

Web Application Bypass Testing

Jeff Offutt, Ye Wu, Xiaochen Du and Hong Huang
Information and Software Engineering
George Mason University
Fairfax, VA 22030, USA
(+1) 703-9934-1651 / 1654
{ofut,wuye,xdu,hhuang2}@ise.gmu.edu

Abstract—Input validation refers to checking user inputs to a program to ensure that they conform to expectations of the program. Input validation is used to check the format of numbers and strings, check the length of strings, and to ensure that strings do not contain invalid characters. Input validation testing (IVT) is particularly important for software that has a heavy reliance on user inputs, including Web applications. A common technique in Web applications is to perform input validation on the client by using HTML attributes and scripting languages such as JavaScript. An insidious problem with performing input validation on the client is that end users have the ability to bypass this validation. *Bypass testing* is a unique and novel way to create test cases that is available only because of the unusual mix of client-server, HTML GUI, and JavaScript technologies that are used in Web applications. This workshop paper presents the issues and concerns that allow bypass testing, the preliminary concepts behind the technique, and some early results on applying it. How effective and useful bypass testing can be in testing Web applications will be determined through ongoing research and automation.

I. INTRODUCTION

The World Wide Web gives software developers a new way to deploy sophisticated, interactive programs with complex GUIs and large numbers of back-end software components that are integrated in novel and interesting ways. Analyzing, evaluating, maintaining and testing these applications present many new challenges for software developers and researchers. Most Web applications are run by users through a Web browser and use HTML to create graphical user interfaces. Users enter data and make choices by entering data through HTML forms and pressing a submit button. Browsers send the data and choices to the software on the server using HTTP *requests*. An important point to note is that HTTP is a “stateless” protocol, that is, each request is independent of previous requests and, by default, the server software does not know whether multiple requests come from the same or different users.

The type of HTTP *request* determines how the user’s data is packaged when sent to the server. Although

HTTP defines a number of request types, this paper only considers GET and POST requests. GET requests package the data as parameters on the URL that are visible in the URL window of most browsers (for example, `http://www.compsac.com?name=george`). POST requests package the data in the actual data packets that are sent to the user.

A common activity of Web applications is to **validate** the users’ data. This is necessary to ensure that the software receives data that will not cause the software to do bad things such as crash, corrupt the program state, corrupt the data store on the server, or allow access to unauthorized users. This type of input validation is crucial for Web applications, which are heavily user interactive, often serve a very large user base, have very high quality requirements, and are always publicly accessible [9]. Because of the fundamental client-server nature of Web applications, input validation can be and is done on both the client and the server.

For traditional client-server software, the client only has binary versions of the software and the program has full control of the execution. Only the client programs can submit the input data and users do not know how or where the data is sent. Once input data are validated by the client software, they cannot easily be changed by the users. On the contrary, users of Web applications can easily bypass client programs and send any data to the servers. This presents a challenge to assure the security and reliability of web applications. At the same time, this provides a convenient mechanism to automate the testing of Web applications.

Bypass testing is a testing technique for Web applications that skips client-side validation mechanism and automatically generates input data to verify the reliability, security, performance and other quality aspects of Web applications. Bypass testing has several advantages; it can be easily automated, it does not require access to the source code of the Web application, and it can reveal faults that can be easily overlooked by many other

techniques.

II. CLIENT-SIDE VALIDATION VS. SERVER-SIDE VALIDATION

HTML pages (whether static `html` files or dynamically created) can include scripting programs such as JavaScript that can respond to user events and check various syntactic attributes of the inputs before sending the data to the server. User events that JavaScript can respond to are defined by the HTML document object model (DOM) and include mouseover events, form field focus events, form field changes, and button presses, among others. Client-side checking is used to check that required fields are filled in, inputs conform to certain restrictions on characteristics such as length, characters used, and satisfaction of syntactic patterns such as email addresses.

Client-side input validation can be done by using the HTML input elements to restrict the size or contents of inputs (syntactic restrictions only), and by writing programs in languages such as JavaScript to evaluate the values before submission (syntactic and semantic restrictions). Specific syntactic restrictions include restricting the length of text boxes and using select, check or radio boxes to restrict the inputs. Specific semantic restrictions include the input must match a specific data type, the input must match a format for specific kinds of data such as phone numbers or email addresses, avoiding the submission of potentially damaging database commands through SQL injection, and enforcing relationships between different input elements. Submission restrictions include setting the type of HTTP request (GET or POST) and limiting the URLs where the submissions go. Ongoing research is refining this list of validation types with the intent to have a complete set of rules to apply.

Client-side checking can be done as soon as a user event is triggered or after the user clicks on a submit button but before the data is submitted. Doing input validation on the client avoids the need for a trip to the server and allows the checking to be defined with the actual form.

Server side checking is done by programs on the server such as CGI/Perl, Java servlets, Java Server Pages, and Active Server Pages. Server side checking can perform all of the checks that client-side checking can, but not until after the user presses the submit button. Server side checking cannot respond to user events, but does have access to the state of the file system and database on the server. When a high level language such as Java is used on the server, server side checking also

provides more robust and flexible ways to check inputs and respond to invalid user inputs than does a scripting untyped language such as JavaScript.

III. BYPASSING CLIENT-SIDE INPUT VALIDATION

Client-side programs expect users to type their values and make their choices by using the keyboard and mouse. However, users can easily bypass the HTML to send values directly to the server software. With GET requests, the users can simply type the parameters into the URL box in their browsers. With POST requests, a simple program can be written on the client that creates and submits the request. There are two valid reasons to bypass HTML forms. One is for convenience; if a Web application is used a lot it might be more convenient to enter the values through another mechanism than through the relatively slow HTML FORM interface. A second reason is for automation; when running multiple tests on a Web application, the test execution can be automated by bypassing the forms. As Marick pointed out, automation of GUI testing is helpful but sometimes too expensive to be useful [6]. Bypassing avoids much of that expense.

The ability to bypass form entry allows another strategy to be used. If the Web application uses client-side input validation, then the users can bypass the validation to violate security. This technique is sometimes used by hackers to try to break into computers or access data by techniques such as injecting SQL statements into databases. Our suggestion in this research is to utilize the ability to bypass client-side checking, thereby supplying invalid inputs to the software to test for robustness and security.

An additional ability that is available when bypassing HTML forms is to override hidden form fields. HTML allows data to be placed into a page with the tag `<INPUT Type="Hidden" ...>`. These fields are not shown to the users in the browsers, but data in the fields are submitted to the server. Bypassing forms allow the additional ability to change or remove the contents of hidden form fields.

IV. BYPASS TESTING FOR WEB APPLICATIONS

To apply bypass testing, the HTML form elements are analyzed and each input element is modeled as a parameter. Values are chosen to violate each of the restrictions that the HTML and JavaScript impose. At present the values are chosen according to simple rules, but we expect to describe the restrictions via regular expressions, then use mutation-style modifications to the expressions to generate invalid inputs.

TABLE I
STRINGS FOR SECURITY BYPASS TESTING.

Potential Illegal Character	Symbol
Empty String	
Commas	,
Single and double quotes	' or ''
Tag symbols	< and >
Directory paths/
Strings starting with forward slash	/
Strings starting with a period	.
Ampersands	&
Control character	NIL, newline
Characters with high bit set	254 and 255
Script symbols	<javascript> or <vbscript>

For the security concerns, we have defined specific rules to violate security constraints. Certain kinds of strings are submitted to try to violate the security. An initial rule set has been adapted from Wheeler [11] and is given in Table I.

Several other rules are applied. For instance, numbers are created to violate the minimum and maximum of the allowed values; numbers with invalid format; special invalid texts. For instance, correctly email addresses require a correctly formed user name and domain name. Invalid email addresses such as those without user names, or with invalid domain names are generated. Also, invalid input parameter names, invalid links and program names are generated.

V. PRELIMINARY EMPIRICAL RESULTS

The preliminary ideas on bypass testing have been applied to a small Web application (STIS) that helps users keep track of arbitrary textual information. STIS uses a database and contains 5 Java bean classes and 17 Java Server Pages, 8 of which process HTML forms. STIS had already been extensively tested by the developers. We generated 158 tests using the bypass technique for STIS. When Web applications receive invalid inputs, there are three types of server responses. (1) The invalid inputs **are recognized** by the server and adequately processed by the server. (2) the invalid inputs **are not recognized** and cause abnormal server behavior, but the abnormal behavior is recognized and automatically processed by server error handling mechanism. (3) The invalid inputs **are not recognized** and the abnormal server behavior is **exposed** directly to the users. The abnormal behavior includes run time exceptions, and the revealing of confidential information to unauthorized clients, among others. Type 1 represents proper server behavior, while type 2 and 3 represent inadequate server behavior, and are considered to be failures by our testing. Of the 158 test cases, all of which represented invalid

inputs, 92 test cases were properly processed by STIS, and 66 resulted in failures in the software. These results both demonstrate the inadequacy of other testing strategies to find these kinds of failures and the importance of bypass testing.

VI. RELATED WORK

This research is closely related to input validation testing [4], which involves statically analyzing the input command syntax as defined in interface and requirement specifications and generating input data from the specification.

HTML forms can also be considered a graphic user interface to access dynamic Web pages and can be augmented by results from GUI testing [8], [7].

A new theoretical model of new coupling and the dynamic flow of Web applications has been proposed in our previous research [12]. A Web application is modeled in intra-component and inter-component levels, and a series of new coverage criteria are proposed.

Other Web application modeling techniques include the Web Modeling Language (WebML) [2], [3], allows Web sites to be conceptually described from the user's view. Kung et al., have developed a model to represent Web sites as graphs, and provide preliminary definitions for developing tests based on the graphs in terms of Web page traversals [5]. Their model includes static link transitions and focuses on the client side with limited use of the server software. Ricca and Tonella proposed an analysis model and corresponding test strategies for Web page analysis [10]. Testing of dynamic aspects of Web application is limited to testing the access of dynamic Web pages under user's interaction. Andrews et al. use hierarchical FSM to model potentially large Web applications. Test sequences are generated based on FSM and using input constraints to reduce the state space explosion [1].

VII. CONCLUSIONS

Bypass testing is a unique and novel way to create test cases that is available only because of the unusual mix of client-server, HTML GUI, and JavaScript technologies that are used in Web applications. This workshop paper presents the issues and concerns that allow bypass testing, the preliminary concepts behind the technique, and some early results on applying it. How effective and useful bypass testing can be in testing Web applications will be determined with ongoing research and automation.

One argument against bypass testing may be that it is not fair to subject software to input that the software is explicitly **not** expected to receive. However, this is

a dangerously misleading assumption. When the inputs are validated or checked on the client, the fact is that real users can bypass those tests. This might be done by accident if JavaScript is turned off, by uses of the software that the original developers did not anticipate such as direct submission of inputs, or by devious intent on the part of hackers. For all these reasons, it is important for Web application developers to understand how their software will behave if the client-side checks are not done.

The fact that the possibility of bypass testing exists may, in fact, provide motivation to Web application developers to check data on the server. This may already be a trend in the industry. Five years ago, many books on Web software advocated checking inputs with JavaScript as a mechanism to reduce network traffic; modern books and instructors usually advocate doing input validation on the server. Nevertheless, major e-commerce and e-service sites still use client-side checking and hidden form fields. We found client-side checking on `amazon.com` and `netflix.com`, and the use of hidden form fields to store sensitive information on `fast-lane.nsf.com`.

A major advantage of bypass testing is that it does not require access to the source of the back-end software. This greatly simplifies the generation of tests and automated tools to support bypass testing. Our current plan is to build tools that parse HTML, discover and analyze the form field elements, parse the client-side checking encoded in the JavaScript, and automatically generate bypass tests to evaluate the server-side software. One open question is whether to limit the tests to the input validation that is already done on the client or expand the tests to check input validation that may be done on the server. An interesting complexity is that the use of dynamic Web pages means that the same URL can produce different forms at different times, depending on the parameters supplied, state on the server, characteristics of the client, and other environmental information. Our atomic section analysis [12] can help with this problem but it relies on access to the Web application source.

REFERENCES

- [1] Anneliese Andrews, Jeff Offutt, and Roger Alexander. Testing Web applications. *Software and Systems Modeling*, 2004. To appear.
- [2] Stefano Ceri, Piero Fraternali, and Aldo Bongio. Web modeling language (WebML): A modeling language for designing Web sites. In *Ninth World Wide Web Conference*, Amsterdam, Netherlands, May 2000.
- [3] Stefano Ceri, Piero Fraternali, Aldo Bongio, Marco Brambilla, Sara Comai, and Maristella Matera. *Designing Data-Intensive Web Applications*. Morgan Kaufmann, December 2002. Information available online at: <http://webml.elet.polimi.it/webml/book.html>.
- [4] J. H. Hayes and J. Offutt. Increased software reliability through input validation analysis and testing. In *Proceedings of the 10th International Symposium on Software Reliability Engineering*, pages 199–209, Boca Raton, FL, November 1999. IEEE Computer Society Press.
- [5] D. Kung, C. H. Liu, and P. Hsia. An object-oriented Web test model for testing Web applications. In *Proc. of IEEE 24th Annual International Computer Software and Applications Conference (COMPSAC2000)*, pages 537–542, Taipei, Taiwan, October 2000.
- [6] B. Marick. When should a test be automated? In *Proceedings of 11th International Software/Internet Quality Week*, May 1998.
- [7] A. M. Memon, M. L. Soffa, and M. E. Pollack. Hierarchical GUI test case generation using automated planning. *IEEE Transactions on Software Engineering*, 27(2):144–155, February 2001.
- [8] Atif M. Memon. GUI testing: Pitfalls and process. *IEEE Computer*, 35(8):90–91, August 2002.
- [9] Jeff Offutt. Quality attributes of Web software applications. *IEEE Software: Special Issue on Software Engineering of Internet Software*, 19(2):25–32, March/April 2002.
- [10] F. Ricca and P. Tonella. Analysis and testing of web applications. In *23rd International Conference on Software Engineering (ICSE '01)*, pages 25–34, Toronto, CA, May 2001.
- [11] David A. Wheeler. *Secure Programming for Linux and Unix HOWTO*. Published online, last accessed, Feb 2004, March 2003.
- [12] Ye Wu, Jeff Offutt, and Xiaochen Du. Modeling and testing of dynamic aspects of Web applications. *Submitted for publication*, 2004. Technical Report ISE-TR-04-01, www.ise.gmu.edu/techreps/.