Secure Login Through Mobile Authentication

Kalpesh Adhatrao, Aditya Gaykar, Rohit Jha, Vipul Honrao
Department of Computer Engineering
Fr. C.R.I.T, Vashi, Navi Mumbai, India
E-mail: {kalpesh.adhatrao, adityagaykar, rohit305jha, mithunhonrao2000}@gmail.com

***Abstract*—Logging in from an unreliable terminal on public networks using common approaches such as by way of keyboard, mouse or on-screen keyboard, is always a security issue as third party keylogger software and hardware devices are able to record key strokes and track mouse movements covertly. On the other hand, communication over a personal mobile device such as a cell phone, smartphone or table PC, is relatively safe for authentication. In this paper, we demonstrate the usage of QR (Quick Response) codes with encrypted access tokens for authenticating a user's login on a mobile device, and then allowing him/her to access the web application on the terminal computer.**

Keywords—login authentication; QR code; keylogger; access token; mobile device;

I. INTRODUCTION

In today's age, the boundary between the virtual and real life is the login process to online web applications. Needless to say, a secure login is of paramount importance. Through our implementation, we have demonstrated the usage of QR (Quick Response) codes, containing encrypted access tokens, to be scanned on mobile devices such as smartphones and tablet PCs. A similar experiment (named 'Sesame') was carried out by Google [1]. The details of the experiment are not known, although a member of the experiment, Dirk Balfanz, did mention some on Google+. In our implementation, the content of the QR code will be an authentication hyperlink which the user will use in his/her mobile browser. From this page in the mobile browser, the user will submit his login details. On successful authorization, the access token is validated for that particular user account. Then, the user is notified on his mobile device to refresh the URL (Uniform Resource Locator) of the page containing the QR code on his desktop browser. If the access token was validated successfully on the web server, then the user is directed to his web account. By default, the lifetime of an access token is 10 minutes. Once the token is validated, its lifetime changes to 2 minutes. Within this time, the URL must be refreshed. The generated token is destroyed from the database either after the user is redirected to his home page on refresh or when the lifetime expires.

The purpose of this procedure is to allow the user a diversion from key stroke logging on alien terminals, so as to protect his/her login details. This login method is more secure than not only the usage of keyboards on terminal computers, but also biometric authentication [2].

The traditional method of login using a keyboard is flawed as a user's account can be accessed by a person using a keylogger (hardware device or software program) to store and retrieve the login details. Another problem difficult to overcome is that of shoulder-surfing, in which a malicious observer looks for passwords, PINs (Personal Identification Numbers) or other sensitive personal information, while the user is entering them [3]. Biometric authentication, which identifies individuals based on physiological and behavioral characteristics, suffer from the disadvantage that they are non-revocable and non-secret and pose a physical threat to the user [4].

II. TECHNOLOGIES APPLIED

A. *CAPTCHA*

Standing for "Completely Automated Public Turing test to tell Computers and Humans Apart", CAPTCHA is a means of automatically generating challenges which intend to provide a problem easy enough for all humans to solve and also to prevent standard automated software from filling out a form [5]. In our implementation, CAPTCHA is an added layer of security, making sure that only a human can ask for mobile authentication.

B. *SHA-1*

Access tokens are 10-character long pseudo-random alphanumeric keys generated through a PHP (PHP: Hypertext Preprocessor) script. This string is then encrypted using SHA-1 (Secure Hash Algorithm - 1), which is available as a built-in function in PHP and generates a 160-bit hexadecimal string [6].

C. *QR Code*

Due to its fast readability and large storage capacity compared to traditional UPC (Universal Product Codes) barcodes [7], the QR code has become popular for a variety of applications including storing URLs. Also, apps for scanning QR codes are available on almost all smartphones and tablet PCs [8]. We have used the Google API (Application Programming Interface) to generate QR codes of access tokens.

D. *Access Token*

An access token is an object encapsulating the security descriptor of a process. The security descriptor identifies an owner (or process) and access rights available to that owner (or process). The access token, encryption, is passed to the browser in the form of a QR code. The token is visible in the address bar with the URL. The entire URL is also displayed under the QR code. Thus if a user is unable to scan the QR code, he/she can use the URL on his/her mobile device.

III. Implemenation

We have implemented our mobile authentication for the website **http://www.compag.in**. On the home page, a link "*Try mobile auth*" directs the user to a page where he/she is asked to fill out a CAPTCHA form. For this purpose, we have used reCAPTCHA by Google [9], which has provision for an audio translation for people with visual disabilities. Additionally, an option to reload CAPTCHAs is available to users. On successful verification, the user is directed to a page containing the access token as part of the URL. The same link is available as a QR code and in the form of text under the QR code. The addition of CAPTCHA prevents attacks by malicious bots.

The access token exists on the server database for a maximum time of 10 minutes. A user must login from his mobile device within this time to be able to access the website from the desktop. If a user fails to authenticate within this time, then he/she must ask for another access token by following the above procedure again. The lifetime of the access token is long enough to allow a user multiple tries, in case his mobile device's browser fails. On opening the link encoded in the QR code on the mobile browser, the user can use the mobile login page of the website. Once properly logged in, a message is displayed on the mobile browser, asking the user to refresh his desktop browser window that still shows the page with QR code. Then, the user's home page on the web site is displayed on the desktop browser. As a result, the validated access token is destroyed, even if the lifetime hasn't expired, thus preventing access by a malicious attacker from another device.

In case the user does not have a mobile device equipped with a barcode scanning app, he/she can type the link displayed under the QR code into the mobile browser and authenticate himself/herself to the server. After this, the user can refresh his/her desktop browser to access his/her home page.

A. Generation of Access Tokens

When a CAPTCHA test is passed by the user, an access token is generated by a PHP script. In this script, a pseudo-random, 10-character long string is generated from alphanumeric characters – 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y and Z. For example, a sample random string is "a95iN1q7Xt".

Then, to this random string, the SHA-1 cryptographic hash function is applied. The result is a 160-bit hexadecimal string, i.e. a 40-character long string. For the above sample string, the result is "7d39da64cad740cd6c416a30d9b811329127abc0".

This is the actual unique access token generated. This access token is reasonably secure considering its size and the pseudo-randomness of generation of the string. Combined with the SHA-1 hash function and the limited lifetime of the token, it would be difficult to determine the value of a valid access token.

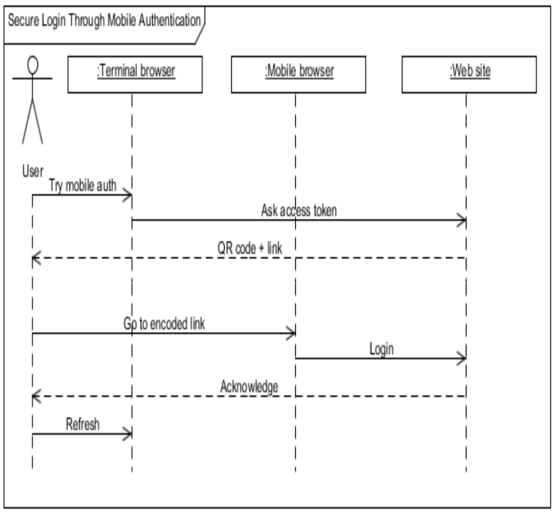

Figure 1: Sequence diagram of user's interaction

B. QR Code Generation

Google has added the QR code generator API to its extensive API of chart generators [10]. An infographics server returns the QR code in the form of an image in response to a URL GET or POST request. All the data required to create the QR code is included in the URL. However, as the QR code in our implementation does not contain a large amount of data, we prefer to use the GET request method [11].

The root URL of the Google Chart API is **https://chart.googleapis.com/chart?**. All infographic URLs start with this root URL and followed by one or more of the following parameter-value pairs:

- **chs** – Size of the image in pixels, in the format *<width>x<height>*. A 350x350 image will be specified as **chs=350x350**.
- **cht** – Type of image. Assigning **cht=qr** denotes a QR code image type to be returned.
- **chl** – This parameter holds as value the data to be encoded, which is URL encoded with hexadecimal notion. For example, is the data is 'Hello World', then the parameter-value pair is **chl=Hello%20World**.
- **choe** – This optional parameter contains the encoding method to use for the data. The possible methods are Shift_JIS, UTF-8 and ISO-8859-1.
- **chld** – Another optional parameter, it sets the error correction level for the QR code. The various levels are **L** (default), **M**, **Q** and **H**, which offer 7%, 15%, 25% and 30% of the QR code to be restored respectively. Given the simplicity of the content in our implementation, we use level L.

For generating the QR code of the access token, we use the following URL that returns an image:

https://chart.googleapis.com/chart?cht=qr&chs=300x300&chl=http://www.compag.in/login/mlogin/7d39da64cad740cd6c416a30d9b811329127abc0&choe=UTF-8&chld=L

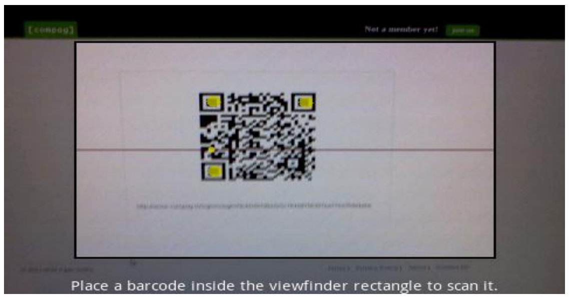

Figure 2: Scanning QR code generated on webpage.

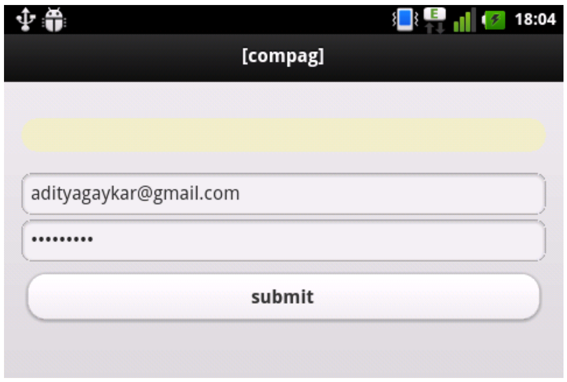

Figure 3: Submitting login details.

C. *Database Manipulation*

The website's database has a table for storing data pertaining to mobile authentication. In this table, attributes for human verification and a hash key exist. Initially, when a user verifies that he/she is a human, a tuple is added to the table. The first attribute for human verification is the hash key, the process of whose generation is already explained. On logging in from the mobile browser, the validity of the link containing the access token, together with the username, is checked. If the tuple containing the access token exists, the attribute for human verification is set and the user is asked to refresh the URL on his terminal browser within 2 minutes. The user should log in from a mobile browser within this time. If he/she doesn't do so, the lifetime expires and the entire procedure would have to be followed again. Once, the page is refreshed, the tuple containing the generated access token which was validated, is removed from the table.

IV. FUTURE SCOPE

The current implementation could be developed as a web service so that several other web applications would be able to support mobile authentication. Another possibility is the integration with existing mobile apps, such that, if a user is already logged in to that web app, it is not required to resubmit login details on receiving an access token. It may also not be necessary to ask the user to refresh his page, as it

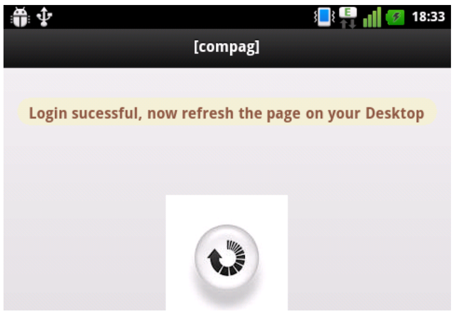

Figure 4: Successful logging in.

could be done so automatically once the authorization is completed on the web server.

V. CONCLUSION

Today's standard methods for logging into websites are subject to a variety of attacks. We have presented and implemented an alternative and secure approach for entering user login details by using mobile devices such as cell phones, smartphones and tablet PCs as alternative input devices.

ACKNOWLEDGMENT

We thank George Jose, Amiraj Dhawan and Ajinkya Pisal for vital contributions to this implementation.

REFERENCES

[1] *Sesame – Google Accounts*, https://accounts.google.com/sesame (January 22, 2012)
[2] Jain, A., Hong, L., & Pankanti, S. (2000). "Biometric Identification". Communications of the ACM, 43(2), p. 91-98. DOI 10.1145/328236.328110.
[3] Kumar, M., Garfinkel, T., Boneh, D., & Winograd, T. (2007). "Reducing Shoulder-surfing by Using Gaze-based Password Entry". http://hci.stanford.edu/research/GUIDe/publications/SOUPS%202007%20-%20Reducing%20Shoulder-surfing%20by%20Using%20Gaze-based%20Password%20Entry.pdf (January 21, 2012)
[4] Weaver, A.C. (2006). "Biometric Authentication". *Computer*, 39 (2), p. 96-97. DOI 10.1109/MC.2006.47.
[5] Grossman, Lev (2008-06-05). "Computer Literacy Tests: Are You Human?". Time (magazine). (December 6, 2008)
[6] *PHP: sha-1 Manual*, http://php.net/manual/en/function.sha1.php (January 21, 2012)
[7] "QR Code – About 2D Code". Denso-Wave. http://www.denso-wave.com/qrcode/aboutqr-e.html (October 3, 2011)
[8] "Global Growth in Mobile Barcode Usage - Q4 / 2010". 3GVision. 5 January 2011. (January 21, 2012)
[9] *reCAPTCHA: Stop Spam, Read Books*, http://www.google.com/recaptcha (January 21, 2012)
[10] *Google QR Chart API*, http://www.qrme.co.uk/qr-code-news/3-newsflash/75-google-qr-code-chart-api.html (January 21, 2012)
[11] *Infographics – Google Code* http://code.google.com/apis/chart/infographics/ (January 21, 2012)