KIT214 Assignment 2 Report  
Jagmeet’s LMS

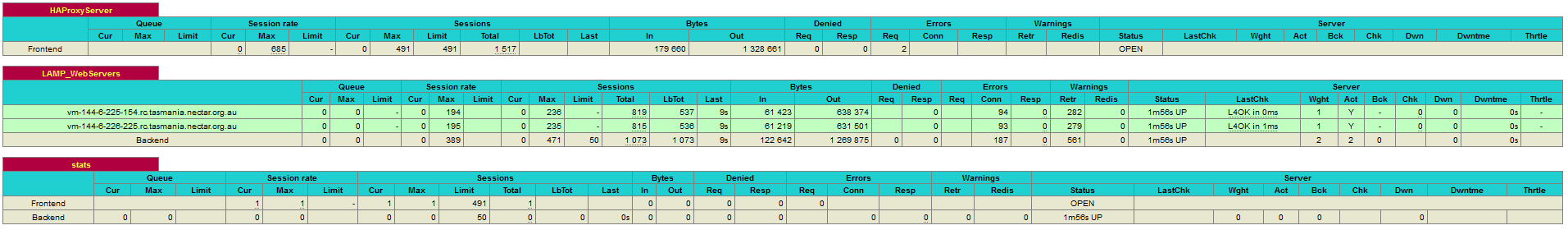
Jagmeet Singh Grewal

Justin Johnson

William Sutton

## 

Load Balancer Stats



## 

## Security

1. SQL Injection

SQL injection vulnerabilities were tested for using whitehat testing methods and in-band SQL injection attacks. On the login page, after selecting ‘Login’ on the initial page, the following text was entered into the ‘password’ field:

’ OR ‘1’=’1

The username field was left blank. When this was entered, the page acted as if the details of a correct user had been entered and proceeded to the search page. The welcome message displayed the name of a correct user, named Jagmeet. The base query sent to the database is as follows:

SELECT Name, email FROM users WHERE Username='$username' AND password='$password'

If a user submits details that match a database entry, the query will return an entry. The application will check if there is one or more entries returned by the database and authenticate the user if this is the case. In the above case, the query became:

SELECT Name, email FROM users WHERE Username='' AND password='' OR '1'='1'

This returns all users, as the conditional section of the query becomes a tautology. The user named Jagmeet was at the top of this list, so the application acted as if this user had logged in. This vulnerability affects all model files which authenticate users:

/Model/model.php

/Model/model2.php

/Model/model3.php

Three techniques were implemented to defend against further SQL injections in this manner. User input variables were sanitised for special characters using the real\_escape\_string function. Prepared statements and parameter binding were also used to prevent attacks such as those using the union operator. Finally, the code was changed so that it would only authenticate a user if the database result contained only one row. The original attack was repeated after this and did not succeed.

1. Cross-Site Scripting

Cross-site scripting vulnerabilities were tested for using whitehat testing methods and persistent XSS attacks. A user was registered with the following name:

<h1>Big Text</h1>

When this user was logged in, it displayed the text ‘Big Text’ as a heading. This shows that the user input was interpreted as HTML code rather than as text, which demonstrates the vulnerability of this application to cross-site scripting.

This attacks affects search.php, which displays the name of the user that just logged in or has just registered. All model files are affected as they save the name of the user to the database.

Two measures were implemented to prevent further XSS attacks. Firstly, HTML tags were filtered for using the strip\_tags() PHP method. User input in the registration was also validated. This included using regular expressions to prevent any potentially dangerous characters from being entered into the database. The Regex now only allows letters and whitespaces. Anything else will not be recognised as valid input.

## III. Load Balancer Performance

Load balancer performance was tested using JMeter and HAProxy stats. Two test plans were used. The first used HTTP request threads. This involved sending the server HTTP requests in three thread groups of 5000 users each. Each group had a timer between it. The first had a constant timer of ten seconds, the second had a Gaussian random timer and the third had a Uniform random timer with a maximum delay of 100 milliseconds. The second test plan used TCP threads. This test was three groups of 10,000 threads, which used the HTTP port number (80). These were to be sent to the load balancer, forwarded to the application servers, then rejected for not being the correct format for an HTTP request. Once these tests were ran, results were graphed and statistics were collected from HAProxy stats. Notable information is as follows:

Maximum number of requests per second: 2594

Average throughput: 16,000.45 bytes/second

Almost all of the HTTP requests were successfully returned, with a sparse few being timed out or receiving 503 (Service Unavailable) errors. The TCP test had a large number of failures, with about 50% of the total threads sent being timed out. This indicates that the load balancer was overwhelmed after the HTTP test and thus had to reject the later TCP requests, even if they were simpler to process.

Request response time for the HTTP test was monitored using Jmeter’s graphing functionality (Fig1). The graph shows a large spike in response times, which gradually declines. This indicates that the load balancer was initially overwhelmed by the number of requests. As the test continued and more requests had been dealt with, it became less overwhelmed as was able to handle newer requests at a quicker rate.

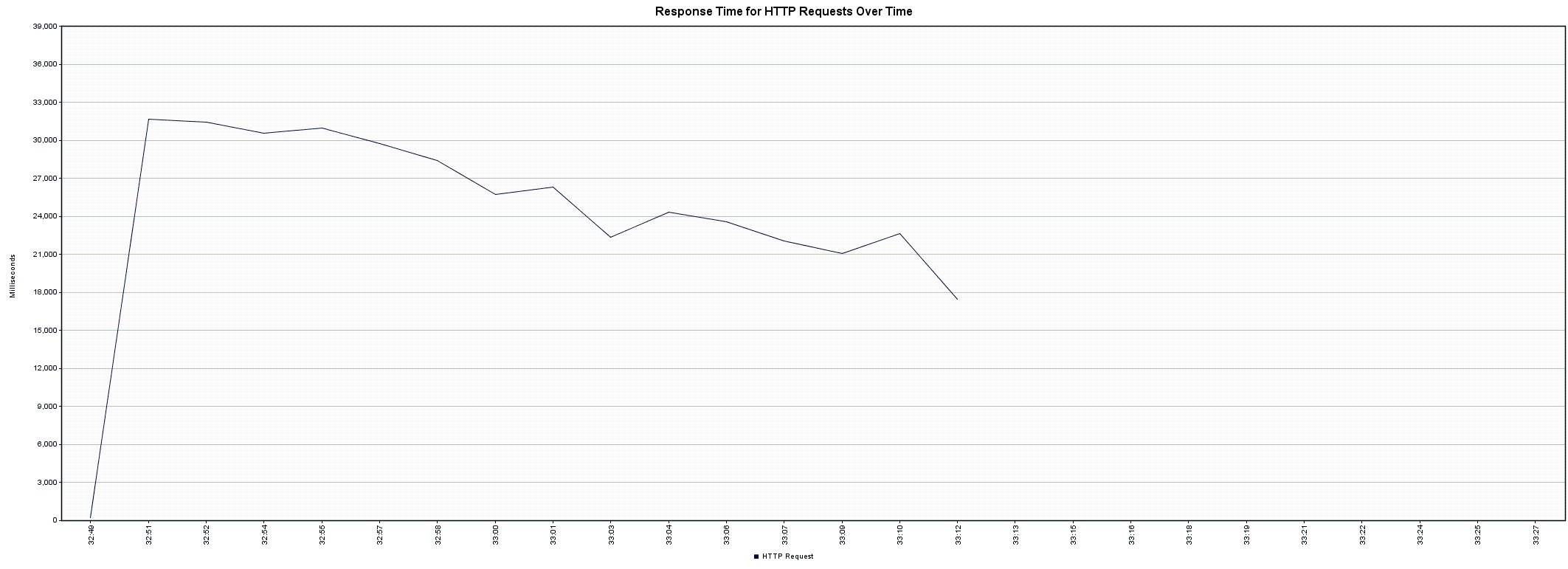


Fig1. Response time for HTTP request response time over the testing period. At the beginning of the test, response times are longer. These gradually decrease until the end of the test.