Hacking And Prevention For Secure Bank System

Pratik Suryawanshi[1], Abhishek Dhole[2], Somanath Maske[3], Supriya Shirsat[4]
[1, 2, 3, 4] Dept of Computer Engineering
[1, 2, 3, 4] RMD Sinhgad School of Engineering

Abstract*- This is a Banking web application which provides various functionalities to User and Admin. Admin can approve or reject user application. Admin can search customers by ac- count number or customer name. Admin can see logs of at- tacks. User can do online transactions like Fund Transfer, Bill Payments (electricity bill, income tax, mobile recharge).The application is prevented by detecting different attacks such as SQL injection, URL injection, Cross site Scripting attack and Brute force attack. User will get randomly generated user- name, password and pin number on his email. After every transaction user will get notification by message. User can see his account details and mini statement.*

I. INTRODUCTION

In this we are using Banking application can do online trans- action, and can detect attacks like SQL injection, Brute force attack, URL injection, Cross site scripting attack. Personal information of user get stored in database in encrypted for- mat. Dynamic password and pin generates and send to user on his Email. After every transaction user get notification by message. User can see his account details and mini state- ments

The Advanced Encryption Standard or AES is a symmet- ric block cipher used by the U.S. government to protect clas- sified information and is implemented in software and hard- ware throughout the world to encrypt sensitive data.

II. MOTIVATION BACKGROUND

We are motivated from the attack on COSMOS bank in Aug,2018. NPCI says cyber fraud due to malware attack on banks IT system. Hacker used middleware attack and transferred 94 crore over in 21 countries.

III. RELATED WORK

Server-side approaches: Cross-site Scripting is essentially an input filtering failure. Consequently, methods have been developed to target malicious inputs even before they reach the web server. Traditionally, web application firewalls

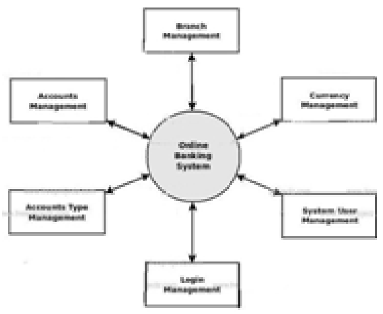

Figure 1: Work Flow Diagram.

(WAFs) are either scanning for attack signatures in the pa- rameters passed on to the web application (including POST- parameters, cookies, etc.), or require an administrator to manually specify a ruleset to match requests against . Both ways can be regarded as an external second in- put filter- ing layer. A first anomaly-based intrusion detection system for web applications was proposed by Kruegel and Vigna in . Their system derives a number of statistical character- istics from observed HTTP requests, regarding the param- eters length, character distribution, structure, presence and order. However, unlike our methods, both approaches con- centrate solely on the incoming query parameters while ig- noring the respective HTTP response, thus either causing unnecessary false positives or missing certain attacks. Is- mail et al. describe an XSS detection mechanism which follows an approach similar to our reflected detector . Us- ing a server-side proxy incoming parameters are checked for contained HTML markup. If such a parameter could be identified, the respective HTTP response is examined if the same HTML markup can be found in the responses HTML content. In comparison to our approach the proposed tech- nique has several shortcomings. The HTML- based match- ing approach is inaccurate, as it fails to identify in-script and attribute-injections. Furthermore, unlike our technique, the proposed detector also does not consider transformation- processes, such as character-encoding or removal filters, that may alter the incoming parameters before their reflection on the outgoing HTML. Taint analysis has been proven to be a powerful tool for detecting code injection vulnerabilities. Taint analysis tracks the flow of untrusted data through the application. All user-provided data is tainted until its state is explicitly set to be untainted. This allows the detection if un- trusted data is used in a security sensible context. Taint anal- ysis was first introduced by Perls taint mode. More recent work describes finer grained approaches towards dynamic taint propagation. These techniques allow the tracking of untrusted input on the basis of single characters. In indepen- dent concurrent works Nguyen-Tuong et al and Pietraszek and Vanden Berghe proposed fine grained taint propagation to counter various classes of injection at- tacks. Halfond et al. describe a related approach (positive tainting) which, un- like other proposals, is based on the tracking of trusted data. Xu et al propose a fine grained taint mechanism that is im- plemented using a C-to-C source code translation technique. Their method detects a wide range of injection attacks in C programs and in languages which use interpreters that were written in C. To protect an interpreted application against injection attacks the application has to be executed by a re- compiled interpreter. However, in any case dynamic taint tracking requires profound changes to either the monitored application or the application server/run-time. Thus, access to the source code of one of these components is required. Also, a taint tracking based solution is necessarily always specific for a certain technology (programming language, application server), while real-life web applications are of- ten composed of heterogeneous systems. Finally, real-time data-tracking always introduces certain performance penal- ties. In comparison, our approach is applicable with all lan- guages and application servers, does not require any changes to the executed code, and is able to monitor highly heteroge- neous set-ups. Also, as we propose a passive offline detector, no performance penalties are introduced.

Client-side approaches: We are proposing detection methods that are positioned exclusively at the server-side. For the sake of completeness, this section lists related ap- proaches that incorporate the client-side web browser: In concurrent and independent work an XSS filter for the Inter- net Explorer browser was implemented which follows an ap- proach that is closely related to our reflected detector: Based on an analysis of outgoing HTTP parameters, signatures are generated which are then checked against the correspond- ing HTTP response. Furthermore, the NoScript- plugin for Firefox provides a simple protection mechanism against re- flected XSS: Outgoing HTTP parameters are checked if they potentially contain JavaScript code. If such parameters are detected, the plugin warns the user be- fore sending the re- spective HTTP request. As the incoming HTTP response is ignored, the plugin produces unnecessary false positives. Both browser-based approaches are unable to detect stored XSS.

With Browser-Enforced Embedded Policies (BEEP) , the web server includes a whitelist-like policy into each page, allowing the browser to detect and filter unwanted scripts. As the policy itself is a JavaScript, this method is very flex- ible and for instance allows the definition of regions, where scripts are disallowed. BEEP re- quires the usage of a mod- ified web browser. does not elaborate how the list of legit- imate scripts is supposed to be compiled. Instead this step is left to the applications developers. As our generic detec- tor is specifically designed to establish the list of legitimate scripts, a combination of the two approaches appears to be promising. Finally, Hallaraker and Vigna in modified Mozil- las SpiderMonkey Engine to track the behaviour of client- side JavaScript. The activity profile of each script then is matched against a set of high-level policies for detecting ma- licious behaviour.

IV. OVERVIEW

Our detection mechanism for reflected XSS is based on the observation that reflected XSS implies a direct re- lationship between the input data (e.g., HTML parameters) and the injected script. More precisely: The injected script is fully contained both in the HTTP request and the HTTP response. Reflected XSS should therefore be detectable by simply matching incoming data and outgoing JavaScript us- ing an appropriate similarity metric. It is crucial to empha- sise that we match the incoming data only against script code found in HTML. Non-script HTML content is ignored. See Section 5 for our script-extraction technique. For the sake of readability we will uses the term parameters as a gener- alized term for all user-provided data in the sequel. We can formulate the problem to be solved as follows:

Problem 1: Given a set of parameters P = p1, p2, ..., pm and a set of scripts S = s1, s2, ..., sn find all matches between P and S in which pi was used to define parts of sj.

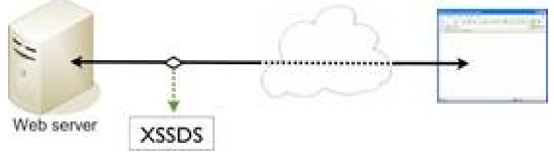

Figure 2: Passive XSS attack detection

V. ADOBTED ALGORITHM

Definition 1: Given a string p = p1p2...pn , the DFA

Dp = (Q, , , s0, F) DFA with Q = s0, s1, ..., sn (states)
: Q Q : (si, wj) = (transitions) sj i ¡ j [jJ : i ¡ jJ ¡ j pj = pjr] F = Q (final states)
s0 Q (starting state) (alphabet)accepts exactly the set of all subsequences of p.

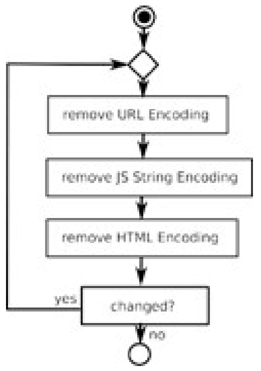

Figure 3: Algorithm for encoding removal

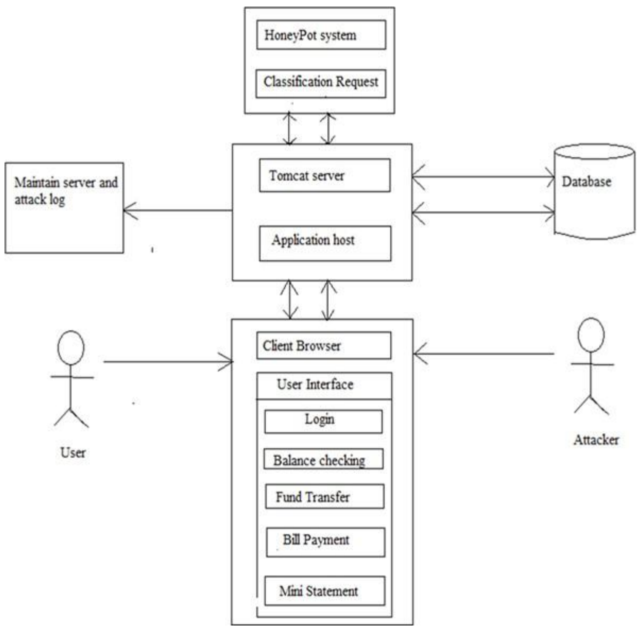

Figure 4: System Architecture.

VI. LIMITATION

The proposed detector relies upon a direct comparison of in- coming HTTP-parameters and outgoing HTML. Stored XSS is therfore not always detectable with this approach: the required direct relationship between HTTP request and re- sponse does not necessarily exist. It might be possible to de- tect the initial exploiting request/response pair, if the given stored XSS takes effect immediately. However, in certain cases, the HTTP request that injects the malicious payload permanently in the application and the poisoned HTML re- sponse are not created consecutively.

VII. METHODOLOGY

Generally, any attack detection system should have two major capabilities: detecting as many attacks as possible whilst having a false-positive rate as low as possible. We assessed the detection abilities of both approaches by apply- ing them to crafted attacks injected into otherwise benign data and real-world attack data of disclosed XSS problems. Furthermore, we measured the false-positive rate by apply- ing the detectors to our collected dataset which we assumed contained no attacks. This time every alarm was counted as a false-positive and logged. Afterwards all alarms were re- viewed by hand to make sure there really were no attacks in the data. The error-rates were divided by the number of pages used for testing in order to remove the influence of different web application sizes.

VIII. CONCLUSION

We described XSSDS a server-side Cross-site Scripting de- tection system. The systems uses two novel detection- ap- proaches that are based on generic observations of XSS at- tacks and web applications. A prototypical implementation demonstrated our approachs capabilities to reliably detect XSS attacks while maintaining a tolerable false positive rate. As our approach is completely passive and solely requires reading access to the applications HTTP traffic, it is appli- cable to a wide range of scenarios and works together with all existing web technologies.

REFERENCES

[1] R. A. Baeza-Yates and G. H. Gonnet. Fast text searching for regular expressions or automaton searching on tries. Journal of the ACM, 43(6):915 936, November 1996.

[2] Blwood. Multiple xss vulnerabilities in tikiwiki 1.9.x. mailing list Bugtraq, http://www.securityfocus. com/archive/1/435127/30/120/threaded, May 2006.

[3] S. Christey and R. A. Martin. Vulnerability type distribu- tions in cve, version 1.1. [online], http://cwe.mitre. org/documents/vuln-trends/index.html, (09/11/07), May 2007.

[4] K. Fernandez and D. Pagkalos. Xssed.com - xss (cross- site scripting) information and vulnerabile web- sites archive. [on- line], http://xssed.com (03/20/08).

[5] D. Gusfield. Algorithms on Strings, Trees, and Se- quences: Computer Science and Computational Biology. Cam- bridge University Press, New York, USA, 1997. ISBN 0521585198.

[6] W. G. Halfond, A. Orso, and P. Manolios. Using posi- tive tainting and syntax-aware evaluation to counter sql injection attacks. In 14th ACM Symposium on the Foun- dations of Soft- ware Engineering (FSE), 2006

[7] O. Hallaraker and G. Vigna. Detecting malicious javascript code in mozilla. In Proceedings of the IEEE In- ternational Conference on Engineering of Complex Com- puter Systems (ICECCS), pages 8594, June 2005.

[8] R. Hansen. XSS (cross-site scripting) cheat sheet - esp: for filter evasion. [online], http://ha.ckers.org/xss. html, (05/05/07).

[9] O. Ismail, M. Eto, Y. Kadobayashi, and S. Yam- aguchi. A proposal and implementation of automatic detection/collec- tion system for cross-site scripting vul- nerability. In 8th In- ternational Conference on Advanced Information Network- ing and Applications (AINA04), March 2004.

[10] T. Jim, N. Swamy, and M. Hicks. Defeating script injection attacks with browser-enforced embedded poli- cies. In 16th International World Wide Web Conference (WWW2007), May 2007.

[11] A. Klein. Cross site scripting explained. White Pa- per, Sanc- tum Security Group, http://crypto.stanford. edu/cs155/CSS.pdf, June 2002.

[12] A. Klein. Dom based cross site scripting or xss of the third kind. [online], http://www.webappsec.org/ projects/articles/071105.shtml, (05/05/07), Sebtember 2005.

[13] J. Kratzer. Jspwiki multiple vulnerabilitie. Post- ing to the Bugtraq mailinglist, http://seclists.org/ bug- traq/2007/Sep/0324.html, September 2007.

[14] C. Kruegel and G. Vigna. Anomaly detection of web- based attacks. In Proceedings of the 10th ACM Con- ference on Computer and Communication Security (CCS 03), pages 251261. ACM Press, October 2003.

[15] G. Maone. Noscript firefox extension. Software, http: //www.noscript.net/whats, 2006.

[16] Misc. New xss vectors/unusual javascript. [on- line], http://sla.ckers.org/forum/read.php? 2,15812 (04/01/08), 2007.