A Proposed System for Preventing Session Hijacking with Modified One-Time Cookies

Annies Minu Sathiyaseelan
M.E. Student
Department of Computer Engineering
St. Francis Institute of Technology
Mumbai-400103, India
anniesminu14@gmail.com

Vincy Joseph
Assistant Professor
Department of Computer Engineering
St. Francis Institute of Technology
Mumbai-400103, India
vincyej@gmail.com

Anuradha Srinivasaraghavan
Associate Professor
Department of Computer Engineering
St. Francis Institute of Technology
Mumbai-400103, India
g.anuradha@sfitengg.org

***Abstract*—Most of TCP connections use HTTP to communicate; so, it becomes mandatory for every server to create a unique identifier for each and every connection. A session is a unique identifier generated by a server that is sent to a client for identifying current interaction session, which is stored in a cookie. A cookie is a short text file for identifying a particular client. Since cookies are transmitted over HTTP, they are visible and prone to attacks such as session hijacking. HTTPS is the most widely used mechanism to protect cookies, but utilizing full HTTPS support is not that easy, especially for applications that are highly distributed due to performance and financial issues. Hence, one-time cookies (OTC) are suggested as an alternative for authentication. OTC prevents various attacks, like for example session hijacking, as they are temporarily stored for a particular period of time or only for a particular session. In this work, we propose a mechanism that uses OTC to prevent an attacker to gain access to a cookie and backend server. A reverse proxy server with OTC, IP, session ID, and browser fingerprinting are used to prevent adversary from capturing session credentials.**

Keywords—HTTPS; one-time cookies; reverse proxy server; session hijacking; TCP

I. INTRODUCTION

The extensively used HTTP works in a common fashion of request–response. First, a client sends a request (which asks for a file) to a server. Next, the server processes the request received from the client and sends back a response to the client. After this, the connection between the client and server is lost and forgotten since HTTP is protocol which is stateless, i.e., the server is not able to distinguish between different connections of different users. An HTTP server treats each and every request it receives independently of any previous requests. However, every web application that is built on top of HTTP needs to store every detail of the state in which it participates, i.e., it should be stateful. A website has a lot of visitors, for example, all online shopping applications over the net need to keep track of shopping carts of their clients; so, in order to identify which request is from which client, web applications need to maintain the state, which can be done with the help of a cookie. A cookie is a piece of information used to record a client's state. They do not contain any information about a user or his/her machine. It is a unique ID given to a client to identify the client. They are stored in a browser or memory or on a hard disk. As a user browses a website for each new page the user visits, which is nothing but a request, the browser is queried for a cookie, and if the cookie ID matches with a website's ID, then the website retrieves the user's information from its server. In this way, requests are processed. Cookies allow a site to store state information about a machine, and this information lets a website remember what state a browser is in. The state is like "your browser has visited the website at least once," and from this, the site knows your ID from that visit. A cookie is widely used to authenticate a user. HTTP cookies generally contain one or a small number of short identifier strings, which allow a server to associate seemingly unrelated requests, such cookies rapidly became essential for managing web session. Unfortunately, the use of cookies introduces a number of risks in security, especially when they are employed as tokens for session authentication. However, because these tokens are static and transmitted "in the clear," an adversary able to intercept them can use these cookies to gain unauthorized access to a user's session. Although session hijacking or "sidejacking" attacks are not new, a major number of web applications are still vulnerable. Quite a few factors such as the explosion of open wireless networks and the release of attack tools like automated attack tools have increased the risk of this threat. The most suggested defense to protect all communications with web applications ("always-on HTTPS") is to use HTTPS. However, using always-on HTTPS can be

978-1-5090-6399-4/17/$31.00

challenging, particularly for distributed systems, due to performance and financial concerns. More prominently, using HTTPS always is not a comprehensive solution; cookies can still be exposed to various attacks due to configuration errors or by attacks against HTTPS and browser [1]. In short, always-on HTTPS does not address the core cause of the problem: cookies are weak session authenticators. To overcome this problem, the authors in [1] proposed a system using one-time cookies (OTC), where not reusable credentials called OTC replace authentication credentials. The OTC scheme generates a set of tokens that are only used once and discarded once used.

In this study, we use a reverse proxy server (RPS) and the concept of OTC, but instead of generating OTC for a particular session, we generate OTC for every request made by a client. A proxy server is a server that acts as an arbitrator between a client and server; whenever a client makes a request, it connects to the proxy server, which then forwards it to the server. However, unlike proxy server where a client is aware that it is getting connected to the proxy, RPS appears as an ordinary server to the client, i.e., the client is not aware of a proxy being present between it and server. RPS collects every request from the client and sends it to the server for request processing. RPS generates session ID, IP, OTC, and browser fingerprinting, which are used to prevent session hijacking.

II. Related Work

A lot of security issues have been raised due to the use of cookies as session authentication tokens. Several surveys [2, 3] have proved multiple problems with web authentication mechanisms, including susceptibility to session hijacking attacks. As a result, security researchers have suggested changes to improve the robustness of authentication cookies. Park et al. [4] and Fu et al. [2] suggested a mechanism using a cookie that provided improved privacy and integrity guarantee by using well-known cryptographic techniques. Moreover, these authors made use of cookie expiration time to reduce the impact of session hijacking attacks. On the other hand, many applications use long expiration time to avoid affecting a user's experience, reducing the effectiveness of this approach. As a substitute to cookies for the identification and authentication of users, Juels et al. [5] proposed the use of cache cookies, which are stored in web browsers by servers (e.g., browser history and temporary internet files). Even though resistant to pharming attacks, cache cookies need HTTPS for protection to prevent active attacks. In addition, HTTPS only protect cookies on a network. An adversary can also steal cookies from a user's computer through many different attacks (e.g., cross-site scripting attacks [6], cross-site tracing attacks [7], and domain-related attacks [8]). Always-on HTTPS is the most recommended defense against session hijacking. To secure communication in an Internet session, Lamport [9] proposed one-way hash chain (OHC) technique—a cryptographic technique that relies on one-time passwords. Specifically, the OHC technique has been utilized in many applications with the goal of reducing the possibility of session hijacking. For example, the authors in [10] proposed a mechanism using OTC, a disposable credential, to replace authentication credentials. To protect a user's session, [9] implemented a framework that ties a session to a current browser by fingerprinting and monitoring an underlying browser and also detecting browser changes at server side. The OTC generates a set of tokens that are used only once and discarded once used. The authors in [11] proposed a hybrid scheme that utilizes one-way hashing and sparse caching techniques, but practically it is not implementable; their research focuses only on hashing, but it does not describe how a session hijack is being prevented.

Not even one of the previously described mechanisms has been widely deployed. Even though many of them prevent session hijacking, they miss the mark to address the necessities of highly distributed web applications, particularly requests' statelessness. As a result, most of the web applications have opted for always-on HTTPS as the main defense against session hijacking attacks. However, always-on HTTPS may be problematic to deploy, particularly in large web applications, because they were not designed for such requirement. Always-on HTTPS not only affects the performance but also impacts existing functionality (e.g., virtual hosting, applications [12], and network content filtering [13]). Therefore, to effectively prevent session hijacking attacks, a more robust, efficient, and practical alternative is needed.

III. Proposed System For Preventing Session Hijacking with Modified OTC

The components of the proposed system are as follows (Fig. 1):

A. User

A user or client is the one who initiates a request. Suppose a user wants to purchase something, he will send a request that contains a user's username and password to the server. After successful authentication, the user will be given an OTC through which he/she will be authenticated for every request he/she makes. Each time a user sends a request, an OTC is sent along with the request.

B. RPS

A proxy server is nothing but a computer that acts as an intermediary between an endpoint device. It is mainly used at a user or client side. However, instead of using a proxy server here at the client side, we use RPS at the sever side. Thus, every request from the user has to pass through RPS. The function of RPS is to obtain IP address, browser fingerprint, set OTC, and session ID, and for each incoming request, RPS will check for IP address, session ID, OTC, and browser fingerprint. If some of these parameters would change, then RPS would redirect to another page.

C. Server

This is the actual server to which a request is sent by a client. The server checks credentials, process all client requests, and sends responses to all clients.

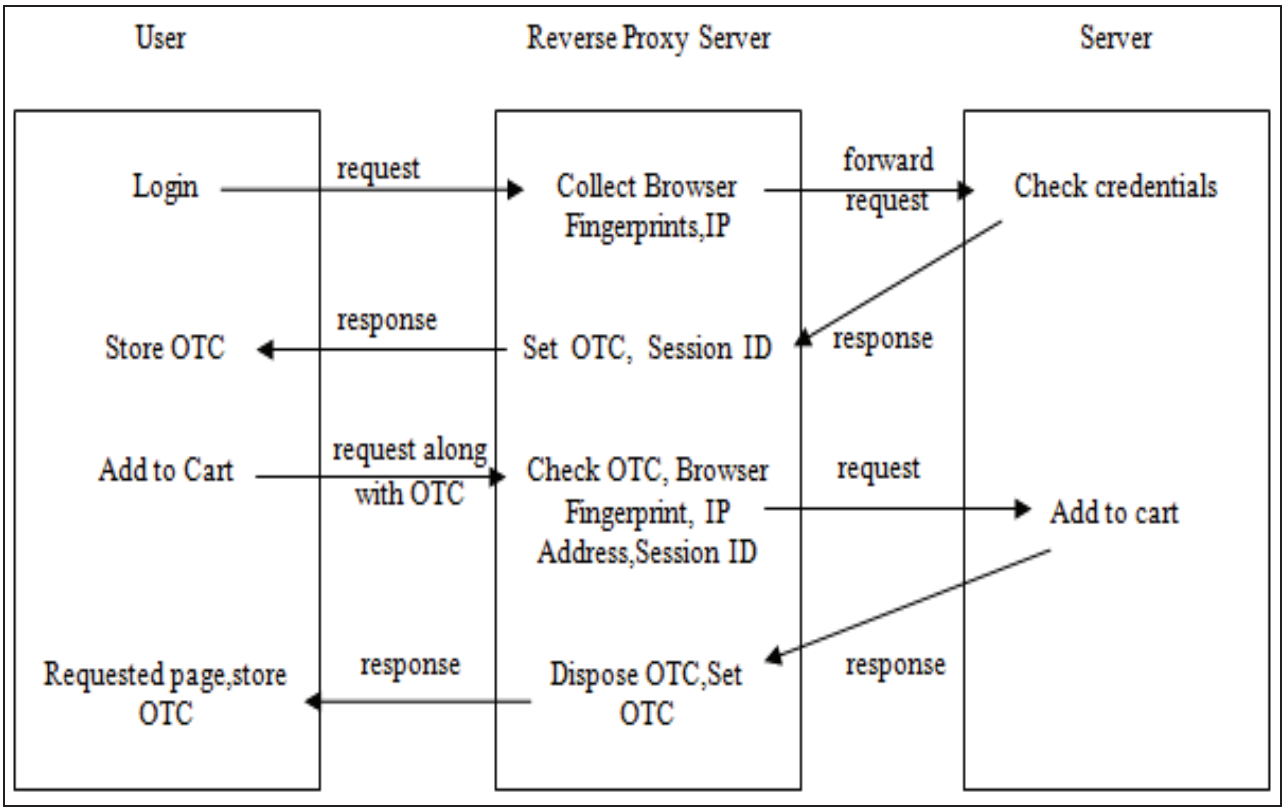

Fig. 1. Block diagram of the proposed system for preventing session hijacking with modified OTC.

The proposed system works as follows (Fig. 1):

1) User enters his/her credentials.
2) A request is sent to RPS, which collects IP address and browser fingerprints from client and forwards the request to server.
3) The server checks the credentials, processes the request, i.e., it fetches the requested page and sends it to the client, but before that, it passes through RPS.
4) RPS creates OTC, session ID, and gives it to the client, and it also forwards the response.
5) User then stores OTC.
6) From now on for every request made by the user, the user sends OTC to RPS.
7) RPS checks it and again creates OTC for every new request made by the user.
8) RPS terminates the session if session ID, IP address, OTC, and browser fingerprint changes.

IV. Conclusion and Future Work

In this paper, we propose the prevention of session hijacking with modified OTC in which RPS generates OTC for every request made by a client. RPS then checks various parameters and terminates a session if it is false. In future, we plan to implement the abovementioned proposed work for preventing session hijacking with modified OTC.

Acknowlegdments

The authors would like to thank the entire staff and colleagues of Computer Engineering Department of St. Francis Institute of Technology for their insightful comments and reviews that motivated us to write the proposed paper.

References

[1] I. Dacosta, S. Chakradeo, M. Ahamad, and P. Traynor, "One-time cookies: Preventing session hijacking attacks with disposable credentials," Georgia Institute of Technology, 2011.

[2] K. Fu, E. Sit, K. Smith, and N. Feamster, "The dos and don'ts of client authentication on the web," USENIX Security Symposium, pp. 251–268, August 2001.

[3] C. Visaggio, "Session management vulnerabilities in today's web," in *IEEE Security & Privacy*, vol. 8, no. 5, pp. 48–56, Sept.–Oct. 2010.

[4] J. S. Park and R. Sandhu, "Secure cookies on the Web," in *IEEE Internet Computing*, vol. 4, no. 4, pp. 36–44, Jul/Aug 2000.

[5] A. Juels, M. Jakobsson, and T. N. Jagatic, “Cache cookies for browser authentication,” *2006 IEEE Symposium on Security and Privacy (S&P'06)*, Berkeley/Oakland, CA, 2006, pp. 5 pp.-305.

[6] “XSS-Stealing Cookies 101” [Online] Available: http://jehiah.cz/a/xss-stealing-cookies-101 [Accessed on March 15,2017].

[7] “Deadliest Web Attacks: Entertaining insights into web security” [Online] Available: https://deadliestwebattacks.com/2010/05/18/cross-site-tracing-xst-the-misunderstood-vulnerability [Accessed on March 15,2017].

[8] A. Bortz, A. Barth, and A. Czeskis, “Origin cookies: Session integrity for web applications,” *Web 2.0 Security and Privacy Workshop (W2SP)*, 2011.

[9] L. Lamport, “Password authentication with insecure communication,” *Communications of the ACM*, vol. 24, no. 11, pp. 770–772, 1981.

[10] T. Unger, M. Mulazzani, D. Fr¨uhwirt, M. Huber, S. Schrittwieser, E. Weippl, “SHPF: Enhancing HTTP(S) session security with browser fingerprinting (extended preprint),” in *Proc. 6th International Conference on Availability, Reliability, and Security*, ARES, 2013.

[11] A. Alabrah and M. Bassiouni, “Preventing session hijacking in collaborative applications with hybrid cache-supported one-way hash chains,” *10th IEEE International Conference on Collaborative Computing: Networking, Applications and Worksharing*, Miami, FL, pp. 27–34, 2014.

[12] “Hotmail always-on crypto breaks Microsoft's own apps” [Online] Available: http://www.theregister.co.uk/2010/11/10/lame_hotmail_encryption [Accessed on March 15,2017].

[13] “Google Moves Encrypted Web Search” [Online] Available: http://www.eweek.com/security/google-moves-encrypted-web-search [Accessed on March 15,2017].