniques like OCR (Optical Character Recognition) technology and segmentation [6–10]. Increasing the complexity of the text based systems by introducing more noise and distortion to make the challenge difficult for bots also makes them less user friendly and less usable to normal users.

Other works in the area of preventing the web bots from using web resources like blogs/forums focus on the analysis of submitted content. Shin et. al. [11] developed a real time classification system to weed out forum and blog spam from bots. They manually inspected the bot posts on a research blog and came up with a set of features which could be used to train a classifier to detect bot postings on forums and blogs. Features used such as anatomy of the post, commenting activity may vary considerably from forum to forum and also as the bots evolve and try to mimic normal human interactions. Moreover, defining an effective feature set requires considerable human effort and cannot be used in fast deployment and detection.

Our work on a novel Image based CAPTCHA - SEMAGE aims at prevention of web bots without hampering usability. Image-based systems were proposed to increase the usability of CAPTCHA systems [12–19]. However, many current state-of-the-art image based systems such as Asirra [12] suffer from lack of flexibility and adaptability. All Assira challenges are just based upon image recognition, requiring the user to identify all cats among a series of images of cats and dogs. Specialized attacks using machine learning techniques have achieved a high rate of success against systems like Asirra, as shown by Golle [20]. Moreover the inherent choice presented to the bot is always binary - an image either a cat or a dog, making it more susceptible to template fitting attacks. We discuss more about template fitting attack in Section 5.2. We propose SEMAGE, a novel image-based CAPTCHA system, which has a two-factor model requiring the user to recognize the image and identify images which share a semantic relationship. The introduction of semantic correlation makes SEMAGE safe from similar machine learning attacks. Other image-based systems like ESP-PIX [13] and SQ-PIX [17] are language dependent and have usability concerns. We survey more CAPTCHA systems and their limitations in Section 2. As we would see, SEMAGE aims to defeat web bots by presenting challenges which are considerable harder for bot to solve but come intuitively to humans.

Web bots have also been widely used by attackers to generate web spam. Many web bots such as XRumer, Magic Submitter and SENuke have been developed for creation of backlinks in Blackhat Search Engine Optimization (SEO) technique, automated content creation on web services or bulk registration of free services through identifying meanings and functions of special HTML elements. Current most common ways of defending web bots are utilizing CAPTCHA. However, CAPTCHAs requires users to solve some explicit challenges, which is typically interactive and intrusive. As these challenges have to become more complex to defense evolved web bots, it has become difficult as well for legitimate users to solve them, resulting in decreased usability.

NOID is also completely accessible since it is completely passive in nature and hence poses no burden on part of the visually impaired. Mobile and touch based hand-held devices where its traditionally difficult to type would also be greatly benefited.

NOID is a novel, one of its kind solution which has the potential to completely replace CAPTCHAs and prevent bots from spoiling the user experience for normal users. The novelty of NOID lies in the fact that it works behind the scene to create challenges for the bot whilst the letting the normal user to carry out his own tasks. The challenge for the bot is to identify the correct HTML input element/parameter to automate a request. We would present and discuss details of NOID design and its important components in Section 8. We also implemented NOID on a proxy and evaluated its effectiveness against state of the art bots like XRumer, UWCS on popular open source forum and blogging platforms like PhpBB and Wordpress. We present the implementation details and evaluation in Section

9. We give a proper threat model in Section 7.1 and discussion limitations and futurework in Section 11.

2. RELATED WORK

CAPTCHA systems, text-based in particular, have been in widespread use as the first line of defence against bots on the web. Recently, with the improvements in Computer vision technology text based systems have become susceptible to bot attacks with a high success rate [6–10, 21]. Hence a lot of work has been put on alternate CAPTCHA systems like image-based [12–19] and audio-based systems [22–25].

2.1 Text Based CAPCTHA Systems

Generally, text-based CAPTCHA systems ask the user to discern letters or numbers. GIMPY is one classic example [26]. Attacks on text-based systems mostly employ OCR (optical character recognition) algorithms. These algorithms first segment the images into small blocks each containing only one letter, and use pattern recognition algorithms to match the letters in each block to standard letter template features [6–8]. The later task is considered a well solved AI problem. In counter-attack to these algorithms, text-based CAPTCHA systems employ the following techniques to enhance robustness [5, 10]:

- Adding noises in the form of scattered lines and dots to the background to counter-attack segmentation algorithms.
- Characters are connected or overlapped so that attacking algorithms cannot correctly segment image into correct blocks.
- Characters are twisted to increase difficulty in character recognition.

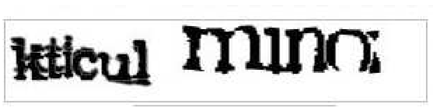


Fig. 2.1.: A text-based CAPTCHA example.

However, all the above techniques increase the difficulty level for humans too. Connecting characters together makes the task harder for humans e.g. characters‘r’ and ‘n’ when connected appear the same as character ‘m’. Twisted characters not only gnaw on user's nerves, but are sometimes impossible to identify correctly. Figure 2.1 shows one such difficult to solve text based challenge.

Text-base system faces one inevitable situation, humans find the CAPTCHA challenge unpleasant as CAPTCHA gets more complicated. This is probably why popular websites such as MSN hotmail opted for simple and clean CAPTCHA , which could be attacked with a rate over 80% [6]. Some systems use distinctive color for each character and add colored background using non-text colors, both of these additions can be easily removed by an automated program, which add no more difficulty for the bot [27].

Popular systems such as 'reCAPTCHA' [28] uses dictionary words that are labelled as unrecognizable by real automatic OCR programs running on real tasks digitizing books, and evaluate correctness by other user's input. However reCAPTCHA too suffers from decreased usability and user satisfaction due to high distortion and noise in the challenge.

2.2 Audio Based CAPCTHA Systems

Audio based CAPTCHA systems [22–25] remedy the fact that visual CAPTCHA systems are not accessible to visually-impaired people. In a typical audio CAPTCHA system, letters or digits are presented in randomly spaced intervals, in the form of audio pronunciation. To make the test more robust against bots, background noises are added to the audio files. These systems are highly dependent on the audio hardware and the user only has a certain small amount of time to identify each character. In some sense, audio CAPTCHA systems can be considered as the acoustic version of text-based systems. Although the visual cues are replaced with acoustic cues and the algorithms vary, the underlying idea of attacking is the same - features are extracted and classified to recognize the letters [29]. The difficulty curve for bot and humans are similar. Thus audio CAPTCHA systems provided neither more user-friendly interface for visually accessible users, nor more robustness against bots.

2.3 Image Based CAPTCHA Systems

Image-based CAPTCHA systems emerged in effort to replace text-based CAPTCHA systems which were growing more complex for humans to solve easily. Security is not the only concern is a good CAPTCHA design. All CAPTCHA systems are a form of HIP (Human Interactional Proofs) and require the users involvement. This also makes usability a key issue too in CAPTCHA design. Tygar et. al. [14] propose the following requirements for a good CAPTCHA system,

- The task should be easy for humans.
- The task should be difficult for computer algorithms.
- The database should be easy to implement and evaluate.

The general basis of image-based CAPTCHA is that images contain more information than texts. It is intuitive for human to catch visual cues but hard for AI algorithms to do visual recognition. ESP-PIX [13] presents a set of images and asks the user to choose a word from a list of words that describes all images. This approach suffers from two drawbacks such as it still depends on text to convey meaning and since all words are written in English, the users success depends on his/her proficiency in English(or any other particular language it migrates to). It is not only language dependent but hard to operate too; a user needs to scan through the whole list of words to find the most proper answer. SQ-PIX [17] also presents user with an image set, but asks the user to select an image of a given object name, and also trace the object in the image. This is also language dependent and the act of tracing around an object with a pointer operated from a hand-held device like mouse cannot be assumed to be easy for all users.

Google's image CAPTCHA "what's up" [15] asks the user to adjust the orientation of an image. This system is language independent, but the adjustment requires a lot of attention and subtle mouse (or other hardware) movement. Some images also have ambiguity as it can be correctly oriented in multiple ways.

Microsoft's Asirra [12] utilizes an existing database on petfinder.com and presents the user with images of cats and dogs and asks the user to identify all images of cats out of 12 pets. This platform is language independent, and requires user to scan through 12 images and click 6 times on average to be correct. Figure 2.2 shows a sample Assira challenge.

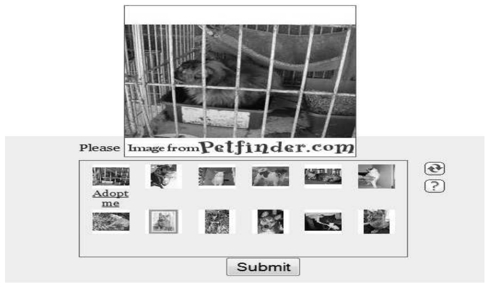


Fig. 2.2.: An Assira challenge: A user is always required to select all cats from images of cats and dogs.

Asirra partners with petfinder.com and gets access their huge database of cats and dogs. But the inherent difficulty for the bot boils down to only recognising classifying each image in either of the two classes: cats and dogs. This makes Assira more vulnerable to machine learning attacks [20] and template fitting attacks. SEMAGE on the other hand has a two-factor design where in order to pass a challenge the user is required to recognize each image and then understand and identify the semantic relationship between a subset of them. Assira only requires the user to solve the first level (i.e., image recognition). Utilizing the semantic correlation between images to create more secure and user-friendly challenges makes SEMAGE more robust.

2.4 Analysis of Inputs

Shin et. al. [11] examined the characteristics of 286 days of forum spam posted at a research blog and developed light-weight features based on spammers' IP, commenting activity and the anatomy of their posts. They then used a SVM classifier trained on these features to detect and weed out spam postings on that blog. Mishne [30] proposed detecting link spam (common in blog comments) by comparing the language models used in the blog post, the comment, and pages linked by the comments. Bhattarai et. al. [31] also used content analysis to characterize spam in blogs. However, content analysis is often application/platform centric (blogs in these cases) and does not apply to web bots doing bulk registrations and logins on other web sites. The features seemingly effective for the mentioned research blog may still need to be validated on multiple other platforms such as discussion boards and forums. For the anatomy of the posts, they could be changed easily by the botmasters to pass detection. Also devising such a set of features requires considerable amount of human effort ruling out fast and dynamic deployment. Schluessler et. al. [32] analyzed input data events such as keystrokes and mouse clicks to detect a bot, which was complementary to our approach. Brewer et. al. [33] used link obfuscation to detect and counter web bots which mimic human clicks by walking random links. This approach though effetive in stopping bots from navigation via random links in the web site, it does not deter bots which know the url of the page they want to go to and generate bulk data processing requests.

3. SEMAGE - A NOVEL IMAGE BASED CAPTCHA

As out first technique, we propose SEMAGE (**Se**mantically **Ma**tching Ima**ge**s), a better CAPTCHA system. In SEMAGE, we present the user with a set of candidate images, out of which a subset of them would be semantically related. The challenge for the user is to identify the similar images based on the context defined by the system. Users are asked to pick up the correct answer set. Note that the images in the correct set need not be images of the same object, a set of semantically related images may be images of entities with different physical attributes but sharing the same meaning in the defined context. Consider for example the user being asked to identify similar images with the context being similar images should have the same origin; the candidate set could contain images of a wooden log, a wooden chair, a matchstick, an electronic item, an animal, a human etc. with the chair, matchstick and log being the similar set.

The challenge in solving a SEMAGE CAPTCHA system is two-fold: (1) a user has to figure out the content of the individual images, i.e., image recognition, (2) and understands the semantic relationships between them and correctly identify the matching images. This challenge solving ability comes naturally to humans as human automatically employ their cognitive ability and common sense without even realizing the inherent difficulty of the task. The same challenge would be difficult for a bot to solve as it would require both understanding images and identifying relationships between them constituting a difficult AI problem. Our two-factor design aims at increasing the difficulty level for a bot and improving usability for humans, without sacrificing the robustness of the system.

What makes SEMAGE novel is the idea of presenting the user with a two-factor challenge of "identifying images with similar semantics under the given context". The idea of choosing images exhibiting semantic similarity has a much broader scope than simple selection of images of animals of the same species (cats in case of Assira). This differentiates SEMAGE from all the other state of the art image-based CAPTCHAs which only require the user to solve the first level which is image recognition. Computers cannot comprehend and identify the semantic content of an image making SEMAGE very robust to bots.

We also implement one very simplified sample instance of SEMAGE using real and cartoon images of animals. The relationship query asks the user to pick up images (real and cartoon) of the same species. This has two immediate benefits: (1) Adds fun factor for the user without adding burden on the recognition part since a human can easily make a connection between a real image of an animal and a cartoon image; (2) Scales up the difficulty level for bots as the cartoon images need not even resemble the real physical attributes of the animal. Moreover, SEMAGE provides an easy to operate interface to indicate correct answers making it an ideal choice for touch based systems and smartphones where typing is more difficult.

A sample simplified SEMAGE challenge is shown in Figure 3.1 which illustrates the idea. A human can easily identify the images marked in a circle as similar but a bot would not be able to relate the real and cartoon images due to difference in shape and texture. Note that this is just one way of creating a SEMAGE challenge. Any other *semantic relationship* can be used as the identifying factor apart from our particular simplified implementation.

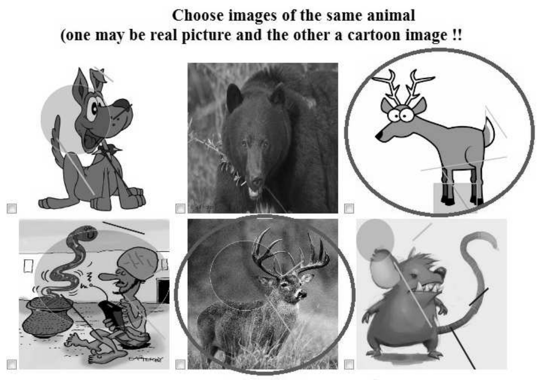

Choose images of the same animal
(one may be real picture and the other a cartoon image !!

Fig. 3.1.: Sample SEMAGE challenge, the encircled images are similar.

The main contributions of this work are as follows:

- We propose SEMAGE, a new image-based two-factor CAPTCHA that has several unique features. The design of a SEMAGE allows easy tuning of the security level and usability level depending on the nature and popularity of the website. The images of the SEMAGE challenges can vary to suit the needs of different websites. In fact in most cases given a labelled database its very easy and intuitive to come up with a defining "semantic relationship" and SEMAGE implementation. We also provide an in-depth security analysis and show how SEMAGE is more robust to many attacks than existing systems.
- We further conduct a large-scale user study with 174 participants using a simple sample SEMAGE implementation. We compare our system with state-of-the-art text-based CAPTCHA system reCAPTCHA [28] and image-based system Asirra [12] on the metrics of usability and fun factor. As discussed in details in section 6.2, results show that our system is easy to use and participants reported a high level of 'fun' factor.

4. SEMAGE DESIGN

We propose SEMAGE, "**SE**mantically **MA**tching Ima**GE**s", a novel image-based two-factor CAPTCHA system which is built upon the idea of semantic relationship between images. The use of semantic meaning of a query has been applied in other fields like web search [34]. We formulate definitions for semantic similarity of images and design a system that uses these concepts to develop a user friendly and secure CAPTCHA system.

4.1 Intuitive Idea

All image-based CAPTCHA systems have two main components: a database of images and a "concept" which uses the database to create challenges. The inherent concept may be as simple as PIX [35] which displays different images of the same object from the database and asks the users to assign an appropriate label or a complex one like Cortcha [19] which uses the database to create inpainted and candidate images and asks the users to place the correct candidate image in the inpainted image.

The idea behind SEMAGE is to use semantic relationships among images as the concept and keep the task of the user to simply identify the semantically similar/related images. The semantic relationship is a concrete description which would bind the similar images. The freedom of choosing the semantic relationship for one's application and database give it the much required customization flexibility. For example, for an electronic e-commerce site, SEMAGE challenge could be formed from the images of the products (an ipod, a zune, tv, heater, refrigerator etc) where the concept would be to ask the users to choose products which do the same thing (ipod and zune in this case, both portable music devices).

SEMAGE presents a set of candidate images with a subset of them sharing an implicit connection or relationship with each other. The challenge for the users is to correctly identify all images in the semantically related subset.

4.2 Defining the Semantic Relationship

We now present the conditions for choosing the “semantically similar” relationship which forms the ‘concept’ for challenge creation. A “Semantic label” could be a term or a relationship which identifies/labels the object. Semantic labels can be directly used to label the database for challenge creation. Let $SL(x)$ denote the function that returns the semantic label of an object x. We consider two images to be “Semantically Matching” if they satisfies any of the following conditions:

- Condition (1): if both images can be identified with the same semantic label. Given two images A and B. A and B are said to be semantically related if $SL(A) = SL(B)$. For example, an image of a computer and a television set can be defined with a semantic label(SL) ‘electronics’.
- Condition (2): both images can be classified under the same semantic label. Given two images A and B, they are semantically related if $\exists T\ s.t.\ SL(A) \subset T\ \&\&\ SL(B) \subset T$, here T denotes some semantic label. For example an image of a lion and a deer can be classified under the semantic label ‘four legged animals’. Similarly, an image of a television set and a computer can be classified under the semantic label ‘electronics’.
- Condition (3): when both images put together they express a uniquely identifiable concept. Given two images A and B and some semantic label C that denotes a set of requirements, A and B are said to be semantically matching if $\{A \cup B\} \models C$ where “$\models$” denotes that the left hand side satisfies the requirements of right hand side. For example, an image of a printer and paper can be defined with a identifiable concept ‘printing’ which becomes the semantic label.

The requirements for a “semantic relationship” gets more generic and the semantic correlation increases as we move from condition item (1) to (3). In order to form a SEMAGE challenge, the images have to be chosen such that only one subset meets any one of the above conditions with preference given to the least generic label. That is, if a set of images contain images which satisfy more than one of the above conditions, the least generic match is the solution required to pass the challenge. Thus, given a set of images where a small subset of images is of fishes and the rest of the images are of other unique animals, the solution to the challenge would be selecting all images of fishes.

The mechanism may seem complicated but as we show below, a system designed to create challenges where all solutions satisfy only one chosen condition is relatively easy to implement. Also the user study in Section 6.2 supports our claim that such a system is intuitive and easy for the normal user to solve. The important thing after one has decided upon the "semantic relationship" is to label the images accordingly. We discuss database generation in Section 4.4.

4.3 Challenge Creation

We develop a simple algorithm to create SEMAGE challenges. First we present the definitions and requirements of the involved parameters as follows.

Let n be the number of images in the challenge and m be the number of similar/related images. Let U be the superset of all image sets in the database. Each challenge set is denoted as S where $num(S) = n$. There exists a 'semantically similar' subset of images R such that every image in R has the same semantic label, i.e., $\forall\, r_i,\, r_j \in R,\, SL(r_i) = SL(r_j) \&\& num(R) = m$. A set of images D with $num(D) = n - m$, and each image in D has a different semantic label than R. Now each challenge set becomes $S = R \cup D$.

We now present a simple algorithm to implement the challenge set as shown in Algorithm 1 . The database would be a collection of semantically labelled images. The algorithm starts with empty sets R and D. We then pick a semantic label at random from the database and populate R with images having the picked semantic label. Then we populate D with images such that each image has a different semantic label than any of the images chosen before. The number of images in the R and D depends on the values of n and m and is customizable. The images in set $R\, and\, D$ are then presented in a random tabular order to the user.

Algorithm 1 : An algorithm to generate SEMAGE challenges from a labelled database

```
$R \leftarrow \phi$
$D \leftarrow \phi$
$A \leftarrow$ Pick an Semantic label at random
**while** $num(R) \neq m$ **do**
    $X \leftarrow$ (pick unique image with label $A$)
    $R = R \cup X$
**end while**
$Y \leftarrow \phi$
**while** $num(D) \neq (n - m)$ **do**
    $Z \leftarrow$ Pick a label at random which is not $A \cup Y$
    $Y \leftarrow Y \cup Z$
    $D = D\cup$ (pick unique image with label $Z$)
**end while**
$S \leftarrow R \cup D$
Randomize(S)
```

4.4 Database

For our implementation, we developed a semi-automated mechanism that populates the database by crawling the internet. One can also consider taking frames from movies and short videos. Both of the above approaches can be considered as semi-automatic and require some manual work to weed out irrelevant images. The drawback of such methods is that an attacker can venture to spend enough time and manual work to reproduce the whole database.

SEMAGE, however, does provide great room to adapt to existing database where tagging already exist. One possible application lies in the e-commerce area. Vendors in e-commerce usually have multiple images of the same product (such as pictures from different angles), multiple styles of the same product (same product of different color, size, packages), and multiple products of the same category. Images are tagged with the product information, and product info is categorized into different classes. Multiple relations can be established among these images and used as the 'relation question'. With the abundance of existing tagging information, we can implement the 'matching' algorithm by adding simple logical changes. Furthermore, some database actually have implemented more sophisticated relations such as ‘relative products’ as a recommendation for users when they browse certain products, thus more sophisticated relationship questions can be implemented based on such information. Using these images not only achieve the security purpose, but also serve as a good form of advertisement.

5. SEMAGE ANALYSIS

5.1 Design Analysis

Usability: Usability with security is the primary focus of SEMAGE. The images contain content that cognitively make sense to the users, and are easy to discern. By drawing on human's vast storage of common-sense knowledge, our design help user spend minimum effort solving the challenge. Moreover, it fits a humans way of thinking - it is natural for human at first sight to see what an image is about, much better than dealing with any details (orientation, certain feature image, etc.). Establishing relationships among objects is another ability humans are natural at, and humans almost automatically dissolve any ambiguity they need to resolve. For example, if a red car is presented with other colored cars, human immediately notice the color difference. However, if the same red car is presented with red buckets, red clothes etc. humans notice the difference in object category. For a computer, both of the steps pose a difficult AI problem. It first needs to do image recognition to determine what the image contains, and tag the image in a pre-determined category. To solve the 'relationship' answer, the computer would not only need vast correctly labelled database, but also complex AI intuition. This creates a great gap in the difficulty level for human and bots.

In addition, SEMAGE provide an easy to operate interface for users to indicate correct answers. Only a few mouse clicks is required to pick up the correct images, this makes SEMAGE to be a good choice of touch based systems and smart-phones where typing is more difficult. This is much easier than tracing an outline of objects (as in SQ-PIX [17]) and typing in letters from a keyboard, especially on mobile platforms.

Language Independence: Our design utilizes the fact that a picture transcends the boundaries of languages. Some CAPTCHA systems also use semantic clues, such as ESP-PIX [13]. However it asks the user to find the right word among a list of English words that describes the content of the image. This limits the audience to people with decent proficiency in the language. Our design is language independent and can used by people across the world. This is especially beneficial for people who are not comfortable using English as a daily language, but access websites in English.

Customization Flexibility: Our design offers several ways to customize the challenge on content, security level and usability level. The image database can be customized to suit the needs and style of the hosting website. For example, for special interest groups, the database can be objects of the theme of the group, such as movie screenshots for a movie rental site or specific products for a e-commerce site. This provides possibility of advertisement of content or fun in the traditionally boring test of CAPTCHA.

It is also easy for web administrators to customize on the security level. The administrator can decide on the size of the candidate image pool, and the size of the correct answer set. For a scheme that present n candidate images and ask the user to pick up k matching images, the success rate of random guessing is $1/C(n, k)$. The increase of the size of answer set does not necessarily decrease the chance of success of a random guessing success when n is small, but as n increases, the probability of a random guess attack goes down. As for the user experience, the time user spent on the CAPTCHA task increases as the size of candidate image pool increase, but the effect of an increased size of answer set on users time is not obvious. We think the optimum choice of n and k might depend on particular content of the images used, and specialized user study can be conducted if such data is desired.

5.2 Security Analysis

We consider an adversary model wherein a bot has access to the unlabelled and uncategorised database of images from which we form our challenges. It is to be noted that given ample time and resources some of the attacks discussed below could succeed but taking a long time defeats the primary purpose of the bot. Our goal as in any CAPCTHA system is to make any attack using the current state of the art as difficult as possible, so that any successful attack would need a major step forward in technology. We now identify and analyse possible ways of attack against our system and how it fares against them.

Attacks using machine learning techniques: Similar techniques used to attack Asirra [20] can be used to attack our system too. The attack on Assira was an attack on the first level of our model namely simple "Image Recognition". In essence, attackers try to get a certain number of correctly labelled images, and train on several different classifiers, either based on color information or texture information. However, solving a SEMAGE challenge not only requires image recognition but also identifying the "semantic relationship". The identification of "semantic relationship" among images is a unsolved AI problem. Moreover, even if the semantic correlation is weak and the semantic label is just the object name, SEMAGE accommodates much more object classes than Asirra (which had only 2), and the attacker will need to build many more types of classifiers accordingly.

Now let us consider a very simple example of "semantic relationship", e.g., "real and cartoon" images of the same animal (as used in Section 6.2). The color and texture data between a cartoon specie and real animal specie varies much more than in between cartoons and real animals, as illustrated in Figure 5.1. While attackers might attempt to train classifier of real animal and cartoon animal independently, the performance decreases as the number of classifiers increase which could be huge for a respectable size database. Thus success rate of attacks using this sort of algorithm is likely to be very low.

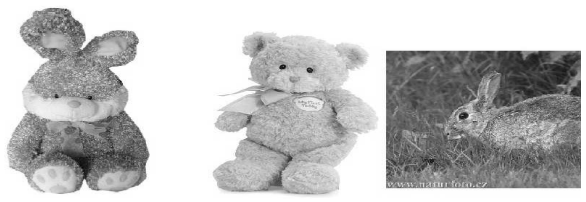


Fig. 5.1.: Showing limitations of the texture-based machine learning attack, (a) shares more commonality with (b) than with (c) , while (a) and (c) are of the same type rabbit.

Attacks using template fitting techniques: In image recognition, one developed area is to fit objects into (visual) feature templates. For example, a chair can be identified if given the template of 'four legs and a horizontal top'. Accordingly, for a rabbit, the feature should probably be ‘upwards pointing long ears’. However, it is much harder to define ‘long’ than ‘upwards’. A deer, with pointy upward ears and would be classified into the ‘rabbit’ template. Furthermore, not all objects have such uniquely identifiable simple feature.

Random guess attack: For a SEMAGE scheme that present n candidate images and ask the user to select k matching images, the success rate of random guessing is $1/C(n, k)$. As shown in Figure 5.2, choosing a low value of n and k could make the system more vulnerable to random guess attacks. On the other hand a low n, k makes the system more user friendly and less frustrating for the user. In case a low n, k system is deployed like our implementation, multiple rounds of SEMAGE could constitute one challenge, such technique is already in use current systems such as reCAPCTHA. A SEMAGE system could also be complemented with other techniques like the Partial Credit Algorithm in [12], which would allow a large n, k and a ‘almost right’ answer can be defined as missing one image in the answer set. Token buckets [12] can also be implemented to prevent brute-force attackers from making a number of continuous random guess attacks.

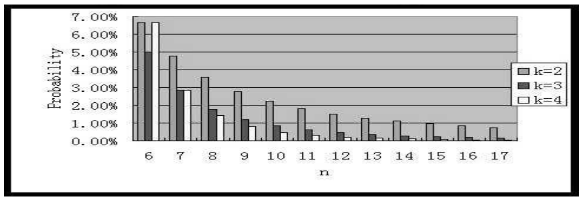


Fig. 5.2.: Random guess attack success rate with respect to k and n.

Attack using the static image name in source: If the source code of the HTML page hosting the challenge uses image names, an attacker could potentially use those names to identify similar images. However, this sort of attack is easily defeated by randomizing the images name in the source. In our system implementation, names of the images in the challenge are in no way exposed to the user. The image names in the html source is randomized when sent to the user.

By creating an attack database using the general relationships used in the system: The attacker might manually identify the general "semantic relationship" used in the system and then search and build an image repository to create an attack database. Using the labelled images of the attack database, a brute force search against the candidate set might yield him a correct 'similar' set. However comparing each image of the challenge with all the images in the attacker's image archive would take lots of time and resources than what would constitute a feasible attack; also this might exceed the maximum time allowed to take a challenge.

Attack by mining textual description of images: Potentially a attacker by using systems like google's goggle[1], an image based search system, might uncover textual descriptions of the candidate image set and then can use the textual descriptions to identify relationships among images. We argue that first of all image recognition or search is still not mature enough for now (very hard problem for unknown images). In addition, identifying relationships among objects even with textual descriptions is a complex AI problem to solve, especially since the correct similar images depend on the context. Such a attack would potentially defeat most the present Image based systems such as Assira, PIX, SQ-PIX but because of the two level design of SEMAGE, the bot would still need to understand and identify the semantic correlation. Having a textual description only possibly solves the problem of image recognition. There may exist images with overlapping descriptions but which are not a part of the 'semantic similar' image set in the context. Consider for example a candidate image set wherein the context is identifying 'four legged' animals among images of insect, deer, lion, human, electronics item and other unrelated objects. Now even with accompanying textual descriptions such a relationship is hard for a bot to find and relate to lion and deer.

[1]`http://www.google.com/mobile/goggles/`

6. SEMAGE EVALUATION

We conducted a large scale user study to evaluate the usability of SEMAGE as compared to Assira and reCAPTCHA. For this purpose, we firstly built a website which would present the users with sample SEMAGE challenges.

6.1 Sample Implementation of SEMAGE

In our sample implementation, each challenge consists of a set of images (the number of images is configurable) where a subset of images would share a distinct relationship/feature with each other. The images are furthermore randomly distorted by introducing noise and changing the texture. Our implementation was carried out in PHP with MySQL being used as the database. Figure 6.1 gives a high level design of the implementation.

Choosing the "semantic relationship": In our particular implementation, the challenge set consists of real and cartoon images of animals with the relationship defining the 'similar' subset being "real and cartoon images" of the same animal. The advantages of choosing the 'real and cartoon' relationship to define "semantic relationship" between images are:

- The relationship between real and cartoon images of the same animal in most cases is subtle and variable. The reason being that the animals may completely differ in visual characteristics such as size, shape and outline in real and cartoon representations
- Humans with inherent capability to relate visibly dissimilar objects would be able to pass the challenge easily whereas the current state of the art bots cannot. We test this assumption of ours in the user study we conduct, discussed in details in section 6.2

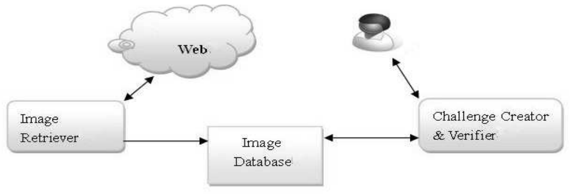


Fig. 6.1.: Overall implementation illustration.

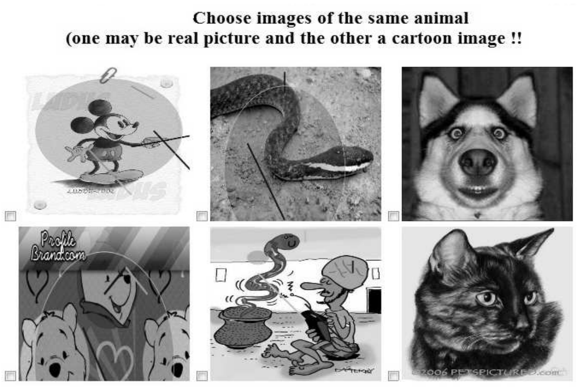

Choose images of the same animal
(one may be real picture and the other a cartoon image !!

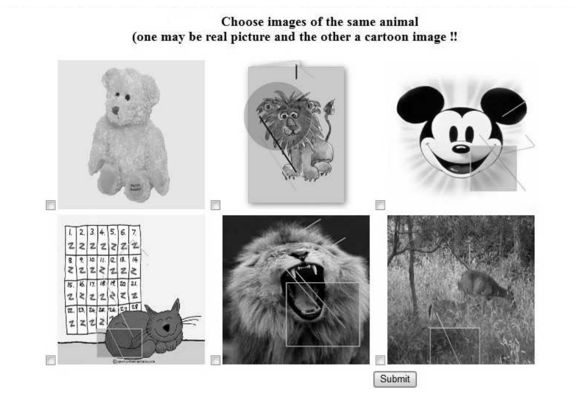
Fig. 6.2.: Screenshot of sample SEMAGE implementation with images 2 and 5 being similar, both snakes.

- Generating a large database is easier. A simple search for an animal on images.google.com yields millions of entries, hence we have a fast and easy way to build up a large database.

Figure 6.2 shows a sample SEMAGE challenge of our simple implementation. The total number of images in one challenge is six with the "semantically similar" set of two images, one a real image and the other a cartoon image of the same animal.

Database Generation: The first step for SEMAGE implementation after defining the semantic relationship between the "similar" images is database generation. An image search and download tool was implemented shown as Image Retriever in Figure 6.1, which searches and downloads the required images from the web. The tool would take in the search keywords (to search for real or cartoon images of the animals), image dimensions, and number of images to download and the label tags. It then automatically downloads the images and stores the local path and concerned labels in the database. A simple search for an animal on images.google.com yields millions of entries, hence we have a fast and easy way to build up a large database. In reality, since the automated search does not always yield relevant results, we manually weed out the irrelevant images from the collection.

Dynamic Noise Addition: To make machine learning attacks based on image classifiers difficult, we randomly introduce noise in the images of the challenge set at each challenge creation phase. We introduce noise in the form of random shapes and color scale alteration in the image with the help of the ImageMagick library. The position of inserting the random shapes varies from the center of the images to its edges. Also scale of color adjustment is also randomly varied to prevent the bot classifiers from easily weeding out the noise. Such random noise introduction makes sure that each image appears with different noise levels. Figure 6.3 shows a SEMAGE challenge after the introduction of noise.



Choose images of the same animal
(one may be real picture and the other a cartoon image !!

Fig. 6.3.: Example of noise addition in our implementation, here we can clearly see noise but still identify images 2 and 5 being similar, both lions. The changes in color scale are not visible due to the black and white nature of images.

Interface: As shown in Figures 6.2 and 6.3, each challenge appears as a tabular strip of images. The title of the tabular strip presents the challenge and then the user needs to click on the similar images and press submit to send the response to the server for verification. We experimented with different layouts e.g. the images being apart from one another, images in a single straight strip, and found that its much easier to identify similar images if they are bunched together in a tabular format.

6.2 User Study Methodology

A comprehensive IRB approved user study was then conducted to gather data about how user friendly SEMAGE is, which is one of the most essential criterion for a CAPTCHA to be deployed in real systems. We also incorporated reCAPTCHA, a text based system and Asirra, an image based system from Microsoft in the user study to carry out a comparative analysis. Both Asirra and ReCAPTCHA are available as a free web service allowing us to easily integrate them in our study. The volunteers took the study remotely and were given a brief 1 page pictorial description of what they need to do to pass a challenge for all the systems. We logged the time taken to complete each challenge as the difference in time between when the test first appears on the screen and the time user clicks on the 'submit' button to submit his attempt. The users were let known of whether they passed or failed the previous challenge before presenting a new one.

A total of 174 volunteers took the study and the population was a mix of graduate and under-graduate students. The subject pool was diverse with most of the users from a non-computer science discipline, with a mix of native and non-native English speakers. The subject pool consisted of 66 females and 108 males. The subject pool were in no way made aware of the fact that SEMAGE is our system. We collected the time taken by each user to complete a challenge for each of the system as described earlier. We monitor the time taken for all attempts irrespective of whether it was successful or not. We also collected number of successful and failed attempts to solve a challenge.

6.2.1 User Study Layout

The user study was carried out via an website with the following sections:

- An initial questionnaire asking the users to rate their familiarity with CAPTCHAs, proficiency in english language and other demographic questions such as sex, age.
- A 1 page pictorial description each of SEMAGE, Assira and reCAPTCHA, showing users how to solve the challenge.
- 5 different challenges from SEMAGE.
- 5 different challenges from Asirra.
- 5 different challenges from ReCAPTCHA.
- A final short questionnaire asking users to rate SEMAGE for Fun factor and ease of use as compared to Assira .

We believe a pictorial description of each of the systems was necessary for fair usage statistics on the Image recognition systems. It was user's first time seeing an Image based CAPCTHA whereas all the users had invariably taken a text based challenge before. Presenting a brief description of what they need to do to pass a challenge would prepare them about each system and allow us to collect even handed usage data. The study took an average of 8.7 minutes to complete.

We divide the usability evaluation in different sections presented below according to the following metrics:

- How fast can a user complete a challenge?
- How many times does the user pass the challenge successfully?
- Does the user consider the system to be fun and easy?

6.2.2 Timing Statistics

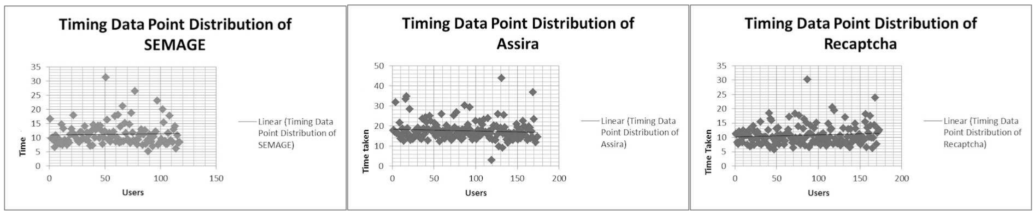
As we see in Table 6.1, users complete text based and SEMAGE challenges faster than Asirra. Each user takes an average of 6 seconds more to complete a Assira challenge.

The distribution plots in Figures 6.4 clearly show that most of the users of SEMAGE finished each challenge in about 11.647 seconds or less, whereas this number is comparatively high for Asirra with most of the users taking around 17.355 seconds. Consistency and uniformity in majority of the data points of the plots show that the timing average was

Table 6.1: Average time taken per challenge for each of the systems in seconds.

	Semage	**Asirra**	**recaptcha**
Time Taken in seconds	11.64	17.35	11.05

not largely affected by some isolated edge cases and it represents the general behavior of the users.



(a) SEMAGE Timing (b) Assira Timing (c) reCAPTCHA Timing

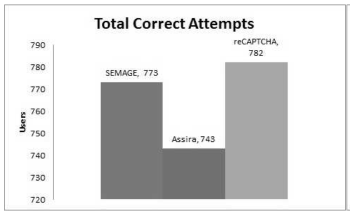
Fig. 6.4.: Timing distribution of each system for all users.

We notice that the average time taken by the users to solve a challenge from SEMAGE is almost same as reCAPTCHA, we think this may be due to the fact that text based CAPTCHA have been in use for a long time and users have gotten used to it whereas users were seeing our system for the very first time.

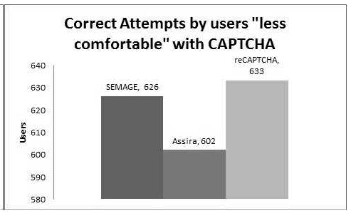
We concede that an Assira challenge consists of more images than a SEMAGE challenge leading to more time spent on completing each challenge. However, Assira needs more images in each challenge set to be secure because of the limited set of differentiating classes of objects (two to be precise, just cats and dogs) whereas there can be theoretically thousands of differentiating classes in our SEMAGE implementation. Moreover, presence of just two differentiating given classes should have made the challenge easier for humans as they simply need to place each image in one of the two categories. SEMAGE on the other hand requires the user to relate two or more images, making it potentially more time consuming. However the timing data clearly shows that taking SEMAGE challenges is easier than it seems because of the natural cognitive ability of humans.

6.2.3 Accuracy Statistics

Simply speaking, the total number of correct attempts for SEMAGE is higher than Asirra, indicating that users are able to correctly solve more challenges of SEMAGE. Figure 6.5a shows a graphical representation of the difference in correct attempts between Assira and SEMAGE. We had also asked the users to rate their familiarity and comfort level with Captchas on the scale of 1 to 5 (with 5 being very comfortable) in the initial questionnaire. As we see in Figure 6.5b all participants, as well as those who voluntarily identified themselves as 'less comfortable' (rated 3 or less) with CAPTCHA systems in general, also show high accuracy with SEMAGE and reCAPTCHA than with Asirra.



(a) Total correct attempts out of 815 attempts



(b) Total correct attempts from 132 users who rated themselves as less comfortable with CAPTCHA

Fig. 6.5.: Accuracy achieved on individual systems.

In order for the system to be deployed in the real world, it should have a high 'Correct Attempts ratio' for humans. The 'Correct Attempts ratio' (C.A.R) is simply the number of correct attempts divided by the total attempts. It signifies how many times a human passes the challenge. Hence closer the ratio is to 1, the better the system is in terms of usability.

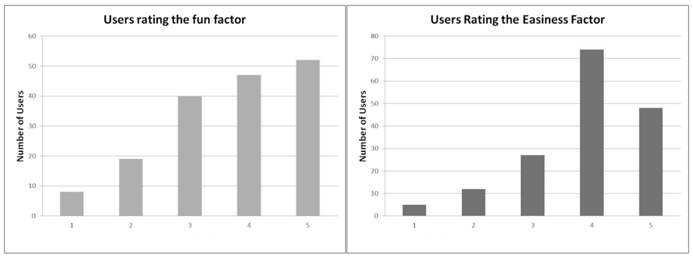
The user study data shows us that our system has a higher C.A.R (0.94) than Asirra (0.91). Users had been familiar with text based CAPTCHA systems, so we expected them to do very well in the reCAPCTHA system. But again, the difference between SEMAGE and the traditional text-based system is almost negligible. This along with the timing data shows that our system likely has a higher usability factor than the current state-of-the-art image-based system (Asirra).

6.2.4 Fun Factor and Ease of Use

After the completion of challenges from the three systems, the users were then asked to compare and rate SEMAGE and Assira on the criterion of Fun and Easiness. There were two separate questions: one for Fun factor and the other for Easiness, which asked them to choose a rating as follows:

- 1, if they found Assira to be way more fun or easy.
- 3, if they found Assira and SEMAGE to be equal on the Fun or Easiness factors.
- 5, if they found SEMAGE to be way more fun or easy.
- 2 or 4, if they were slightly inclined towards Assira or SEMAGE respectively.

These factors gave us a more subjective indicator of usability.



(a) Users rating the Fun factor (b) Users rating the Easiness

Fig. 6.6.: We asked users to comparatively rate SEMAGE and Assira on the metrics of fun and easiness, Rating 1,2 indicate Assira to be more fun and easy, rating 3 indicate both systems are equal and rating 4,5 indicate SEMAGE to be more fun and easy.

We can clearly see from Figure 6.6a that majority of the users (58.92 %) choose rating 4 and 5 indicating a high fun factor with SEMAGE. Only 16.07% choose rating 1 and 2 indicating Assira was better while the rest considered them to be equally fun. This clearly supports that more users found SEMAGE to be a system which was more fun to solve than Assira.

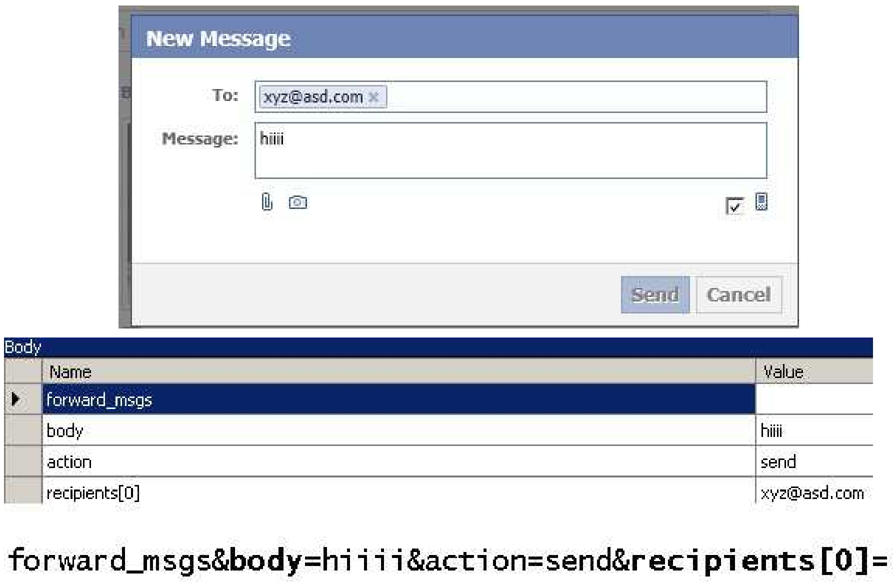
Figure 6.6b shows the rating distribution for the Easiness factor. 72.61% of the users rated 4 and 5 indicating SEMAGE to be easier than Assira. Only 10.72% of the users rated 1 and 2 indicating Assira to be more easy while 16.66% of the users rated 3 indicating they considered both systems to be equally easy. These metrics as well as the timing and accuracy results shown previously clearly demonstrate that SEMAGE is a highly user-friendly CAPTCHA.

7. NON-INTRUSIVE WEB BOT DEFENSE SYSTEM

Many websites have successfully attracted users' great attention, through providing favorable web services such as information searching, social networking, online blogs, web-based games and so on. These services enable user interaction with remote servers through receiving users' requests and sending back corresponding responses. However, through masquerading real human users' requests, attackers have also designed and utilized web bots to launch attacks on the websites such as illicitly registering accounts [1] and automatically posting web spam [36].

In fact, many web bots such as XRumer, Magic Submitter, and SENuke have been developed for creation of backlinks in Blackhat Search Engine Optimization (SEO) technique, automated content creation on web services or bulk registration of free services through identifying meanings and functions of special HTML elements. As one of the most powerful and popular forum spam bots, XRumer can be directly used to automatically register forum accounts and send spam with the aim of boosting search engine rankings [11, 37]. Popular online social networking websites such as Twitter and Facebook have also become web bots' attacking targets. Figure 7.1 shows a real example that attackers can automatically send spam through Facebook Message Box by utilizing a web debugger, named Fiddler to identify HTML elements and to submit bulk HTTP requests.

Current most common ways of defending against web bots are utilizing CAPTCHA (Completely Automated Public Tests to tell Computers and Humans Apart) to distinguish bots from real human users [1–5]. However, CAPTCHA requires users to solve some *explicit* challenges, which is typically interactive and intrusive. Also, the robustness of CAPTCHA system highly relies on the intrinsic difficulties of artificial intelligence challenges, which could be compromised through achieving artificial intelligence breakthroughs [2, 3]. As those challenges have to become more complex to defend evolved web bots, they have become more difficult as well for legitimate users to solve, resulting in decreased usability. Other machine learning technique based approaches to defend web bots



To: xyz@asd.com

Message: hiiii

Body

	Name	Value
▶	forward_msgs	
	body	hiiii
	action	send
	recipients[0]	xyz@asd.com

```
forward_msgs&body=hiiii&action=send&recipients[0]=
```

Fig. 7.1.: An example of using Fiddler to automatically send spam through Facebook Message Box.

are usually time-consuming to implement and error-prone. This kinds of work identify web bots through investigating bot behaviors and designing distinguishable features [11]. However, finding and designing an effective feature set require considerable time and human efforts to obtain bots' behavioral characters. Even an effective feature set is achieved, false positives are still difficult to avoid, leading to bad user experience. In addition, bots could still evade those detection features by better mimicing of real human users' behavior.

In this work, we design a novel server-side and *non-intrusive* web bot defense system, named **NOID**, to prevent web bots from accessing web resources. Specifically, NOID focuses on defending against web bots that can automatically submit bulk requests to remote servers by imitating real users' actions of filling out HTML forms and clicking submission buttons. The basic intuition of NOID is to leverage the fact that web bots need to pre-identify HTML elements to obtain corresponding parameters to fabricate normal users' requests. If the web server randomizes HTML element parameters in each session, web bots would fail in sending bulk automated requests based on hard coded parameters.

Thus, to prevent web bots from uniquely identifying HTML elements for later automation, NOID randomizes name/id parameter values of essential HTML elements such as “input textbox”, “textarea” and “submit button” in each HTTP form page. In addition, to prevent powerful web bots from identifying special user-action HTML elements by analyzing the content of their accompanying HTML Label tags – the “label text”, we enhance NOID by adding a component, named **Label Concealer**, which hides label indicators by replacing “label text” in Label tags with randomized images.Also, to further prevent more powerful web bots from identifying HTML elements by recognizing their context (e.g., relative positions or surrounding elements in the DOM structures) of the webpages, we enhance NOID by adding another component, named **Element Trapper**, which obfuscates important HTML elements’ surroundings by adding “decoy elements” (duplicate HTML elements) without compromising usability. These decoy elements are hidden from human eyes (or easy for human to skip) but make it difficult for bots to identify correct elements.

In this way, while NOID can defend against web bots at least as robust as CAPTCHA, it is totally transparent to end users and does NOT require normal users to validate themselves as human via explicit actions.

The main contributions of this work are as follows:

- We propose NOID, a novel server-side and *non-intrusive* web bot defense system. Due to NOID’s flexibility, it can be deployed either as a proxy, middleware, or a server side component according to web administrators’ goals.
- Through analyzing possible evolving techniques that can be utilized by powerful web bots to bypass NOID, we design an enhanced version of NOID by adding image labels and decoy HTML elements.
- We implement a proxy-based NOID system and evaluate it against five current state-of-the-art web bots on four popular open source web platforms on popular free platforms. According to our evaluation, NOID can prevent all these web bots from automatically sending spam on these web platforms with very reasonable overhead.

7.1 Threat Model

In our work, we focus on defending against those web bots that can automatically submit data submission requests to remote web servers through imitating real human users' browsing behavior. We define data submission requests as those requests which contain users' submitted data through filling out HTML forms (e.g., a user authentication request containing values of username and password).

We next briefly introduce the operation mechanism of this types of web bots. In order to fabricate data processing requests to remote servers as real human, those web bots have to first recognize submission requests sent by real users through using their web browsers to fill out HTML forms. Particularly, each input HTML element of a HTML form such as a textbox, textarea, checkbox, submit buttons are uniquely identifiable in the HTML source of the web page by a "name/id" attribute. Then, when a real user submits data through filling out a form, the browser generates HTTP requests containing parameters with key-value pairs, which will be process by remote servers. The keys are "name/id" attributes of input elements and the values are users' input data. Thus, through parsing and identifying unique and invariable "name/id" attributes of those input HTML elements, web bots can be programmed to fabricate and submit requests containing customized data. As Figure 7.1 shows, a facebook message request can be fabricated through identifying message body parameter"body" and recipient parameter "recipients[0]".

7.2 Web Bots' Threats

Through automating HTTP requests, attackers can utilize web bots to create more in-links to a particular web site to improve its search ranking, to post spam or malicious links, to illicitly vote, and to automate online game tasks. In this section, we summarize several major web bots' threats.

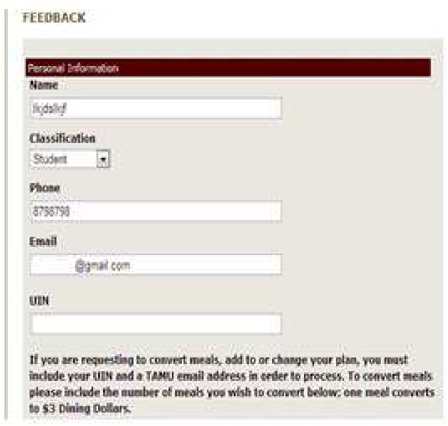
Comment Form Spamming: Attackers can utilize web bots to customized information on the comment sections in forums, blogs or online social networks [11, 38–41]. These information may be spam, malicious links or other websites' links for the purpose of SEO.

Actually, it leads to bad user experience and has become a big security issue for many web services.

Email Field Spamming: Email spam bots have been traditionally used by attackers to send email spam through harvesting email addresses. The effectiveness of those bots is highly restricted by email spam filter systems, which detect email spam by analyzing the reputation of senders or mail content. However, through utilizing web bots to automatically submit content in the email field in the "Feedback" or "Contact us" forms, attackers can send spam to victims through targeted websites' mail servers. We next introduce two prominent types of email form spamming: spam injection and spam source hijacking.

(1)*Spam injection.* As seen in Figure 7.2(a), many web sites create "Feedback" or "Contact Us" forms to communicate with their users through email. Typically, once users fill out their email address in email forms contained in such webpages, the websites will set users' email addresses in the "reply to" header of emails. However, through inserting email header "bcc:" and spam content with email addresses in the email fields, attackers can utilize websites' email servers to forward spam emails to victims [36, 42]. This threat may both bring potential safety hazard to other users and damage the reputation of websites' mail servers.

(2)*Spam source hijacking.* Many websites design hidden email fields such as "send to" in the "Feedback" or "Contack Us" forms to directly send emails to aforementioned addresses through their email servers. However, web bots could automatically fill out those hidden forms with victims' email addresses to send spam to victims through websites' email servers without websites' approval. For example, Figure 7.2(a) shows a real Feedback form used in some university's official website. From Figure 7.2(b), we observe that the form has hidden fields such as "mail to", "mail from", "subject", which can be easily utilized by web bots to send spam to victims with the identity of the website's email server. Actually, such attacks are fairly dangerous, since those spam emails sent from a trusted sender may affect more users.



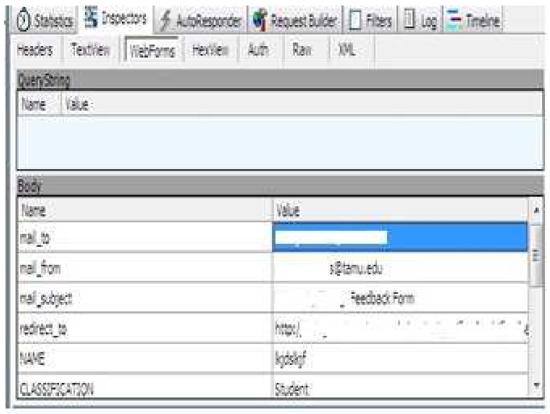
FEEDBACK

Personal Information

Name	lkjdslkjf
Classification	Student
Phone	8798798
Email	[redacted]@gmail com
UIN	

If you are requesting to convert meals, add to or change your plan, you must include your UIN and a TAMU email address in order to process. To convert meals please include the number of meals you wish to convert below: one meal converts to $3 Dining Dollars.

(a) Feedback form



Name	Value

Body

Name	Value
mail_to	[redacted]
mail_from	[redacted]s@tamu.edu
mail_subject	[redacted] Feedback Form
redirect_to	http:/[redacted]
NAME	lkjdslkjf
CLASSIFICATION	Student

(b) Hidden email fields

Fig. 7.2.: A case study of email filed spamming.

Online Vote Cheating: Poll bots, a special type of web bots with an aim at skewing poll results, can be easily obtained through Internet [43]. A particular example of bots cheating a poll is about the question "which is best graduate school in computer science?" on the website named Slashdot.com [3]. It turns out to be a battle of bots between CMU and MIT rather than a fair voting. Needless to say, poll bots violate the sanctity of the poll and render it practically useless.

Web-based Game Automation: With the emergence of many popular web-based games, many web bots are developed to automate certain repetitive tasks in the games. This behavior essentially violate the policy of fair competition of online games. In addition, although most of advanced games such as "Lord of Ultima" and "Mafia wars" are based on Ajax requests, the underlying idea of sending requests by submitting values in certain distinct"id/name" parameters still remains.

8. NOID DESIGN

In this section, we present the system design of NOID.

8.1 System Overview

As described in Section 7, the intuition behind NOID is that web bots need to pre-identify those unique id/name parameter values of the HTML elements to automatically interact with remote servers. The name/id parameter of a HTML element is a unique value in the web page and is generally constant, since it is used by the server-side logic as well. However, if NOID could hide correct name/id parameter values of those "critical" elements in the webpages, it will be difficult for web bots to identify those elements. (The "critical" HTML elements refer to elements such as "textbox", "textarea", "submit button" or other input elements in a HTML form, which can submit users input to the remote server.)

In fact, NOID hides the name/id parameter values of such elements by randomizing them in the source. In this way, it is impossible for web bots to simply use pre-programmed/hardcoded name/id attribute values to fabricate requests. In order to avoid adding complexity to the server-side logic of the web applications, NOID could be implemented as a middleware between the server and client. The NOID may reside on the server itself or be a proxy, acting as an interceptor proxy and catching and relaying all communication between the server and client. This independence of the NOID from the server and client allows it be universally applicable to different server technologies as a plugin, which can be switched on and off. Thus, it will be transparent to end users and need few server-side modifications.

With this intuition, NOID is designed with three major components: *Element Tagger*, *Randomizer* and *De-randomizer*. As illustrated in Figure 8.1, once the remote server replies to the client any response of webpage source code that is related to HTML forms such as HTML, Javascript and CSS, NOID will intercept the response and utilize the component of *Element Tagger* to tagging those "critical html elements" in the source. These elements are usually used to achieve web service functions, such as posting content on specific webpages (comments, posts, blogs etc), collecting users' email addresses and so on. Then, *the randomizer* replaces parameter values of those tagged elements in the source with newly randomized values, and reply the randomized HTTP form page to the client. Once the client receives the randomized form page and submits back their data processing requests, *the de-randomizer* restores original parameter values of those R-tagged elements and send back to the server. We next describe the design of each element in details.

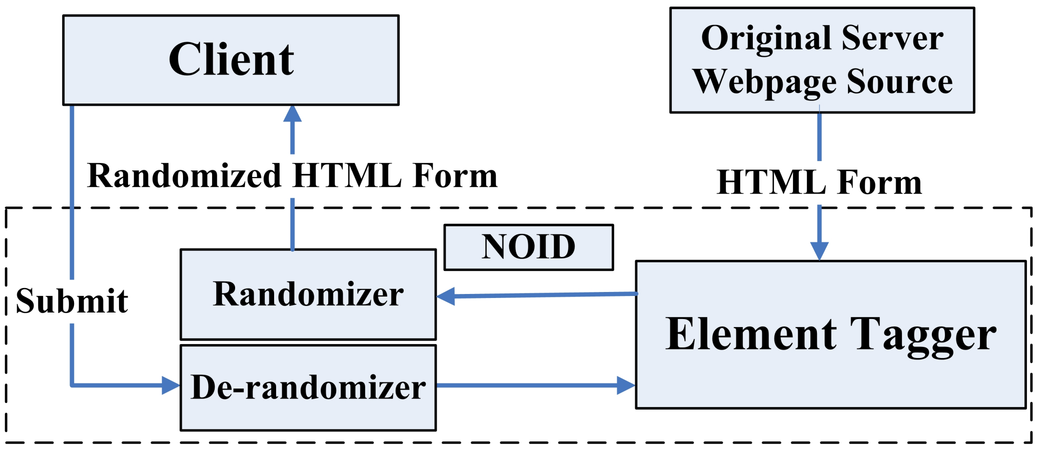


Fig. 8.1.: The system architecture of NOID.

8.2 Tagging Critical Elements

It is true that NOID can randomize all html elements in the webpage source to better defend web bots. However, complex webpages normally contains many HTML elements and most of them, such as navigation elements (buttons and links), are difficult to be explored by attackers to launch attacks. Thus, in order to more efficiently randomize server response, NOID can be used to randomize those "critical elements" rather than all HTML elements. Thus, to tag critical elements, Element Tagger will first identify critical HTML elements and then annotate those elements.

Identify Critical HTML Elements: To identify critical html elements, we give our definition of "Critical HTML elements". We define any HTML element (including hidden elements or Ajax request parameter) as cortical, if it meets at least one of the following conditions: a) it allows users to post content in the webpages; b) it is used by remote servers in forming email headers; c) it is used by the server to modify/update other server resources such as a database, a file, or in-memory states; d) The value of the element forms part of the parameters of a request to the server.

Then, those critical html elements in the webpage source can be either selectively identified by webpage developers or automatically identified through a program according to their functions[1].

Annotate Critical HTML elements: After knowing which elements are critical, we need to annotate the parameter values of those elements in the webpage source, which will be randomized later. This could be done either by adding a unique and specific prefix with those parameters or specifying totally new parameter values.

In our work, we choose the first approach, i.e., we annotate "name/id" parameter values of critical elements by adding a unique and specific prefix string, named as "R-tag". Specifically, in our implementation, we use the unique and special string of $\_RaNmE\_$ as R-tag, which is not normally used in the webpage source, aiming at prevent any unwarranted errors. For example, if a submission button with an id value of "submit" (e.g., "$< button\ type = button \quad id = submit >$") is identified as critical, Element Tagger will annotate this element as "$< button\ type = button \quad id = \_RaNmE\_submit >$". Since the parameters of those elements can also be used in the presentation code such as javascript and CSS, NOID will also annotate critical elements in those presentation code with R-tags.

[1]In our preliminary work, we choose the first approach to implement and leave the second approach in our future work. We also note that if a webpage has very few critical elements, NOID can selectively choose some non-critical elements to tag to increase security robustness.

8.3 Randomization

After tagging critical elements, the randomizer will randomize those elements with R-tags. In order to guarantee NOID to be stateless (i.e., NOID does not need to save original values to de-randomize), the randomizer uses a symmetric encryption scheme to randomize the parameters. As seen in Figure 8.2, the entire process of randomization can be divided into four main steps. (The details of Randomization algorithm can be referenced in Appendix A.)

Generate Master Key. The system periodically generates a master key in every "T" time slots, which is used to encrypt those annotated parameters. The master key generation should be fast and efficient, since a busy web server may service millions of requests per day. Thus, we adopt Xorshift random number generator [18], which generates sequences of random numbers basically by applying the "xor" operation on a seed number with a bit shifted version. The seed number could be a random counter or a specific system state value of the server such as system time and number of current processes. Whenever a new master key is generated, the randomizer uses the same master key for the entire session. This ensures that longer sessions do not break due to the change of the master key, since the change of the master key during each session may cause errors for the de-randomization.

Generate Client Session Key. After generating the master key, for each session, the randomizer transforms the master key to a specific client session key. This client session key is generated by hashing the string of the combination of the master key and some specific client-side identifiers such as IP address, tcp port number or session ID. The hashing algorithm can be SHA1 or MD5.

Generate Randomized Parameter Values. After obtaining the client session key, the randomized parameter values can be calculated by using the symmetric "XOR" cipher on the R-tagged parameter value and the client session key. We adopt the simple "XOR" cipher, due to its simplicity and high efficiency.

Append R-tag. The randomizer will finally append R-tags again with randomized parameter values and send them to the client. This step aims at providing the indicator that which elements need to be de-randomized by the de-randomizer.

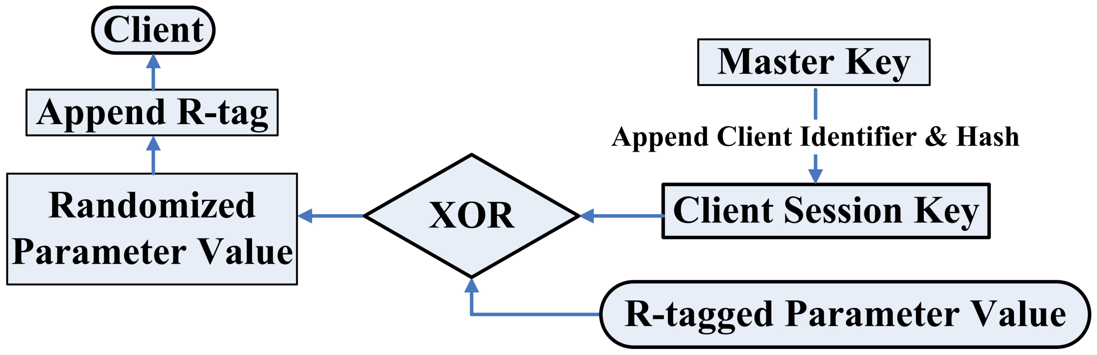


Fig. 8.2.: System flow of the Randomizer.

In this way, NOID does not need to keep any temporary data to carry out the randomization/de-randomization. The stateless nature of NOID ensures that it is lightweight and efficient. We next present how NOID de-randomizes users' submission according to their received randomized source.

8.4 De-randomization

Once users send a submission request[2] to the server according to their received randomized webpage source, the De-randomizer will analyzes reverse those "name/id" parameter values with R-tag prefix back to original parameter values. Figure 8.3 shows the system flow of the De-randomizer, which mainly contains four steps. (The details of De-Randomization algorithm can be referenced in Appendix A.)

Find R-tagged Parameter Values. When the De-randomizer receives users' requests, it will find out all parameter values of the HTML elements that have R-tag prefix. These values will be de-randomized later.

[2]The request can be an ajax call, a submitted form, or a HTTP GET request with randomized parameters.

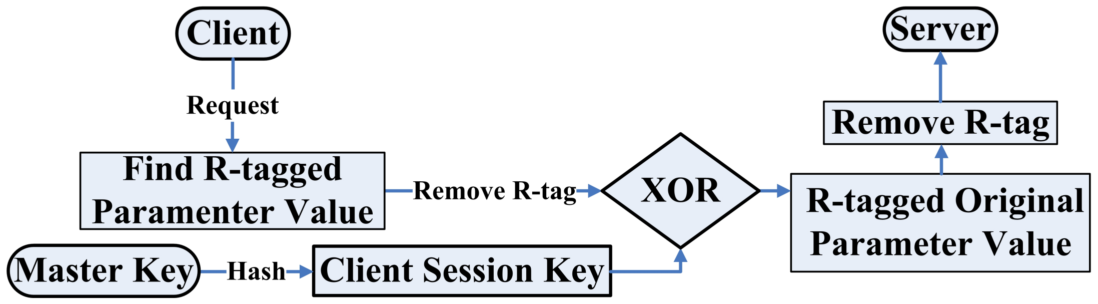


Fig. 8.3.: System flow of the De-randomizer.

Remove R-tags. Since the randomizer will append R-tag prefix to those randomized parameter values before sending them to the client. To de-randomize, the De-randomizer will remove those R-tag prefix in the parameter values first.

Calculate Client Session Key. Then, the De-randomizer will retrieve current master key and use client identifier information to calculate client session key. The way to calculate this client session key is similar to the one that used in the Randomizer. Thus, since both the master key and those identifier information will not change during the session, we can obtain the same client session key that is used to randomize parameter values.

Generate Original Parameter Values. In this step, the De-randomizer will generate original parameter values with R-tag prefix by using the "XOR" cipher on the randomized parameter values and the client session key. Then, the De-randomizer will remove R-tag prefix to generate original parameter values and send them back to the sever.

It is very important to realize that though this component on its own cannot prevent advance bots, this component is paramount for the success of the Obfuscation component. In its absence, the bot master can easily manually identify the critical elements and use the corresponding name parameter in automating requests. The randomization subsystem ensures that the values of the name/id parameters of the critical elements differ from session to session and are hard to predict. In fact the success of the entire system thus depends on the robustness and security of this subsystem.

8.5 Enhancing NOID

However, bots do not always use hardcoded name/id parameters of the HTML elements, advanced bots can retrieve the name/id parameters of the important HTML elements from the page source by parsing the source. In all, there can be three possible ways by which a bot identifies an HTML element of concern specially in a form.

Directly using a hardcoded (previously identified) name/id parameter value. The botmaster manually determines the name/id parameter values of the important elements and programs the bot to use them in bulk data processing requests. This is already defeated by the basic NOID.

Content Analysis of the source. Each HTMl form has label HTML elements to inform the end user about the purpose of input HTML elements in the form. The bots can then identify the correct input HTML elements in form by the content description in the label associated with the input element. We cannot randomize the label text because its visible and of importance to the end user.

Via context hints present in the source. A bot may be made aware of the relative location of the HTML element with respect to other elements in the page by the botmaster. Hence, identifying and retrieving the correct name/id parameter boils down to moving through the DOM structure along a pre-identified path.

In this section, we present techniques which enhance the basic NOID and make it more robust and secure. (The details of Label Concealer and Element Trapper algorithms can be referenced in Appendix A.)

8.5.1 Label Concealer

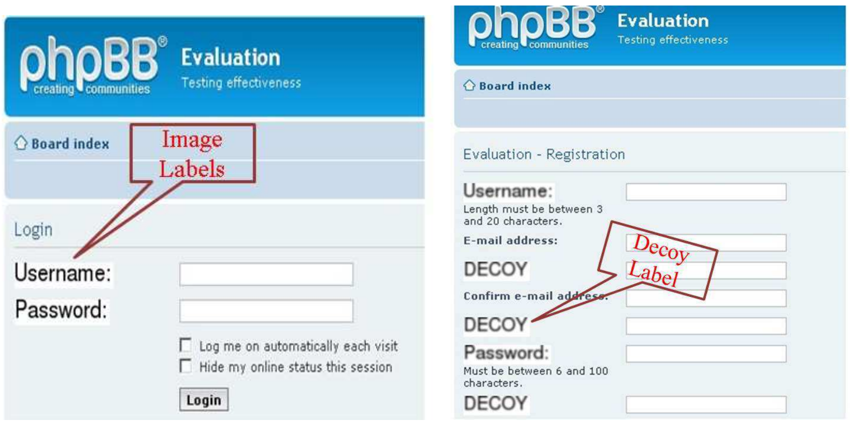
Insertion of Image Labels in Original Elements. A label text is a string inside "<label></label>" HTML tags associated with a HTML element where the string specifies the purpose of the element to the end user. The association is made by either setting the "for" attribute of the label tag "$<$ $l$abel for=id$>$" to the id of the element or by placing the element between the label start and end tags. For example, in a typical login form, the label texts “Username” and “Password” are placed near the corresponding textbox to facilitate the user to fill up the login form correctly. NOID cannot change the content of the label text as its visible and of importance to the end user. A bot on the other hand could use this label text to figure out the name/id attributes of the original HTML element. NOID uses Image substitutions to render this mode of bot operation ineffective.

The image label are created in real time, in our implementation we use ImageMagick - a free Image processing library to create image labels. The overhead from image generation is small and we discuss it further in the Evaluation Section 9. The label text is substituted by an image of the label text with random noise addition. This introduces another hard AI problem for the bot, which now has to use state of the art computer vision techniques to figure out its content.

8.5.2 Element Trapper

Creation of Decoy Elements. Decoy elements are nothing but duplicates of the original element with a random name/id attribute, introduced randomly near the original element in the source. In order to prevent confusion among the normal users these decoy elements are either hidden or have an accompanying label image indicating its decoy nature. Decoy elements can be hidden from the web page by setting the “display” or “position” property via css statements in the source. A bot could potentially still parse the css of the web page for all candidates (original and decoy) to determine the original one. Hence, NOID also introduces decoy elements which would be visible to the normal user but with an accompanying label image indicating its decoy nature. A visible decoy elements with an accompanying image label would indicate to the normal user that he does not need to interact with it. The decoy image label could either have a text saying “Do not fill this”, “invalid”, “decoy” etc. or it may simply be a symbol. There are different ways of indicating this to the user and may vary with web sites and geography.

Identifying the decoy image labeled elements is an easy cognitive task for humans but a tough AI challenge for the bots. A simple task of identifying even a single element of interest would require the bot to solve multiple CAPTCHA-like problems increasing the cost of filling up a simple form manifold and hence unsustainable. Moreover, by introduction of noise and randomly changing the decoy label text/symbol NOID could make the challenge even more difficult for bots. In our design, NOID uses both Hidden and Image Labeled decoy elements, with each of them configurable to be on/off. Figure 8.4 shows phpBB pages where NOID obfuscates the element location by decoy element insertion and conceals label text with images.


phpBB creating communities
Evaluation
Testing effectiveness
Board index
Image Labels
Login
Username:
Password:
Log me on automatically each visit
Hide my online status this session
Login
phpBB creating communities
Evaluation
Testing effectiveness
Board index
Evaluation - Registration
Username:
Length must be between 3 and 20 characters.
E-mail address:
Decoy Label
DECOY
Confirm e-mail address:
DECOY
Password:
Must be between 6 and 100 characters.
DECOY

(a) Concealing label text (b) Decoy element insertion

Fig. 8.4.: NOID inserting decoy elements with accompanying image labels and concealing original labels.

8.5.3 Effectiveness of Enhanced NOID

The continuing success of the text based CAPTCHA's have proved that it is a difficult problem to solve. NOID, with the use of its decoy element and label concealer insertions, multiplies the complexity and cost of submitting a web form for bots. A bot now has to solve multiple CAPTCHA-like challenges instead of just one. Moreover, since a normal user does not need to explicitly solve a CAPTCHA, it improves usability. In case of a normal CAPTCHA, both the bots and humans have to explicitly pass a test to be considered human. Whereas with NOID, we present challenges only for the bots. The challenges are implicit and do not warrant a written solution from the end user.

9. NOID EVALUATION

In this section, we evaluate our designed techniques of defending against web bots by implementing a prototype system of NOID. We next present our implementation, experiment setup and evaluation methodology.

9.1 Implementation

NOID is designed to be incorporated to existing web sites without any major changes. Most of the web sites behind a proxy can be stacked on the proxy without making any substantial infrastructure changes. Thus, we implement our prototype of NOID as an intercepting proxy, which sits in the middle between the server and the client and relays HTTP communications between them. As presented in Section 8, aiming at providing flexibility, each component of NOID - Randomizer/De-randomizer, Element Trapper and Label Concealer can be configured on/off dynamically.

To achieve this goal, we modify the source code of a powerful web proxy called Privoxy [44] to create a NOID enabled proxy. Privoxy is an open source and non-caching web proxy with advanced filtering capabilities, allowing modifications of web pages and HTTP headers. It is primarily meant to be designed as a client side proxy, which is used to filter out banners and ads from the web page. It still has the capability to be run in an intercepting mode. And all of Privoxy's actions of filtering ads and banners are defaultly turned off. The source code is modified to include a module for NOID. We use Pcre [45] module in the Privoxy to search, replace and substitute content, through implementing perl compatible regular expressions in C. The pcre module allows us to use regular expressions to search for R-tag annotated parameters and to replace them with randomized values. The enhanced components of NOID is implemented by using ImageMagick's C API, which allows for creating small thumbnail type label images and introducing noise. The decoy elements and image labels are added to HTTP form pages by issuing substitution commands through Pcre. We then hooked NOID in Privoxy's request/response processing module.

The HTTP form pages are buffered, modified and then sent to the client. For evaluation, the web servers incorporating with our prototype are setup in a Virtual Machine.

9.2 Experiment Setup

To evaluate our prototype of NOID, we incorporate it as a simple intercepting proxy in local installations of four popular web platforms: phpBB, Simple Machine Forums (SMF) and Wordpress and Buddypress [46]. PhpBB and SMF are two of the most popular open-source forum/discussion platforms [47]. Wordpress starts as a blogging service but has evolves to be a powerful platform to design websites and content management system [48–50]. BuddyPress is a open source endeavour to provide easy setup of websites with social networking features, which is built upon Wordpress. Since its debut in 2008, it has been adopted by a number of websites such as Solo Practice University and hMag [36]. However, web bots have misused those free services offered by these open-source platforms and have targeted on those websites built upon these platforms and caused great displeasure among to legitimate users [40, 41]. These four web platforms are chosen with an aim at representing most popular, state-of-the-art and open-source systems, which have a history of being plagued by bots.

For web bots, we evaluate our prototype of NOID according to five state-of-the-art web bots: XRumer, Magic Submitter, Ultimate Wordpress Comment Submitter (UWCS), SENuke and Comment Blaster. XRumer is one of the most effective and widely used free web bot [37]. Magic Submitter allows users for automatically submitting bulk messages on blogs and Online Social Networks such as Facebook and Twitter [51]. Particularly, Table 9.1 shows specific types of web platforms that are targeted those web bots, respectively. Ultimate Wordpress Comment Submitter and SENuke are two commercial web bots mainly designed for the purpose of SEO. Comment Blaster is a tool that allows users to automatically send bulk comments or messages on the web platforms.

After setting up the experiments, we evaluate our prototype of NOID according to its Security Effectiveness and Performance Overhead.

Table 9.1: Experiment setup of web bots and their targeted web platforms.

Platform	Platform Type	Web Bot
PHPBB	Forum, Content Management	XRumer, Magic Submitter, SENuke
Simple Machine Forums (SMF)	Forum, Content Management	XRumer, Magic Submitter, SENuke
WordPress	Blogs, Websites	XRumer, Magic Submitter, SENuke, UWCS, Comment Blaster
BuddyPress	Social Networking Addon to Wordpress	XRumer, Magic Submitter, SENuke, UWCS, Comment Blaster

9.3 Security Effectiveness

NOID is evaluated on forum and blogging instances of popular open source systems against six state-of-the-art web bots. The settings on these created web site instances are liberal, allowing guest/unregistered users to create threads and to post comments. The web bots XRumer, Magic Submitter, UWCS, Comment Blaster and SENuke can run against these instances to establish effectiveness. All of the above bots succeeded in creating new posts/comments automatically.

As describe in Section 9.2, NOID is then incorporated as an intercepting proxy in the server side for the web sites. The source code of important web pages such as Login page, Thread Posting page, Comment page and Registration page of these instances are modified to annotate critical HTML elements with R-tags. We then run these bots to register, login and post threads/comments on NOID enabled instances of the web sites. As seen in Table 9.2, NOID is able to completely defend agaisnt all of the above bots. Particularly, UWCS and Comment Blaster target blogs and hence could not be evaluated on forum platforms.

Table 9.2: The effectiveness of using NOID to defend against different web bots on different web platforms. "Yes" implies that NOID can successfully defend against the specific bot on that platform. "N/A" implies that the specific bot does not target on the respective platform.

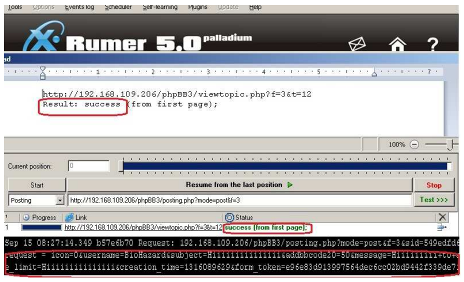
Web Platform	Type	XRumer	Magic Submitter	SENuke	UWCS	Comment Blaster
phpBB	Forum, Content Management	Yes	Yes	Yes	N/A	N/A
SMF	Forum, Content Management	Yes	Yes	Yes	N/A	N/A
WordPress	Blogs, Websites	Yes	Yes	Yes	Yes	Yes
BuddyPress	Social Networking Addon to WordPress	Yes	Yes	Yes	Yes	Yes

To validate that NOID without enhanced components would be effective enough to counter most of current bots, we present a component wise evaluation on Xrumer. Figure 9.1 shows XRumer's status according to different NOID configurations. Figure 9.1(a) shows a successful attempt by XRumer on a unmodified phpBB3 thread creation page with NOID OFF. XRumer first issues a request and then verifies that the post was successful. The lower inset of each figure shows the request body captured in privoxy. Figure 9.1(b) then shows that with NOID turned on and R-tag appended to the thread creation form elements, XRumer fails to identify the html elements. This results in an empty request body as shown in the lower inset of Figure 9.1(b). The status unknown signifies that XRumer had reached an unknown page which it cannot identify. We then turned off the Label Concealer feature of the Obfuscation subsystem of NOID and tried to create a thread using XRumer. With only the Randomization and Element Trapper components, NOID was able to defeat XRumer as we can see from Figure 9.1(c). We then turned off the entire Obfuscation subsystems of NOID (i.e., Label Concealer and Element Trapper) and with only randomization active, NOID was again successful in defeating XRumer as seen in Figure 9.1(d).

9.4 Performance Overhead

Page Loading Time. We first examine time overhead of NOID by quantifying the metric of "Page Loading Time (PLT)", which is the time interval (in seconds) between the timestamp of sending the request and the timestamp of completely loading a webpage by the browser. Specifically, we use Mozilla Firefox's addon LORI [52] to calculate the time used to load and display an entire page. Thus, a higher PLT increased by using NOID implies a bigger overhead. Table 9.3 shows PLT used for loading different types of webpages on three web platforms, when NOID (without Label Concealer) is off and on. NOID without Label Concealer contains only the Randomization and Element Trapper (hidden decoy element) components. This table also records the number of elements that are R-tagged, which may affect the values of PLT.

(a)Success

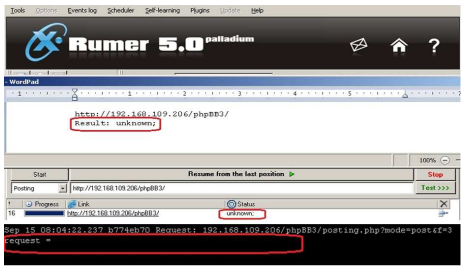


```
http://192.168.109.206/phpBB3/viewtopic.php?f=3&t=12
Result: success (from first page);
```

```
1   http://192.168.109.206/phpBB3/viewtopic.php?f=3&t=12   success (from first page);
```

```
Sep 15 08:27:14.349 b57e6b70 Request: 192.168.109.206/phpBB3/posting.php?mode=post&f=3&sid=549edfd
request = icon=0&username=BioHazard&subject=Hiiiiiiiiiiiiiii&addbbcode20=50&message=Hiiiiiiiii+to+
_limit=Hiiiiiiiiiiiiiii&creation_time=1316089629&form_token=e96e83d913997564dec6cc02bd9442f339de7
```

(b)Unknown

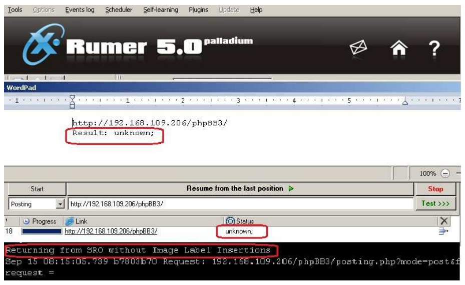


```
http://192.168.109.206/phpBB3/
Result: unknown;
```

```
16   http://192.168.109.206/phpBB3/   unknown;
```

```
Sep 15 08:04:22.237 b774eb70 Request: 192.168.109.206/phpBB3/posting.php?mode=post&f=3
request =
```

(c)Unknown

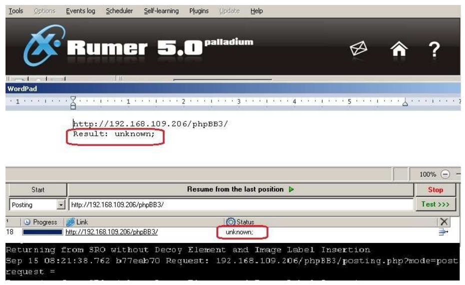


```
http://192.168.109.206/phpBB3/
Result: unknown;
```

```
18   http://192.168.109.206/phpBB3/   unknown;
```

```
Returning from SRO without Image Label Insertions
Sep 15 08:15:05.739 b7803b70 Request: 192.168.109.206/phpBB3/posting.php?mode=post&f
request =
```

(d)Unknown



```
http://192.168.109.206/phpBB3/
Result: unknown;
```

```
18   http://192.168.109.206/phpBB3/   unknown;
```

```
Returning from SRO without Decoy Element and Image Label Insertion
Sep 15 08:21:38.762 b77eab70 Request: 192.168.109.206/phpBB3/posting.php?mode=post
request =
```

Fig. 9.1.: Snapshots of requests from XRumer sent to the server to post a new phpBB thread. (a) shows a successful post when NOID is OFF, (b) shows a failed attempt when NOID (with enhanced components) is ON, (c) shows a failed attempt, when Label Concealer is OFF and (d) shows a failed attempt when NOID (without enhanced components) is ON.

Table 9.3: PLT for different types of pages, when NOID is off/on. (Time is in seconds.)

Platform	Page	R−tag	PLT (Off)	PLT (On)
phpBB	New Thread	6	2.554	3.012
	Post Reply	6	2.536	3.206
	User Login	5	1.036	1.96
Wordpress	Comments	5	0.8	1.412
	Register	3	0.799	0.991
	Login	4	0.785	0.921
Buddypress	Login	4	0.75	0.9
	Comment	5	0.891	1.521

We can see that the average overhead per page caused by NOID is acceptable, which is 0.47 seconds. In fact, the number of R-tagged elements affects the times of randomizing parameter values, inserting decoy elements, and concealing label elements. Thus, the number of R-tagged elements will have a direct impact on the PLT. From Table 9.3, we can also calulate that the average overhead per R-tagged element is around 0.08 seconds.

We next evaluate PLT overhead of complete NOID with and without Label Concealer. We acknowledge that the more images inserted by NOID, the more time is needed to create/process images at server side, download additional image bits from the server and load them into the browser at the client. This overhead is again linear to the number of R-tagged elements. Table 9.4 shows Page Loading Times (PLT) that are used for different types of pages, when NOID is enhanced without/with inserting images.

Table 9.4: PLT for different types of pages, when Label Concealer OFF and ON. (Time in seconds.)

Platform	Page	R−tags	PLT (Off)	PLT (on)
phpBB	New thread	6	3.012	3.699
	Post Reply	6	3.206	3.869
	User Login	5	1.96	3.463
Wordpress	Comment	5	1.412	1.920
	Register	3	0.991	1.779
	Login	4	0.921	1.785
Buddypress	Login	4	0.9	1.750
	Comment	5	1.521	2.810

We find that the average overhead of enhanced NOID with image insertions (Label Concealer) is 0.894 seconds, which is still relatively acceptable. Especially, our implementation is based on a virtual machine with limited processing capability. We expect the performance to be better on a real server. In addition, since each PLT is collected using the browser with empty cache, the time overhead can decease even more in a real-world communication session with cache information.

WebPage Size. Since enhanced NOID inserts decoy elements and image labels in the webpages, we next examine the increased sizes of webpages by using enhanced NOID. Specifically, we also use Mozilla Firefox's addon LORI [52] to measure the webpage size. It is noted that the number of R-tagged elements will affect the number of increased webpage size. Table 9.5 shows the sizes of different types of pages on phpBB3, when

NOID is off and on. (All these webpages have the same number of R-tagged elements, which is 6.)

Table 9.5: The sizes of different types of pages on phpBB3, when NOID is Off/On.

Page	**Size (Off)**	**Size (On)**
New Thread	60.94KB	129.79KB
Post Reply	62.24KB	128.56KB
User Login	29.69KB	95.30KB
Register	22.57KB	93.73KB

We find that the average increased size of these web pages with 6 R-tagged elements is 67.99 KB, which is also acceptable, especially with the consideration that the size of a typical label image is round 8 to 15 KB. In addition, the page size overhead of NOID would not increase, due to the increased size of original web page source.

10. CONCLUSION

In this work, we present SEMAGE (semantically matching images) and a Source Randomization and Obfuscation engine to prevent web bots from illegitimately accessing web resources. We propose SEMAGE as an improvement on robustness and usability from the current CAPCTHA systems. With NOID, we introduce a new mechanism to defeat bots, build entirely around the operating mechanics of bots. Our threat model includes every possible way a bot could issue an automated request by analysing the web page and NOID aims to defeat all of the operating modes.

SEMAGE aims to be a effective and user friendly barrier and presents a set of candidate images and asks user to choose a set of images that fit a certain relation. The challenge is layered in that both knowledge about semantic meaning of images and relationship between the subjects of images is required. The challenge comes naturally to humans as it incorporates light-weighted visual and cognitive task. However, the layering scheme provides double protection against bot attacks. It is easy to understand and the interaction interface is simple and efficient. CAPTCHA systems constantly seek an optimum trade-off point on security and usability. SEMAGE provides great room for customization by the website administrators. They can tune on various factors such as the size of candidate images size, answer size to adjust the usability and security level according to the need of particular website. Moreover SEMAGE can be targeted towards touch based smartphones and devices where typing to solve a text based CAPCTHA is difficult. Website administrators can also determine the content of the images and cater towards their promotional needs. The database can be populated especially for SEMAGE, or adapted from existing database. E-commerce is one area where SEMAGE database can be easily built and SEMAGE can be utilized for both security and advertisement purposes.

In this work, we also introduce a novel server-side and non-intrusive web bot defense system, named NOID. It is designed to pose *implicit* challenges to web bots and not to require legitimate users to explicitly solve a challenge as a CAPTCHA does. We also provide enhanced NOID by adding two more components aiming at defeating more powerful web bots to automatically send bulk requests to the web servers. To be more flexible, NOID's components can be customized and turned on/off dynamically according to the requirements of a particular website. Our evaluation results show that NOID can defeat current state-of-the-art web bots on popular web platforms. Also, the performance overhead of NOID is acceptable for better security protection and usability.

11. LIMITATIONS AND FUTURE WORK

Generating a vast and correct database is always a challenge for image-based CAPTCHA systems. In our simple SEMAGE implementation we crawl the web to automatically gather and label images. However not all images returned by the crawler were relevant, some were even objectionable. We then manually weeded out the irrelevant images. Such manual labor is time consuming and would pose a big problem when the database content is regularly updated. There can also be legal issues in directly using the crawled images.

SEMAGE by the virtue of its design though, does not require the database to be built in such a way. Websites like e-commerce services, movie rental services can easily use the available image database with a suitable "semantic relationship". However, further work is required to create a large, correct database automatically to allow widespread deployment in real world.

In this work we introduced the concept and technique of creating CAPTCHAs using "semantic relationships" between objects and then implemented a simple system for demonstration. Our implementation does not reach the full potential of SEMAGE and we plan to build a more robust, high semantic correlation based SEMAGE systems as future work.

Web frameworks are complicated and enabling NOID for a specific web page and identifying critical HTML elements might need thorough understanding of the framework. We plan to develop an Automated Critical Element Identifier and Tagger as one future work. In this work, we did not consider accessibility properties of an HTML element such as tabindex which allow the user to use the tab key to navigate among the elements. Potentially an attacker can establish one to one relationships between tabindex value and HTML elements. Either tabindex can be disabled or its values could be randomly changed to deter such approaches.

The probability of random guess attack is seemingly high for pages with only one or two r-tagged elements. The alternative is for the developer to r-tag other non-critical (or all) elements in the page. Also, with all the components of NOID turned on, multiple insertions of decoy elements further decrease the success probability of random guess attacks.

The page size overhead from NOID may look substantial for small pages but with the increasing network capacity and speed we believe that a $60 - 70$ KB overhead is totally acceptable. With image label insertions being the main source of Page Load Time as well as page size overhead, we propose two possible ways of improving upon it: (1) To improve timing performance, developers can always choose to create images in advance and NOID could just insert noise. (2) NOID is highly configurable; we could only use image label insertions when necessary (e.g., for suspicious users).

REFERENCES

[1] H. S. Baird and K. Popat. "Human Interactive Proofs and Document Image Analysis". *Proc. of the 5th International Workshop on Document Analysis Systems*, Sept. 2002.

[2] L. V. Ahn, M. Blum, and J. Langford. "Telling Humans and Computers Apart Automatically". *Commun. ACM, vol. 47, pp. 56-60*, Feb. 2004.

[3] L. V. Ahn, M. Blum, N. J. Hopper, and J. Langford. "CAPTCHA: Using Hard AI Problems for Security". *Proc. of Eurocrypt*, May 2003.

[4] Y. Rui and Z. Liu. "Excuse Me, But Are You Human?". *Proc. of the eleventh ACM international article on Multimedia*, Nov. 2003.

[5] K. Chellapilla, K. Larson, P. Simard, and M. Czerwinski. "Designing Human Friendly Human Interaction Proofs (HIPs)". *Proc. of the SIGCHI article on Human Factors in Computing Systems*, June 2005.

[6] J. Yan and A. S. El Ahmad. "A Low-Cost Attack on a Microsoft CAPTCHA". *Proc. of the 15th ACM article on Computer and Communications Security*, Oct. 2008.

[7] G. Mori and J. Malik. "Recognizing Objects in Adversarial Clutter—Breaking a Visual CAPTCHA". *Proc. of the article on Computer Vision and Pattern Recognition*, June 2003.

[8] K. Chellapilla and P. Simard. "Using Machine Learning to Break Visual Human Interaction Proofs (HIPs)". *Advances in Neural Information Processing Systems*, Dec. 2005.

[9] "Breaking Text Captcha". `http://www.blackhat-seo.com/2008/how-to-break-captchas/`, Dec. 2010.

[10] A. S. El Ahmad, J. Yan, and L. Marshall. "The Robustness of a New CAPTCHA". *Proc. of the Third European Workshop on System Security*, Sept. 2010.

[11] Y. Shin, M. Gupta, and S. Myers. "Prevalence and Mitigation of Forum Spamming". *Proc. of INFOCOM*, Apr. 2011.

[12] J. Elson, J. R. Doucerur, J. Howell, and J Saul. "Asirra: A CAPTCHA that Exploits Interest-Aligned Manual Image Categorization". *Proc. of the 14th ACM article on Computer and communications security*, Oct. 2007.

[13] "ESP-PIX". `http://server251.theory.cs.cmu.edu/cgi-bin/esp-pix/esp-pix`, Sept. 2010.

[14] M. Chew and J. D Tygar. "Image Recognition CAPTCHAs". *Proc. of the 7th International Information Security article (ISC)*, Sept. 2004.

[15] R. Gossweiler, M. Kamvar, and S. Baluja. "What's Up CAPTCHA?: A CAPTCHA Based on Image Orientation". *Proc. of the 18th International article on World Wide Web*, Apr. 2009.

[16] R. Datta, J. Li, and J. Z. Wang. “IMAGINATION: A Robust Image-Based CAPTCHA Generation System”. *Proc. of the 13th Annual ACM International article on Multimedia*, Nov. 2005.

[17] “SQ-PIX”. `http://server251.theory.cs.cmu.edu/cgi-bin/sq-pix`, June 2010.

[18] P. Matthews and C. C. Zou. “Scene Tagging: Image-Based CAPTCHA Using Image Composition and Object Relationships”. *Proc. of the 5th ACM Symposium on Information, Computer and Communications Security*, Apr. 2010.

[19] B. B. Zhu, J. Yan, Q. Li, C. Yang, J. Liu, N. Xu, M. Yi, and K. Cai. “Attacks and Design of Image Recognition CAPTCHAs”. *Proc. of the 17th ACM article on Computer and Communications Security*, Oct. 2010.

[20] P. Golle. “Machine Learning Attacks Against the Asirra CAPTCHA”. *Proc. of the 15th ACM article on Computer and Communications Security*, Oct. 2008.

[21] E. Bursztein, S. Bethard, C. Fabry, D. Jurafsky, and J. C. Mitchell. “how good are humans at solving captchas? a large scale evaluation”. *Proc. of 2010 IEEE Symposium on Security and Privacy (Oakland’10)*, May 2010.

[22] T. Y. Chan. “Using a Text-to-Speech Synthesizer to Generate a Reverse Turing Test”. *IEEE International article on Tools with Artificial Intelligence*, Nov. 2003.

[23] J. P. Bigham and A. C. Cavender. “Evaluating Existing Audio CAPTCHAs and an Interface Optimized for Non-Visual Use”. *Proc. of the 27th International article on Human Factors in Computing Systems*, May 2009.

[24] “Audio and Visual CAPTCHA”. `http://www.nswardh.com/shout/`, Sept. 2010.

[25] H. Gao, H. Liu, D. Yao, X. Liu, and U. Aickelin. “An Audio CAPTCHA to Distinguish Humans from Computers”. *Proc. of the 2010 Third International Symposium on Electronic Commerce and Security*, Sept. 2010.

[26] “GIMPY Project”. `http://www.captcha.net/captchas/gimpy/`, June 2010.

[27] J. Yan and A. S. El Ahmad. “Usability of CAPTCHAs or Usability Issues in CAPTCHA design”. *Proc. of the 4th Symp. on Usable Privacy and Security*, July 2008.

[28] “reCAPTCHA Official Site”. `reCaptchaOfficialSite:http://www.google.com/reCAPTCHA`, June 2010.

[29] E. Bursztein, R. Beauxis, H. S. Paskov, D. Perito, C. Fabry, and J. C. Mitchell. “The Failure of Noise-Based Non-Continuous Audio Captchas”. *IEEE Symposium on Security and Privacy*, May 2011.

[30] G. Mishne. “Blocking Blog Spam with Language Model Disagreement”. *Proc. of the First International Workshop on Adversarial Information Retrieval on the Web (AIRWeb)*, Sept. 2005.

[31] A. Bhattarai, V. Rus, and D. Dasgupta. "Characterizing Comment Spam in the Blogosphere Through Content Analysis". *IEEE Symp. on Computational Intelligence in Cyber Security*, Apr. 2009.

[32] T. Schluessler, S. Goglin, and E. Johnson. "Is a Bot at the Controls?: Detecting Input Data Attacks". *Proc. of the 6th ACM SIGCOMM Workshop on Network and System Support for Games*, Sept. 2007.

[33] D. Brewer, K. Li, L. Ramaswamy, and C. Pu. "A Link Obfuscation Service to Detect Webbots". *IEEE International article on Services Computing, vol. 0, pp. 433-440*.

[34] R. Guha, R. McCool, and E. Miller. "Semantic Search". *Proc. of the 12th International article on World Wide Web*, Apr. 2003.

[35] L. V. Ahn. *"Human Computation"*. Phd. dissertation, Carnegie Mellon Univ., Sept. 2005.

[36] "SPAM Injection". `http://www.web-form-buddy.com/html-wfb/spam-injection.html`, Sept. 2010.

[37] Y. Shin, M. Gupta, and S. Myers. "The Nuts and Bolts of a Forum Spam Automator". *Proc. of the 4th USENIX article on Large-scale Exploits and Emergent Threats*, Feb. 2011.

[38] "Web Form Spam Alive and Kicking". `http://blog.trendmicro.com/web-form-spam-alive-and-kicking/`, Dec. 2010.

[39] Y. Niu, Y. Wang, H. Chen, M. Ma, and F. Hsu. "A Quantitative Study of Forum Spamming Using Context Based Analysis". *Proc. Network and Distributed System Security (NDSS) Symposium*, Feb. 2007.

[40] "Flooded by Spam/Splog Registrations". `http://wordpress.org/support/topic/plugin-buddypress-unusable-flooded-by%-spamsplog-registrations`, May 2011.

[41] "Automatic Registrations and Spam on Wordpress/Buddypress". `http://buddypress.org/community/groups/how-to-and-troubleshooting/forum/topic/automatic-registrations-spam/`, Sept. 2010.

[42] "Form Hijacking". `http://anders.com/projects/sysadmin/formPostHijacking/`, Nov. 2010.

[43] "Poll Bots Request and Use". `http://www.scriptlance.com/projects/1290407615.shtml`, May 2010.

[44] "Privoxy". `http://www.privoxy.org/`, May 2011.

[45] "PCRE". `http://www.pcre.org/`, May 2011.

[46] "BuddyPress - Social networking in a Box". `http://buddypress.org/`, May 2011.

[47] "PHPBB - the Most Widely Used Open Source Bulletin Board System in the World". `http://www.phpbb.com/`, May 2011.

[48] "Wordpress - WordPress is Web Software You Can Use to Create a Beautiful Website or Blog". `http://wordpress.org/`, May 2011.

[49] "Wordpress Powers 14.7 Percent of World's Sites". `http://www.h-online.com/open/news/item/WordPress-powers-14-7-per-cent-of-the-top-million-%web-sites-1327356.html`, May 2011.

[50] "Top Powered by Wordpress Sites". `http://www.tripwiremagazine.com/2009/11/20-remarkable-examples-of-websites-powered-by-wordpress.html`, May 2011.

[51] "Magic Submitter". `http://www.magicsubmitter.com/`, May 2011.

[52] "Lori: Life-of-Request Info". `https://addons.mozilla.org/en-US/firefox/addon/lori-life-of-request-info/`, May 2011.

APPENDIX A
NOID ALGORITHMS

Algorithm 2 : Algorithm for Randomization.

```
$response \leftarrow$ HTTP response body
**if** $response$ contains r-tag **then**
   $listParameters \leftarrow$ r-tag annotated parameters from $response$
**else**
   Return $response$
**end if**
{/* Create the final key */}
$masterKey \leftarrow$ Retrieve current Master Key
$tmpKey \leftarrow masterKey +$ client specific ipaddr/port/sessionid
$finalKey \leftarrow hash(tmpKey)$
{/* Randomize the r-tagged parameters, substitute in response and send to client */}
$i \leftarrow 0$
$randomizedParam \leftarrow$ ""
**while** $num(listParameters) \neq i$ **do**
   $i \leftarrow i + 1$
   $param \leftarrow listParameters[i]$
   $randomizedParam \leftarrow param \oplus finalKey$
   {/* Prefix the r-tag again so as to let the de-randomizer know what to re-randomize
   */}
   $randomizedparam \leftarrow$ r-tag $+originalparam$
   substitute listParameter[i] in $response$ with $randomizedParam$
**end while**
Return $response$
```

Algorithm 3 : Algorithm for De-randomization.

```
$request \leftarrow$ HTTP request body
if $request$ contains r-tag then
    $listParameters \leftarrow$ r-tag annotated parameters from $request$
else
    Return $request$
end if
{/* Recreate the final key */}
$masterKey \leftarrow$ Retrieve current Master Key
$tmpKey \leftarrow masterKey+$ client specific ipaddr/port/sessionid
$finalKey \leftarrow hash(tmpKey)$
{/* De-randomize the r-tagged parameters to get the originals, substitute in request and
send to server */}
$i \leftarrow 0$
$originalParam \leftarrow$ ""
while $num(listParameters) \neq i$ do
    $i \leftarrow i + 1$
    $param \leftarrow listParameters[i]$
    $param \leftarrow$ strip r-tag from $param$
    $originalParam \leftarrow param \oplus finalKey$
    {/* Assuming that the server logic is not r-tag annotated, NOID need to send the
    actual parameter */}
    $originalparam \leftarrow$ strip r-tag from $originalparam$
    substitute listParameter[i] in $request$ with $originalParam$
end while
Return $request$
```

Algorithm 4 : Algorithm for Element Trapper and Label Concealer.

```
$response \leftarrow$ HTTP response body
**if** $response$ contains r-tag **then**
   $listParameters \leftarrow$ r-tag annotated parameters
   $listElements \leftarrow$ parse response and get strings of HTML input elements containing r-tags with label tag
**else**
   Return $response$
**end if**
$i \leftarrow 0$
$decoyElem \leftarrow$ ""
$element \leftarrow$ ""
$decoyParam \leftarrow []$
**while** $num(listElements) \neq i$ **do**
   $i = i + 1$
   **while** $element$ in $listElements$ **do**
      $decoyElem \leftarrow copy(element)$
      $decoyParam \leftarrow (r - tag + random(string))$ {Add the r-tag prefixed decoy parameter to list}
      $decoyElem \leftarrow$(Replace the r-tagged parameter in $decoyElem$ with $decoyParam$)
      $response \leftarrow$(Insert $decoyElem$ in $response$ randomly above or below the original element)
   **end while**
   **while** $param$ in $decoyParam$ **do**
      **if** $< RandomCondition >$ **then**
         $response \leftarrow$ (Insert CSS command to hide element with id $param$ and remove corresponding label tag)
      **else**
         $response \leftarrow$ Replace label text of element with $decoyParam$ with invalid image
      **end if**
   **end while**
   **while** $realParam$ in $listParam$ **do**
      $response \leftarrow$ Replace label text of element with $realParam$ with image
   **end while**
**end while**
Return $response$
```

VITA

Shardul Vikram received his Bachelor of Engineering degree in Electronics and Telecommunications from National Institute of Technology at Silchar in 2007. He entered the Computer Science program at Texas A&M University in September 2009 and received his Master of Science degree in December 2011. His research interests include web security and prevention of web bots.

Shardul may be reached at Building 88, Microsoft Corp, Redmond, WA, 98052. His email is shardul.vikram@hotmail.com.