Fault Tolerant System for Dynamic Web Applications

[1]Mr. Shinde Santaji Krishna , [2]Dr. D. K. Kaushik , [3]Dr.Shashank Dattatraya Joshi
[1]Research Scholar, [2]Research Supervisor, [3]Research Supervisor
[1,2]Department of Computer Engineering, Shri Jagdishprasad Jhabarmal Tibrewala University, Vidyanagari, Jhunjhunu, Rajasthan, India
[3]Research Supervisor, Shri J.J.T.University Rajasthan and Professor, Department of Computer Engineering, Bharati Vidyapeeth, Deemed University College of Engineering,Pune, Maharashtra, India

---

***Abstract* - Web script crashes and deformed progressively created website pages are normal faults, and they genuinely affect the ease of use of Web applications. Traditional systems for page acceptance cannot deal with the dynamically generated pages that are ever show on today's Internet. Proposed system implements a technique a for dynamic web applications implemented in ASP.net. The system produces tests consequently, run the tests catching consistent demands on inputs. Our proposed system executes the method for the Web applications for C#.net, Vb.net. Module produces test inputs for a Web application, screens the application for accidents, and approves that the yield fits in with the HTML particular.**

***Keywords* - ASP.Net, Dynamic analysis, HTML, Faults, Web applications**

I. INTRODUCTION

Dynamic test generation tools, such as Cute[1], DART [2], and EXE [3], create tests by executing an application on real input values, and then producing extra input values. Such techniques have not been practical in the domain of web applications. Due to the dynamism of the programming languages, these pose distinct challenges, their use of persistent state, the use of implicit input parameters and their complex patterns of user interaction [4]. The proposed work extends dynamic test generation to the domain of web applications, which dynamically create web (HTML) pages during execution. These pages are presented to the user in a browser. Proposed work applies these techniques in the context of the web development languages and languages for server-side web programming.

Execution failures may happen, for example, when a web application reads a file that is not exist or calls a function that is not defined. In such situations, depending on the strictness of the failure, the output of HTML that includes an fault message and execution of the application may be stopped. When output is not in well form HTML that indicates, HTML failure occurs. HTML failures are not as important because Web browsers are designed to tolerate some degree of malformedness in HTML. The execution failures are important than HTML failures. 1. Browsers' attempts to compensate for malformed web pages may lead to crashes and security weaknesses. 2. Standard HTML renders faster. 3. Malformed HTML is less portable across browsers and is weak to breaking or looking strange when shown by browser versions on which it is not tested. 4. A browser might succeed in showing only part of a malformed web page while silently discarding valuable information. 5. Search engines may have suffering indexing malformed pages [5].

Web script crashes and twisted powerfully created website pages are basic lapses, and they truly affect the convenience of Web applications. Current devices for site page approval can't deal with the alterably created pages that are universal on today's Internet. We display an element test era method for the area of element Web applications. The system uses both consolidated solid and typical execution and unequivocal state model checking.

Proposed system stretches out dynamic test era to the area of web applications that rapidly make web (HTML) pages amid execution, which are ordinarily displayed to the client in a program. Instrument applies these systems in the setting of the scripting dialect ASP.Net for server-side Web programming.

Notwithstanding dynamic substance, current Web applications might likewise produce a critical application rationale, commonly as JavaScript code that is executed on the customer side. Our systems are principally centered on server-side ASP.Net. It despite the fact that we do some negligible investigation of customer side code to decide how it summons extra server code through client interface components, for example, structures. Our objective is to discover two sorts of disappointments in web applications: execution disappointments that are shown as accidents or warnings amid system execution, and HTML disappointments that happen when the application creates distorted HTML.

There are two general methodologies to discovering blames in web applications: static examination and element investigation (testing). In the setting of Web applications, static methodologies have restricted potential in light of the fact that
1) Web applications are regularly composed in dynamic scripting dialects that empower on-the-fly production of code, and 2) control in a Web application normally streams by means of the produced HTML content (e.g., catches and menus that oblige client cooperation to execute), instead of singularly through the dissected code. Both of these issues posture enormous difficulties to methodologies focused on static examination. Testing of element Web applications is additionally complicated in light of the fact that the information space is extensive, and applications ordinarily oblige numerous client collaborations.

II. LITERATURE REVIEW

We exhibit another device, named DART, for consequently testing programming that consolidates three fundamental systems. Together, these three methods constitute Directed Automated Random Testing or DART for short. Amid testing, DART discovers standard mistakes, for example, program crashes, attestation infringement, and non-end. Preparatory investigations to unit test a few illustrations of C projects are extremely empowering [2].

EXE, a compelling bug-discovering apparatus that naturally produces inputs that crash genuine code. Since EXE's imperatives have no estimates, sustaining this cement information to an un instrumented adaptation of the checked code will make it take after the same way and hit the same bug (accepting discourage ministic code).EXE functions excellently on genuine code,, determining bugs alongside inputs that trigger them in: the BSD and Linux bundle channel executions, the udhcpd DHCP server, the pcre general outflow library, and three Linux document frameworks [3].

In unit testing, a project is decayed into units that are accumulations of capacities. The current work creates a technique to speak to and track stipulations that catch the conduct of a typical execution of an unit with memory charts as inputs. Additionally, a proficient obligation solver is proposed to encourage incremental era of such test inputs. At long last, CUTE, a device actualizing the system is depicted together with the consequences of applying CUTE to true cases of C code [1].

Web script crashes and contorted alertly created site pages are regular slips, and they truly affect the ease of use of Web applications. Current instruments for site page approval can't deal with the powerfully produced pages that are pervasive on today's Internet. Their module Apollo executes the method for the PHP programming dialect. Apollo produces test inputs for a Web application, screens the application for accidents, and approves that the yield complies with the HTML detail[4].

Server-side writing computer programs are one of the key innovations that backing today's WWW surroundings.. The estimate acquired by the analyzer can be utilized to check different properties of a server-side system and the pages it creates. To exhibit the viability of the investigation, they have actualized a string analyzer for the server-side scripting dialect PHP. The analyzer is effectively connected to freely accessible PHP projects to discover cross-site scripting vulnerabilities and to approve pages they create rapidly[8].

SPIN is an effective check framework for models of dispersed programming frameworks. It has been utilized to catch plan lapses in applications running from abnormal state portrayals of appropriated calculations to itemized code for controlling phone trades[6].

A Veriweb is a tool for naturally discovering execution ways of dynamic Web applications. Veriweb was intended to consolidate the adaptable route capacities of catch replay tools with the high state of automation gave by Web crawlers. They have talked about architecture, finding algorithms for discovering Web applications, and Smart Profiles and related strategies for consequently filling forms[7].

Fuzz testing is a successful procedure for discovering security vulnerabilities in programming. They have executed this calculation in SAGE (Scalable, Automated, and Guided Execution); another apparatus are utilizing x86 direction level following and copying for white box fluffing of discretionary document perusing Windows applications.. Strikingly, without any organization particular learning, SAGE identifies the Ms07-017 ANI weakness, which was missed by broad discovery fluffing and static investigation devices. Moreover, while still in an early phase of advancement, SAGE has effectively found 30+ new bugs in huge transported Windows applications including picture processors, media players, and document decoders. A few of these bugs are conceivably exploitable memory access infringement [9].

They concentrate in space on those variables and values that are important for the disappointment, and in time of those minutes where reason moves happen minutes where new applicable variables start being disappointment causes: "At first, variable a was 3; in this manner, at shell sort(), variable a[2] was 0, and subsequently, the system fizzled." In their assessment, reason moves a spot the errors instigating abscond twice and additionally the best systems known[10].

They present an approach that holds a guarantee for the future of active invariant inference: utilizing typical execution, all the while with concrete test execution so as to acquire conditions for invariants. It joins the favorable circumstances of invariant induction through static examination, with the quick reasonableness of observing invariants by executing tests composed of programmers who exercise genuine conditions [11].

III. MOTIVATION

Novelty in this work is the induction of information parameters, which are not showed in the source code, yet which are intuitively supplied by the client (e.g., by clicking catches in created HTML pages). The fancied conduct of Asp.net applications is normally attained by an arrangement of cooperations between the client and the server. We handle this issue by improving the consolidated solid and typical execution strategy with unequivocal state model checking focused around programmed element recreation of client cooperations. To mimic client association, proposed hardware shops the condition of the nature's domain (database, sessions, and treats) after every execution, examines the yield of the execution to discover the conceivable client choices that are accessible, and restores the earth state before executing another script focused on a discovered client alternative

Web developers widely recognize the status of creating legal HTML. Many websites are checked using HTML validator. However, HTML validator can only point out problems in HTML pages, and are by them incapable of finding the faults in applications that generate HTML pages. Checking the dynamic Web applications requires checking that the application creates a valid HTML page on all possible execution path. In practice, even professionally developed and carefully tested applications often contains several faults

Proposed system introduces a computerized procedure for discovering disappointments in HTML-producing web applications. Our procedure is focused on dynamic test era, utilizing joined solid and typical (concolic) execution, and obligation illuminating, apparatus that executes our method in the connection of the freely accessible Asp.Net web applications.

Proposed system first executes the Web application under test with void information. Amid every execution, this device screens the project to record way requirements that reflect how include qualities influence control stream. Furthermore, for every execution, Proposed system figures out if execution disappointments or HTML disappointments happen (for HTML disappointments, a HTML validator is utilized a prophet). Proposed system consequently and iteratively makes new inputs using the recorded way imperatives to make inputs that practice distinctive control stream. Most past methodologies for concolic execution just recognize "standard lapses, for example, accidents, and attestation disappointments. Our methodology recognizes such standard lapses too, additionally utilizes a prophet to locate determination infringement in the application's yield.

Problem Definition

To develop a proposed system to detect and resolve execution and HTML errors in Web applications.

The proposed system finds the faults in Web applications using a technique that is based on error detection and correction. The proposed system discovers the two types of errors in web applications that include execution errors and HTML errors. Execution errors are showed as accidents or warnings during system execution, and HTML errors that happen when the application produces distorted HTML

Objective

1. To detect HTML and execution faults in web applications
2. To resolve HTML and execution faults in web applications

IV. PROPOSED SYSTEM

Proposed system is implemented technique for Web Technology. This module consists of following major components, input provider, ASP.Net parser, fault detector, fault resolver and report generation illustrated in Fig. 1

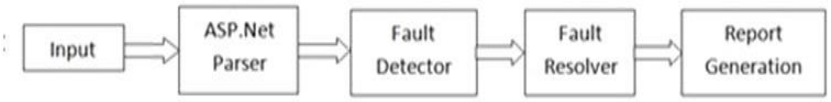

Fig. 1. Proposed Framework for Fault Detection and Correction for ASP.net Web Applications.

Input:

Proposed system use input as a web applications that are implemented in ASP.Net technology. This section is used to provide the input for a proposed system.

ASP.net Parser:

ASP.net parser use asp.net web sites as an input and they parse all the code from design file (xxx.aspx) and logic file (x.aspx.cs /x.aspx.vb) and create a source file for error detection.

Fault Detector:

The basic objective of proposed work is to find two types of faults i.e. HTML faults and execution faults. ASP.Net Fault Detection uses the input file from parsers and finds out the HTML faults and execution faults from input files. The output from the PHP/ASP.net Fault Detection is a list of HTML and execution faults and also the path of the error containing pages.

Fault Resolver:

Error detector detects or finds the faults in respective web applications. Error resolver checks the fault list and provides the appropriate solution and does the corrections.

Report Generation:

This section is used to create fault detection and correction report of respective web application

.

Snapshots:

Detect the faults:

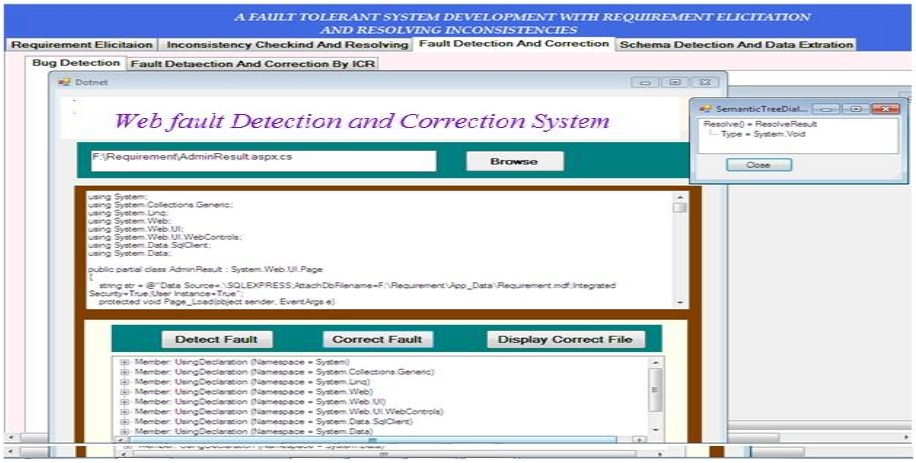

Fig. No. 2 Shows detection faults

Display Corrected file:

Fig. No. 3 Shows the corrected file

V. RESULT ANALYSIS

We have taken four datasets that include Schoolmate, Faqforge, Webchess1, and Phpsytem. Figure 2 shows a time taken for the detection of faults by the traditional system and time required for detection and correction of faults for proposed system. Proposed system takes less time for detection of execution and HTML faults present in the web applications as compared to a traditional system. Figure 4 shows number of faults detected by the proposed system and traditional system. Figure 5 shows the accuracy of fault detection of proposed system and traditional system. Accuracy of fault detection obtained by the proposed system is better than a traditional system.

Table No.1 Time required for finding faults in traditional and proposed system

Dataset	Proposed System (s)	Traditional System (s)
Schoolmate	91	915
Faqforge	86	621
Webchess1	146	1478
Phpsytem	35	1068

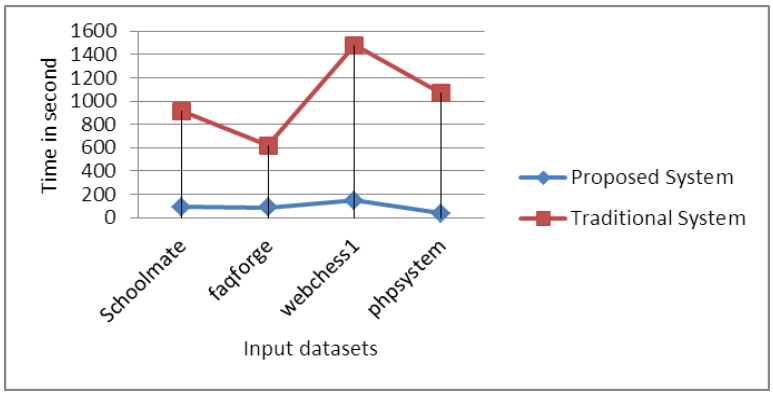

Fig. No. 2 Graph of time for fault detection

Table No.2 Fault detected by traditional and proposed system

Dataset	No of faults Detected	
	Proposed System	Traditional System
Schoolmate	143	123
faqforge	47	93
webchess1	55	31
phpsystem	17	9

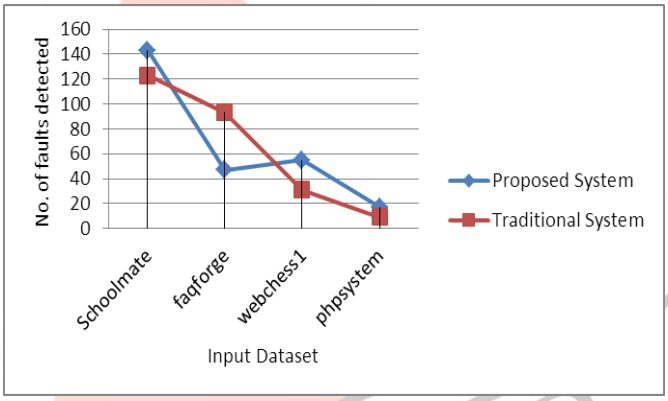

Fig. No. 4 Graph of number of faults detected

Table No. 3 Accuracy proposed system

Dataset	Actual Error in file	Errors detected by Proposed system
Schoolmate	235	232
faqforge	53	50
webchess1	6	6
Phpsystem	128	128

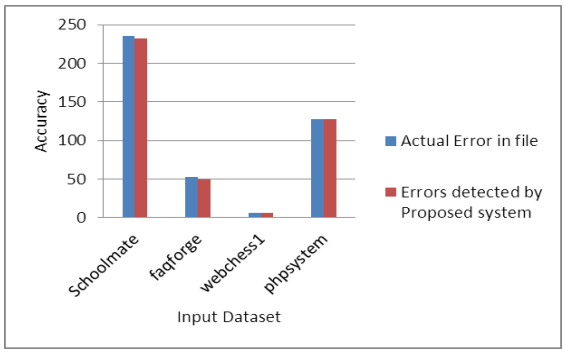

Fig. No. 5 Graph of accuracy of fault detection of proposed system

VI. CONCLUSION

We have presented a technique for finding faults in ASP.Net. The work is innovative in several aspects. The aim of proposed system is to give a new approach to detecting and correct faults (i.e. HTML and execution). This system has utilized the dynamic inputs to check the fault events and perform automated examination to minimize the extent of the bug prompting inputs. Traditional system only detects faults in ASP.Net but proposed the system detects static, dynamic bugs as well as HTML faults happened in web applications implemented in ASP.net. We have tested our system on four dataset that includes Schoolmate, Faqforge, Webchess1, and Phpsytem. Proposed system detects and corrects HTML errors and execution errors in less time as compared to the traditional system. The accuracy of fault detection of proposed system is better than that of the traditional system. Hence, the accuracy of the web applications has improved by proposed system.

VII. REFERENCES

[1] K. Sen, D. Marinov, and G. Agha, "CUTE: A Concolic Unit Testing Engine for C," *Proc. ACM SIGSOFT Int'l Symp. Foundations of Software Eng*., pp. 263-272, 2005.

[2] P. Godefroid, N. Klarlund, and K. Sen, "DART: Directed Automated Random Testing," *Proc. ACM SIGPLAN Conf. Programming Language Design and Implementation*, pp. 213-223, 2005.

[3] C. Cadar, V. Ganesh, P.M. Pawlowski, D.L. Dill, and D.R. Engler, "EXE: Automatically Generating Inputs of Death," *Proc. Conf. Computer and Comm. Security*, pp. 322-335, 2006.

[4] Shay Artzi, Adam Kie _zun, Julian Dolby, Frank Tip, Danny Dig, Amit Paradkar, "Finding Bugs in Web Applications Using Dynamic Test Generation and Explicit-State Model Checking" *IEEE trans. on Software Engg. vol. 36, NO. 4*, July/Aug 2010

[5] F. Zoufaly,, "Web Standards and Search Engine Optimization (SEO)—Does Google Care About the Quality of Your Markup?" 2008.

[6] G.J. Holzmann, "The Model Checker SPIN," *Software Eng., vol. 23, no. 5, pp. 279-295*, 1997.

[7] M. Benedikt, J. Freire, and P. Godefroid, "VeriWeb: Automatically Testing Dynamic Web Sites," *Proc. Int'l Conf. World Wide Web*, 2002.

[8] Y. Minamide, "Static Approximation of Dynamically Generated Web Pages," Proc. Int'l Conf. World Wide Web 2005..

[9] P. Godefroid, M.Y. Levin, and D. Molnar, "Automated Whitebox Fuzz Testing," *Proc. Network Distributed Security Symp., pp. 151-166*, 2008

[10] H. Cleve and A. Zeller, "Locating Causes of Program Failures," *Proc. Int'l Conf. Software Eng., pp. 342-351*, 2005.

[11] C. Csallner, N. Tillmann, and Y. Smaragdakis, "DySy: Dynamic Symbolic Execution for Invariant Inference," *Proc. Int'l Conf. Software Eng*., pp. 281-290, 2008.