CLICK-BASED CAPTCHA PARADIGM AS A WEB SERVICE

Abdalnaser Algwil [*]

Faculty of Information Technology, Alasmarya Islamic University, Zliten, Libya
** Corresponding author: a.algwil@it.asmarya.edu.ly*

ABSTRACT

Captcha has become a standard security mechanism by which to deter the misuse of services on the Internet. More precisely, text-based Captchas have been the most popular form for some considerable time. Although many Captchas have been broken, other improved forms of Captchas have been developed. Click-based Captcha is a new category of these improved schemes. However, most such Captcha schemes are just proposals presented in the literature and have not been produced as services in the real world. This paper presents how a Click-based Captcha scheme could be realized as a RESTful Web service that offers Captcha challenges via simple APIs to consumers.

Keywords: Captcha, Security, Web service.

1. INTRODUCTION

Some malicious automated programs are designed in order to abuse free online services on the Internet such as signing up for free webmail accounts, posting to forums and blogs, and unauthorized access to certain online resources. Therefore, Captcha, which is an acronym for ***C**ompletely **A**utomated **P**ublic **T**uring tests to tell **C**omputers and **H**umans **A**part*, is a technique that has rapidly developed in use on the Internet for the purpose of protecting the Internet services against adversarial attacks by malicious programs [1].

Although, text-based Captchas have been the most popular form, many of them have been broken [2]. This motived Captcha designers to make text-based Captchas very difficult to solve, even for ordinary humans. Unsurprisingly, with the ever-increasing evolution of the attacks and defenses, the distinguishable gap between machine and human abilities in deciphering traditional textual Captchas seems to be inadequate in its current form, often making these schemes either insecure against state-of-the-art attack technologies, or secure but unusable for humans.

On the other hand, several types of clicked-based Captchas have been introduced in the literature, especially on small-screen devices, which have become one of the most important means to access the Internet in recent years. However, a usable Captcha, for security purposes, should also be practicable in the real world. Practicality, in this context, refers to the requirement that Captcha challenges must be generated and evaluated automatically in real time, without the need for human intervention, and using a minimal amount of computational and network resources. In fact, practicality can significantly influence both the security and usability of a Captcha. That is, there are many successful side-channel attacks against Captcha schemes in which attackers have exploited fatal flaws in the design and implementation of the Captcha systems rather than solving the underlying AI problem such as an attack in [3]. Furthermore, a flawed implementation of a Captcha design can deteriorate the usability aspects significantly, for example, the following scenarios would have a negative influence on Captcha usability:

- When Captcha challenges need an unreasonable time to generate or validate,
- If Captcha application is incompatible with certain operating systems, browsers or devices, or
- Captcha implementation requires specific libraries or software which are universally unavailable for all client's browsers.

To the best of our knowledge, there is no published work in the literature that explains how to build a real web service for such click-based Captcha schemes. In this paper, we demonstrate the architecture of the REST-based Clickable Captcha service that can automatically generate and validate Captcha challenges without the need for human intervention.

The remaining paper is organized as follows: Section 2 presents Background and related work. Section 3 outlines Captcha practicality, including the reason behind a Captcha service, Rest-based Captcha architecture, and Captcha implementation and APIs. Section 4 concludes the paper.

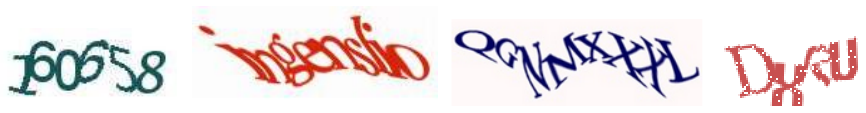
2. BACKGROUND AND RELATED WORK

Over the last two decades, several kinds of Captcha have been developed for equilibrating between usability and security issues. Generally speaking, the various types of Captcha can be classified under four categories [4], [5] as follows:

- Text-based schemes.
- Sound-based schemes.
- Image-based schemes.
- Cognitive-based schemes.

2.1 Text-based Captcha

Text-based Captchas have been the most popular used to protect online services. In a text-based Captcha scheme, challenge appears as a distorted image of text and users are required to recognize and retype the text. However, many of text-based Captchas suffer from inadequacies in terms of *security*, *usability*, or the balance between them. Deficiencies in the *security* issue arise as a result of the fact that a large number of existing Captcha schemes currently deployed on the Internet have been successfully broken by attackers [6][7]. This was the key motivation for designers to use complicated anti-segmentation techniques that are resistant to automated attacks. However, this difficulty seems to be reaching its limits, and in turn negatively affects the *usability* of the scheme because it was also hard for humans to solve the challenges. Therefore, simultaneously achieving a reasonable equilibrium between both *usability* and *security* aspects in the same Captcha scheme is critical, but unfortunately remains an elusive issue. Fig.1 shows some samples of text-based Captchas taken from Google, Microsoft, Baidu and eBay, respectively.



(a) Google (b) Microsoft (c) Baidu (d) eBay

Fig. 1. Examples of Text-based Captcha Schemes

In recent years, various non-text-based Captchas have been designed to minimize user frustration and to be more suitable for touch-friendly input methods, particularly with the increasing use of mobile devices to access the internet. A report from comScore [8] reveals that mobile devices are poised to override ordinary computers as a key mechanism used by people to access the Internet. This report revealed that mobile devices account for more than one-third (approximately 37%) of time spent online.

Another study by Reynaga and Chiasson [9] found that existing Captcha schemes, in their present state, are not suitable for mobile device as a result of having considerable usability problems with regard to Smartphones that frustrate users and lead to errors. Accordingly, it is becoming increasingly difficult to ignore the use of Captchas on mobile devices.

2.2 Image-based Captcha

To solve an image-based Captcha challenge, a user is required to classify or recognize certain images or objects as distinct from others according to their properties. Although these tasks are fairly easy for humans, the current state-of-the-art in computer vision is still unable to accomplish many semantic image understanding tasks, such as describing image content in terms of objects, actions, feelings, and so on. In general, image-based Captcha can improve usability levels across different devices, as users usually interact with these systems using pointing devices such as a mouse, stylus, and so on.

Many Captcha proposals were introduced on image-based Captcha. For example, Implicit Captcha [10], in which the user is required to click on a particular part of a natural image (e.g., click on mountain top of the image depicted in Fig.2 (a)) or click on a specific word in an image comprised of a number of randomly distributed words.

Collage Captcha [11], is another Captcha proposal that is built on image semantics. In this Captcha scheme, users are asked to click a certain object picture's has been presented according to a text description, as shown in Fig.2 (b).

In the same context, Mohamad and Nisar [12] also proposed a new image-based Captcha, dubbed image flip Captcha, which is based on the classification of flipped images from non-flipped images. Fig.2 (c) depicts an example of the image flip Captcha.

Lin et al. [13] proposed a new Captcha scheme for mobile devices called "Captcha Zoo". In this scheme, two similar types of 2D animals (i.e., target animals and clutter animals) are generated from 3D models and randomly drawn on a green grass-textured background. Fig.2 (d) shows a sample challenge where the target animal is a horse and the clutter animal is a dog. A user passes the test by clicking on all the target animals (i.e., horses).

Touclick [14] is a universal Chinese Captcha scheme designed to be mobile friendly, and works as follows. Two Chinese characters are randomly placed on background picture. A hint which contains the two characters is displayed below the puzzle as shown in Fig.2 (e). Users are required to find out the two characters in the puzzle and click them in the same order as they appear in the hint.

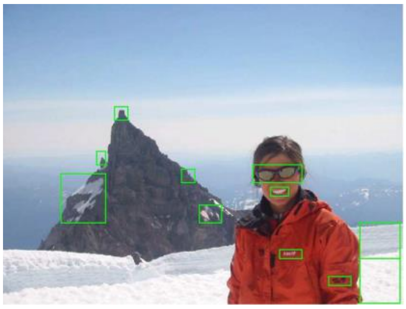
All of the above clickable captcha proposals that based on images are designed to be mobile-devices friendly.

2.3 Audio-based Captcha

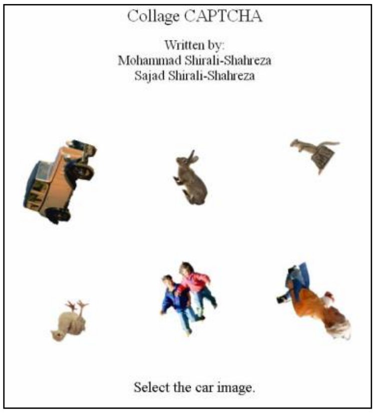
In this kind of Captcha, an audio clip composed of a sequence of distorted words or letters and/or digits spoken by several people at randomly spaced intervals is rendered to the user, who is prompted to transcribe the characters spoken in the audio clip. Audio-based Captcha schemes are usually used as a complementary alternative to visual-based Captcha tests for visually impaired users.

2.4 Cognitive-based Captcha

Cognitive Captcha schemes typically require semantic interpretations of their challenges that rely on the semantics of the images, language constructs, solving riddles, or on performing arithmetic operations. Without a doubt, these cognitive functions are very difficult for computers to solve. PlayThru Captcha [15], Math Captcha [16], and CAPTCHaStar [17] are examples on cognitive Captcha as shown in Fig.3.



(a) Implicit Captcha [10]

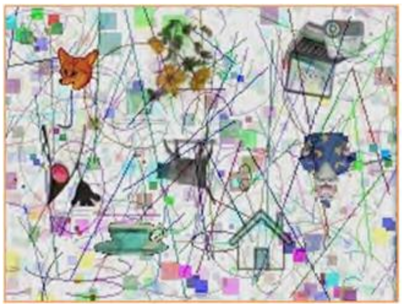


Collage CAPTCHA

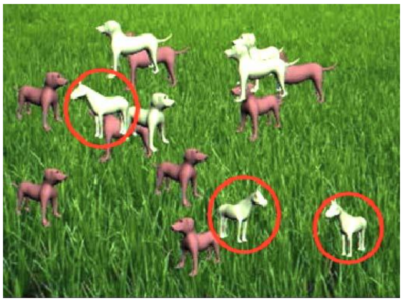
Written by:
Mohammad Shirali-Shahreza
Sajad Shirali-Shahreza

Select the car image.

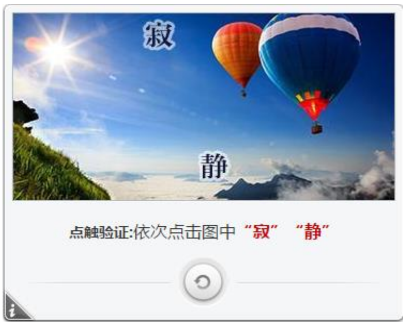
(b) Collage Captcha [11]



(c) Image Flip Captcha [12]



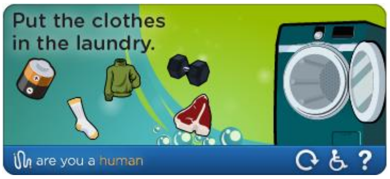
(d) Captcha Zoo [13]



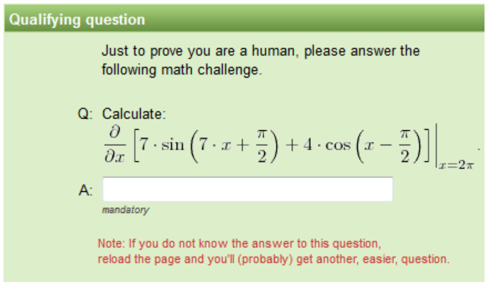
点触验证:依次点击图中 "寂" "静"

(e) Touclick Captcha [14]

Fig. 2. Examples of Image-based Captcha Schemes



(a) PlayThru Captcha [15]



Just to prove you are a human, please answer the following math challenge.

Q: Calculate:

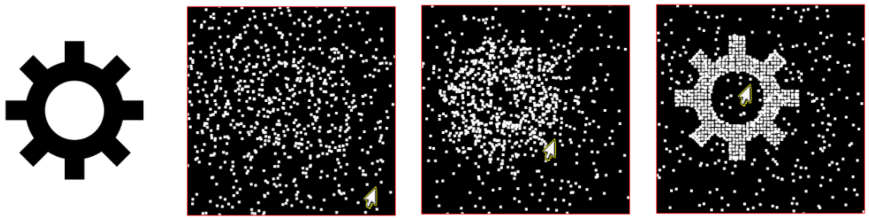
$$\frac{\partial}{\partial x}\left[7\cdot\sin\left(7\cdot x+\frac{\pi}{2}\right)+4\cdot\cos\left(x-\frac{\pi}{2}\right)\right]\Big|_{x=2\pi}$$

A:

mandatory

Note: If you do not know the answer to this question, reload the page and you'll (probably) get another, easier, question.

(b) Math Captcha [16]



(i) Sample Picture (ii) An Unsolved Challenge (iii) Almost Solved (iv) Correctly Solved

(c) CAPTCHaStar[17]

Fig. 3. Samples of Cognitive-based Captcha

There are many other alternative Captcha proposals that have been devised in the literature. In fact, it has become an active research area in the exploration of alternative Captcha designs. On the contrary, Captcha practicality has not been sufficiently highlighted. Hence, the next section presents how to develop a Captcha scheme as a RESTful Web service.

3. CAPTCHA PRACTICALITY

Given the importance of the practicality of any Captcha scheme, this paper presents how to realize a clickable Captcha scheme in practice. In fact, developing a Captcha service from scratch is not a trivial task, especially with scarce resources to do so. Nevertheless, the REST-based Web service has been developed to encapsulate any clickable Captcha scheme and provide a simple and consistent Application Programming Interface that offers Captcha challenges as a service to other web applications. However, before presenting the details of the captcha service, we will briefly explain what the Web service actually is, and why a Captcha service.

3.1 Web service

On the conceptual level, a Web service is a software system designed to provide interoperable application-to-application interaction over a network using standard web technologies and without human intervention. It allows different applications to communicate with each other, regardless of the programming languages, platforms, and operating systems originally used to develop them. This is because all communication and data exchange are done in standard formats such as XML or JSON.

The Web service, which is typically based on the consumer/provider model, consists of two main components: client (*consumer*) side and server (*provider*) side. In the Captcha context, the client side sends users' requests to fetch new Captcha challenges from the server side, renders them, and sends the users' answers back to the service provider for the verification. At the other end of the scale, the Captcha provider (*server*), which is responsible for the generation and validation of the Captcha challenges, receives and processes the clients' requests and then returns the results to the service consumer. Typically, any Web service provides Web APIs that permit clients to access various functionalities provided by the service.

Since architectural decisions in any distributed software design have significant impacts on performance, maintainability and scalability, the REST architectural style [18] has been used to develop the captcha design as a Web service. REST (REpresentational State Transfer) is a client/server architecture style that simply allows clients to request and access data and functionality (Resources) on the server using Uniform Resource Identifiers (URIs) - typically links on the Web - over HTTP as underlying protocol, and then receive proper responses in diverse data formats. Although, the REST-based services can support many data formats (e.g., XML, JSON, plain text, and so on), JSON is the most predominant one used in Web services to return responses.

In fact, REST-based Web services, or RESTful web services, are lightweight, simple, maintainable and scalable, and are becoming the most popular and predominant architecture employed by Web developers to build APIs for web-based applications. Service architects and developers prefer the RESTful approach as it is easier to build, maintain, extend and consume than using SOAP-based services. In addition, RESTful Web services are language and platform independent and can be straightforwardly developed with minimal or no developer tooling.

All interactions in RESTful service are stateless, which means that server-side components do not keep track of the application states of the clients; instead, each client request should include all the necessary data to be processed independently from any previous or other concurrent client requests. Since no request to the RESTful service is dependent on any other, any server on the service provider-side can receive and serve any client request, which in turn improves Web service performance and makes it more reliable, less complicated to design and implement, and horizontally scalable. Furthermore, owing to their lightweight nature, RESTful services are particularly useful and well suited for providing content to the limited bandwidth and resource-constrained devices such as mobiles.

3.2 Why a Captcha service

A Captcha widget can be installed in a Web page through one of three methods: *Firstly*, Web developers can build their own Captcha implementations. *Secondly*, Web designers can install one of the available pieces of Captcha software (either open source or commercial) on their websites. *Thirdly*, using an external Captcha Web service to generate and evaluate Captcha challenges such as the reCaptcha service from Google. In fact, the first approach is generally a bad idea, and not recommended at all as it is typically weak, costly, time-consuming and more prone to fatal flaws. Although the second method looks better than the first, it places an extra burden on both the webmaster and the application server. This means that the webmaster will be responsible for frequently following up the Captcha software updates to combat any cracks. Moreover, installing Captcha software in a Web application will increase the workload on the Web server. That is, the generation and validation of the Captcha is performed on the application server, which in turn consumes an increased amount of server resources (such as CPU and memory). However, the third method (using a Captcha service) seems to be preferred due to its simplicity. That is, the Web developers only need to add some small pieces of code to integrate a Captcha widget into their web page. The Captcha creation and evaluation is originally performed on a third-party server side. In fact, this service-driven approach for developing Captcha schemes has several advantages over other approaches:

- It offers Captcha challenges that are more reliable, usable and robust. The Captcha service is typically developed by experts who are familiar with the current techniques of Captcha breaking as well as flaws and vulnerabilities in the Captcha design that could be exploited by attackers.

- Using a Captcha service can reduce the workload of a Web application server. That is, the generation and evaluation processes of the Captcha are performed on the Captcha service provider's server side. This in turn allows for significant savings of the application server's resources, especially with high-traffic websites that need large numbers of Captcha challenges daily.
- The Captcha service usually provides a simple API that allows web developers to integrate Captcha challenges into their web pages with only a few lines of code, without having to consider complex algorithms or implementation details. Moreover, the API decouples service consumers from the service implementation so that the service implementation can be modified without having to change client code as long as the service API does not change.
- All Captcha system software is hosted on the Captcha service provider's server side, which in turn can bring with it many benefits to service consumers. For example, Captcha updates can be deployed automatically, quickly and easily so that webmasters do not have to worry about dealing with Captcha penetration or following-up with updates. Another benefit would be that the generation of the Captcha image in most Captcha schemes requires specific libraries or software (e.g., Image Processing Library), which are basically hosted/installed on the Captcha server side and therefore do not need to be distributed or installed on the consumer servers.

- Some Captcha services are offered for free, such as reCaptcha from Google.

As a consequence, the logical constructs of a RESTful Captcha service (i.e., the Captcha service architecture) and its physical manifestation (i.e., the APIs and implementation) will be presented in the following sections.

3.3 Rest-based Captcha architecture

Despite the large number of publications on Captcha (mostly about security and usability aspects), there is a clear paucity in research showing how the Captcha mechanism works as a Web service. This, in fact, is one of the key reasons that led us to build the Captcha service. The Captcha system architecture is shown in Fig.4. It illustrates the main components of the RESTful Captcha service and how they interact with each other. The Captcha system provides both Captcha generation and verification services to web sites. While the clickable Captcha generator is responsible for creating and populating new Captcha challenges in the database, APIs provide a simple method to serve these challenges to clients as well as evaluate user responses. It is worthwhile mentioning that both generation and validation operations are carried out in complete automation without any human intervention.

However, to use the Captcha service, a web site must first subscribe to the service through an enrolment process on the Captcha website in order to obtain a unique pair of API keys (i.e., a *site key* and *secret key*). The site key is used to uniquely identify a website, whilst the secret key is used for authenticated communication between the application server and the Captcha server side during the validation process.

Interactions between an end user, an application server and the Captcha service can be summarized as per the following:

- An end-user sends a request to a Web application server to fetch a web page (step1). The server sends back a page in which a Captcha JavaScript is embedded (step 2).
- Driven by the JavaScript, the user's browser sends an Ajax request to the Captcha server to retrieve a challenge (step 3). In fact, each request carries certain parameters with it to specify the challenge configuration options, in addition to the site key which identifies the web page identity.
- Once the Captcha challenge (i.e., a Captcha image and unique token) is retrieved (step 4), the end-user can solve and submit their solution to the web server (step 5).
- The web server forwards this solution to the Captcha server side (step 6). The solution forwarded by the web server is typically a series of XY-coordinates of user's clicks accompanied by the secret key, unique token and certain user identity information such as IP address and user agent.
- The Captcha server side verifies the solution and returns a verification result (step 7). The result, which is in JSON format, is either True, False or an Error Message if a failure occurs.
- According to the verification result, the web server accepts or denies the user's request (step 8).

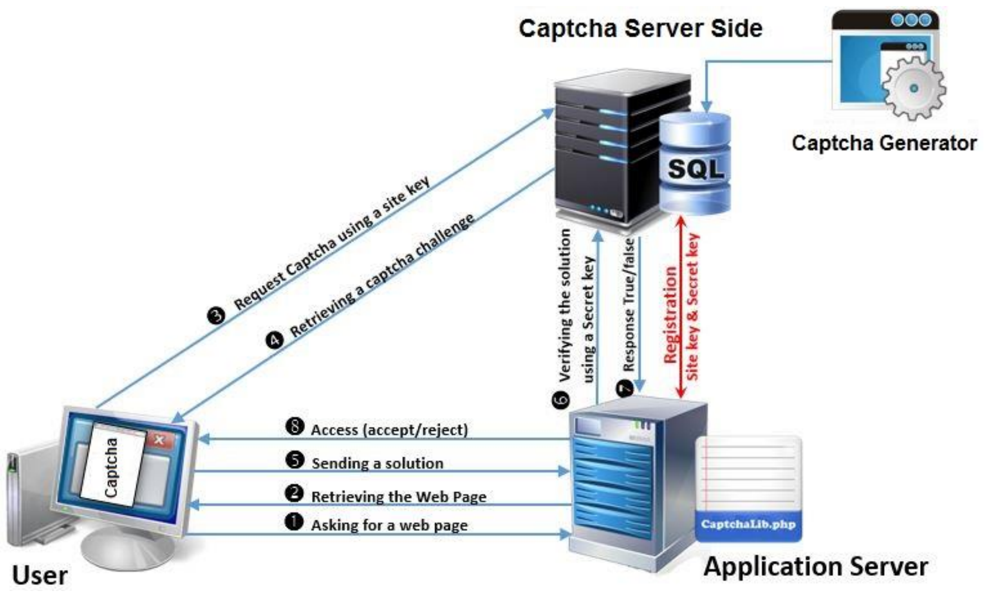


Fig. 4. The Architecture of the RESTful Captcha Service

3.4 Clickable Captcha implementation and APIs

The implementation and its API represent the real physical manifestation of the Captcha Service. While the service implementation is principally represented in the computational logic of the executable source code, data and service configuration information, the API defines the requests that the Captcha service can receive and the responses it would return to clients. In other words, the API describes the functionalities supported by the Captcha service which can be invoked remotely by a client. Additionally, the service interface also defines the structure and format of the requests that can be processed, as well as the responses that would be produced by the service. Therefore, the following sections illustrate in detail how to construct such a service.

3.4.1 Implementation technologies

The following free technologies are used to develop the RESTful Captcha service:

On the server side. Linux as the operating system, Apache as the web server, PHP as the programming language, and MySQL as the database are used to build applications on the back end.

On the client side. JavaScript (Ajax technique), CSS, and HTML are used to develop web pages and applications on the front end.

Data exchange between server and client. JSON is used to transmit data between the server side and client side.

3.4.2 Click-based generator

The generator represents the heart of the Captcha system. It is responsible for the creation of Captcha challenges. The Captcha generator is designed to generate challenges in a completely automated manner without the need for human intervention. The generator can be any of the click-based Captcha schemes suggested in [10], [11], [12], [13], [14]. In fact, We have developed a prototype Captcha generator for ClickText Captcha suggested in [19] to implement our RESTfull service architecture. Our Captcha generator can create a new challenge in a realistic time frame (i.e., it needs approximately 75 ms for a non-distorted clickText challenge and less than 200 ms for a distorted one) using a computer with 8 GB RAM, Intel(R) Core(TM) i7-4790 CPU @3.60GHz and PHP7. Thus, it can generate challenges on-the-fly and serve them up directly to clients when requested. However, a decision was made to use the Captcha generator to produce pre-generated challenges and store them in a database for later use.

In general, the workflow of issuing a new challenge by a Captcha generator can be broken down to the following steps:

- With the assistance of the graphics library, a new image is created with a predefined size.
- A Captcha algorithm is applied in order to construct a Captcha image.
- If required, a certain level of distortion is applied on the whole Captcha image.
- A keyword of a target object is selected and then drawn along with the question text on the canvas.
- The location of each candidate object is accurately defined by calculating a convex hull for every target object according to the keyword in the resultant image. Each target object is typically represented by a vertex list of its convex hull.
- A list of candidate objects along with their convex hull are serialized so as to be storable in the database. We will refer henceforth to this serialization as a “challenge solution”.
- For each new challenge, a unique identifier (ID), keyword and the challenge solution are stored in the database as a new record. However, the corresponding Captcha image keyed with that ID is saved in specific directory on the file system.

3.4.3 Data repository

The Captcha service data is saved in two storage areas: the database and the filesystem. While challenge entities and customer registration information are stored in the database, Captcha images are stored directly on the filesystem.

In fact, storing an image as an ordinary file on the filesystem (i.e., outside of the database) with a path stored in database has several advantages. It provides simplicity and high performance with less load on the database. That is, this approach decreases the database size and then makes the processing faster, leading to better performance; in addition, images can be stored in completely separate storage locations on the server side. Another benefit of this approach is that the server can serve images up from the filesystem directly and efficiently, bypassing the database driver. Furthermore, images can be processed and manipulated by external systems/services (e.g., use compression tools in order to reduce image sizes). In contrast, storing images in the database causes a rapid inflation in size, taking up a lot of space, extra coding and is more time consuming, which in turn adversely affects database performance.

For best performance and avoidance of issues associated with storing large numbers of images (e.g., hundreds of thousands or millions) in one big directory, a balanced tree of subdirectories is used to spread images out so that there are no more than several thousand images in each subdirectory.

As mentioned before, the Captcha generator may generate challenges on-the-fly. However, the pre-generation of Captcha challenges using a database has many advantages over those generated on-the-fly. For instance, Captcha challenges can be served up faster to clients. Moreover, this can reduce the computational workload on the Captcha server side during high-traffic time intervals. Another advantage is that the Captcha generator and service APIs can be decoupled into different hosting environments in order to improve the performance and availability of the service.

3.4.4 API keys

Only valid consumers should be able make API calls to the Captcha service. For this, Web developers must first register their websites with the Captcha service in order to obtain a unique pair of API keys. The API key pair comprises of a *site key* and a *secret key*. One of the characteristics of this mechanism is that it is simple and easy to implement. That is, each key consists of a long unique string (i.e., 40 alphanumeric characters) that are randomly generated upon the website registration and act as a unique identifier for that website. Then, each request made by the registered website to the Captcha server must be accompanied by either a site key or a secret key.

While the site key is used to uniquely identify a website when retrieving a new Captcha challenge from the Captcha server, the secret key is used to provide an authenticated communication between the application server and the Captcha server side to prevent abuse of the RESTful Captcha service when verifying a user’s response at the validation stage. In fact, the site key can be known by an end user who obtains it from the application server (step 2 in Fig.4), and uses the key to retrieve a new challenge from the Captcha server side (step 3 in Fig.4). However, the secret key must be kept safely so that it is known only to the Captcha server side and application server (step 6 in Fig.4).

Although the site key can be exposed to the public, it is tightly associated with a particular site to identify the origin of the request. More precisely, when registering a new Captcha account, the Captcha service will ask the web developer to provide the domain name(s) of the website where the Captcha site key would be used. Then, when an end-user visits this website and sends a request to the Captcha server side to retrieve a new challenge, the Captcha service will verify whether the domain name of the website that uses this site key is the same as the domain name that owns the site key. If a difference in domains is found, a site key error code is issued to the user's browser with no new challenge will be retrieved. Accordingly, API keys can be used to track and control Captcha's API usage. That is, this approach can be used to limit the maximum number of Captcha requests that the API key owner may send to the Captcha service within a specific timeframe (say a day). In fact, APIs with such rate limiting features can restrict and prevent denial of service attacks.

3.4.5 Server side

The main functionalities of the Captcha server-side supports both initial requests for serving up Captcha challenges to clients and subsequent verification requests for evaluating supplied user answers. Therefore, the Captcha service provides two different APIs: the *Captcha retrieval API* and the *Captcha validation API*.

Captcha retrieval APIs. This is responsible for receiving a user request, validating the request, contacting the database to retrieve a new challenge and then delivering a token for the new challenge to the client. Additionally, it is also responsible for retrieving a Captcha image from the server's filesystem and streaming it to the user's browser, as shown in Fig.5.

Once a Captcha image is retrieved in the browser, the user can solve the challenge and submit their solution to the application server, which in turn forwards the solution to the Captcha server side. In fact, the application server sends a new Captcha verification request consisting of the challenge's token, the user's solution, the user's IP address, and the user agent and secret key to the Captcha validation API for verification.

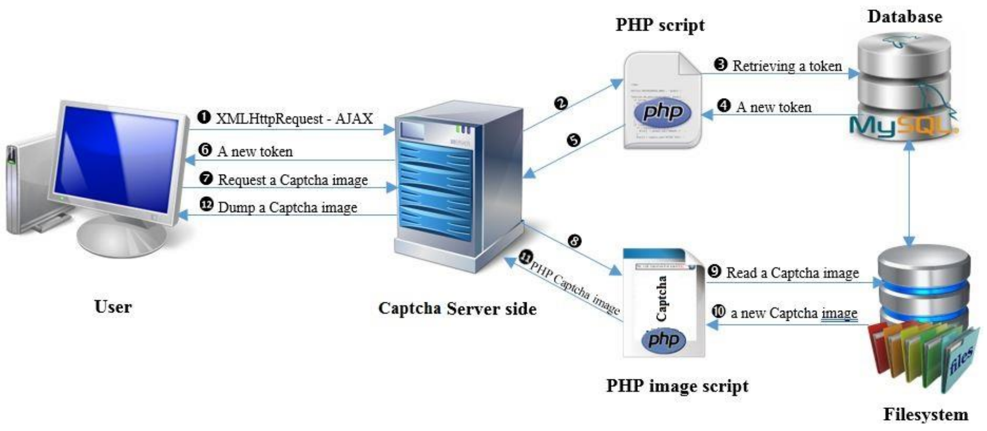


Fig. 5. The Process of Retrieval of a Captcha Image From the RESTful Captcha Service

Captcha validation API. This is responsible for receiving a Captcha verification request from an authenticated web server, validating the request, comparing the supplied user solution with the appropriate data stored in the database, and then delivering a true-or-false answer as a JSON response to the web server.

In the case of any suspicious behaviour in the API requests, for example invalid API keys, false/malformed token or if the IP address requesting challenge is different from IP address submitting the answer, a particular error response code is returned to the client site.

3.4.6 Client side

Client-side's operations are functionalities performed on a web server (i.e., service consumers) and a user's browser in relation to Captcha system as a security mechanism. These functionalities typically include displaying a

Captcha challenge within a web page, the challenge being solved by an end-user, and receiving a solution from the end-user and sending it back to the Captcha server side for verification.

3.4.7 Security considerations for RESTful Captcha web service

It is beyond question that a weak implementation of any Captcha scheme may yield an insecure protection mechanism. A good example of such cases is the fatal security flaws in the implementation of the CCaptcha service [3]. Therefore, we have carefully taken all the required countermeasures to avoid such vulnerabilities when Captcha service was implemented.

The Captcha interface is designed to be as simple, and at the same time, as restrictive, as possible. This simplicity will not only reduce the programmatic burden on web developers, but also makes it easier to test and audit these interfaces. However, with this simplicity, design decisions were carefully considered to present it in such a way that would reduce threats to negligible concerns.

For example, some Captcha schemes use a Session Identifier (SID) to keep track of the state of the client's interaction with the server. In some such schemes, for instance, a user can gain multiple access to a protected service based on a session ID that was essentially authorized by successfully solving a single Captcha test [20], [21]. This, in fact, may compromise the security of the system, allowing it to be ultimately hacked. That is, attackers can somehow hijack the SID and then exploit it to bypass the Captcha.

By contrast, our Captcha scheme, which is designed as a RESTful web service, is completely stateless. Although this can overcome all hazards associated with sessions, it may also carry new security concerns with it.

For that, we take appropriate precautions and countermeasures to secure the communication between the server side and client side.

For example, before using Captcha challenges, a website should first register with the Captcha service in order to obtain a unique pair of API keys. The purpose of these keys is to specify the consumer's identity, avoid verification abuse, and determine the maximum number of Captcha challenges that can be used by each authorized web server.

Nevertheless, attackers could still send requests with malformed data whose content has been crafted to access sensitive data or the back-end systems on which the Captcha scheme is running. For example, an attacker may inject malicious SQL instructions into a database query (i.e., *SQL injection attack*) in order to bypass the verification process. The general rule of thumb is that with adequate validation and sanitization of user-supplied data in place, most security concerns are alleviated, and some are practically removed. Therefore, all parameters that are supplied by a user request or an external source are perfectly validated and filtered from any questionable datum, in addition to rejecting any request that is not faultlessly established.

In addition, several countermeasures have been taken to bolster the service against potential known attacks. For instance, each Captcha challenge is randomly created with a one-time token that accompanies the Captcha throughout its life cycle, from creation to disposal. Both are valid for use only once and their lives end during the verification process. This, in fact, prevents *replay attacks* and *abuse of verification process*. Furthermore, for each challenge issued, the IP address which requests the challenge is compared with the IP address that submits the answer; if a difference is detected, then the request is discarded, and the IP addresses are temporarily blocked. In fact, this can observably limit *relay attacks*. Moreover, the lifespan of each challenge is valid only for a limited amount of time in order to reduce the window of opportunity within which an attacker may guess a solution. Besides, a detailed history of all requests and responses can be logged in order to detect any anomalies and for further review. Indeed, such logs can reveal some important facts about attacks or attempted attacks on the system. For example:

- Too short intervals between time of challenge presentation and time of solution submission (say a few milliseconds) would strongly indicate that the system is breached.
- A vast number of requests in a certain amount of time from a single IP address would strongly imply an attempted attack.

The use of some deterrence methods against such attempts can make the task slower and harder for aggressors. For example:

- Using the rate limiting strategy, which limits the number of requests that a Captcha scheme accepts within a set amount of time.
- Using the timed lockout policy for many requests coming from the same IP address in a short time.
- An automatic adaption of the Captcha system against nefarious activities. That is, with increasing, repeated attempts that come from the same user in a short time, the service can easily be adapted to raise the security bar to a higher difficulty level.

Because automated attacks aim to solve as many challenges as possible in the shortest possible time, these restrictions will slow down such attacks significantly, making them ultimately ineffective.

4. CONCLUSION

In this paper a RESTful service architecture for any clickable image-based Captcha proposal is presented. We illustrated the main components of the RESTful Captcha service and how they interact with each other. The Captcha system provides both Captcha generation and verification services to web sites. Using a Captcha Web service to generate and evaluate Captcha challenges is more reliable, usable and robust, in addition to reduce the workload of a Web application server. Captcha service API also simplifies integrating Captcha challenges into web pages without having to consider complex algorithms, implementation details or update versions. We also discussed in details all security considerations for the RESTful Captcha web service in order to prevent side channel attacks, which exploit vulnerabilities in the practical side of Captcha development. Finally, we would like to point out that whereas writing this paper, we are developing an open source Captcha web service to be available to the public.

REFERENCES

[1] K. Chellapilla, K. Larson, P. Simard, and M. Czerwinski, "Computers beat Humans at Single Character Recognition in Reading based Human Interaction Proofs (HIPs)," in *Proc. CEAS*, CA, USA, 2005.

[2] M. Kumar, M.K. Jindal, M. Kumar, "A Systematic Survey on CAPTCHA Recognition: Types, Creation and Breaking Techniques,", *Arch Computat Methods Eng*, vol. 29, no. 2, pp. 1107–1136, Mar. 2022.

[3] A. Algwil and J. Yan, "Failures of Security APIs: A New Case," in *Proc. FC*, BB, Barbados, 2016, pp.283–298.

[4] J. Yan and A. S. El Ahmad, "Usability of CAPTCHAs or usability issues in CAPTCHA design," in *Proc. SOUPS*, NY, USA, 2008, pp.44–52.

[5] Y. Xu, et al. "Security and Usability Challenges of Moving-Object CAPTCHAs : Decoding Codewords in Motion," in *Proc. USENIX*, Bellevue, USA, 2012, pp.49-64.

[6] J. Nian, P. Wang, H. Gao and X. Guo, "A deep learning-based attack on text CAPTCHAs by using object detection techniques,", *IET Information Security*, vol. 16, no. 2, pp. 97–110, Mar. 2022.

[7] A. Atri, A. Bansal, M. Khari and S. Vimal, "De-CAPTCHA: A novel DFS based approach to solve CAPTCHA schemes,", *Computers & Electrical Engineering*, vol. 97, p. 107593, Jan. 2022.

[8] ComScore, “Mobile Future in Focus 2013,”, ComScore Inc., New York, NY, USA, Feb. 22, 2013. [Online]. Available: http://www.comscore.com/Insights/Presentations_and_Whitepapers/2013/2013_Mobile_Future_in_Focus.

[9] G. Reynaga and S. Chiasson, “The Usability of Captchas on Smartphones,” in *Prec. SECRYPT*, Reykjavík, Iceland, 2013, pp.1-8.

[10] H. Baird and J. Bentley, “Implicit Captchas,”, in *Proc. SPIE*, San Jose, CA, USA, 2005, pp.191–196.

[11] M. Shirali-Shahreza and S. Shirali-Shahreza, “Collage Captcha,”, in *Proc. ISSPA*, Sharjah, UAE, 2007, pp.1–4.

[12] M. T. Banday and N. A. Shah, “Image Flip Captcha,”, *ISeCure*, vol. 1, no. 2, pp. 105-123, Jul. 2009.

[13] R. Lin, S. Huang, G. B. Bell, and Y. Lee, “A New Captcha Interface Design for Mobile Devices,”, in *Proc. AUIC*, Perth, Australia, 2011, pp. 3–8.

[14] Y. Shen, R. Ji, D. Cao, abd M. Wang, “Hacking Chinese Touclick Captcha by Multi-Scale Corner Structure Model With Fast Pattern Matching,”, in *Proc. MM*, Orlando, FL, USA, 2014, pp.853–856.

[15] M. Mohamed, et al., “A Three-Way Investigation of a Game-Captcha: Automated Attacks, Relay Attacks and Usability,”, in *Proc. ASIA-CCS*, Kyoto, Japan, 2014, pp.195–206.

[16] C. J. Hernandez-Castro and A. Ribagorda, “Pitfalls in Captcha Design and Implementation: The Math Captcha, a Case Study,”, *Computers&Security*, vol. 29, no. 1, pp. 141–157, Feb. 2010.

[17] M. Conti, C. Guarisco, and R. Spolaor, “CaptchaStar! A Novel Captcha Based on Interactive Shape Discovery,”, in *Proc. ACNS*, London, UK, 2016, pp.611–628.

[18] R. T. Fielding, “Architectural Styles and the Design of Network-Based Software Architectures,” Ph.D. dissertation, Dept. Info&Com. Sci., California Univ., Irvine, USA, 2000.

[19] B. B. Zhu, J. Yan, G. Bao, M. Yang, and N. Xu, “Captcha as graphical passwords—A new security primitive based on hard AI problems,”, *TIFS*, vol. 9, no. 6, pp. 891–904, Jun. 2014.

[20] J. Elson, J. R. Douceur, J. Howell, and J. Saul, “Asirra: A Captcha That Exploits Interest-Aligned Manual Image Categorization,”, in *Proc. CCS*, New York, NY, USA, 2007, pp.366–374.

[21] G. S. Kalra, “Attacking Captchas for Fun and Profit,”, McAfee Foundstone Professional Services, 2012.