A Mini-project Report
on

HTTP Session Management : Architecture and Cookie Security

carried out as part of the course Web Technologies and Application(IT302)

Submitted by

S Ashish Bharadwaj	***(12IT63) V Sem B.Tech (IT)***
Keerthi Prasad N	***(12IT34) V Sem B.Tech (IT)***
Aneesh S	***(12IT06) V Sem B.Tech (IT)***
Suhas H S	***(12IT85) V Sem B.Tech (IT)***
Ronan Le Gall	***(14EX01)***

in partial fulfillment for the award of the degree

of

BACHELOR OF TECHNOLOGY
In

INFORMATION TECHNOLOGY

Department of Information Technology

National Institute of Technology Karnataka, Surathkal

November 2014

CERTIFICATE

This is to certify that the project entitled **"HTTP Session Management : Architecture and Cookies Security"** is a bonafide work carried out as part of the course **Web Technologies and Applications (IT302)**, under my guidance by (1), (2), (3), (4) (5), students of V Sem B.Tech (IT) at the Department of Information Technology, National Institute of Technology Karnataka, Surathkal, during the academic semester V, in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in Information Technology, at NITK Surathkal.

Place:

Date: Signature of the Instructor

(1)	***S Ashish Bharadwaj***	***(12IT63) V Sem B.Tech (IT)***
(2)	***Keerthi Prasad N***	***(12IT34) V Sem B.Tech (IT)***
(3)	***Aneesh S***	***(12IT06) V Sem B.Tech (IT)***
(4)	***Suhas H S***	***(12IT85) V Sem B.Tech (IT)***
(5)	***Ronan Le Gall***	***(14EX01)***

DECLARATION

I hereby declare that the project entitled “**HTTP Session Management : Architecture and Cookies Security**” submitted as part of the partial course requirements for the course Web Technologies and Applications (IT302) for the award of the degree of Master of Technology in Information Technology at NITK Surathkal during the __Jul - Nov 2014___ semester has been carried out by me. I declare that the project has not formed the basis for the award of any degree, associateship, fellowship or any other similar titles elsewhere.

Further, I declare that I will not share, re-submit or publish the code, idea, framework and/or any publication that may arise out of this work for academic or profit purposes without obtaining the prior written consent of the course Faculty Mentor and Course Instructor.

Place: National Institute of Technology Karnataka, Surathkal

Date: 19-11-2014

Signature of the Student:

(1) S Ashish Bharadwaj
12IT63

(2) Keerthi Prasad N
12IT34

(3) Aneesh S
12IT06

(4) Suhas H S
12IT85

(5) Ronan Le Gall
14EX01

Abstract

Maintaining security over the web has become one of the most important concern on the network. The traditional method of maintaining the user state using cookies is not safe anymore. Capturing and manipulating these cookies causes threat to security. So it’s important to maintain integrity and confidentiality of the cookies. In this paper, we propose a method which does the same. All works, to ensure security over the web were carried out with studying the behavior and implementation of reverse proxy. It not only helped in security but also in efficiency of the system. Reverse proxy employs cookie as its ticket, so it can easily handle the user identification of enterprise. We discusses how to revise the transmitting cookie value of Reverse proxy and adjust it to the communication between background server and client entry, in order to maintain the effective cookie session between the client and the server. In order to improve efficiency, the reverse proxy balances the load among various application servers.

KEYWORDS: Web application, HTTP, cookies, Reverse Proxy, Session Management, Node.js and Integrity Cookie Digit.

.

TABLE OF CONTENTS

TOPIC	Page no
1. Introduction	1
2. Literature Survey	3
2.1 Background	3
2.2 Outcome of Literature Survey	5
2.3 Problem Statement	7
2.4 Objectives	8
3. Methodology	9
4. Implementation	10
5. Conclusion and Future works	16
Reference	17

LIST OF FIGURES

Page no

Fig 1. Cookie Exchange Process 1

Fig 2. An Apache Based Reverse Proxy Implementation 2

Fig 3. RP Cookie and AS Cookie structure 5

Fig 4. Sequence Diagram of the execution 7

Fig 5. The Application in works and the cookie in the Browser 10

Fig 6. From Client to Reverse Proxy (WireShark) 11

Fig 7. Reverse Proxy to Application Server 11

Fig 8. Application Server to Reverse Proxy 12

Fig 9. Reverse Proxy to the Client 12

Fig 10. Latency and Throughput analysis on Application Server without the Reverse Proxy 13

Fig 11. Latency and Throughput analysis on Application Server using Reverse Proxy 14

Fig 12. Heavy Request Analysis 14

1. Introduction

In computer networks, a **Reverse Proxy** is a type of proxy server that retrieves resources on behalf of a client from one or more servers. These resources are then returned to the client as though they originated from the server itself (or servers themselves). While a forward proxy acts as an intermediary for its (usually nearby) associated clients and returns to them resources accessible on the Internet, a reverse proxy acts as an intermediary for its (usually nearby) associated servers and only returns resources provided by those associated servers.

Reverse proxy employs cookie as its ticket, so it can easily handle the user identification of enterprise. This project discusses how to revise the transmitting cookie value of Reverse proxy and adjust it to the communication between background server and client entry, in order to maintain the effective cookie session between the client and the server. The project analyzes the reason for the cookie is easy to attack, and works out a scheme of relaying the secure session, which covers the defect in protecting cookie.

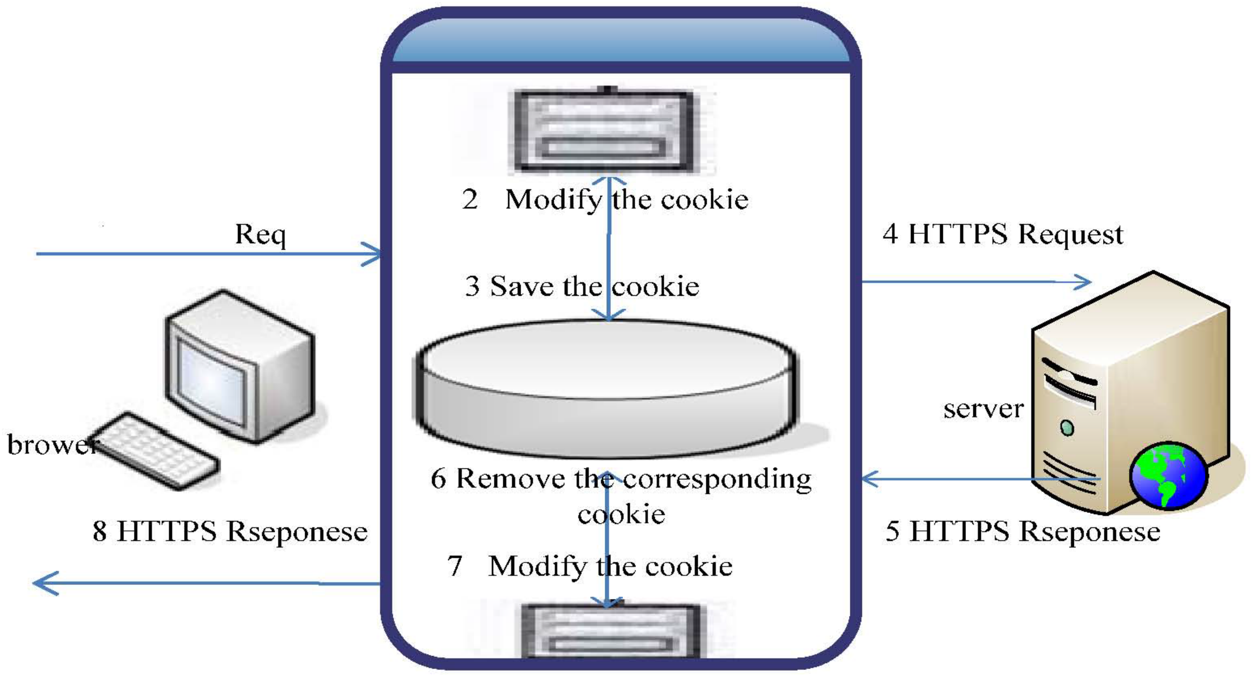


Fig 1. Cookie Exchange Process

Fig 1. Represents the traditional cookie exchange process in the current world scenario. The Server in the middle is the Reverse Proxy Server which forwards the request from the client to the server and back from the server to the client. Reverse Proxy modifies and stores the cookies that it receives from the Application Server and sends it back to the client. When the client sends back the cookie it modifies the cookie and checks in its cookie table for a valid result. The authentication process is done as above in the current world.

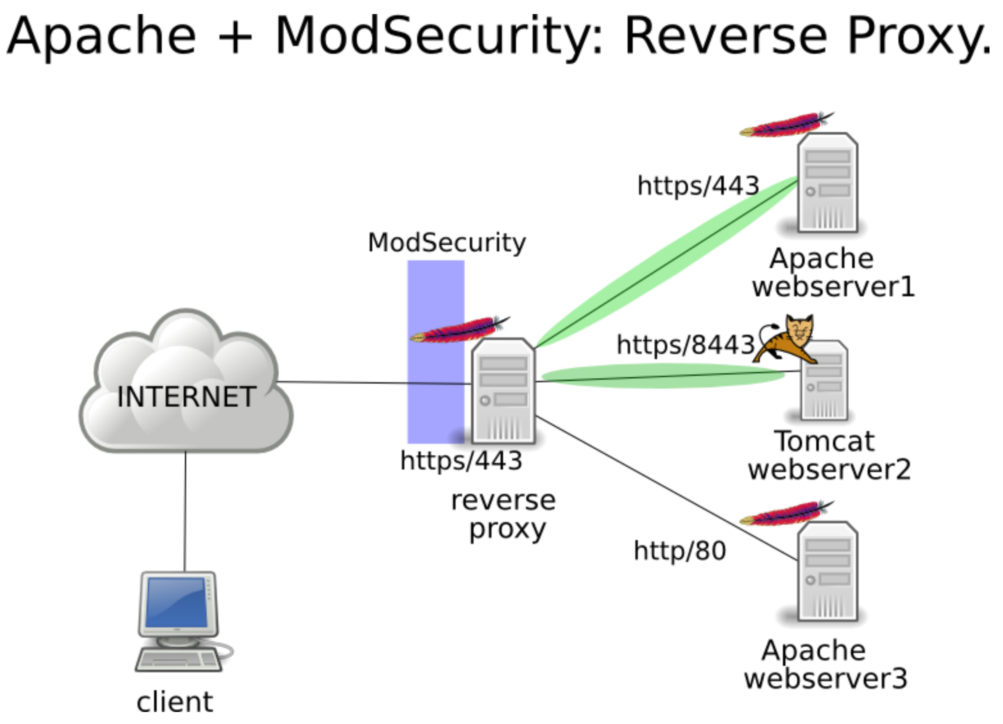


Fig 2. An Apache Based Reverse Proxy Implementation

Fig 2. Shows the working of a Reverse Proxy. It masks the multiple servers for the client. Thus the client has no need or idea of which server is processing his request. This helps in load balancing and scalability of the Entire Application. The terms are defined below.

Load balancing distributes workloads across multiple computing resources, such as computers, a computer cluster, network links, central processing units or disk drives. Load balancing aims to optimize resource use, maximize throughput, minimize response time, and avoid overload of any single resource. Using multiple components with load balancing instead of a single component may increase reliability through redundancy. Load balancing usually involves dedicated software or hardware, such as a multilayer switch or a Domain Name System server process.

Scalability is the ability of a system, network, or process to handle a growing amount of work in a capable manner or its ability to be enlarged to accommodate that growth.For example, it can refer to the capability of a system to increase its total output under an increased load when resources (typically hardware) are added. An analogous meaning is implied when the word is used in an economic context, where scalability of a company implies that the underlying business model offers the potential for economic growth within the company.

Reverse Proxy allows load balancing as when it receives a request from the client, it has a track of which of the n Application Servers are busy and the current load on each of them. Hence it can decide which Application Server that the request must be forwarded to. A simple round robin might suffice. Where the Reverse Proxy assigns Servers to a request in a circular fashion one after the other. But in very specific scenarios, a more complex scheduling algorithm might be needed. This algorithm decides which Application Server the request needs to be given to. Hence the Reverse Proxy helps in load balancing and thus the application becomes more scalable and thus it can handle more number of requests in a given time.

2. Literature Survey

2.1 Background

Managing internal servers cookies was implemented by an open source Apache module called “mod-but” .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, cookie session hijacking, etc.

HTTP cookies headers are detailed in "HTTP state management mechanism" standard 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.

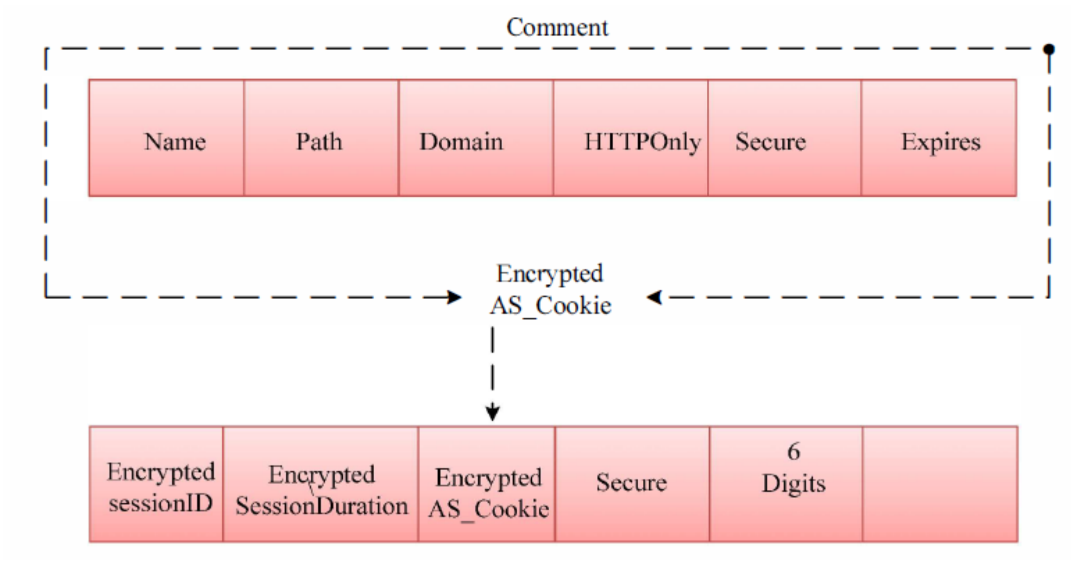


Fig 3. RP Cookie and AS Cookie structures

2.2 Outcome of Literature survey

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. Fig 4. represents the scenario of our approach for managing
HTTP session based on cookies.

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 RPcookie. This token allows a current user to be identified without proceeding a new authentication. Therefore the RPcookie 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.

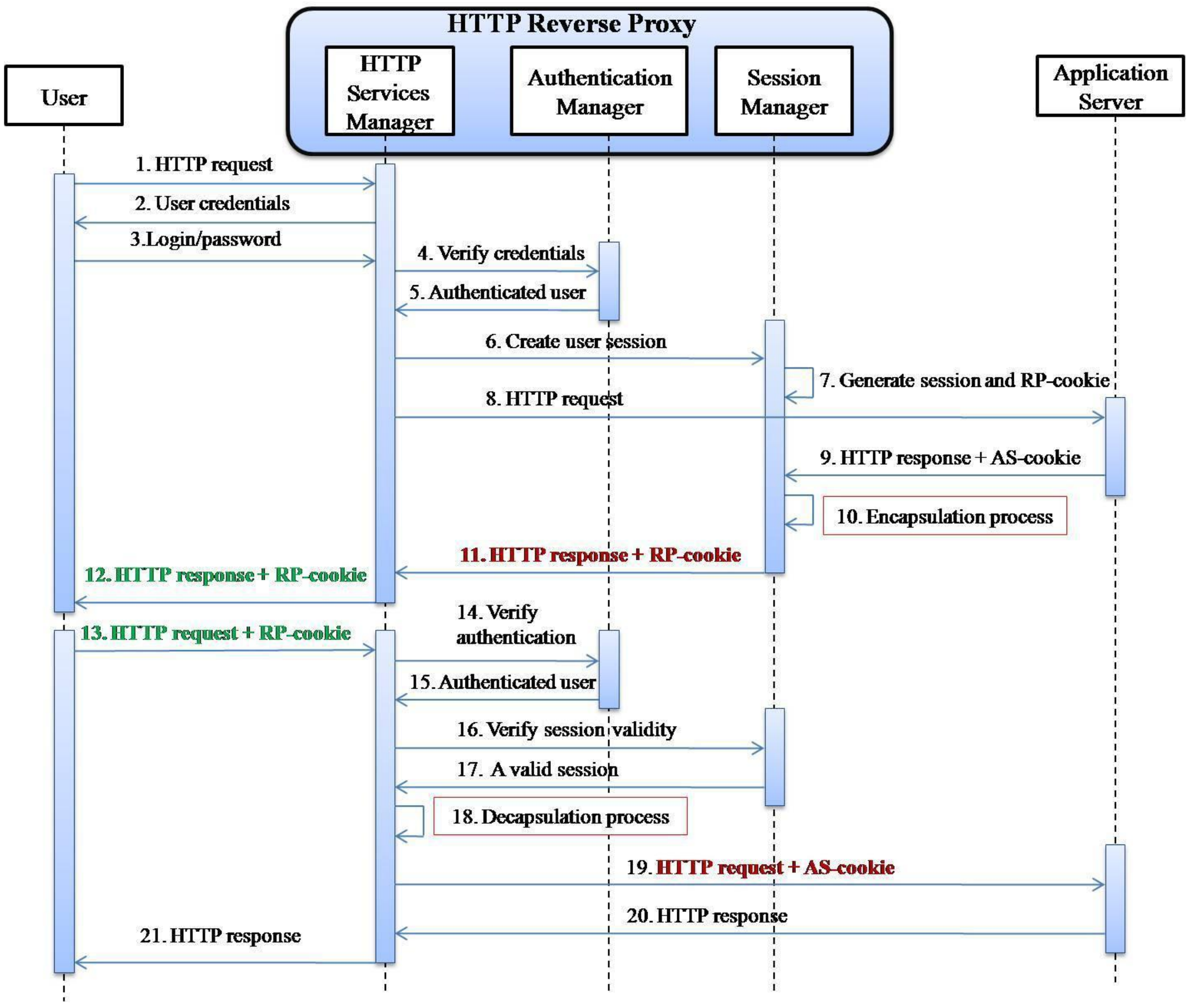


Fig 4. Sequence Diagram of the execution

2.3 Problem Statement

Implement a Reverse Proxy using any of the suitable technologies and modules, and connect an Application Server to the Client. The Reverse Proxy should handle all the authentication processes that are necessary for a successful user login. The Reverse Proxy must also handle the cookies that are sent from the client. Whenever a client sends a cookie AS Cookie, the Reverse Proxy must validate the authenticity of the cookie using ICD mechanism and perform the decryption process. On completion, it must forward the thus obtained AS Cookie to the desired Application Server. The Application Server is built in such a way that it accepts the AS Cookie to be valid and performs the Business Logic for the user Session. When the user logs in for the first time, the Reverse Proxy must be able to validate username to the entered password and forward request to the server. The Application Server, reads this request and sends a Set-Cookie header back to the Reverse Proxy. The Reverse Proxy thus accepts the given AS Cookie, and encapsulates all the AS Cookie values, encrypts them, and calculates ICD, appends it to the encrypted value and sends the thus obtained RP Cookie to the Client.

2.4 Objectives

Main objective of the project is to implement the paper "HTTP Session Management : Architecture and Cookies Security". The following analysis is aimed at:

1. ICD (Integrity Cookie Digit) calculation and its integration in the Reverse Proxy Level.
2. Benchmark Analysis on number of requests handled by the Application Server when Reverse Proxy is used.
3. Benchmark Analysis on number of requests handled by Application Server when Reverse Proxy is not used.
4. Benchmark Analysis for a new Server with heavy computation on a single request with load being divided amongst Reverse Proxy and Application Server
5. Benchmark Analysis for heavy computation on single request with load totally on Application Server.

3. Methodology

Tools Used:

1. node.js
2. jade template
3. bootstrap.css (frontend)
4. HTML 5
5. ApacheBench for Benchmark Analysis on the Server
6. Cryptography modules of node.js

The above tools were used to implement a reverse proxy. http_proxy opensource module of node.js was used to build a reverse proxy server which could handle forwarding of requests from Application Server to the Client and vice-versa.

It also handles cookie management, ICD calculation, cookie validity verification.

ICD calculation methodology is as follows :

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 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 .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.

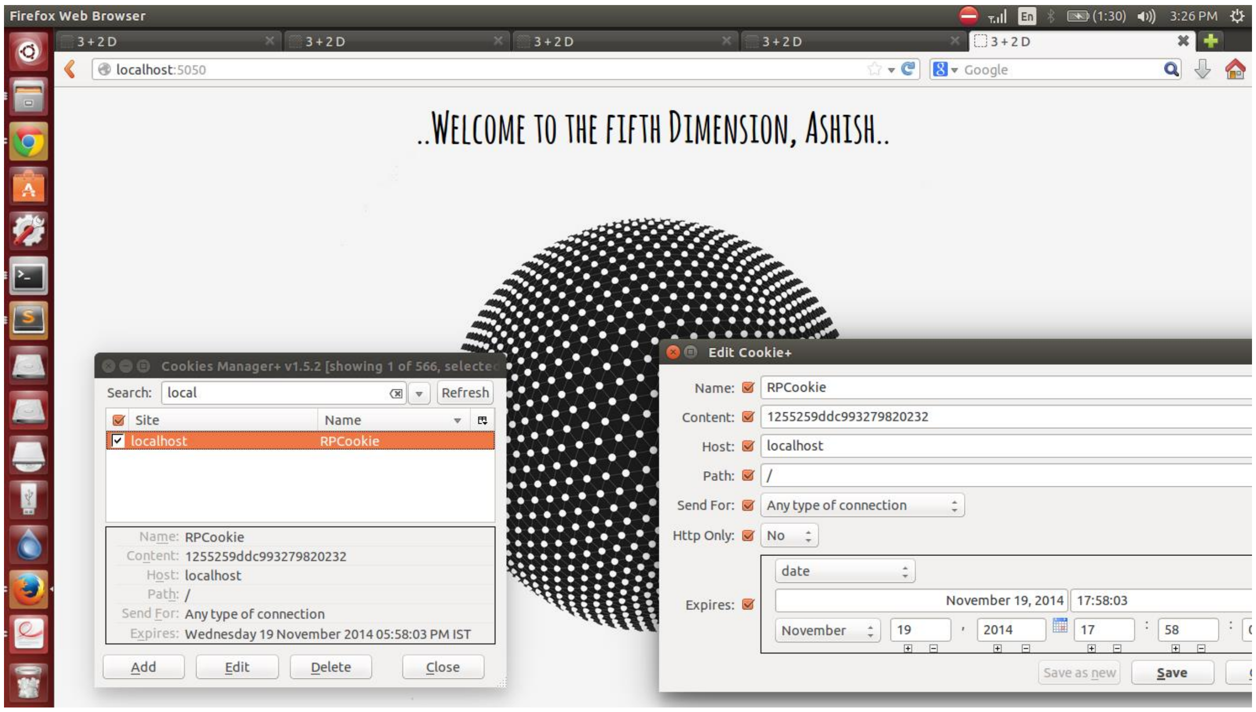
The objective of the “ICD” mechanism is to provide cookies integrity protection. Cookie manipulation is a well-known attack that allows an attacker to access to a context for which he is not authorized. When the Reverse Proxy receives an invalid cookie such as an old-timer cookie or a spoofed cookie, 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 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.

4. Implementation

4.1. Work Done

An application was written using node.js satisfying all the above requirements. Another application was written without a reverse proxy but performing the same task of cookie management. Both these applications were tested using ab tool and the results have been shown.

Another application was built with a reverse proxy to handle heavy requests. In this, the load was divided between the Reverse Proxy and the Application Server. Another server where load was totally on Application Server was made and the latency and throughput were analysed.



..WELCOME TO THE FIFTH DIMENSION, ASHISH..

Cookies Manager+ v1.5.2 [showing 1 of 566, selecte

Search: local

Site	Name
localhost	RPCookie

Name: RPCookie
Content: 1255259ddc993279820232
Host: localhost
Path: /
Send For: Any type of connection
Expires: Wednesday 19 November 2014 05:58:03 PM IST

Edit Cookie+

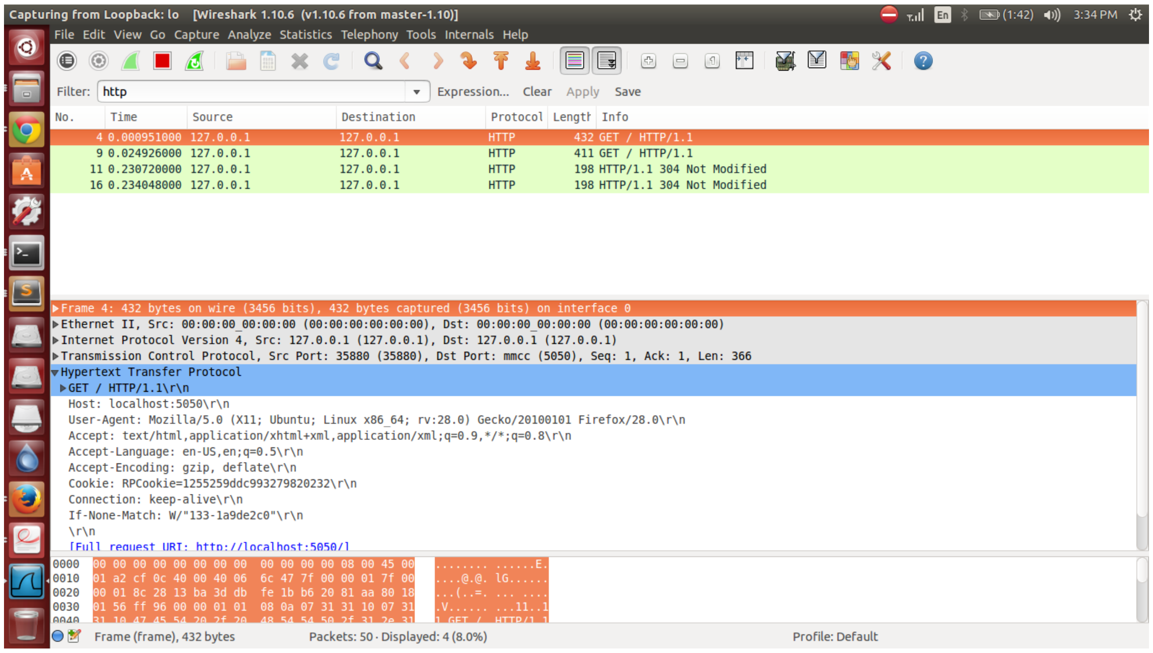
Name: RPCookie
Content: 1255259ddc993279820232
Host: localhost
Path: /
Send For: Any type of connection
Http Only: No
Expires: date
November 19, 2014 17:58:03
November 19, 2014 17 : 58

Fig 5. The Application in works and the cookie in the Browser

4.2 Results and Analysis

Application Server is on port 3000

Reverse Proxy is on port 5050


Capturing from Loopback: lo [Wireshark 1.10.6 (v1.10.6 from master-1.10)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter: http
Expression... Clear Apply Save

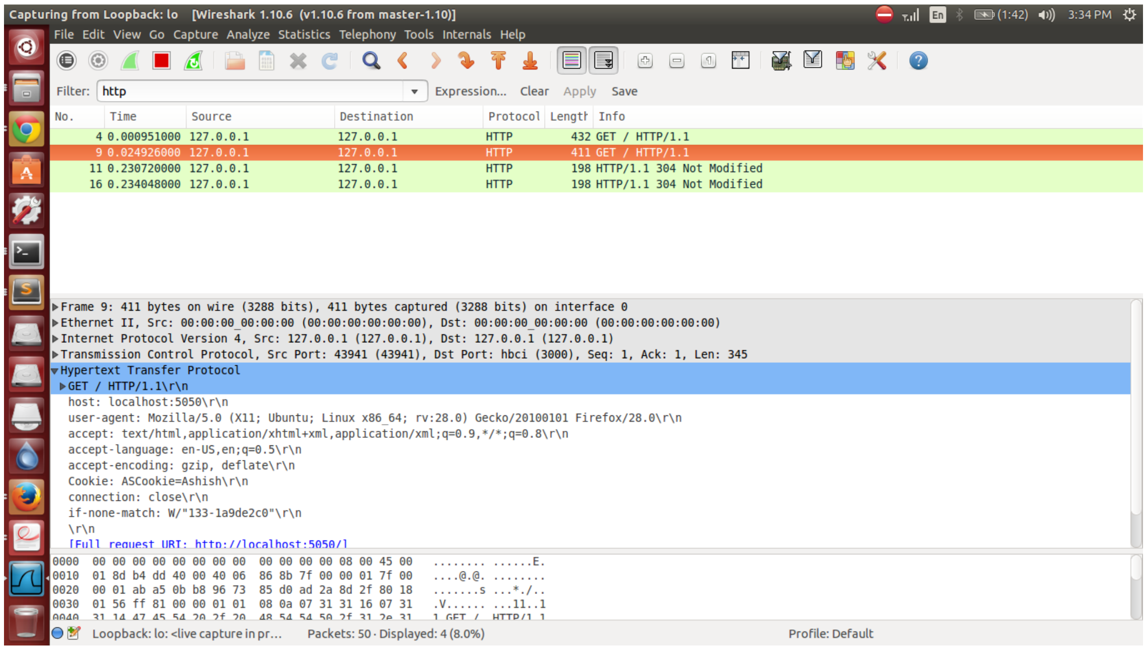
```
No.   Time        Source     Destination  Protocol Length Info
    4 0.000951000 127.0.0.1  127.0.0.1    HTTP      432 GET / HTTP/1.1
    9 0.024926000 127.0.0.1  127.0.0.1    HTTP      411 GET / HTTP/1.1
   11 0.230720000 127.0.0.1  127.0.0.1    HTTP      198 HTTP/1.1 304 Not Modified
   16 0.234048000 127.0.0.1  127.0.0.1    HTTP      198 HTTP/1.1 304 Not Modified
```

```
▶Frame 4: 432 bytes on wire (3456 bits), 432 bytes captured (3456 bits) on interface 0
▶Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
▶Internet Protocol Version 4, Src: 127.0.0.1 (127.0.0.1), Dst: 127.0.0.1 (127.0.0.1)
▶Transmission Control Protocol, Src Port: 35880 (35880), Dst Port: mmcc (5050), Seq: 1, Ack: 1, Len: 366
▼Hypertext Transfer Protocol
 ▶GET / HTTP/1.1\r\n
  Host: localhost:5050\r\n
  User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:28.0) Gecko/20100101 Firefox/28.0\r\n
  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n
  Accept-Language: en-US,en;q=0.5\r\n
  Accept-Encoding: gzip, deflate\r\n
  Cookie: RPCookie=1255259ddc993279820232\r\n
  Connection: keep-alive\r\n
  If-None-Match: W/"133-1a9de2c0"\r\n
  \r\n
  [Full request URI: http://localhost:5050/]
```

```
0000  00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 00   ........ ......E.
0010  01 a2 cf 0c 40 00 40 06  6c 47 7f 00 00 01 7f 00   ....@.@. lG......
0020  00 01 8c 28 13 ba 3d db  fe 1b b6 20 81 aa 80 18   ...(..=. ... ....
0030  01 56 ff 96 00 00 01 01  08 0a 07 31 31 10 07 31   .V...... ...11..1
0040  31 10 47 45 54 20 2f 20  48 54 54 50 2f 31 2e 31   1.GET /  HTTP/1.1
```

```
Frame (frame), 432 bytes     Packets: 50 · Displayed: 4 (8.0%)     Profile: Default
```

Fig 6. From Client to Reverse Proxy (WireShark)


Capturing from Loopback: lo [Wireshark 1.10.6 (v1.10.6 from master-1.10)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter: http
Expression... Clear Apply Save

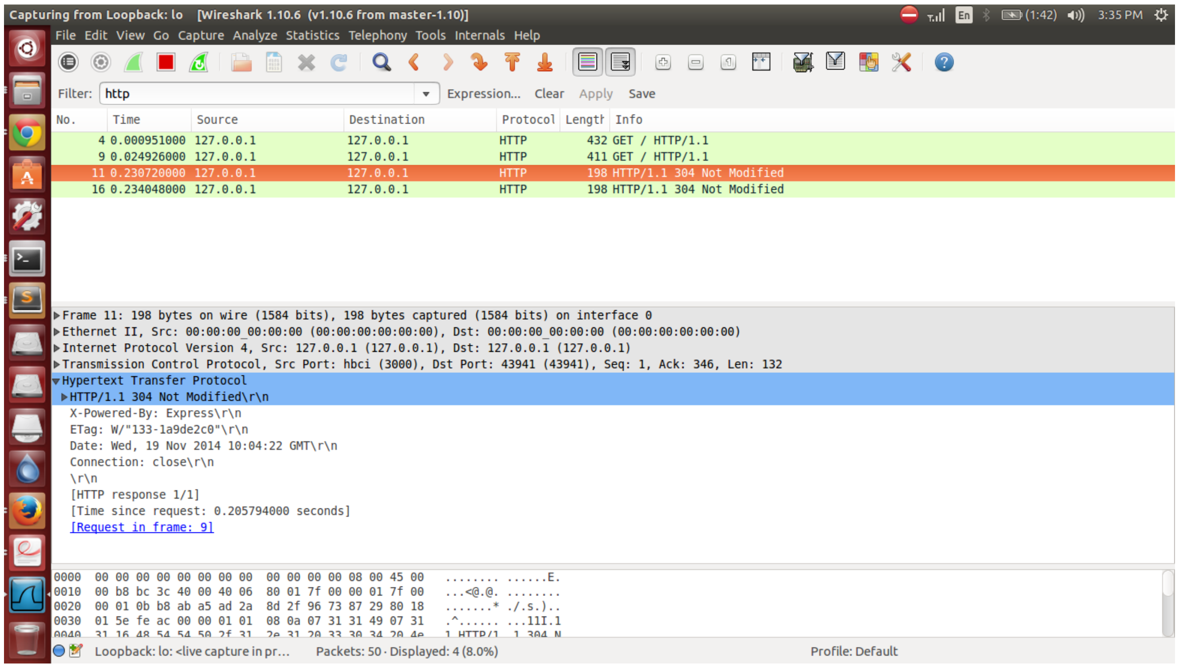
```
No.   Time        Source     Destination  Protocol Length Info
    4 0.000951000 127.0.0.1  127.0.0.1    HTTP      432 GET / HTTP/1.1
    9 0.024926000 127.0.0.1  127.0.0.1    HTTP      411 GET / HTTP/1.1
   11 0.230720000 127.0.0.1  127.0.0.1    HTTP      198 HTTP/1.1 304 Not Modified
   16 0.234048000 127.0.0.1  127.0.0.1    HTTP      198 HTTP/1.1 304 Not Modified
```

```
▶Frame 9: 411 bytes on wire (3288 bits), 411 bytes captured (3288 bits) on interface 0
▶Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
▶Internet Protocol Version 4, Src: 127.0.0.1 (127.0.0.1), Dst: 127.0.0.1 (127.0.0.1)
▶Transmission Control Protocol, Src Port: 43941 (43941), Dst Port: hbci (3000), Seq: 1, Ack: 1, Len: 345
▼Hypertext Transfer Protocol
 ▶GET / HTTP/1.1\r\n
  host: localhost:5050\r\n
  user-agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:28.0) Gecko/20100101 Firefox/28.0\r\n
  accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n
  accept-language: en-US,en;q=0.5\r\n
  accept-encoding: gzip, deflate\r\n
  Cookie: ASCookie=Ashish\r\n
  connection: close\r\n
  if-none-match: W/"133-1a9de2c0"\r\n
  \r\n
  [Full request URI: http://localhost:5050/]
```

```
0000  00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 00   ........ ......E.
0010  01 8d b4 dd 40 00 40 06  86 8b 7f 00 00 01 7f 00   ....@.@. ........
0020  00 01 ab a5 0b b8 96 73  85 d0 ad 2a 8d 2f 80 18   .......s ...*./..
0030  01 56 ff 81 00 00 01 01  08 0a 07 31 31 16 07 31   .V...... ...11..1
0040  31 14 47 45 54 20 2f 20  48 54 54 50 2f 31 2e 31   1.GET /  HTTP/1.1
```

```
Loopback: lo: <live capture in pr...     Packets: 50 · Displayed: 4 (8.0%)     Profile: Default
```

Fig 7. Reverse Proxy to Application Server


Capturing from Loopback: lo [Wireshark 1.10.6 (v1.10.6 from master-1.10)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter: http
Expression... Clear Apply Save

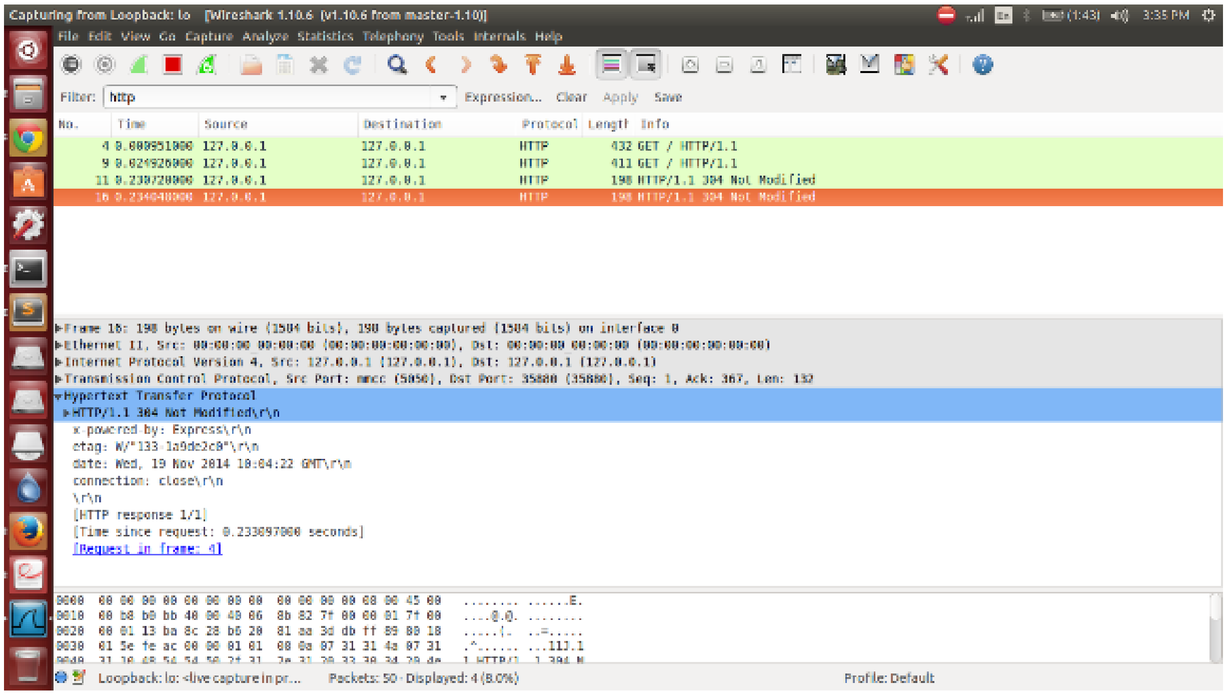
```
No.  Time         Source     Destination  Protocol Length Info
   4 0.000951000  127.0.0.1  127.0.0.1    HTTP     432 GET / HTTP/1.1
   9 0.024926000  127.0.0.1  127.0.0.1    HTTP     411 GET / HTTP/1.1
  11 0.230720000  127.0.0.1  127.0.0.1    HTTP     198 HTTP/1.1 304 Not Modified
  16 0.234048000  127.0.0.1  127.0.0.1    HTTP     198 HTTP/1.1 304 Not Modified
```

```
▶Frame 11: 198 bytes on wire (1584 bits), 198 bytes captured (1584 bits) on interface 0
▶Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
▶Internet Protocol Version 4, Src: 127.0.0.1 (127.0.0.1), Dst: 127.0.0.1 (127.0.0.1)
▶Transmission Control Protocol, Src Port: hbci (3000), Dst Port: 43941 (43941), Seq: 1, Ack: 346, Len: 132
▼Hypertext Transfer Protocol
 ▶HTTP/1.1 304 Not Modified\r\n
  X-Powered-By: Express\r\n
  ETag: W/"133-1a9de2c0"\r\n
  Date: Wed, 19 Nov 2014 10:04:22 GMT\r\n
  Connection: close\r\n
  \r\n
  [HTTP response 1/1]
  [Time since request: 0.205794000 seconds]
  [Request in frame: 9]
```

```
0000  00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 00   ........ ......E.
0010  00 b8 bc 3c 40 00 40 06  80 01 7f 00 00 01 7f 00   ...<@.@. ........
0020  00 01 0b b8 ab a5 ad 2a  8d 2f 96 73 87 29 80 18   .......* ./.s.)..
0030  01 5e fe ac 00 00 01 01  08 0a 07 31 31 49 07 31   .^...... ...11I.1
0040  31 16 48 54 54 50 2f 31  2e 31 20 33 30 34 20 4e   1.HTTP/1 .1 304 N
```

Loopback: lo: <live capture in pr... Packets: 50 · Displayed: 4 (8.0%) Profile: Default

Fig 8. Application Server to Reverse Proxy


Capturing from Loopback: lo [Wireshark 1.10.6 (v1.10.6 from master-1.10)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter: http
Expression... Clear Apply Save

```
No.  Time         Source     Destination  Protocol Length Info
   4 0.000951000  127.0.0.1  127.0.0.1    HTTP     432 GET / HTTP/1.1
   9 0.024926000  127.0.0.1  127.0.0.1    HTTP     411 GET / HTTP/1.1
  11 0.230720000  127.0.0.1  127.0.0.1    HTTP     198 HTTP/1.1 304 Not Modified
  16 0.234048000  127.0.0.1  127.0.0.1    HTTP     198 HTTP/1.1 304 Not Modified
```

```
▶Frame 16: 198 bytes on wire (1584 bits), 198 bytes captured (1584 bits) on interface 0
▶Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
▶Internet Protocol Version 4, Src: 127.0.0.1 (127.0.0.1), Dst: 127.0.0.1 (127.0.0.1)
▶Transmission Control Protocol, Src Port: mmcc (5050), Dst Port: 35880 (35880), Seq: 1, Ack: 367, Len: 132
▼Hypertext Transfer Protocol
 ▶HTTP/1.1 304 Not Modified\r\n
  x-powered-by: Express\r\n
  etag: W/"133-1a9de2c0"\r\n
  date: Wed, 19 Nov 2014 10:04:22 GMT\r\n
  connection: close\r\n
  \r\n
  [HTTP response 1/1]
  [Time since request: 0.233097000 seconds]
  [Request in frame: 4]
```

```
0000  00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 00   ........ ......E.
0010  00 b8 b0 bb 40 00 40 06  8b 82 7f 00 00 01 7f 00   ....@.@. ........
0020  00 01 13 ba 8c 28 b6 20  81 aa 3d db ff 89 80 18   .....(.  ..=.....
0030  01 5e fe ac 00 00 01 01  08 0a 07 31 31 4a 07 31   .^...... ...11J.1
0040  31 10 48 54 54 50 2f 31  2e 31 20 33 30 34 20 4e   1.HTTP/1 .1 304 N
```

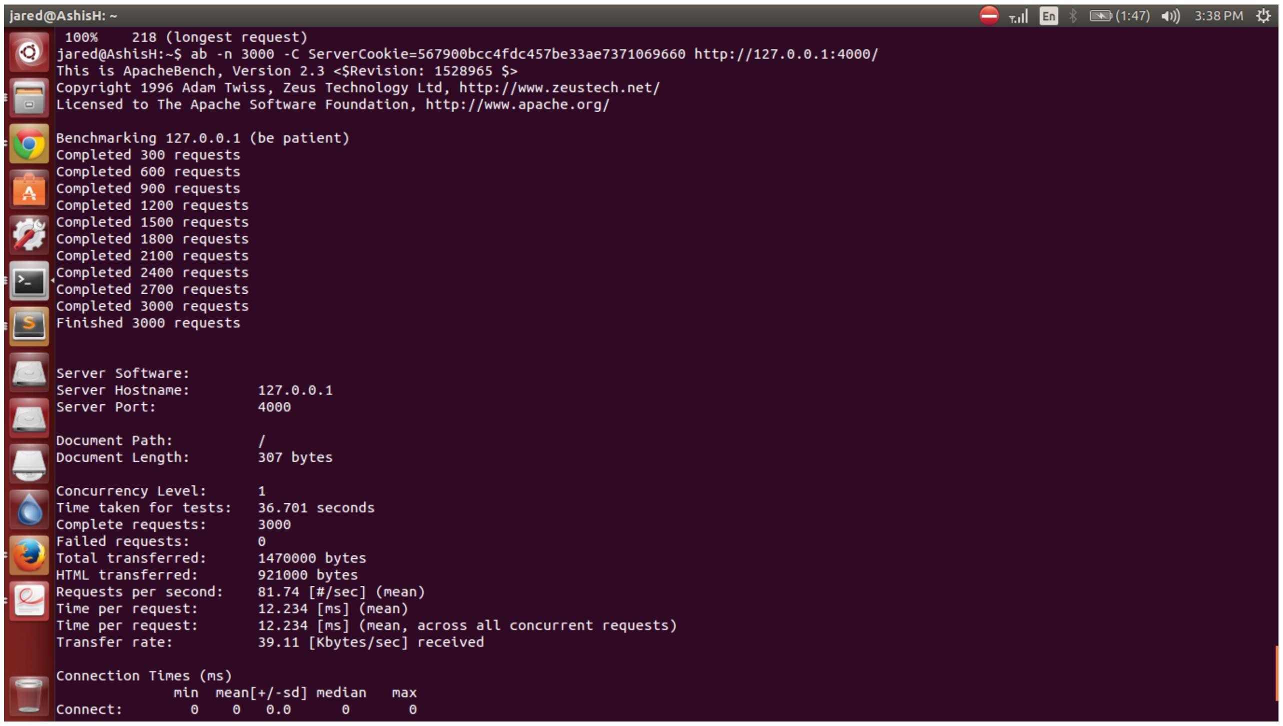
Loopback: lo: <live capture in pr... Packets: 50 · Displayed: 4 (8.0%) Profile: Default

Fig 9. Reverse Proxy to the Client

Observe in the before figures, the cookie and set-cookie headers and also the source and destination ports.

Below is the screenshots of the Benchmarking analysis done.

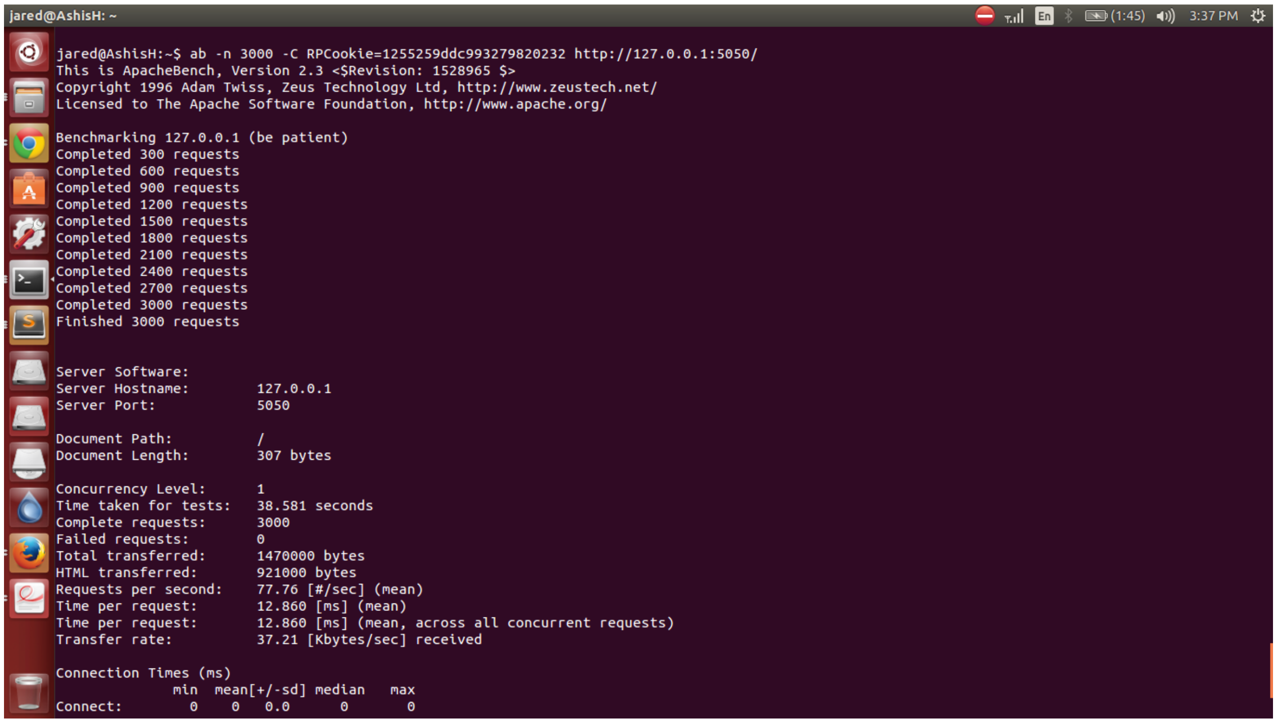
ab –n 3000 –C ServerCookie=567900bcc4fdc457be33ae7371069660 http://127.0.0.1:4000/



```
jared@AshisH: ~
 100%    218 (longest request)
jared@AshisH:~$ ab -n 3000 -C ServerCookie=567900bcc4fdc457be33ae7371069660 http://127.0.0.1:4000/
This is ApacheBench, Version 2.3 <$Revision: 1528965 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 127.0.0.1 (be patient)
Completed 300 requests
Completed 600 requests
Completed 900 requests
Completed 1200 requests
Completed 1500 requests
Completed 1800 requests
Completed 2100 requests
Completed 2400 requests
Completed 2700 requests
Completed 3000 requests
Finished 3000 requests


Server Software:
Server Hostname:        127.0.0.1
Server Port:            4000

Document Path:          /
Document Length:        307 bytes

Concurrency Level:      1
Time taken for tests:   36.701 seconds
Complete requests:      3000
Failed requests:        0
Total transferred:      1470000 bytes
HTML transferred:       921000 bytes
Requests per second:    81.74 [#/sec] (mean)
Time per request:       12.234 [ms] (mean)
Time per request:       12.234 [ms] (mean, across all concurrent requests)
Transfer rate:          39.11 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   0.0      0       0
```

Fig 10. Latency and Throughput analysis on Application Server without the Reverse Proxy

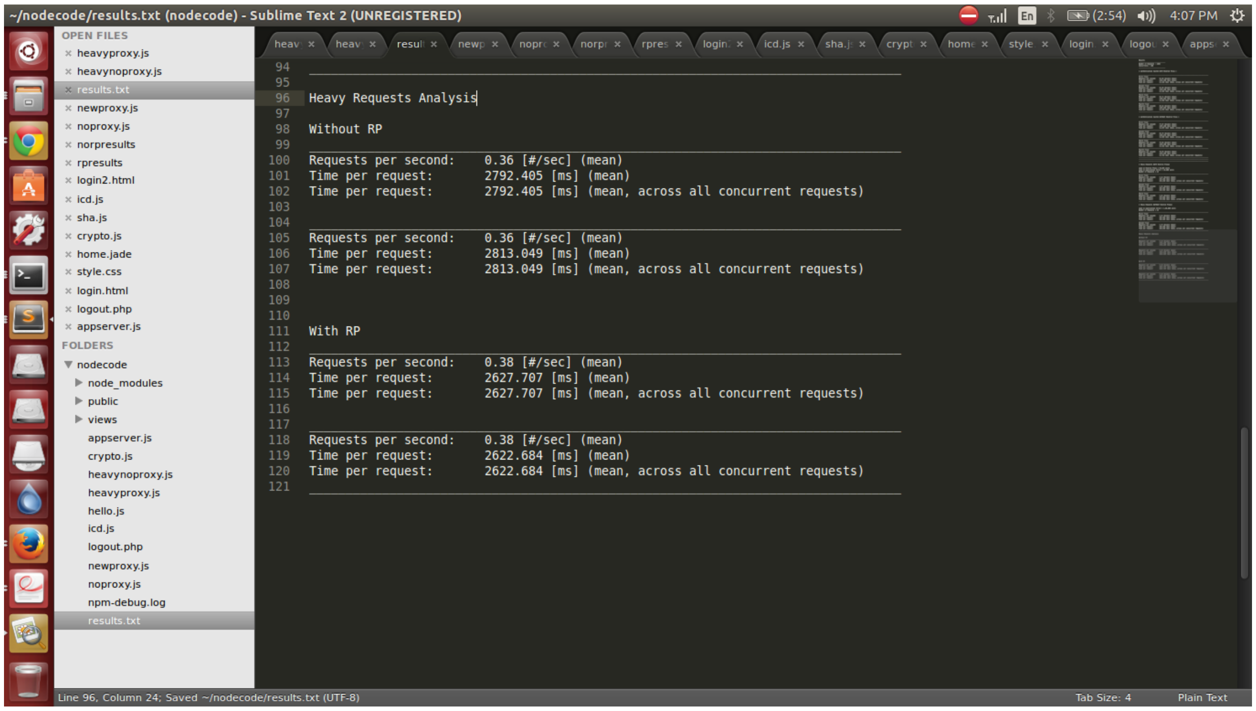
ab –n 3000 –C RPCookie=1255259ddc993272820232 http://127.0.0.1:5050/



```
jared@AshisH:~$ ab -n 3000 -C RPCookie=1255259ddc993279820232 http://127.0.0.1:5050/
This is ApacheBench, Version 2.3 <$Revision: 1528965 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 127.0.0.1 (be patient)
Completed 300 requests
Completed 600 requests
Completed 900 requests
Completed 1200 requests
Completed 1500 requests
Completed 1800 requests
Completed 2100 requests
Completed 2400 requests
Completed 2700 requests
Completed 3000 requests
Finished 3000 requests


Server Software:
Server Hostname:        127.0.0.1
Server Port:            5050

Document Path:          /
Document Length:        307 bytes

Concurrency Level:      1
Time taken for tests:   38.581 seconds
Complete requests:      3000
Failed requests:        0
Total transferred:      1470000 bytes
HTML transferred:       921000 bytes
Requests per second:    77.76 [#/sec] (mean)
Time per request:       12.860 [ms] (mean)
Time per request:       12.860 [ms] (mean, across all concurrent requests)
Transfer rate:          37.21 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   0.0      0       0
```

Fig 11. Latency and Throughput analysis on Application Server using Reverse Proxy

Below is the testing for Heavy requests.



```
Heavy Requests Analysis

Without RP

Requests per second:    0.36 [#/sec] (mean)
Time per request:       2792.405 [ms] (mean)
Time per request:       2792.405 [ms] (mean, across all concurrent requests)


Requests per second:    0.36 [#/sec] (mean)
Time per request:       2813.049 [ms] (mean)
Time per request:       2813.049 [ms] (mean, across all concurrent requests)



With RP

Requests per second:    0.38 [#/sec] (mean)
Time per request:       2627.707 [ms] (mean)
Time per request:       2627.707 [ms] (mean, across all concurrent requests)


Requests per second:    0.38 [#/sec] (mean)
Time per request:       2622.684 [ms] (mean)
Time per request:       2622.684 [ms] (mean, across all concurrent requests)
```

Fig 12. Heavy Request Analysis

A total of 1,00,000 iterations were made on each request, with RP handling half the load

Thus, from the analysis we can conclude that Reverse Proxy built on node.js performs better when the size of each request is increased

4.3. Innovative Work

1. Reverse Proxy has been implemented using node.js unlike in the paper where it has been implemented using apache mod_but module.

2. Reverse Proxy on node.js has proven to be faster when a number of requests with each request having heavy load is made than a normal standalone Application Server.

3. Throughput also shows a significant increase besides the latency.

5. Conclusions and future works

With the proxy-based SSG, the scheme can solve such problems as the cookie session relay, existing source identification, integrated control, and cookie attack replay. Then the main work is: lower the possibility of replaying cookie and targeting session by means of Session Duration, which can not solve completely the problem of synchronism. Use IP address as Session ID but client will have trouble using the dynamic IP. 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. On Analysis, we have proved that reverse proxy s better for implementing a heavy requests.

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.