SQLAS: Tool To Detect And Prevent Attacks In PHP Web Applications

Vandana Dwivedi[1], Himanshu Yadav[2] and Anurag Jain[3]

[1]Department of Computer Science & Engineering , RITS ,Bhopal (India)
[2]Department of Computer Science & Engineering , RITS ,Bhopal (India)
[3]Department of Computer Science & Engineering , RITS ,Bhopal (India)

ABSTRACT

Web applications become an important part of our daily lives. Many other activities are relay on the functionality and security of these applications. Web application injection attacks, such as SQL injection (SQLIA), Cross-Site Scripting (XSS) and Cross-Site Request Forgery (XSRF) are major threats to the security of the Web Applications. Most of the methods are focused on detection and prevention from these web application vulnerabilities at Run Time, which need manual monitoring efforts. Main goal of our work is different in the way it aims to create new systems that are safe against injection attacks to begin with, thus allowing developers the freedom to write and execute code without having to worry about these attacks. In this paper we present SQL Attack Scanner (SQLAS) a Tool which can detect & prevent SQL injection Attack in web applications. We analyzed the performance of our proposed tool SQLAS with various PHP web applications and its results clearly determines the effectiveness of detection and prevention of our proposed tool. SQLAS scans web applications offline, it reduces time and manual effort due to less overhead of runtime monitoring because it only focus on fragments that are vulnerable for attacks. We use XAMPP for client server environment and developed a TESTBED on JAVA for evaluation of our proposed tool SQLAS.

KEYWORDS

Web applications vulnerability, SQL injection attack, Cross-side scripting, Cross-site request forgery PHP, WWW

1. INTRODUCTION

Web application security is primary issue in today's web world. The Web serves as a convenient portal between end-users and company resources such as user accounts. Web application permits user to link up with server by data exchange, which might be access web's database or other resources of system with the help of different browsers. Such applications that facilitate a rich online experience (personal financing, social networking, business transactions, etc) are burdened with securing the privacy of sensitive data while permitting data exchange. Inputs to web application are known as Injection Attack if they belong to vulnerable category, these inputs include XSS and SQL [1]. SQL and XSS are top most web application security thread listed by OWASP in 2010[1]. Attack happens when

Injection attacks happens once input passes from the web browser to the web server application, probably onto the database and even back to the user's browser; and this input includes malicious values/scripts that may alter the activities of the online web application and produced surprising results. A typical winning attack begins within the client-side browser wherever an internet application is rendered, giving AN assailant chance to input malicious information via the browser. This information is then sent to the server, wherever it should reach the back-end information via a query, leading to a SQL Injection attack, or it should be sent back to the attacker's client-side browser for execution, leading to a Cross-Site Scripting Attack. These attacks become susceptible information or provide unauthorized access to system resources forthwith.

Attacks will abuse on an innocent user once the previously stored malicious information is retrieved and becomes a part of a query or the rendered web content. One extreme measure to counter such attacks would be to prohibit any user input to net applications; such a measure is impractical for any net application that interacts with end-users. an internet application while not input has terribly restricted functionality; it doesn't have the power to query the user so as to show pertinent info. Thus, there's no access to back-end databases to retrieve personal info (e.g., a checking account balance) neither is there the aptitude to perform transactions (e.g., on-line bill pay). Therefore, the first measure for countering injection attack involves characteristic attainable malicious inputs and fixing them, so rendering them benign. this is often named because the cleaning methodology.

In this research work we focused on the detection and prevention of SQL injection vulnerability in web applications written in PHP. We choose SQL injection and PHP web applications for several reasons. First, many web applications have SQL injection vulnerability. SQL injection attack could combine with other attach techniques, which occurs very frequently, and could cause huge financial losses for organizations and companies. In 2013, OWASP Top Ten Project rated SQL injection as the number one attack [1]. Second, PHP is a popular web development programming language, but web applications developed in PHP are the most vulnerable web applications according to the study conducted by Positive Technologies [2]. The study compared the security vulnerabilities of web sites on PHP, ASP.NET, and Java caused by inappropriate software implementation. It showed that 81% of sites in PHP contain critical security vulnerabilities, and 91% contained medium-risk vulnerabilities. Last, but not the least, the tool developed in the research can be used for educational purpose. Students can use this tool to check SQL injection vulnerability on their course projects.

This paper is organized as follows. Section two, includes some related work in web application Vulnerability domains. Section three, discusses the approach to web application security, provides a brief overview of the proposed method and explains the model. In Section four, we explain the proposed system. In section five, we evaluate the performance of SQLAS as discussed followed by a conclusion and future work.

2. LITERATURE SURVEY

This work is expounded to some areas of active analysis, such as rationalization and victimization specifications for bug finding, vulnerability analysis, and attack detection for web applications. during this section, we tend to describes the foremost seminal and interesting works in these areas with the intent of highlight the most trends and achievements in these areas of research, and to position the work drawn throughout this paper at intervals the context of previous work. A tool WebSSARI [3], revealed in 2004, and is the key work that applies static taint propagation analysis to finding security vulnerabilities in PHP applications. WebSSARI targets three specific varieties of vulnerabilities: XSS, SQL Injection, and general script injection. WebSSARI used flow-sensitive, intra-procedural analysis supported a lattice model and sort state. Previously discussed all the PHP language is extended with two type-qualifiers, specifically tainted and unblemished, and therefore the tool keeps track of the type-state of variables. The tool uses three user-provided files, known as prelude files: a file with preconditions to any or all sensitive functions (i.e., the sinks), a file with post conditions for acknowledged cleansing functions, and a file specifying all attainable sources of doubtful input. So as to undamaged the contaminated information, the information needs to be processed by a cleansing routine or to be forged to a secure kind. Once the tool determines that tainted information reaches sensitive functions, it mechanically inserts runtime guards, that area unit cleansing routines.

Another approach conjointly supported static taint propagation analysis, to the detection of input validation vulnerabilities in PHP applications is delineated in [4, 5]. A flow sensitive, inter-procedural and context-sensitive information flow analysis is employed to spot intra-module XSS and SQL injection vulnerabilities. These approaches are imposed in a very tool, known as PIXY that is that the most complete static PHP analyzer in terms of the PHP options shapely. To the simplest of our information, it's the sole publicly-available tool for the analysis of PHP-based applications

The work by Xie and author [6], revealed in 2006, describes a three-level approach to search out SQL injection vulnerabilities in PHP applications. In beginning symbolic execution is used to model the impact of statements within the fundamental blocks of intra-procedural management Flow Graphs (CFGs). Then, the ensuing block outline is employed for intra procedural analysis, wherever a typical reachable analysis is employed to achieves goal. In conjunction with substitute data, every block outline contains a group of locations that were unblemished within the given block. The block summaries area unit composed to come up with a perform outline, that contains the pre- and post-conditions of perform.

The research by Su associated Wassermann [7] is another example of an approach which uses a model of “normality” to find injection attacks, like XSS, XPath injection, and shell injection attacks. However, this implementation, known as SqlCheck is meant to find SQL injection attacks solely. The approach works by trailing substrings from user input through the program execution. The trailing is enforced by augmenting the string with special characters, which spot the beginning and so the finish of every substring. Then, dynamically-generated queries area unit intercepted and checked by a changed SQL programmed.

Recently, applicability and scalability of model checking approaches in the domain of web applications started being explored by the research community. In particular, in 2008, a work describing the QED system was published [8]. QED identifies XSS and SQL injection vulnerabilities that arise as a result of the interaction of multiple modules of a servlet-based web application. The system uses explicit model checking to find XSS and SQL injection vulnerabilities and uses a number of heuristics to scale the approach to large applications. To find vulnerability, the tool needs to be supplied with a specification of a vulnerability (written in PQL) and a set of inputs to the application under test. Then, the Java PathFinder [15] model checker is used to execute the application using a sequence of user requests that are generated based on user input values provided by an analyst. Vulnerability is found when a match to a PQL query is found by the model checker. In general, the approach proposed in this work can be applied to detect vulnerabilities other than taint-based ones if an analyst is able to provide the tool with a specification of a vulnerability specifying patterns of events (such as program method calls) that need to occur on a program path.

Another example is the work by Paleari et al. [9] that takes a look at a new and unexplored class of vulnerabilities in the domain of web applications. In particular, the paper looks at race condition vulnerabilities that can arise in web applications interacting with a back-end database. A race condition may occur in a multi-threaded environment between two database queries if data accessed by one query can be modified by another one. In a multi-threaded application, the shared data in the database might not be consistent between the two queries if code that was designed to be executed sequentially is executed concurrently. The authors propose a dynamic approach to identify this class of vulnerabilities, in which all database queries generated by a running program are logged and analyzed (offline) for data dependencies.

A good example of an approach based on a model of expected behavior is the work of Halfond and Orso, whose tool is called AMNESIA [10]. AMNESIA is particularly concerned with detecting and preventing SQL injection attacks for Java-based web applications. In the static analysis phase, the tool builds a conservative model of predictable SQL queries. Then, at execution-time, dynamically-generated queries are checked against the resulting model to identify instances that violate the intended structure of a query. AMNESIA uses Java String Analysis (JSA) [19], a static analysis technique, to build an automata-based model of the set of legitimate strings that a program can produce at given points in the code. AMNESIA also leverages the approach proposed by Gould, Su, and Devanbu [11] to statically check type correctness of dynamically-generated SQL queries.

SQL DOM [12] uses database queries encapsulation access to databases confidentially. SQL DOM use a type-checked API which found query building process is efficient. Consequently by API they apply coding best practices such as input filtering and strict user input type checking. The disadvantage of the methods is that developer should learn new programming paradigm or query-development process. SQL-IDS [13] focused on writing specifications for the internet application that explain the intended configuration of SQL statements that are produced by the application, and in automatically monitoring the execution of these SQL statements for violations with respect to these specifications.

A proxy filtering system that intensifies input validation rules on the data flowing to a Web application is called Security Gateway [14]. In this technique for transferring parameters from web-page to application server, developers should use Security Policy Descriptor Language (SPDL). So developer should know which data should be filtered and also what patterns should apply to the data. SQLPrevent [15] is exists of a Hyper Text Transfer Protocol (HTTP) request interceptor. Core data flow is corrected when SQLPrevent is installed into a web server. The Hyper Text Transfer Protocol (HTTP) requests are saved into the existing thread-local storage. Then, SQL interceptor intercepts the SQL queries that are made by internet application and send them to the SQLIA detector module. Thus, Hyper Text Transfer Protocol request from threads local storage is fetched and analyzed to determine whether it restrains a SQLIA. The malicious SQL statement would be prevented to be sent to database, if it is doubtful to SQLIA. Rather than this there may many other modern methods has also been proposed for preventing SQLI attacks [16], [17], [12], [18], [19], [20], [21].

Swaddler [22] is a distinctive way to notice the anomaly-based attacks in internet applications. Swaddler analyses the inside status of an internet application and discovers the connections between the application’s very important execution points and also the application’s internal state. With this, Swaddler is capable to acknowledge attacks that commit to get associate application in associate inconsistent, abnormal state, like violations of the projected work flow of an internet application. Swaddler could be a unusual approach to the detection of attacks against internet applications. The approach relies on a close characterization of the inner state of an internet application, by suggests that of variety of anomaly models. A lot of exactly, the inner state of the appliance is monitored throughout a learning part. Throughout this part the approach derives the profiles that describe the traditional values for the application’s state variables in important points of the application’s elements. Then, throughout the detection part, the application’s execution is monitored to identify abnormal states. The approach has been enforced by instrumenting the PHP interpreter and has been valid against real-world applications.

Zhendong Su [23] presents the primary formal definition of command injection attacks within the context of internet applications, and provides a sound and complete rule for preventing those supported context-free grammars and compiler parsing techniques. Our key examination is that, for AN attack to succeed, the input that gets propagated into the info query or the output document should amendment the meant grammar structure of the query or document. This definition and rule are general and apply to several varieties of command injection attacks. This approach is confirmative with SQLCHECK, An execution for the setting of SQL command injection attacks. They evaluated SQLCHECK on real-world internet applications with consistently compiled real-world attack information as input. SQLCHECK made no false positives or false negatives, incurred low runtime overhead, and applied straightforwardly to internet applications written in numerous languages.

3. PROPOSED METHOD

In this paper we are proposing a new tool for SQL injection Detection and Prevention mechanism for the PHP web applications. However our planned technique will be equally effective for each simple and complex data structure. The SQL Attack Scanner (SQLAS) is a tool to detect and prevent attacks in PHP web applications, SQLAS is simple and really easy to implement. During this technique all the data validations rules are going to be during a secure place. The data validation rules also will be organized into some XML format and that they are referred to as XML-rules. Whenever server receives any input from client the server can verify the whole XML script supported the verification rules already written within the server.

The main distinction between SQLAS and therefore the alternative SQL injection detection and prevention methods is that our method validates the complete incoming wed application data at a time. Moreover the XML-rules are going to be written and stored on an individual basis. That creates the developer task easier, what they have to do is simply to verdict the validation task for validating any web application data. Presently one validation function written for an application. The validation data can verify the incoming XML scripts supported the foundations written within the XML-rules. For every application the incoming XML scripts can have registered format for various user requests. XML-rules are going to be written on an individual basis for every kind of incoming XML scripts and therefore the incoming XML script should succeed the validation method. This method can primarily divide the data validation of a web application from the application development division. The developer at present ought not to worry about the SQL injection attacks and data validity. The validation parts of data are going to be maintained by a separate cluster which can manage the XML-rules. This is often conjointly useful as a result of the traditional web developers are going to be utterly unaware regarding the safety rules of the application.

Figure 2.1 explains the essential flow of SQLAS. Once user can submit any query then rather than submitting straightforward data, it'll submit all the data in XML format. Essentially it'll send one XML script to the server. The server can receive the XML script, analyze it and obtain the initial data.

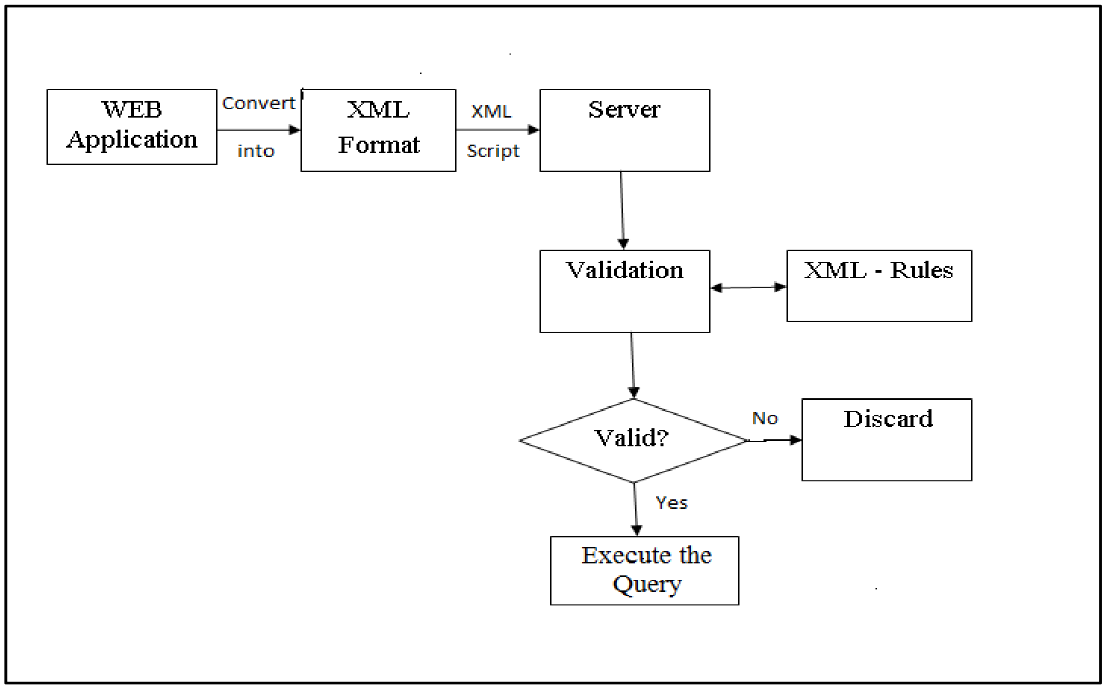

Fig.: 2.1 Flow Graph of proposed SQLAS

The Proposed SQLAS algorithm is as follow:

Algorithm 1: Algorithm SQLAS
Input: A web application.
Output: List of possible SQL injection threats.

- STEP 1: Select the Web page from web application folder.
- STEP 2: Convert it into XML script format (which is already known by server)
- STEP 3: Send this XML script to server

- STEP 4: Server receives XML script and checks its validity with pre-specified XML-Rules.
- STEP 5: Each SQL statement can have variables which are static, dynamic or composite. For each SQL statement do the following operations:

```
if if any variable is composite then
  for all composite variables in the statement do
      recursively verify the composite variable until all its dependent variables are either
      static or dynamic.
  end for
  if any of the dependent variable found as dynamic then
     the SQL statement is may not be safe.
  else
     the SQL statement is safe.
  end if
  else if if any variable is dynamic then
      the statement is may not be safe.
   else
   if all variables are static the statement is safe
end if
```

- STEP 6: if SQL statement is safe or valid than execute the SQL query.
- STEP 7: otherwise discard the SQL query.
- STEP 8: Exit.

4. EXPERIMENTAL ANALYSIS

For the Experimental analysis of proposed method we developed a Testbed, which is coded in JAVASCRIPT and able to run SQL queries of web Applications. To create Client server environment in PHP web application we used XAMPP 1.8.2 version. XAMPP is a tool , it is free and open source cross-platform web server solution stack package, consisting mainly of the Apache HTTP Server, My SQL database, and interpreters for scripts written in the PHP and Perl programming languages.

To evaluate the effectiveness of SQLAS : the SQL Injection Testbed was used. Fig 3 shows representation of Testbed. The SQL Injection Testbed has been used for evaluating various web applications developed in PHP. It examined number of test cases on an open source web applications. It contains two types of test cases: the XML rules from which SQL queries get validate and the other is Return error message by the server. Table 1 summarizes the Web applications. The first column contains the names of the Web applications. The second column contains its descriptions.

Table 1: Web Application Descriptions

Application Name	**Description**
(EMS) Employee Management System	An application for employees management in office developed in PHP
E-Lib	Library automation based on PHP
Project Evaluator	A PHP based academic application for managing students project
Online Shopping	Online shopping PHP web application
Medicine-S	An application for medical shops

Table 2 describes these vulnerable web applications. The first column shows the names of the Web applications, the second column shows total number of input for each application, third column shows Total malicious inputs, fourth column represents Total attacks detected and the fifth column shows the detection rate.

Table 2: Analysis of Results on the SQL Injection Testbed

Application	**Total inputs**	**Total malicious inputs**	**Total attacks detected**	**Rate of detection**
(EMS) Employee Management System	200	75	75	100%
E-Lib	250	92	91	98.9%
Project Evaluator	300	157	157	100%
Online Shopping	220	87	87	100%
Medicine-S	225	87	87	100%

5. CONCLUSION

In this work, we have converse about the concept of SQL injection which has become a common problem of all the web-based applications. We have summarized a variety of techniques existing to protected data from SQL injections, which can modify the program behavior permitting the attacker to retrieve and modify the database information. In this paper our research work presents a tool called SQLAS which can detect the vulnerability spots in the source code. Also, it provides the developer with an additional option of checking the syntax. Detection is done by validating the SQL queries using general validation procedure based on XML rules and the nature of the injection type. Any possibility of vulnerability or violation of the analyzed nature is reported as a warning message or error message to the developer. The warning message or error message includes the injection type description and the line number of vulnerable spot in the source code. The concepts explained in this work assist the Developer to modify the SQL statements and make the code attack free. We conclude by highlighting the robust features of the efficient tool SQLAS, which can detect the error during the development statically and can protect web applications from the future SQL injection.

We believe that the ideas presented in this research work can be further extended to include new injection types. This work also paves way for the development of vulnerability detection services, which can be used by developers to detect vulnerability spots in the source code. We feel the area of SQL injection vulnerabilities is wide open for research and we conclude by suggesting that this step to verify the code against SQL injection vulnerabilities can be added in the checklist for performance review in the static code analysis of the source code of data driven applications.

REFERENCES

[1] OWASP 2010 top ten," 2010. [Online]. Available: http://www.owasp.org

[2] Sergey Gordeychik, et al Web application vulnerability statistics for 2010- 2011 (2012) Positive Technologies. Available at: http://www.ptsecurity.com/download/ statistics.pdf..

[3] 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 12th International World Wide Web Conference (WWW'04), pages 40–52, May 2004.

[4] N. Jovanovic, C. Kruegel, and E. Kirda. Pixy: A Static Analysis Tool for Detecting Web Application Vulnerabilities. In Proceedings of the IEEE Symposium on Security and Privacy, May 2006.

[5] N. Jovanovic, C. Kruegel, and E. Kirda. Precise Alias Analysis for Static Detection of Web Application Vulnerabilities. In Proceedings of the ACM SIGPLAN Workshop on Programming Languages and Analysis for Security (PLAS'06), June 2006.

[6] Z. Su and G. Wassermann. The Essence of Command Injection Attacks in Web Applications. In Proceedings of the 33rd Annual Symposium on Principles of Programming Languages (POPL'06), pages 372–382, 2006.

[7] Y. Xie and A. Aiken. Static Detection of Security Vulnerabilities in Scripting Languages. In Proceedings of the 15th USENIX Security Symposium (USENIX'06), August 2006.

[8] M. Martin and M. Lam. Automatic Generation of XSS and SQL Injection At-tacks with Goal-Directed Model Checking. In Proceeding of the 17th USENIX Security Symposium, pages 31–43, July 2008.

[9] Java pathfinder. http://javapathfinder.sourceforge.net/

[10] R. Paleari, D. Marrone, D. Bruschi, and M. Monga. On race vulnerabilities in web applications. In Proceedings of the 5th Conference on Detection of Intru-sions and Malware & Vulnerability Assessmen t, DIMVA, Paris, France, Lecture Notes in Computer Science. Springer, July 2008

[11] W. Halfond and A. Orso. AMNESIA: Analysis and Monitoring for NEutraliz-ing SQL-Injection Attacks. In Proceedings of the International Conference on Automated Software Engineering (ASE'05), pages 174–183, November 2005

[12] A. Christensen, A. Møller, and M. Schwartzbach. Precise Analysis of String Ex-pressions. In Proceedings of the 10th International Static Analysis Symposium (SAS'03), pages 1–18, May 2003

[13] C. Gould, Z. Su, and P. Devanbu. Static Checking of Dynamically Generated Queries in Database Applications. In Proceedings of the 26th International Con-ference of Software Engineering (ICSE'04), pages 645–654, September 2004.

[14] R. A. McClure and I. H. Kr¨uger, "Sql dom: compile time checking of dynamic sql statements," in Proceedings of the 27th international conference on Software engineering, ser. ICSE '05, 2005, pp. 88–96.

[15] K. Kemalis and T. Tzouramanis, “Sql-ids: a specification based approach for sql-injection detection,” in Proceedings of the 2008 ACM symposium on Applied computing, ser. SAC ’08. ACM, 2008, pp. 2153–2158.

[16] D. Scott and R. Sharp, “Abstracting application-level web security,” in Proceedings of the 11th international conference on World Wide Web, ser. WWW ’02, 2002, pp. 396–407.

[17] P.Grazie, “Phd sqlprevent thesis,” Ph.D. dissertation, University of British Columbia(UBC) Vancouver, Canada, 2008.

[18] M. Cova, D. Balzarotti, V. Felmetsger, and G. Vigna, “Swaddler: An approach for the anomaly-based detection of state violations in web applications,” 2007.

[19] S. W. Boyd and A. D. Keromytis, “Sqlrand: Preventing sql injection attacks,” in In Proceedings of the 2nd Applied Cryptography and Network Security (ACNS) Conference, 2004, pp. 292–302.

[20] W. G. J. Halfond, A. Orso, and P. Manolios, “Using positive tainting and syntax-aware evaluation to counter sql injection attacks,” in Proceedings of the 14th ACM SIGSOFT international symposium on Foundations of software engineering, ser. SIGSOFT ’06/FSE-14, 2006, pp. 175–185.

[21] V. Haldar, D. Chandra, and M. Franz, “Dynamic taint propagation for java,” in Proceedings of the 21st Annual Computer Security Applications Conference, ser. ACSAC ’05, 2005, pp. 303–311.

[22] G. Buehrer, B. W. Weide, and P. A. G. Sivilotti, “Using parse tree validation to prevent sql injection attacks,” in Proceedings of the 5th international workshop on Software engineering and middleware, ser. SEM ’05, 2005, pp. 106–113.

[23] Z. Su and G. Wassermann, “The essence of command injection attacks in web applications,” SIGPLAN Not., vol. 41, no. 1, pp. 372–382, Jan. 2006.