MUTAWEB-Mutation Testing Tool for Servlet based Web Applications

S.Suguna Mallika[1], Samuel Vineeth[2], Rohith Rangaraju[3] and Shabana Begum[4]
[1]Assoc. Professor, CVR College of Engineering/CSE Department, Hyderabad, India
suguna.kishore@gmail.com
[2,3,4] B.Tech. Students, CVR College of Engineering/CSE Department, Hyderabad, India
[2]samuelvineethcool@gmail.com, [3]rangaraju963@gmail.com, [4]shabbz.45@gmail.com

Abstract: **Mutation testing of web applications requires more sophistication and newer operators for greater efficiency to detect defects early in the testing cycle. A plethora of mutation testing tools are available for performing mutation testing. However for performing mutation testing of web applications there is only one tool available which is also available for web applications developed using Java Server Pages. The tools are also not addressing defects related to session management, cookie management while the logical, relational, operators and their corresponding mutant code are being tested. In the current work, an endeavour to implement some novel mutation operators pertaining to servlet based web applications has been made and a simple tool to implement some of the novel operators has been made successfully. This work will certainly assist the web application testers in quickly realising some key defects pertaining to session management and cookie management which might have been otherwise overlooked by the developers given the faster release cycles of the web applications.**

Index Terms: **mutation testing, web applications, testing tool, session management, cookie management.**

I. Introduction

Mutation testing is one of the testing techniques which is based on seeding mistakes into the code at known points and thereby observing the results of the test cases run. If the output is as expected with the original code, then it is indicative of a slip in the original code as the output is supposed to deviate from the expected result. If the output does not deviate from the expected result, then the mutated code which is called the mutant is said to be live otherwise dead. Dead mutants direct us towards a flawless code whereas a live mutant signifies a buggy code. To assure that the code is buggy, a retest of the code with an increased input sample data is performed, of which the results are further analysed. Suppose the results are found deviating from the expected results despite the increased sample data, then a back tracking is performed to analyse the buggy code and derive at the root cause for the defect [1].

Mutation testing can also be employed in testing the web applications, which are difficult to test in contrast to the standalone applications given the aspects of web application development like heterogeneity of development environment, cross platform deployment, browser incompatibility et.al. [2]. For performing mutation testing of web applications defining some operators for various possible defects that can be unearthed is done in earlier works where approximately 150 operators are proposed in various works for testing of applications [10]. Around 5 operators are implemented in the current work for testing of web applications.

There are many tools which perform mutation testing on various standalone applications. Applications like MuClipse, PIT, Jumble, etc., perform mutation testing on Java programs and applications like Cosmic ray and Mutpy does the same on python programs. But there are less number of tools which apply mutation process on web based applications .A tool which performs mutation testing on Java based applications (Servlets and JSPs) was proposed. This tool checks the status of mutation process by comparing log files which are generated by the tool. However, not all mutation operators generate log files. After applying mutation and running the updated application, the difference can be directly seen on the output (Webpages). For applications where applying mutation does not show any difference in the output, then log files before and after mutation are generated and both log files are compared to tell whether the mutant is live or not.

Section II provides a discussion on some of the existing mutation testing tools and their key features. Section III discusses the five novel mutation operators implemented as part of this MUTAWEB for testing web applications. Section IV provides conclusions and a peek into the future enhancements possible with this work.

II. Related Work

Various tools were earlier proposed and implemented for mutation testing of different applications [2, 3, 4, 5, 6, 7, and 10]. A total of 6 tools were taken into study for the development of MUTAWEB, the summary of which is presented here.

A. MuClipse tool

It is a mutation testing tool for java language. It can only mutate java based classes. It is a plug-in .To use this tool first it should be installed in eclipse. It provides a user interface in which different mutation operators are displayed in the form of check boxes. After choosing them, these operators will be applied on the existing java code (this is called mutating code). Junit is used here to run the tests on original code and mutated code. Muclipse compares both the results and if the results are same then the mutant is alive or else mutant is killed. Muclipse maintains the mutant score and displays them to the tester [5].

B. Judy tool

It is a mutation testing tool that supports only java language. It is a command line tool. This tool also covers all the branches in a given lines of code. From generation of byte code to execution of mutants, all can be done by this tool. Junit is used to run the tests for both original and mutated code. This was developed by Madeyski and Radyk [6].

C. WebMuJava

It is web mutation testing tool. It is an extension to Mujava. This tool was developed by Upsorn Praphamontripong, Jeff Offutt, Lin Deng, and JingJing Gu. This tool mutates Servlets and jsps. The mutated files are compiled and included in webapp and then the web app is tested. The tests for normal testing are completely different when compared to this mutation testing. Here the tests are created manually in the form of requests and these were stored using htmlunit, selenium, jwebunit. After that, these tests are applied automatically on the web application and the output is compared every time to generate the mutation score (Which is no of mutants killed to the total no of mutants inserted) [3].

D. Cosmic-ray

It's a mutation testing tool in python. It is an Open source command line tool which is not fully developed, and contributions through GitHub are in progress. This tool mostly mutates the code at AST (Abstract Syntax Tree) level. Developed by Austin Bingham, a founding director of Sixty North, a software consulting, training, and application development company Mutpy offers a new range of mutation operators for the testing of python programs in an efficient manner [9].

E. Mutpy

It is a mutation testing tool in python desktop applications and also web applications (DJANGO WEB DEVELOPMENT FRAMEWORK FOR PYTHON). This tool supports unit test module, and generates reports which can be human readable format. This Mutpy tool does not provide any user interface. It is a command line based tool. At present this tool supports 27 mutation operators [7].

F. Jumble tool

It is a mutation testing tool which mutates the code a byte code level. This tool works faster as it will work under bytecode level. This tool supports JUnit to perform tests on java classes. This tool returns mutation score and no of mutants for which the tests failed for the user for analysis. This tool does not return the mutants for which the tool passed. Jumble was developed in 2003-2006 by a commercial company Reel Two [4].

G. PIT tool

It is a mutation testing tool developed by Coles. It is open source tool. This tool generates mutants quickly. There are 4 phases: mutant generation, test selection, mutant insertion, and mutant detection. Like jumble tool pit also performs mutation at bytecode level. Tool is used as a command line tool as well as an eclipse plugin. The latest version of PIT released is 1.1.4 [8].

A summary of the tools under study is presented in the Table 1 highlighting the language in which each tool is developed and the languages supported by each tool.

TABLE 1
SUMMARY OF TOOLS UNDER STUDY

S no.	Tool Name	Language Developed	Language Supported	references
1	MuClipse	Java	Java	[5]
2	Judy	Java	Java	[6]
3	WebMuJava	Java	Web applications	[3]
4	Cosmic-ray	Python	Python, Django	[9]
5	Mutpy	Python	Python	[7]
6	Jumble	Java	Java byte code	[4]
7	PIT	Java	Java	[8]

III. IMPLEMENTATION

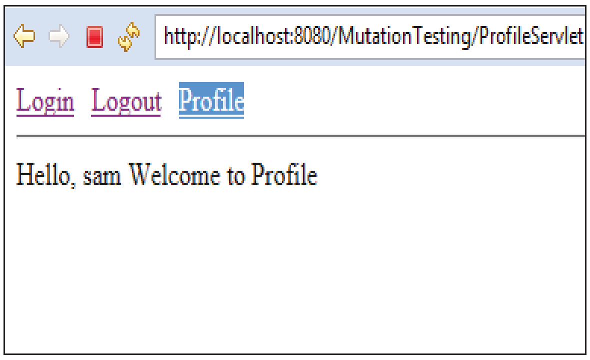
In the present work, 5 operators have been incorporated and tested with mutating the web application to demonstrate the error discovery.

Initially, the web application under test should be placed in the folder where the testing tool is present. Its web.xml is updated with the files of the application. The main page of the application is executed. Here, the file to be mutated is given as input and the type of mutation operator is to be selected. Some operators require generation of log file before mutation and a section for doing the above process is provided. The log file generation code is inserted into the file and the updated application has to be executed in order to write log code into a log file. Then another section which also takes input as a file name and type of operator is provided which now applies mutation and also modifies the logger code inserted previously. The application is executed again in order to generate another log file. A point to be noted is that the file name and the type of operator selected before and after applying mutation must be the same. After both the log files are generated, a servlet code which compares the contents of both log files is executed. The status of the mutation is displayed (Live or dead). After this, the contents of both log files are cleared. Before mutation is applied, a copy of that file is created and after executing the log checking servlet, the contents of mutated file are updated with its original contents.

Some operators do not require log file generation and the results of mutation can be seen in webpages. The application where the change is likely to be seen is to be noted down manually. Then after applying mutation operator, the updated application is re executed and the differences are written to a log file. Comparing both those web pages manually, we can say whether the mutant is alive or not. Again, the mutated file contents are updated with its original contents. This is the basic and overall working of MutaWeb tool.

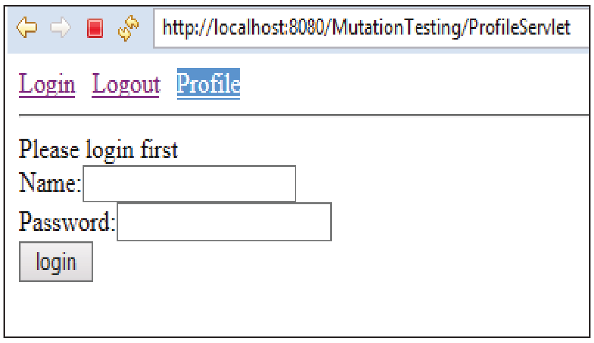
A. DSID: Session Invalidation Function Deletion

When a user logs out from an application, the session information of the user has to be destroyed. So if anyone tries to open the profile url even after logging out, the server will redirect the user to the login page. This operator will delete the method which performs the session invalidation process. This is a security constraint where applications opened in public area networks are prone to these problems if the session operations are not handled correctly. Figure 1 shows the the rendering of profile page of a sample application under test. Figure 2 refers to the rendering of the profile page requesting the user to re login as the session is expired. Figure 3 refers to the rendering of the same page which is unexpected after the introducing the mutated code.



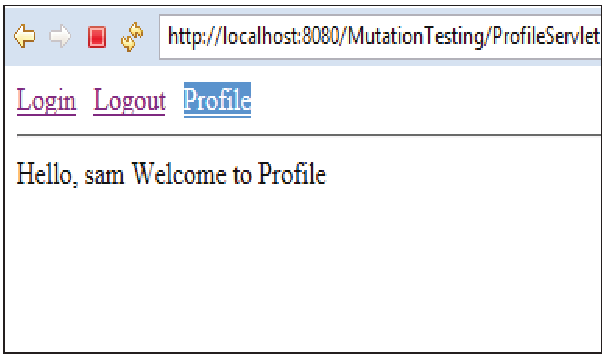
Hello, sam Welcome to Profile

Figure 1. Profile before logout and before applying mutation



Please login first
Name:
Password:

Figure 2. Profile after logout and mutation applied



Hello, sam Welcome to Profile

Figure 3. Profile after logout and after applying mutation

B. DACD (AddCookie Method Deletion):

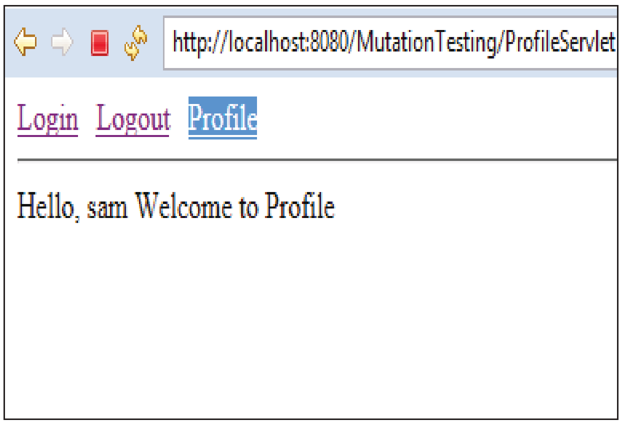
Cookies are used for storing information like username, password, session ID, etc. All these are stored in the form of key value pairs. Once a Cookie table is created, objects of any name can be created and that can modify, add and delete cookie table contents. So an operator which deletes a method which adds cookie information into cookie table is implemented.

Usually cookie information is not displayed on the webpages. The user of that web application might not the difference in the content displayed before and after applying this DACD mutation operator. In this case, log files are introduced. Two log files are created, one for storing information before applying mutation and the other one for storing information after mutation operation is applied. The contents of both the log files are compared and its status is displayed. If contents of both the log files are same then we say mutant is alive else mutant is dead.

C. DHBR (HTTP Boolean Replacement):

When a session is created, usually a session variable with the username or ID is set. After the user clicks logout, the session is destroyed and the session variable’s value is set to null. The parameter to this session creation or accessing method is a Boolean operator. This operator will invert the Boolean parameter in the method and run the mutation process. In both the cases, this method will try to access the current session. The difference comes after this step. If the parameter is true, then it will create a new session if a current session does not exist. If the parameter is false, then it will not create a new session even if a current session does not exist.

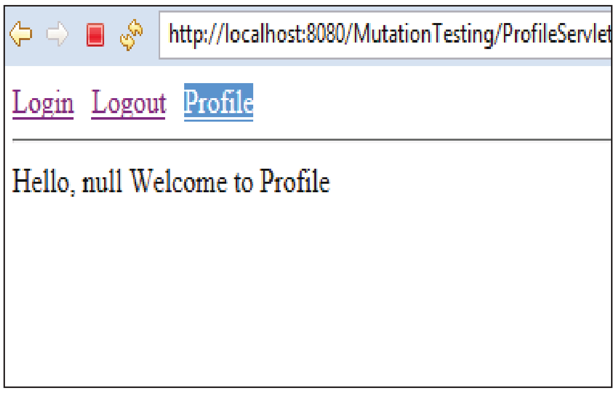
In any part of the application, the session maybe accessed by using the session creation or accessing method. So when we modify the Boolean variable from false to true and if that session variable is accessed, then its value is set to null. Here, the difference can be seen in the output (webpages) and there is no need of any log file concept.


http://localhost:8080/MutationTesting/ProfileServlet

Login Logout Profile

Hello, sam Welcome to Profile

Figure 4. Profile before Mutation


http://localhost:8080/MutationTesting/ProfileServlet

Login Logout Profile

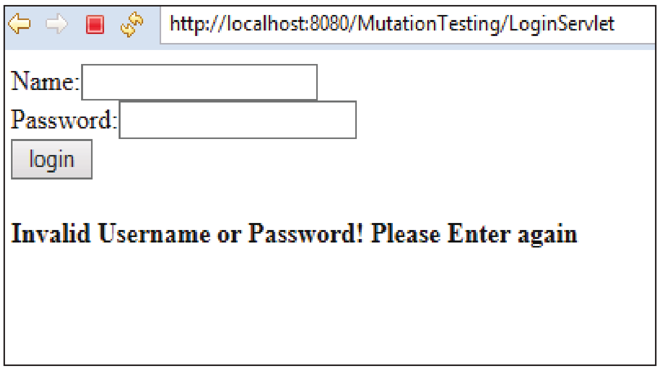
Hello, null Welcome to Profile

Figure 5. Profile after Mutation

D. DFIR (Forward Include Replacement):

In Web Applications, redirecting to other pages is a common feature. However, the control is not transferred to the redirected web page. Transferring the control feature is provided by the RequestDispatcher class in Servlets. When "include" attribute is used for redirecting to other webpage, and then the control is transferred to the called webpage and after its execution gets over, the control is returned back to the called webpage. So all the statements after the method call are executed. In the case of usage of "forward" tag, the control is not returned back to the called webpage but all the statements are after the method call are not executed. The statements related to response objects are not executed as control is transferred to the other webpage.

This operator will invert "include" to "forward" and test the application whether it is working well with the redirection of pages and transfer of control operations. Figure 6 and Figure 7 show the rendering of the login page with and without the mutated code.


http://localhost:8080/MutationTesting/LoginServlet

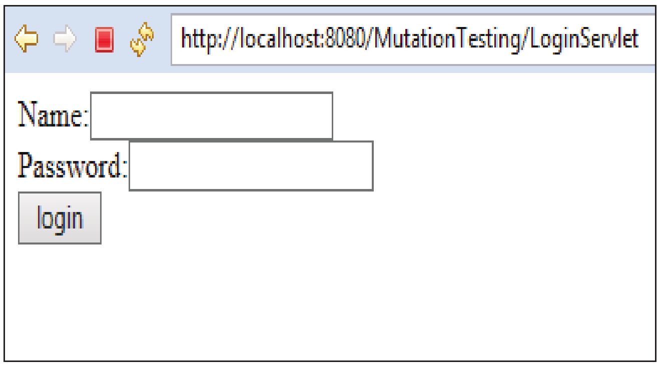
Name:

Password:

login

Invalid Username or Password! Please Enter again

Figure 6.Login before Mutation


http://localhost:8080/MutationTesting/LoginServlet

Name:

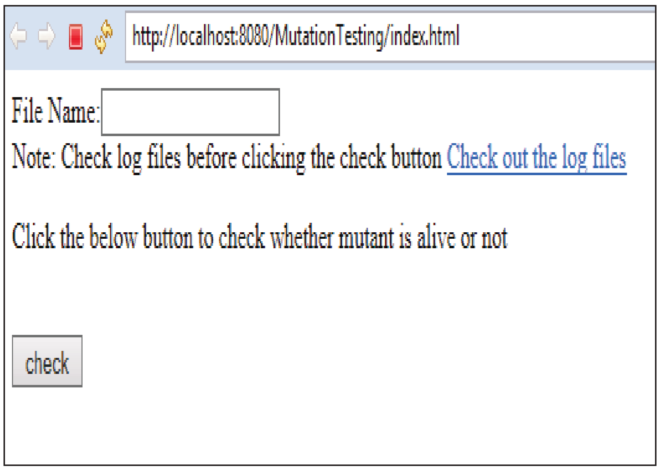
Password:

login

Figure 7.Login after Mutation

E. DRDUR (RequestDispatcher RLReplacement):

Like action string in html form tag and responseObject.sendRedirect method are used for redirecting to other pages, RequestDispatcher also does the same thing but provides an additional feature to transfer control to other webpage. However, the developer should also check whether the program sent a request to the correct expected webpage. So this operator will apply mutation in such a way that it will replace an existing URL in the RequestDispatcher with another URL. Figures 8 and 9 demonstrate the testing of requeste dispatches method.


http://localhost:8080/MutationTesting/index.html

File Name:

Note: Check log files before clicking the check button Check out the log files

Click the below button to check whether mutant is alive or not

check

Figure 8. Servlet Redirection before Mutation

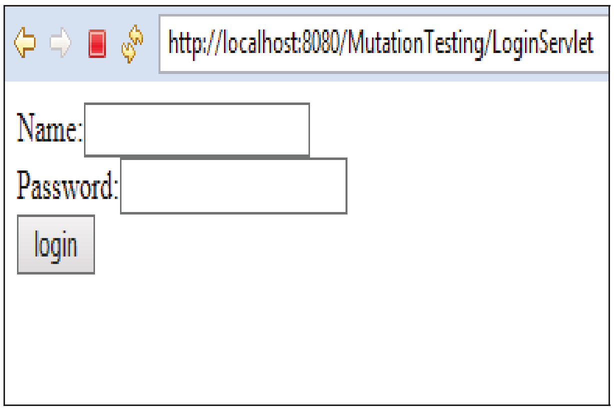

http://localhost:8080/MutationTesting/LoginServlet

Figure 9. Servlet Redirection after Mutation

IV. CONCLUSIONS

The above mutation operators are focussed on the session and cookie management of servlets. The tool developed could serve as a first hand aid to showcase some of the overlooked errors in the code as soon as the testing commences. However there is scope for inclusion of some more operators pertaining to servlet based web applications and the efficiency of operators to be measured against conventional testing strategies.

There is a scope for improvement of the tool to make it working dynamically online rather than like a standalone tool which is currently the need of the industry.

REFERENCES

[1]. M R Woodward, Mutation testing its origin and evolution, Information and Software Technology, Volume 35, No 3, March 1993.
[2]. Yuan-FangLi, Paramjit K.Das, DavidL.Dowe. "Two decades of Web application testing—A survey of recent advances". Information Systems 43 (2014) 20–54.0306-4379 & 2014Elsevier.
[3]. Upsorn Praphamontripong, Jeff Offutt, "Applying Mutation Testing to Web Applications", ICSTW '10 IEEE Proceedings of the 2010 Third International Conference on Software Testing, Verification, and Validation, April 2010 pages 132-141.
[4]. Sourceforge, "Jumble" http://jumble.sourceforge.net/, 2007.
[5]. B.H. Smith and L. Williams, "An Empirical Evaluation of the Mujava Mutation Operators," in Proceedings of the 3rd workshop on Mutation Analysis(MUTATION '07), published with the proceedings of the 2nd testing. Academic and Industrial Conference Practice and Research Techniques (TAIC PART '07). Windsor, UK: IEEE Computer Society, 10-14 September 2007, pp. 193-202.
[6]. L. Madeyski, N. Radyk, "Judy - a mutation testing tool for java " IET Software, Volume 4, Issue 1, Feb. 2010.
[7]. Mutpy, https://bitbucket.org/khalas/mutpy
[8]. PIT, "http://pitest.org/"
[9]. Cosmic Ray, "https://github.com/sixty-north/cosmic-ray".
[10]. Upsorn Praphamontripong, Jeff Offutt, Lin Deng, "An Experimental Evaluation of Web Mutation Operators", IEEE Ninth International Conference on Software Testing, Verification and Validation Workshops (ICSTW), April 2016.