HTTP Session Management: Architecture and Cookies Security

Ines Ayadi * †, Ahmed Serhrouchni *, Guy Pujolle ‡ and Noëmie Simoni*
*ENST, Telecom ParisTech, 46 rue Barrault - 75013 Paris
Email: (iayadi, Ahmed, Simoni)@telecom-paristech.fr
†Sup'Com, Cité Technologique des Communications - Ariana Tunisie
Email: Ayadi.Ines@supcom.rnu.tn
‡ Lip6, University Pierre and Marie Curie, 4 place Jussieu - 75005 Paris
Email:Guy.Pujolle@lip6.fr

***Abstract*—Web applications are an important target for security attacks. Most of these applications make use of cookies to maintain user state. Many attacks are carried out over these cookies in order to compromise network security. In this paper, we propose an architecture and a method of cookies security. This method aims to enforce cookies with integrity and confidentiality services. It was necessary to review the behavior of Reverse Proxy in order to apply these contributions. The approach has been quantitatively and qualitatively validated. The results of this validation are analyzed in this article.**

***Index Terms*—Web application, HTTP, Cookies, Reverse Proxy, SSO architecture, Session management, Integrity, Performance, Apache, Benchmarking.**

I. Introduction

This work was carried out under the project AdminProxy [1]. This project aims to design a management platform to control access to Web applications servers. This project is supported by System@tic government program. Due to its simplicity and efficiency, HTTP protocol [2] is the primary used protocol on the World Wide Web. Therefore, access to Web-based applications has become more widespread. However, most Web applications contain security vulnerabilities, and so, exposing web servers directly on Internet causes a rapid increase in Web applications attacks.

Firewalls and Reverse Proxies [3] are the most used patterns to ensure a secure access to Web applications. Placing a firewall in front of Web servers can protect against network-level vulnerabilities by filtering application traffic. Nevertheless, firewalls don't prohibit attacks employing application-level vulnerabilities. Therefore, integration of a Reverse Proxy is required to protect against application-level attacks.

A Reverse Proxy [3] is a Web application firewall [4]. It is placed on the network perimeter in order to hide internal Web servers. Its main functionality consists in parsing, extracting and transforming application-level transactions. Through the use of a Reverse Proxy, unifying access control and centralizing user logging becomes possible. In our case, the purpose of this proxy is also to handle end-to-end HTTP sessions. However, HTTP is a stateless protocol [2] that is based upon a request/response transaction model. As a result, maintaining user session is not achieved. To cope with this situation and to keep state user, cookies headers are defined in [5].

In [6], an approach for managing a secure HTTP session through cookies within a Reverse Proxy has been proposed. The present paper describes the implementation and validation of this approach. Furthermore, we evaluate the performance of this approach and its effects on the Reverse Proxy behavior.

This paper is organized as follows. Section II describes the overall architecture. Problem statement and the related work are described in Section III. Section IV details HTTP cookies-based sessions approach. Security consideration is detailed in Section V. Section VI shows an implementation and validation of this approach. Experiments and performance results are presented in Section VII. Conclusion is offered in Section VIII.

II. Overall Architecture

This section proposes a new intermediary platform to evolve the Reverse Proxy classic architecture to another one more advanced. The Reverse Proxy is a mandatory proxy placed in front of internal Web servers. This mandatory proxy protects the access to Web server resources. It deals HTTP requests and responses via a filtering process. In addition, it can use the application-level transactions data in order to determine a pertinent filtering decision. Users access to server resources through a secure HTTP connection based on TSL protocol [7]. The request will be transparently redirected to the Reverse Proxy. The mandatory analyzes the given request and checks its validity. Then, it sends this request to the adequate internal Web server in order to respond to the user needs. Figure 1 represents an overview of the presented architecture.

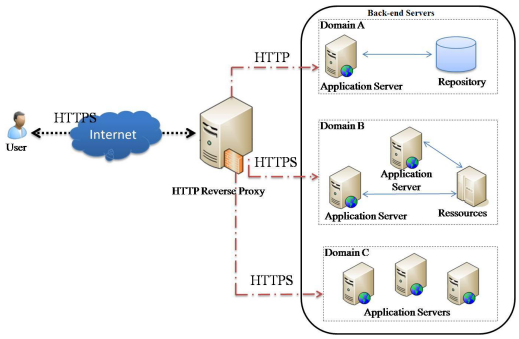

Fig. 1. Overview architecture

978-1-4577-0737-7/11/$26.00 ©2011 IEEE

This work offers new features to Reverse Proxy pattern. These features allow unifying authentication process, managing cookies and maintaining HTTP session. Figure 2 presents the different components of the advanced Reverse Proxy architecture.

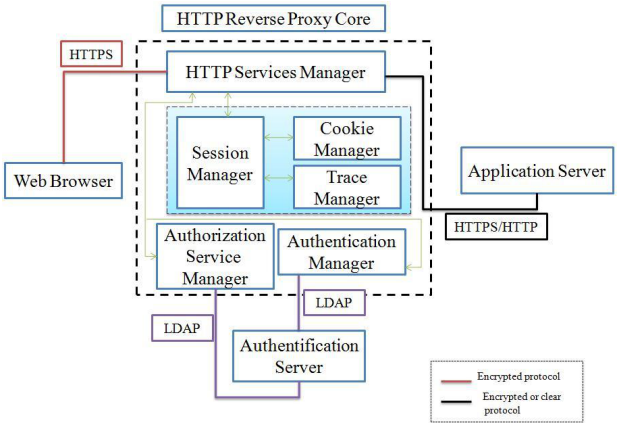

Fig. 2. Reverse Proxy components

This architecture consists of several principal components.

The "HTTP Services Manager" is the core of the Reverse Proxy that manages HTTP transactions. By applying filtering rules, this element analyzes requests received from the Web browser and determines their validity. Then, it transmits the valid request to the back-end server. In addition, this component can analyze and modify the response received from the application server. Users are created in the authentication server such as OpenLDAP directory [8].

The main component of the present architecture is the "Session Manager" element that handles end-to-end secure HTTP session. A unique session is generated between a given user and all internal servers for which he is authorized. A session token is required in order to associate the user with a given session. In this work, this token is an HTTP cookie [5], [9]. So, this platform includes a "Cookie Manager" element. This component generates the cookie token in order to identify the user in his session. Furthermore, it manages also the cookies received from internal servers. Functions of this component will be illustrated in section IV.

Authentication process is an indispensable mechanism to check users' identity. This process is provided by a simple login and password. The "Authentication Manager" verifies the client credentials stored in the LDAP directory. This process will be realized one time in order to provide an SSO technology.

The "Authorization Service Manager" provides the control access for authenticated client. In fact, a given client can only access to applications for which it is authorized. To do this, this component recovers the access rights from LDAP repository where users information and access rights are defined.

Finally, another principal function of this architecture is transactions traceability. The "Trace Manager" tracks all HTTP transactions from the user to the internal servers. This function provides monitoring capabilities to the architecture and the use of such functionnality in preventing possible security issues.

III. Problem and Related Work

A. Problem Statement

As shown by [6], maintaining end-to-end sessions requires managing two types of cookies: Reverse Proxy cookie, called RP-cookie through this work, and internals servers' cookies, called AS-cookie (Application Server Cookie) through this present paper. The main problem of using cookie to maintain user session is how to manage internals servers' cookies. In fact, a RP-cookie is generated by the Reverse Proxy in order to establish sub-session for a given client. AS-cookie are cookies received in response from the internals servers. The Reverse Proxy transmits these two cookies to the user browser. Whenever a browser sends an HTTP request from the client to the same origin Web server, only RP-cookie will be send. In fact, the browser returns cookie to origin Web server whose domain matches the cookie domain. In other words, they return the cookie to Web servers that created them. Usually, the Reverse Proxy has not the same domain as the back-end servers [3]. That is why the browsers return only the RP-cookie. In this case, the request is transmitted to the origin server without the AS-cookie. Consequently, the server considers that no session has been previously maintained for the given client. Indeed, when a server receives a request that includes a cookie, the server is able to use the information stored in it in order to retrieve user state session. Unfortunately, this is not the case and end-to-end user session can not be maintained.

B. Related Work

Managing internal servers cookies was implemented by an open source Apache [10] module called "mod-but" [11]. This module stores internal cookies in Reverse Proxy shared memory. This approach requires supplementary tables for storing these cookies. Access to cookies stored in memory can cause traffic overflow. Therefore, Reverse Proxy performances are decreased. In fact, the Reverse Proxy is the common entry point for all HTTP requests and responses. Thus, using added tables for storing internals cookies can cause bottleneck effects. Another inconvenient of this approach is high probability of name cookie confusion. Indeed, internals servers can use the same name for cookies. The Reverse Proxy has to manage the several internal cookies that have the same name. To do this, this approach requires an additional configuration on the Reverse Proxy. In fact, this require a complexity of Reverse Proxy configuration in order to clarify the location of storing each internals cookies. Other disadvantages include security concerns. This solution presents two types of vulnerabilities. First, the RP-cookie is stored in clear text in the user computer. Therefore, confidentiality and integrity are not guaranteed. Second, internals cookies are also stored in clear text in the Reverse Proxy memory. Therefore, many types of attack can be performed such as spoofed cookie [4], cookie session hijacking [12], etc.

IV. Session Management with Cookies

A. An Overview of HTTP Cookies

HTTP cookies headers are detailed in "HTTP state management mechanism" standard [5] of the Internet Engineering Task Force. These specifications show that cookie mechanism allows Web server to maintain a persistent state for Web client. To do this, two HTTP headers are defined: Set-cookie and Cookie. These headers are a set of HTTP transactions between the client and the server. If the Web server gets a request from the client, it creates a user's context and stores state information such as user credentials, user preference and user profile.

A Set-cookie is added by the server in HTTP response in order to associate a token session to a given user. The browser stores cookie information in its hard disk. If the client sends a new request to the server, the browser subjoins a cookie header. This follows to the server that a session has already maintained. In order to provide persisting session, the cookie header should be send back to the Web server in all subsequent requests.

A Set-cookie header consists of the following attributes: "Name", "Domain", "Path", "Comment", "Expiry", "Secure" and "Version". The "Name" field describes the cookie's name. Each "Name" attribute is associated to a "Value". The couple "Name=Value" is unique and required for each cookie. The "Expiry" element represents the cookie's lifetime. The "Comment" flag allows server to store more information about the user. The "Domain" attribute indicates the domains for which the cookie is valid. The "Path" attribute specifies the subset URL to which the cookie should be applied. User browser uses the domain and path attributes to determine the cookies that should send back to an appropriate internal server. The "Secure" value allows sending cookie only in secure transmitted channel by applying SSL or TLS with HTTP. The last attribute is the "Version" that indicates the HTTP protocol version.

B. Proposal Session Management Approach

This section proposes a new approach, called encapsulation/decapsulation process. This solution provides a new mechanism to manage HTTP session based on cookies and to improve server overload performance. Generate a RP-cookie allows the Reverse Proxy to handle session between him and the clients. However, this is not enough because handling the sub-sessions between the Reverse Proxy and the internal servers is also a required issue. In fact, the Reverse Proxy must behave as a Web client in frontal of back-end servers. Thus, it must handle the cookies received from all internal servers. We propose a new approach that manages the internal cookies. This approach is composed on four steps: First, the Reverse Proxy concatenates all cookies received from the same internal server. Second, it encrypts the concatenation value with his secret key. Then, the encrypted will be encapsulated in a specific attribute of RP-cookie. Finally, the Reverse Proxy adds the RP-cookie into Set-cookie header and sends the response to the client.

In this approach, the RP-cookie has a specified syntax and semantics. Two attributes are added to cookie's syntax. The first attribute is the "capsule" that contains encrypted internals cookies. The second is called "Integrity Cookie Digit (ICD)" that provides integrity cookie service. This attribute allows the Reverse Proxy to control cookie integrity and to ensure the source authentication. This will be detailed in section V. Figure 3 represents the scenario of our approach for managing HTTP session based on cookies.

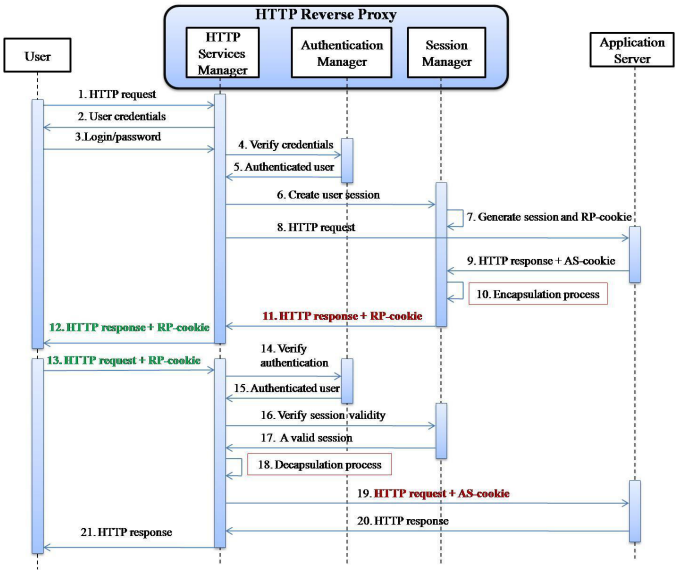

Fig. 3. Encapsulation/decapsulation process scenario

A new HTTP session will be created only if the user is successfully authenticated. This session is specified by a unique identifier, a life time period and user's information. Next, a token session is generated and it represents the RP-cookie. This token allows a current user to be identified without proceeding a new authentication. Therefore the RP-cookie header must be added in all HTTP responses in order to provide session persistent.

The "HTTP Services Manager" sends the request to the adequate Web server. After receiving the response from the server, the "Session Manager" analyzes the response headers. If this response contains a cookie header, so the encapsulation process must be executed. In effect, the "Session Manager" encrypts all cookies received from the internal server. Then, the encrypted value will be encapsulated in the RP-cookie attribute. Therefore, the "capsule" attribute consists of the encrypted AS-cookie.

When the user sends a second request to the internal server, he does not need to provide his credentials again because the request must contain the RP-cookie. Thus, the "Session Manager" verifies the validity of the current session and executes the decapsulation process. This process consists of extracting the AS-cookie from the RP-cookie, decrypting its values and sending this cookie in HTTP request to the internal server.

If the internal server has not a set-cookie header in its response, the encapsulation process will not be made. This is a particular case that the Web server does not need to send the cookie header.

V. Security consideration

A. Security Menaces of Cookie

Cookies contain sensible information such as user profile, authorization information and credentials information. Several researches [12], [13], [14] study cookie headers weakness. There are two types of cookies threats: client side threats and the network threats.

These tokens are stored in the hard disk or memory of client. Thus, cookies can be easily manipulated by many techniques attacker's. In effect, data stored in cookies are in clear text. So they are vulnerable to be modified and copied from user computer. Many techniques are developed to steal cookies information without user connivance such as cross-site scripting [15], cross-site request forgery [16], etc. Additionally, attacker can use the stolen cookie to impersonate another authorized client for accessing to a Web application. This type of attacks called "spoof an authentication cookie" [4].

The network threats can take place if cookies transmitted in clear text. Attackers can so read and modify cookies information. The main solution against this attack is to use a secure transmitted channel by using TSL protocol [7]. This protocol is widely used by several Web servers in order to assure integrity and confidentiality of transmitted date in the network.

B. Integrity Cookie Mechanism

This section describes the new attribute cookie called "ICD". This attribute ensures cookie information integrity. ICD calculation based on a cryptographic hash function and composed of three steps. The first step is the generation of a MAC (Message Authentication Code) value. To do this a HMAC function [17] will be applied to all RP-cookie attributes. This function uses a hash function and a secret key known only by the Reverse Proxy. The second steps consist of a truncation function that inspired from HOTP standard [18]. This function takes the MAC value as input and it calculates the decimal value of the lower four bits of last byte of the MAC value. Furthermore, it extracts four bytes from the MAC value starting at the byte whose number is the calculated decimal value. The last step is a "modulo" function that takes the output of truncation function and generates the "ICD" value. The output of this function has six digits long. This result provides an efficient way for adding integrity service without increasing cookie overhead.

VI. Implementation and validation

The implementation of encapsulation/decapsulation process is essentially based on "mod-but" Apache module.

A. Apache Configuration

The architecture described in section II is based on an HTTP Reverse Proxy. We carried out a comparative study of Reverse proxies. Apache, configured as Reverse Proxy, has been chosen in the present implementation for the following reasons. First, Apache is the most popular Web server in the World Wide Web [10]. In addition, this server has a high performance because of its modular features. Furthermore, it is freely available and so, it is possible to make changes to its modules. During the implementation, we use Apache 2.0.64 and we set it up as an HTTP Reverse Proxy, using "Mod-proxy" module [19]. This module allows Apache to behave as a mandatory proxy for internal servers. In this case, Apache can transparently treat requests received from clients and transmit them to the adequate back-end servers.

B. Implementation of Encapsulation/Decapsulation Process

The implementation of this solution is performed by modifying the "mod-but" Apache module [11]. This module provides several features that facilitate the deployment of the proposal such as managing HTTP cookies headers, generating Reverse Proxy cookies, etc. In fact, the original code source implements session management features such as session identification, session validity and session data. As well, this code generates a static Reverse Proxy cookie (RP-cookie) that is associated with a given session. However, other several changes are necessarily made in order to achieve encapsulation/decapsulation process.

As mentioned above, the principal goal of our approach is to optimize Reverse Proxy managing session function and to ensure secure end-to-end sessions. For this reason, the main changes of "mod-but" consists are twofold. Firstly, we removes the storage internal servers' cookies implementation and we added a suitable code that implements the encapsulation/decapsulation process. Secondly, we studied and implemented control cookies integrity service.

C. ICD Verification Process

The objective of the "ICD" mechanism is to provide cookies integrity protection. Cookie manipulation is a well-known attack that allows an attacker to access to a context for which he is not authorized. When the Reverse Proxy receives an invalid cookie such as an old-timer cookie or a spoofed cookie, the application returns an URL pointing to an error Web page. This is done by calculating "ICD" value and comparing it with the one received in HTTP request. For testing this, we installed Paros [20] proxy in the client side. We manually changed the cookie value field. Then, we sent the amended request to the Reverse Proxy. After "ICD" calculation, this proxy detects that the received cookie is not conform to the one previously send and it calls the user authentication process.

VII. Performance Evaluation

This section presents experimental tests to evaluate the performance of our proposed approach. We show a comparative analysis of the present proposal and the one described in

Apache "mod-but" module. These measurements examine how the encapsulation/decapsulation process affects the Reverse Proxy performance.

As mentioned above, one disadvantage of "mod-but" solution includes security concerns, that is, confidentiality and integrity of internal servers' cookies had not been provided. So to have a fair comparison between the cost of our approach and that of "mod-but" approach, we have added the latter with integrity control and confidentiality mechanisms. We then evaluate the Reverse Proxy loads performance for these two approaches.

A. Web Workload Characterizations

Several works attempt to study overall Web applications performance that depend on clients activity, network characteristics and servers behavior. [21], [22], [23], [24], [25], [26], [27], [28] show the effects of client characterizations on Web server performance, during a session browsing. For example, user side caching mechanism has been evaluated by some works to show its effects on Web server performance. Furthermore, [29], [30], [31], [32], [33] describe how network characteristics, such as HTTP protocol characterizations and TCP packet level characterizations, can affect performance evaluation of Web server. In addition, Web server behavior is studied in many researches. Indeed, evaluation of Web server performance depends on workload characterizations such as request file types (static and dynamic files), transfer sizes, response size, clients numbers, clients domain and other statistics characterizations described in [34], [35], [36], [37], [38]. Another number of studies such as [39], [40], [41] evaluate Web server performance using synthetically generated loads.

To evaluate the performance of different Web servers, several benchmarking tools have been developed. SPECweb96 [42], WebStone [43], Surge [38], httperf [44] and Apache benchmarking [45] are the most popular and available tools. Most of these current tools are simple benchmarks that generate the workload by sequentially creating HTTP requests in each experimental test. Some complex tools, such as Surge, focused on more advanced workload characterizations (depend on client activity, network characteristics and server behavior) to improve the evaluation of server testing performance.

In the present paper, tests focused on Reverse Proxy performance. Observations attend to evaluate the effects of the number of users on the Reverse Proxy behavior and to determine the capacity of the mandatory to handle many simultaneous users sessions. To perform this goal, a simple testing tool is sufficiently adequate. In this work, the Apache benchmarking tool, *ab*, is used to analyze server proxy performance and to show how many requests per second can be served.

B. Description of the Benchmark

[46] provides steps for managing performance testing. We use the methodologies of this guidance in order to have efficient experimental tests that focused on evaluation of Reverse Proxy performance.

These tests are run in a UNIX environment. The proxy machine is a 2.4 GHz Intel 2 Duo processor and 3534 MB of RAM. The observations are set by using *ab* [45] tool for Apache benchmarking. This tool provides Web server performance by measuring three essential parameters: time per request, number of requests per second and throughput (kilo bytes per second). These parameters are well studied in the present work in order to compare performances of our approach and "mod-but" approach. In fact, the time per request measures the amount of time that the Reverse Proxy takes to process a request. The number of requests per second shows the number of requests treated by the Reverse Proxy per unit of time. This measures the capability of processing demands at a given time versus number of clients. Throughput represents the average number of bytes transferred every second from the server to all concurrent users.

These tests are performed by configuring the number of requests to 10000, the maximal number of clients to 300 and the time between two tests to 5 second. In addition, the file size used in these tests is 8031 bytes. Indeed, measurements are taken by increasing the number of clients by ten in each test. Therefore, thirty experiences are performed. The results are shown in figures 4, 5 and 6.

C. Experimental Results

1) Request Generation Rate Comparison: In this section, we study the number of requests treated by the Reverse Proxy per second. This test shows measurements and experimental results of the two approaches. The first one is the "mod-but" implementation and the second is our approach. Figure 4 shows plots of the number of requests per second as a function of the number of concurrent clients. The X-axis lists the concurrent client numbers, while the Y-axis gives the number of requests per second. We can observe that, under

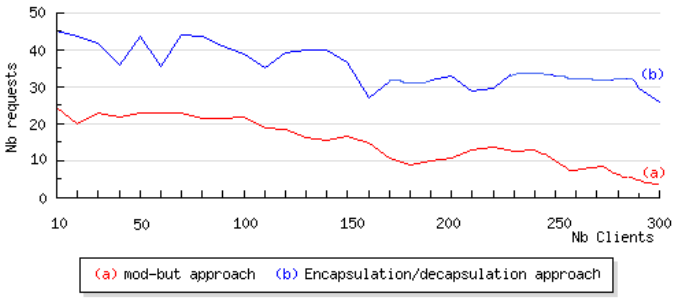

Fig. 4. Request rate versus number of clients

our approach, the Reverse Proxy can treat between 28 and 45 requests per second. This is to be compared to a number of serviced requests varying between 8 and 25 per second with "mod-but". In both cases, number of requests decreases as the number of allowed concurrent clients increases. However, performances of approach (b) are better than those of approach (a). We note that the number of processed queries in our approach is approximately twofold of "mod-but" approach.

We also note that with the “mod-but” approach , the platform is saturated if the number of clients reaches the maximum (300)and the number of requests becomes about two requests per second.

2) Latency Comparison: This section shows the processing time per request varies as a function of the number of clients in both “mod-but” approach and encapsulation/decapsulation approach. In the following figure, processing time is measured in milliseconds, the X-axis lists the number of concurrent clients and the Y-axis gives the number of requests per second.

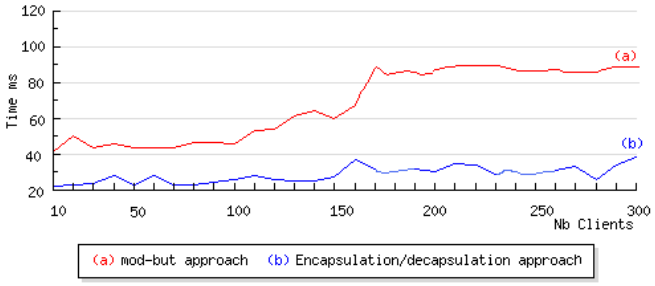

Fig. 5. Time per request measurements

We conclude from figure 5 that the processing time per request is much larger in the “mod-but” approach than ours. Under our solution, latency increases slowly, whereas under the “mod-but”, it widely increases with the number of concurrent clients. Indeed, if the number of clients is less than half maximal number of allowed simultaneous clients, the time per request is approximately stable and varies between 20 and 28 ms in our approach. However, “mod-but” takes longer time that is varied between 41 and 60 ms per request. When a large number of concurrent clients accessing the Reverse Proxy, it takes between 30 and 40 ms to service each request with our approach. Nevertheless, it strongly increases with the “mod-but” approach and it can take up to 85 ms per request. This proves that the encapsulation/decapsulation is an optimal solution that improves Reverse Proxy loads performance and reduces the bottleneck probability.

3) Throughput Comparison: Throughput parameter evaluates the overload behavior of the Reverse Proxy. Figure 6 demonstrates the variance of throughput versus the number of clients in both “mod-but” approach and encapsulation/decapsulation approach. The X-axis lists the number of concurrent clients and the Y-axis gives throughput in kilo bytes per second.

As expected in figure 6, the throughput rate generally degrades as the concurrent clients number increases. We observe the throughput improvements achieved by encapsulation/decapsulation process. In this process, the Reverse Proxy throughput varies between 250 and 400 Kbps. It declines initially slowly with few numbers of clients, and then, it declines more rapidly reaching about 150 concurrent clients. However, for a “mod-but” approach, the results are getting worse and they are less than 100 kbps even with a single client.

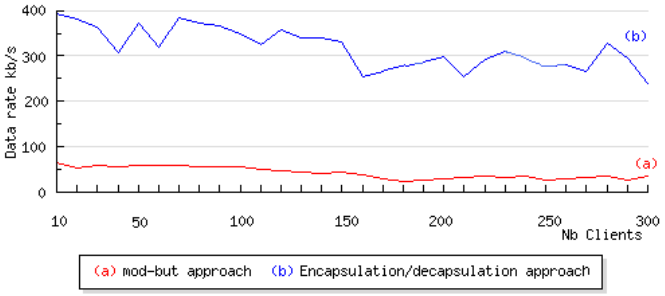

Fig. 6. Throughput measurements

D. Summary of Benchmark Results

The results presented above illustrate a significant variance performance of the two approaches in throughput, latency and request generation rate. These results show that encapsulation/decapsulation approach consistently outperforms the “mod-but” approach. Indeed, our approach ensures the lowest latency to client requests. Furthermore, it maintains a high throughput as number of concurrent clients increases. In addition, it provides a high generation request rate that is approximately the twofold of “mod-but” approach. However, under “mod-but” approach, throughput degrades rapidly and latency also rises significantly versus a high number of concurrent clients. These results demonstrate that, to alleviate bottlenecks, Reverse Proxy must include encapsulation/decapsulation approach to manage HTTP users sessions. These results affirm our hypothesis that this process can achieve better performance.

VIII. Conclusion

This paper has introduced an architecture model for securing Web-based applications. This architecture is based on an HTTP Reverse Proxy that protects back-end Web servers from unauthorized use. The present work is focused on a new Reverse Proxy function for managing end-to-end HTTP sessions. This function provides a way for maintaining user state. To this end, we presented the encapsulation/decapsulation process that consists in transparently handling a secure user session based on cookies. This process is mainly focused on securing internal servers cookies management by ensuring integrity and confidentiality services.

Apaches modules had been used for the implementation of the present architecture. The encapsulation/decapsulation approach has been quantitatively and qualitatively validated by using benchmark tests. Simulations evaluate the effects of this approach on Reverse Proxy behavior. The results show the efficiency of this approach which significantly alleviate Reverse Proxy loads.

References

[1] “Admin proxy,” 2008, available at `systematic-paris-region.org/fr/projets/admin-proxy`.

[2] J. M. H. F. L. M. P. L. R. Fielding, J. Gettys and T. Berners-Lee, “Hypertext Transfer Protocol – HTTP/1.1,” Internet Engineering Task Force, 1999.

[3] P. Sommerlad, "Reverse proxy patterns," in *European Conference on Pattern Languages of Programming (EuroPLoP)*, 2003.

[4] "Owsap consortium," 2010, available at `http://www.owasp.org/`.

[5] D. Kristol and L. Montulli, "HTTP State Management Mechanism," Internet Engineering Task Force, 2000.

[6] S. A. A. I. Pujolle, G., "Secure session management with cookies," *Information, Communications and Signal Processing (ICICS 2009)*, pp. 1–6, 2009.

[7] T. Dierks and E. Rescorla, "The TLS protocol version 1.1," 1999.

[8] "Openldap foundation," 2010, available at `http://www.openldap.org/`.

[9] D. M. Kristol, "Http cookies: Standards, privacy, and politics," *ACM Trans. Internet Techn.*, vol. 1, no. 2, pp. 151–198, 2001.

[10] "The apache software foundation," 2010, available at `http://www.apache.org/`.

[11] I. Buetler", ""mod_but"," 2006, compass Security AG, available at `http://www.but.ch/mod_but/`.

[12] D. Endler, "Brute-Force Exploitation of Web Application Session IDs," *Retrieved from http://www. cgisecurity. com*.

[13] D. Xu, C. Lu, and A. Dos Santos, "Protecting web usage of credit cards using one-time pad cookie encryption," in *Computer Security Applications Conference, 2002. Proceedings. 18th Annual*, 2002, pp. 51–58.

[14] D. Kristol, "HTTP Cookies: Standards, privacy, and politics," *ACM Transactions on Internet Technology (TOIT)*, vol. 1, no. 2, p. 198, 2001.

[15] D. Endler, "The evolution of cross site scripting attacks," *Whitepaper, iDefense Inc.(May 2002) http://www. cgisecurity. com/lib/XSS. pdf*.

[16] C. Karlof, U. Shankar, D. Tygar, D. Wagner, and C. Karlof, "Locked cookies: Web authentication security against phishing, pharming, and active attacks," 2007.

[17] H. Krawczyk, M. Bellare, and R. Canetti, "HMAC: Keyed-hashing for message authentication," 1997.

[18] H. F. N. D. R. O. M'Raihi D., Bellare M., "HOTP: An HMAC-Based One-Time Password Algorithm," Internet Engineering Task Force, 2005.

[19] "Mod-proxy, the apache software foundation," 2010, available at `http://httpd.apache.org/docs/2.0/mod/mod-proxy.html`.

[20] Chinotec Technologies Company, 2004, available at `http://www.parosproxy.org/`.

[21] W. Feng and H. Chen, "Performance optimization for web caching based on input modeling," *Journal of Computational Methods in Science and Engineering*, vol. 9, pp. 149–157, 2009.

[22] P. Barford, A. Bestavros, A. Bradley, and M. Crovella, "In World Wide Web, Special Issue on Characterization and Performance Evaluation Changes in Web Client Access Patterns Characteristics and Caching Implications," 2008.

[23] A. Bestavros, P. Barford, A. Bradley, and M. Crovella, "Changes in web client access patterns: Characteristics and caching implications," *World Wide Web*, vol. 2, no. 1, pp. 15–28, 1999.

[24] J. Mogul, "The Case for Persistent-Connection HTTP Abstract," 2008.

[25] A. Balamash, M. Krunz, and P. Nain, "Performance analysis of a client-side caching/prefetching system for Web traffic," *Computer Networks*, vol. 51, no. 13, pp. 3673–3692, 2007.

[26] M. Kellar, K. Hawkey, K. Inkpen, and C. Watters, "Challenges of capturing natural Web-based user behaviors," *International Journal of Human-Computer Interaction*, vol. 24, no. 4, pp. 385–409, 2008.

[27] A. Mahanti and C. Williamson, "Web proxy workload characterization," *Progress Report, Computer Sciences Dept, Univ. of Saskatchewan*, 1999.

[28] T. Kelly and D. Reeves, "Abstract Optimal Web cache sizing: scalable methods for exact solutions ," 2008.

[29] S. Khaunte and J. Limb, "Statistical characterization of a world wide web browsing session," 1997.

[30] J. Heidemann, K. Obraczka, and J. Touch, "Modeling the performance of HTTP over several transport protocols," *IEEE/ACM Transactions on Networking (TON)*, vol. 5, no. 5, pp. 616–630, 1997.

[31] H. Nielsen, J. Gettys, A. Baird-Smith, E. Prud'hommeaux, H. Lie, and C. Lilley, "Network performance effects of HTTP/1.1, CSS1, and PNG," in *Proceedings of the ACM SIGCOMM'97 conference on Applications, technologies, architectures, and protocols for computer communication*. ACM, 1997, p. 166.

[32] H. Balakrishnan, V. Padmanabhan, S. Seshan, M. Stemm, and R. Katz, "TCP behavior of a busy Internet server: Analysis and improvements," in *IEEE INFOCOM*, vol. 1. Citeseer, 1998, pp. 252–262.

[33] E. Nahum, M. Rosu, S. Seshan, and J. Almeida, "The effects of wide-area conditions on WWW server performance," *ACM SIGMETRICS Performance Evaluation Review*, vol. 29, no. 1, pp. 257–267, 2001.

[34] M. Arlitt and C. Williamson, "Web server workload characterization," *ACM SIGMETRICS Performance Evaluation Review*, vol. 24, no. 1, pp. 126–137, 1996.

[35] I. Boston, A. Cc, A. Bestavros, R. Carter, M. Crovella, C. Cunha, A. Heddaya, and S. Mirdad, "Application-Level Document Caching in the Internet," 1995.

[36] H. Braun and K. Claffy, "Web traffic characterization: an assessment of the impact of caching documents from NCSA's web server* 1," *Computer Networks and ISDN Systems*, vol. 28, no. 1-2, pp. 37–51, 1995.

[37] A. Williams, M. Arlitt, C. Williamson, and K. Barker, "Web workload characterization: Ten years later," *Web content delivery*, pp. 3–21, 2005.

[38] P. Barford and M. Crovella, "Generating representative web workloads for network and server performance evaluation," in *Proceedings of the 1998 ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems*. ACM, 1998, p. 160.

[39] J. Almeida, V. Almeida, and D. Yates, "Measuring the behavior of a world-wide web server," Citeseer, Tech. Rep., 1996.

[40] G. Banga and P. Druschel, "Measuring the capacity of a web server," in *Proceedings of the USENIX Symposium on Internet Technologies and Systems (USITS)*, 1997, pp. 61–71.

[41] G. Banga and J. Mogul, "Scalable kernel performance for Internet servers under realistic loads," in *Proceedings of the annual conference on USENIX Annual Technical Conference*. USENIX Association, 1998, p. 1.

[42] "The standard performance evaluation corporation," 2003, available at `http://www.spec.org/osg/web96`.

[43] G. Trent and M. Sake, "WebSTONE: The first generation in HTTP server benchmarking," *Silicon Graphics, Inc. whitepaper, February*, 1995.

[44] D. Mosberger and T. Jin, "httperfa tool for measuring web server performance," *ACM SIGMETRICS Performance Evaluation Review*, vol. 26, no. 3, pp. 31–37, 1998.

[45] "Apache http server benchmarking tool," 2009, available at `http://httpd.apache.org/docs/2.0/programs/ab.html`.

[46] J. Meier, C. Farre, P. Bansode, S. Barber, and D. Rea, "Performance testing guidance for web applications: patterns & practices," 2007.