Cross-site scripting (XSS) attack – case study on malicious scripts

Manish M, Dr Manjunath M
Department of MCA
RV College of Engineering
Bengaluru -560059.

Abstract

Cross-site scripting is one of the major problems occuring in Web Applications. With more connected devices which use different kind of web applications for pretty much every job, the danger of XSS attacks is increasing rapidly. In these web applications, attacker steals victim's session details and other important info by exploiting XSS vulnerabilities. Present XSS prevention methods are dynamic and static analysis and proxy based methods etc which aren't effective enough. Cross-Site Scripting (XSS) vulnerabilities are being exploited by the attackers to steal web browser's resources like cookies and credentials by injecting the malicious JS code on the victim's web application. Since browsers support the execution of commands embedded in web content to enable dynamic web pages, attackers make use of this feature to enforce the execution of malicious code in a user's browser. The analysis of detection and prevention of Cross-Site Scripting (XSS) helps to avoid this attack. We describe a method to detect and stop this and hence eliminate Cross-Site Scripting attack.

Keywords—*Cross-site scripting; web security; web applications; XSS attacks;*

I. Introduction

The World Wide Web (WWW), refers to as the web platform, has evolved into a large-scale system composed by countless various applications and its services. In the beginning, only static web pages were used which use to provide static information expressed in text and graphics. As the Internet grew, the web sites become more and more professional and dynamic. Now web applications are used to generate dynamic web pages and have become a dominant method for implementing and providing access to online services and have become truly pervasive in all kinds of business models and organizations. Today, most of the systems such as Social Networks, health care, blogs, banking, or even emergency response, are relying on these applications. Users use these web applications for communicating with others via messaging, reading email , managing their photos and files, for editing and viewing videos, creating spreadsheets and presentations.

Hence, Internet has become an essential part of our daily life. So there has to be addition mechanisms to ensure security for internet users. Therefore providing a safe networking environment is absolutely necessary. If vulnerability occurs in websites, a lot of users will be attacked. Cross-site scripting (XSS) attack is one of the most common vulnerabilities seen during the usage of web applications. Cross-site

scripting attack is usually executed by manipulating a vulnerable web site so that it returns a malicious JavaScript to its users. When the malicious code executes inside any one of the victim's browser, the attacker can fully compromise their interaction with the web application. To achieve those purposes, automatic tools and security system have been implemented, but none of them are complete or accurate enough to ensure an absolute level of security on web application. A mechanism which is easily deployable to to detect and prevent this Cross-site scripting (XSS) attack is absolutely essential.

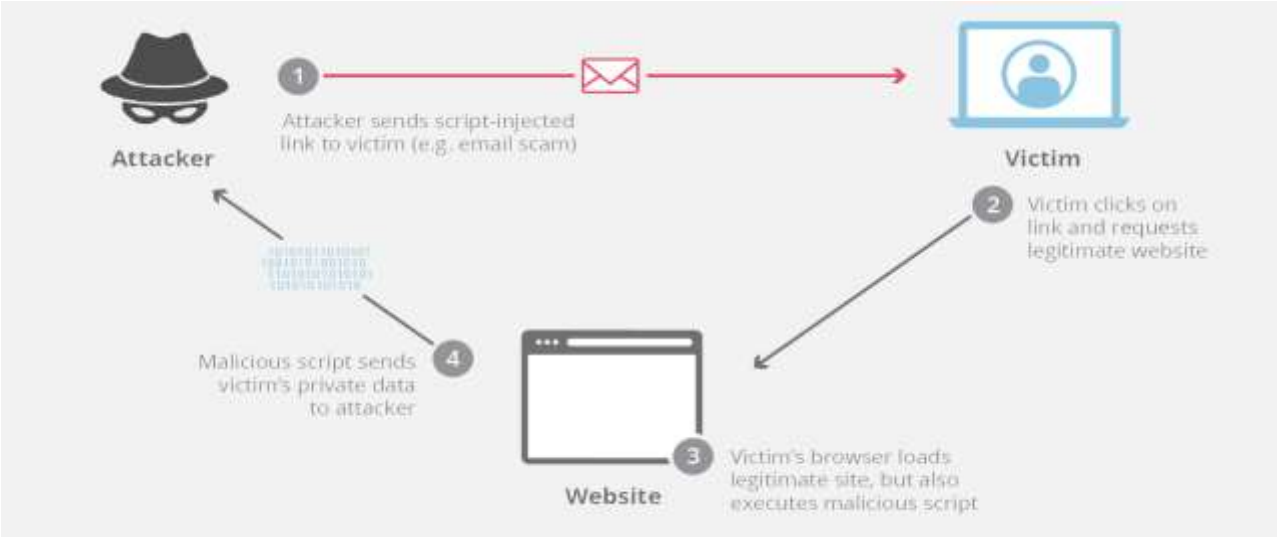

Fig 1. Cross-site Scripting Attack

II. Overview of Cross-site Scripting Attack

Generally, cross-site scripting refers to the hacking technique that uses vulnerabilities in the code of a web application to allow a hacker to send malicious content from an end-user and collect some type of data from them. XSS can be defined as a security exploit in which a hacker can embed malicious script (JavaScript, VBScript, ActiveX, HTML, or Flash) into a vulnerable dynamic page, by executing the script on his system in order to collect data. Use of XSS attack might compromise private information of the end user, manipulate or steal cookies, create requests that may be mistaken for those of a valid user, or even execute malicious code on the end user systems. The data is by default formatted as a hyperlink containing malicious content which is distributed over any possible means on the internet.

III. Threats faced due to Cross-site Scripting Attack

Cross-site scripting poses severe application risks that include the following:

1.Session hijacking : such as adding JavaScript code to a webpage that forwards cookies to an attacker.

2.Misinformation : such as adding "For more info call 1-800-BADGUY"to any web page".

3.Defacing web site : such as adding "This Company is a terrible company" to any page.

4.Inserting hostile content to a page : such as adding malicious ActiveX controls.

5.Phishing-attacks : such as adding login form posts and other data entry forms to third party sites.

6.Takeover of the user's browser : such as adding JavaScript code to redirect the user to site attacker wishes to.

7.Pop Up Flooding : Malicious scripts may make your website inaccessible and also browsers crash or inoperable.

8.Scripts can spy on what the user does such as history of the sites they have visited and track the information they posted to a web site and also can access personal data such as credit card details or bank account details.

9.Access to business data such as bid details or construction details.

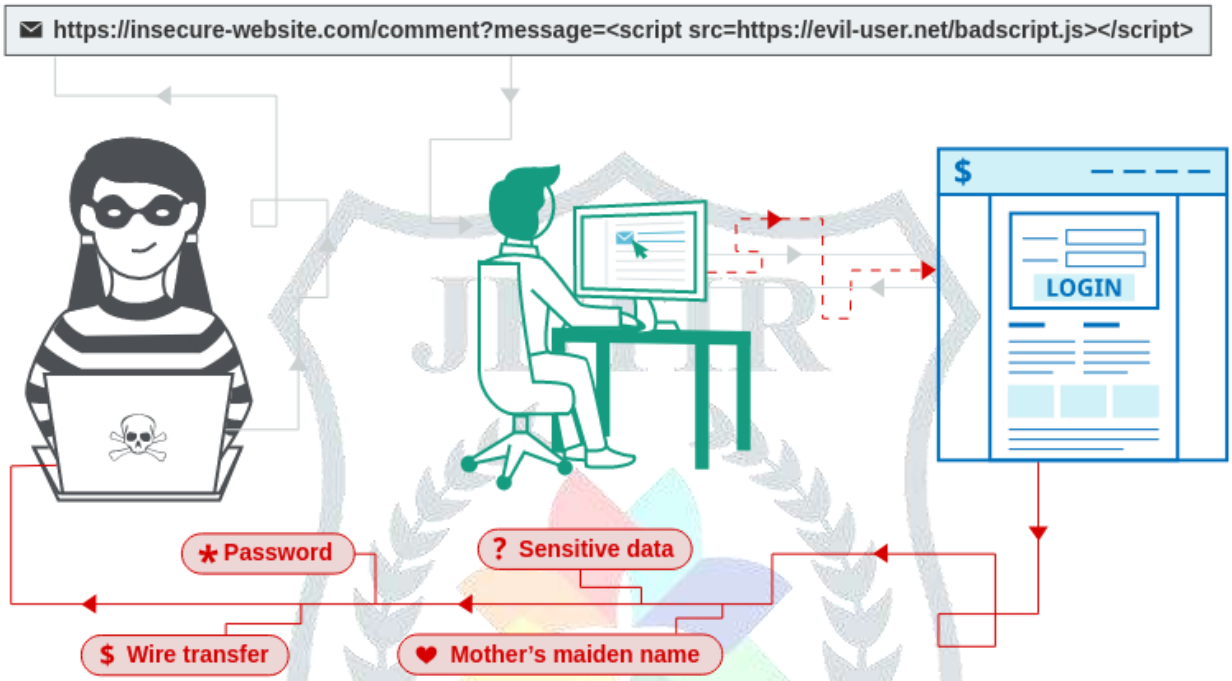

Fig 2. Example of mis-use of login form to steal user's information

IV. Types of Cross-Site Scripting Attack

There are three different types of XSS attacks: nonpersistent, persistent, and DOM-based.

Persistent XSS Attack:

In the persistent XSS attack , an attacker injects the malicious code into the page persistently and meaning the code will be stored in the target servers as html text, such as in a database, in a comment field or messages posted on forums, etc. and this code will be stored in the page which is shown to the user later. If the user victim goes to the page which is embedded with XSS attack code, the code will execute on the user's browser, which in turn sends the user's sensitive info from his site to the attacker's site. The persistent XSS attack is also known as stored XSS attack. Compared with reflected attack, this type of XSS does more serious harm. If the stored xss vulnerability is successfully exploited by attackers it will persistently attack the users until administrator remove this vulnerability. The following Figure 3.1 shows architecture of exploiting the persistent XSS attack by a malicious attacker.

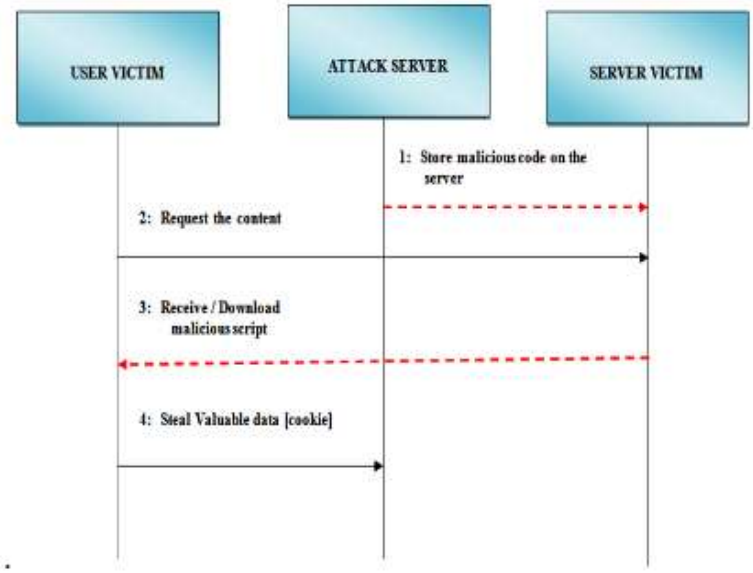

Fig 3.1 Persistent XSS attack

Non-persistent XSS Attack:

Non-persistent cross-site scripting vulnerability is a common type of XSS attack. The attack code is not persistently stored, but it is immediately reflected back to the user. It is also known as reflected XSS attack. In this type, the injected code is sent back to the user off the server, such as in an error message, search result or any other response that includes some input sent to the server as part of the request.

For this, the attacker sends a link to the user victim for eg. via email, if the user victim clicks on that particular link, the vulnerable web application displays the requested web page with the information passed to it in the link sent to the user. This information contains the malicious code which is now part of the web page displayed which is sent back to the web browser of the user, where it's executed. The following Figure 3.2 is an architecture which shows the sequence of steps of exploiting the non-persistent XSS vulnerability by a malicious attacker.

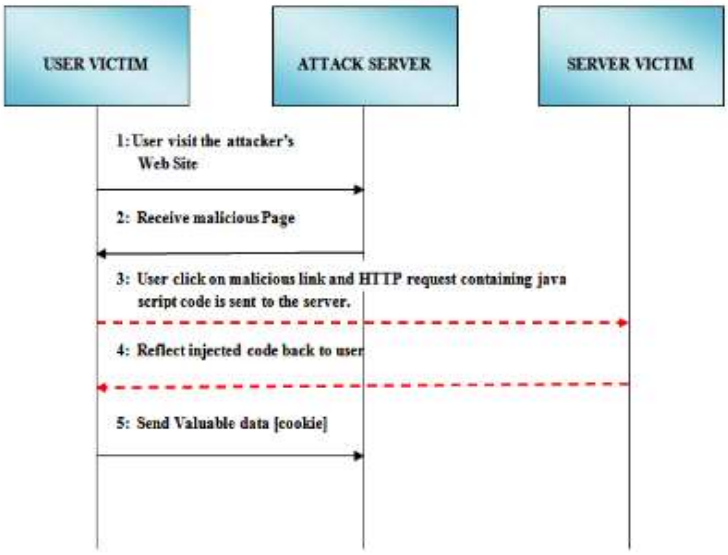

Fig 3.2 non-persistent XSS attack

DOM based XSS attack:

DOM based cross-site scripting attacks are executed by modifying the DOM environment in the client-side instead of sending malicious code to server. DOM is short form for Document Object Model and its a platform-independent and language neutral interface. DOM allows scripting to change HTML or XML document. It can be modified by a hacker's scripting or program. Hence, DOM-based XSS attack uses DOM's vulnerability to make the XSS come true. This type of XSS vulnerability is completely different from the reflected or stored XSS attack and it doesn't inject malicious code into a web page. So its the problem of the insecure DOM object which can be controlled from the client side in the web page or application. For this reason, hackers let the attack payload execute in the DOM's environment to attack the user side. The following Figure 3.3 is an architecture which shows the sequence of steps of exploiting the DOM-based XSS vulnerability by a malicious attacker.

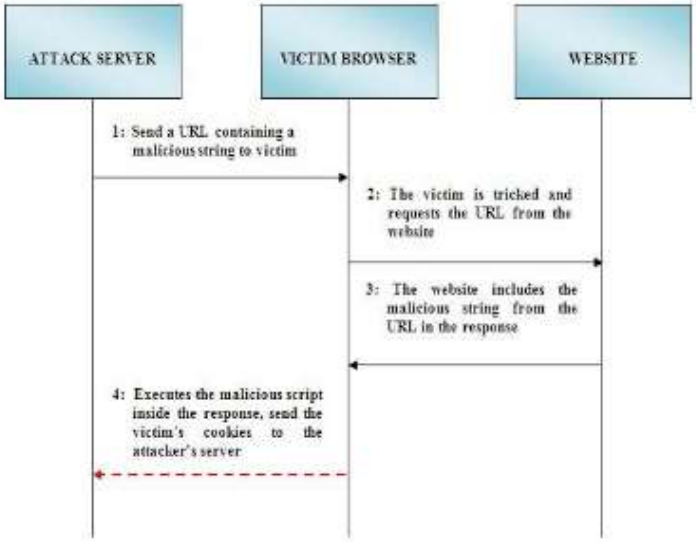

Fig 3.3 DOM-based XSS attack

V. Detection of Cross-site Scripting Attack

In order to detect XSS attack, filtering and other detection methods are being researched. This section explains the work related to it.

Static Analysis Approach:

Analysis of String:

Analyzing reflective code in Java programs and checking errors in dynamically generated SQL queries using finite state automata as a target language representation to analyze Java.This approach is not that efficient than the other string analyzing methods because the source of data was not tracked and it must

also determine Finite State Automata between each operations, so not a practical approach to find XSS vulnerabilities. More often web applications have their own scripts and there are several other ways to invoke a JavaScript interpreter. This approach is not at all practical to find XSS vulnerabilities.

Software Testing Approaches:

Using software-testing techniques to detect XSS vulnerabilities such as black-box testing, fault injection, and behavior monitoring to web application. This approach combines user experience modeling like black-box testing and user-behavior simulation.

Since, these approaches are applied to identify errors in development cycle they may be unable to provide instant protection to web application. They cannot guarantee the detection of all flaws as well.

Dynamic Analysis Approach:

Browser-Enforced Embedded Policies Approach:

A white list of all benign scripts is given by the particular web application to browser to protect it from malicious code. This is a good idea which allows only the scripts in the provided in the list to run. However, since there is are differences in the parsing mechanism of different browsers, a successful filtering system of one browser may not be successful for other browser. But a modification is required to be done in all the browsers to enforce this policy. So it suffers from scalability problem from the web application's point of view. Each client needs to have this modified version of browser on their system.

Syntactical Structure Approach:

A successful injection attack ensures that there is a change in the syntactical structure of the exploited entity. This approach checks the syntactic structure of output string to detect malicious payload. Enlarge the user's input with metadata to track this substring from source to sinks. This metadata helps the modified parser to check the syntactical structure of the dynamically generated string by indicating start and end position of the user's given data. This approach was found to be quite successful while it detects any injection vulnerabilities other than XSS.

Proxy-based Approach:

Noxes is a web proxy which protects against transferring of sensitive information from victim's site to third party sites. This is an application-level firewall todetect and block malware. There is a control provided to users on every connection coming or leaving the local machine. If any connection is mismatched with the firewall's rules then the firewall prompts the user to decide whether the connection needs to be blocked or allowed. Similar kind of approaches has been followed. It is not sufficient to blacklist a link to prevent cross-site Scripting attacks. Proxy-based approach does not provide a method to find errors, in addition it requires a watchful configuration. This approach could definitely increase false positives as they protect the unpredictable link with no examination of the fault.

Interpreter-based Approaches:

This approach uses instrumenting interpreter to track untrusted data at the character level and for identifying vulnerabilities that use context-sensitive string evaluation at each susceptible sink. This approach is sound and can detect vulnerabilities as they add security assurances by modifying the interpreter.

VI. Prevention of Cross-site Scripting Attack

Server Side Solution:

There are two types of testing tools – Black-box web application testing tools and white-box vulnerability scanners which are successfully used in real time practice. These kind of tools can generally help in identifying cross site scripting vulnerabilities. Its likely that some remain undetected, which recommend additional safeguards for web application. An application level firewall located on a security gateway between client and server,applies all security relevant checks and transformations. Wurzinger in 2009 introduced a secure web application proxy for server side solution to detect and prevent cross site scripting attacks which is comprised of reverse proxies that intercepts HTML responses. it also consists of a modified browser which is utilized to detect script content. Shahriar and Zulkernine (2009) applied the conceptual idea of mutation based testing technique to generate adequate test data sets for testing XSS vulnerabilities. They addressed XSS vulnerabilities related to web applications that use Hypertext Preprocessor (PHP) and JavaScript code to generate dynamic HTML contents and their mutants. Their approach includes a strong mechanism for identifying scripts at the server side and also removes any script in the output that is not intended by the web application.Few defense mechanisms for cross site scripting attacks were proposed by Prithvi Bisht and Venkatakrishanan (2008) based on input validation that can effectively prevent these attacks on the server side. They proposed a new framework called XSS GUARD that which has been designed to prevent XSS attacks on the server side which works by dynamically learning all the set of scripts which a web app intends to create for any HTML request. Researchers also developed the WebSSARI tool(Web Security via Static Analysis and Runtime Inspection), which performs type based static analysis to identify potentially vulnerable code sections and instrument them with runtime guards.

Client-side Solution:

A proposed client-side solution is detecting malicious Java Scripts consisting of a browser-embedded script auditing component and Id's that processes the audit logs and compares them to signatures of known malicious attacks.

Using this its possible to detect various kinds of malicious scripts and not only XSS attacks. For each type of attack a signature must be crafted, which means that the system is defeated by original attacks. Client side cross site scripting protection – Noxes tool is a client-side web proxy which passes on all

web traffic and serves as an application level firewall to the browser. Noxes requires user specific configuration as well as user interaction when a suspicious event occurs during the usage of a browser. A number of software-testing techniques can be used whose main aim is to cover and fix web vulnerabilities such as XSS. Philipp Vogt,et al.,(2007) proposed a solution to prevent XSS attacks on the client side by tracking flow of sensitive information from inside of the browser. Stephen Chong in 2007 developed a compiler that uses policies to automatically partition the program into JavaScript code and Java code running on the browser and server respectively. Their system protects the code and data on the client side. This data and code can be replicated across the client and server to provide both security and performance. Mike Ter Louw and Venkatakrishnan in 2009 presented a new XSS defense strategy that is widely deployed in existing web browsers for providing effective security. This approach eases the trust placed on browsers for interpreting untrusted content. They have evaluated blueprints against a barrage of stress tests that demonstrates strong resistance to attacks and provides excellent compatibility with web browsers with reasonable performance overheads. Based on the analysis of client side information, Nuo Li in 2010 proposed a new approach which generates test inputs for User Input Vectors(UIV). They used input field information to generate valid inputs and then procure valid inputs to generate invalid test inputs. Their approach is effective than the currently existing vulnerability scanners in finding semantics related vulnerabilities of UIV in web applications.

Conclusion

The use of the web paradigm is becoming an upcoming strategy for application software companies which permits the design of extensive applications which can be potentially used by thousands of customers from simple web clients. Moreover, the existence of new technologies for the improvement of web features allows software engineers the conception of new tools which are not longer restricted to specific operating systems. Existing approaches used to secure traditional applications are not always sufficient when its addressing the web paradigm and often leaves end users responsible for the protection of key aspects of a web service. This situation must be prevented since if its not well managed, it could allow improper uses of a web application and also lead to a violation of its security requirements. In this paper we focus on the types of Cross-Site Scripting attacks against the web application security. This attack mainly injects malicious code into a web application, in order to compromise the trust relationship between a web application's site and it's user.

If the vulnerability is successfully exploited, then it allow attackers to execute script in the victims browser, which can hijack user's sessions, deface web sites being used, insert hostile content, and also conduct phishing attacks..We survey in this paper the XSS attacks that affect current web applications and also discuss existing approaches for its detection and prevention of XSS attack. We discuss the above mentioned approaches and their limitations, as well as their deployment and applicability.

References

1. M. Singh and P. Kumar, "An Analytical Study on Cross-Site Scripting," 2020 International Conference on Computer Science, Engineering and Applications (ICCSEA), 2020.

2. G. Shanmugasundaram, S. Ravivarman, and P. Thangavellu, “A study on removal techniques of Cross-Site Scripting from web applications,” 4th IEEE Spons. Int. Conf. Comput. Power, Energy, Inf. Commun. ICCPEIC 2015, pp. 436–442, 2015.

3. S. K. Mahmoud, M. Alfonse, M. I. Roushdy and A. M. Salem, "A comparative analysis of Cross Site Scripting (XSS) detecting and defensive techniques," 2017 Eighth International Conference on Intelligent Computing and Information Systems (ICICIS), 2017, pp. 36-42, doi: 10.1109/INTELCIS.2017.8260024.

4. V. K. Malviya, S. Saurav and A. Gupta, "On Security Issues in Web Applications through Cross Site Scripting (XSS)," 2013 20th Asia-Pacific Software Engineering Conference (APSEC), 2013, pp. 583-588, doi: 10.1109/APSEC.2013.85.

5. K. Pranathi, S. Kranthi, A. Srisaila and P. Madhavilatha, "Attacks on Web Application Caused by Cross Site Scripting," 2018 Second International Conference on Electronics, Communication and Aerospace Technology (ICECA), 2018, pp. 1754-1759, doi: 10.1109/ICECA.2018.8474765.

6. G. A. Di Lucca, A. R. Fasolino, M. Mastoianni and P. Tramontana, "Identifying cross site scripting vulnerabilities in Web applications," Proceedings. Sixth IEEE International Workshop on Web Site Evolution, 2004, pp. 71-80, doi: 10.1109/WSE.2004.10013.

7. Engin Kirda, Christopher Kruegel, Giovanni Vigna, and Nenad Jovanovic. Noxes: A client-side solution for mitigating cross site scripting attacks. In Proceedings of the 21st ACM Symposium on Applied Computing (SAC), 2006.

8. H. Takahashi, K. Yasunaga, M. Mambo, K. Kim, and H. Y. Youm, “Preventing abuse of cookies stolen by XSS,” in 2013 Eighth Asia Joint Conference on Information Security, 2013, pp. 85–89.

9. A. Shrivastava, S. Choudhary and A. Kumar, "XSS vulnerability assessment and prevention in web application," 2016 2nd International Conference on Next Generation Computing Technologies (NGCT), 2016, pp. 850-853, doi: 10.1109/NGCT.2016.7877529.

10. Y.-W. Huang, F. Yu, C. Hang, C. H. Tsai, D. Lee and S. Y. Kuo, “Securing web application code by static analysis and runtime protection,” In Proceedings of the 13 th International World Wide Web Conference, (2004).