[volume 1 Issue 2 March 2013]

Page No. 61-68 ISSN :2320-7167

[International Journal Of Mathematics And Computer Research]

Survey on Preventing Input Validation Vulnerabilities in Web Applications through Automated Type Analysis

Mounika B, A. Krishna Chaitanya

(Mtech Computer Science and Engineering in Vardhaman College of Engineering)

mouni2089@gmail.com

(Associate professor in Vardhaman college of Engineering, Hyderabad)

Chaituit2004@gmail.com

ABSTRACT- *In the last twenty years, web applications have grown from simple, static pages to complex, full-fledged dynamic applications. simple web applications today may accept and process hundreds of different HTTP parameters to be able to provide users with interactive services. Unfortunately, web applications are also frequently targeted by attackers, and critical vulnerabilities such as Front-end and Back-end are still common. Much effort has been taken from the past few years to reduce these vulnerabilities. The current technique focused on sanitization is not able to prevent new forms of input validation vulnerabilities such as HTTP parameters pollutions and are runtime overhead. In this paper a technique for preventing these front end and back end vulnerabilities is developed which is based on automated data type detection of input parameters. This novel technique is referred to as IPAAS which automatically and transparently augments.*

Keywords- *Input Validation, Pattern matching, Sanitization, SQL injection attacks, Type learning, XSS Attacks.*

1. INTRODUCTION

Web applications have become targets for attackers due to their wide usage by users. Web applications vulnerabilities include so many attacks, among them the front end and back end vulnerabilities remain most attention in the research and is focused on reducing these vulnerabilities. These vulnerabilities include XSS and SQL vulnerabilities. Both of these vulnerabilities manifest at a fundamental level as a failure to preserve the integrity of HTML documents and SQL queries. In XSS vulnerability it allows attacker to inject malicious HTML elements which includes malicious client side code. SQL injection attack allows the attacker to enter incorrectly filtered data that is embedded into SQL statement and the statement is modified in such a way that it violates the web application data integrity.

One approach for preventing this vulnerability is automated sanitization of malicious input. This approach uses filters or sanitizers that are applied to user data which prevents injection of dangerous elements into HTML documents or SQL queries. But it is difficult to achieve complete and correct sanitizer coverage.

Two methods for reducing HTML and SQL vulnerabilities are

1) Output Sanitization and
2) Input Validation.

1.1 Output Sanitization

Output sanitization is a automated, robust, and context aware to web browsers and databases and is a best solution for preventing front end and back end vulnerabilities. In this technique the sanitizers are automatically applied to untrusted data before its use in document or query construction. If an output sanitizer decides that the data computed from untrusted data is safe, then it is actually safe to give it to the user or submit it to the database.

Unfortunately, output sanitization is not a best solution. In order to achieve correctness and complete coverage of all locations where untrusted data is used to build HTML documents and SQL queries, it is necessary to construct an abstract representation of these objects in order to track output contexts. But, this requires the direct specification of documents and queries in a domain-specific language or else the use of a language amenable to precise static analysis. While new web applications have the option of using a secure-by-construction development framework or templating language, legacy web applications do not have this luxury. furthermore, many web

developers continue to use insecure languages and frameworks for new applications.

1.2 Input Validation

In abstract input validation is the process of assigning semantic meaning to unstructred and untrusted inputs to an application and ensuring that these inputs respect a set of constrints describing a well formed inputs. This involves checking the inputs that are applied to web application against the specifification of legitimate values. The main goal of the input validation is finding out the program correctness rather than preventing the attacks.

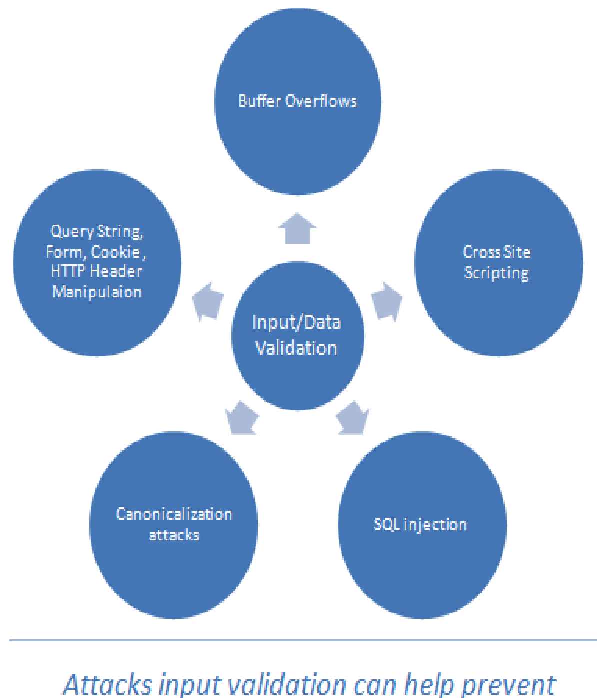

Fig 1: attacks that are prevented by input validation

Input validation mainly focuses on validating the untrusted input and provides less assurance of protecting the database vulnerabilities. But this approach still fails to validate the malicious input. Despite of these drawbacks, input validation has many advantages. First, the application of input validators has prevented vulnerabilities to some extent. Second, it is simple to achieve complete coverage of untrusted input data. Fig shows different types of attacks that are prevented by input validation. Fig 1: attacks that are prevented by input validation.

2. LITERATURE SURVEY

There are many types of techniques that are proposed from past decade for reducing the vulnerabilities. Fonseca et al studied how software faults relate to web applications security. His results shown that only small set of software fault results in SQL and HTML vulnerability. They have demonstrated that the attacks are occurred because of missing function calls to sanitization or input validation functions.

Weinberger et al showed hoe effective web application frameworks are sanitizing user supplied input to defend applications against XSS attacks. In this work he compared the sanitization work against the features that popular web applications require. His work also focused on output sanitization as a mechanism for detecting and reducing XSS attacks.

Vulnerability detection approaches identify vulnerabilities through tracking the flow of user inputs to sensitive sinks. Static and dynamic analysis techniques are generally used for user input tracking. Static analysis-based techniques suffer from low precision as these techniques generally overestimate the tainted-ness of inputs. Dynamic analysis-based techniques such as model checking and concolic execution produce zero false positive in principal. But these techniques are generally complex and expensive.

One more technique in this approach is automating the task of generating test vectors for exercising input validation mechanisms. Sania is a system which is used in the development and debugging phases. It automatically generates SQL injection attacks based on the syntactic structure of queries found in the source code and tests a web application using the generated attacks. Saxena et al. proposed Kudzu, which combines symbolic execution with constraint solving techniques to generate test cases with the goal of finding client-side code injection vulnerabilities in JavaScript code. Halfond et al. used symbolic execution to infer web application interfaces to improve test coverage of web applications. Several papers propose techniques based on symbolic execution and string constraint solving to automatically generate XSS and SQL injection attacks and input generation for systematic testing of applications implemented in C.

Different techniques have been proposed for detecting and preventing front end and back end vulnerabilities. Most of the research being carried out, nowadays, pertaining to detection or prevention of SQL attacks. in general, this is divided into three categories (1) Runtime HTTP requests, (2) Design-time web application source code, and (3) Runtime dynamically generated SQL statements. In order to detect SQL attacks, some researchers employ only one type of data while some others employ two.

One more technique is an advanced query based multi-tier approach, for detecting SQL attacks and other web attacks, is being proposed, designed from the base, after realizing the complexities involved in these attacks. Knowledge based dynamic query generation techniques have been implemented in the designed application, which learns from the history of previously occurred attacks over the system. The system gains efficiency with time. It also maintains a list of common attacks which helps detect a larger number of attacks at an improved rate.

The easiest and the most effective client-side solution to the XSS problem for users are to deactivate Java Script in their browsers. Unfortunately, this solution is often not feasible because a large number of web sites use JavaScript for navigation and enhanced presentation of information.

Previous approaches to identifying SQLI and XSS vulnerabilities and preventing exploits include defensive coding, static analysis, dynamic monitoring, and test generation. Each of these approaches has its own merits, but also offers opportunities for improvement. Defensive coding is error-prone and requires rewriting existing software to use safe libraries. Static analysis tools can produce false warnings and do not create concrete examples of inputs that exploit the vulnerabilities. Dynamic monitoring tools incur runtime overhead on the running application. Black-box test generation does not take advantage of the application's internals, while previous white-box techniques have not been shown to discover unknown vulnerabilities

A Multi-Agent System has been explored for the automated scanning of websites to detect the presence of XSS vulnerabilities exploitable by a stored XSS attack. It works by finding the input points of the application susceptible of being vulnerable to a stored–XSS attack then Injecting selected attack vectors at the previously detected points. Finally it checks the web application for the injected scripts in order to verify the success of the attack. It is not capable runtime detection and prevention of attack also it can be used for attack detection only, i.e. no mechanism for prevention.

Preventative techniques for mitigating XSS and SQL injection vulnerabilities focus either on client-side mechanisms, or on server-side mechanisms. Client-side or browserbased mechanisms such as Noxes, Noncespaces, or DSI make changes to the browser infrastructure aiming to prevent the execution of injected scripts. Each of these approaches requires that end-users upgrade their browsers or install additional software; unfortunately, many users do not regularly upgrade their systems.

Wassermann and Su proposed a system that checks at runtime the syntactic structure of a query for a tautology. AMNESIA checks the syntactic structure of queries at runtime against a model that is obtained through static analysis. XSSDS is a system that aims to detect XSS attacks by comparing HTTP requests and responses. While these systems focus on preventing injection attacks by checking the integrity of queries or documents, we focus on input validation. Recent work has focused on automatically discovering parameter injection and parameter tampering vulnerabilities.

3. BACKGROUND

This section describes SQLI and XSS Web-application vulnerabilities and illustrates attacks that exploit them.

3.1 SQL Injection.

A SQLI vulnerability results from the application's use of user input in constructing database statements. The attacker invokes the application, passing as an input a (partial) SQL statement, which the application executes. This permits the attacker to get unauthorized access to, or to damage, the data stored in a database.

Fig 2: SQL injection attack

The above fig 2 shows SQL injection attack. Attackers can use existing vulnerabilities in the web server logic to inject the data or string content that contains the exploits and then use the web server to relay these exploits to attack the back-end database. To prevent this attack, applications need to sanitize input values that are used in constructing SQL statements, or else reject potentially dangerous inputs

3.2 XSS(Cross Site Scripting)

The problem with the current JavaScript security mechanisms is that scripts may be confined by the sand-boxing mechanisms and conform to the same-origin policy, but still violate the security of a system. This can be achieved when a user is lured into downloading malicious JavaScript code (previously created by an attacker) from a trusted web site. Such an exploitation technique is called a cross-site scripting.

For example, consider the case of a user who accesses the popular trusted.com web site to perform sensitive operations (e.g., on-line banking). The web-based application on trusted.com uses a cookie to store sensitive session information in the user's browser. Note that, because of the same origin policy, this cookie is accessible only to JavaScript code downloaded from a trusted.com web server. However, the user may be also browsing a malicious web site, say www.evil.com, the Fig 3 shows this Scenario.

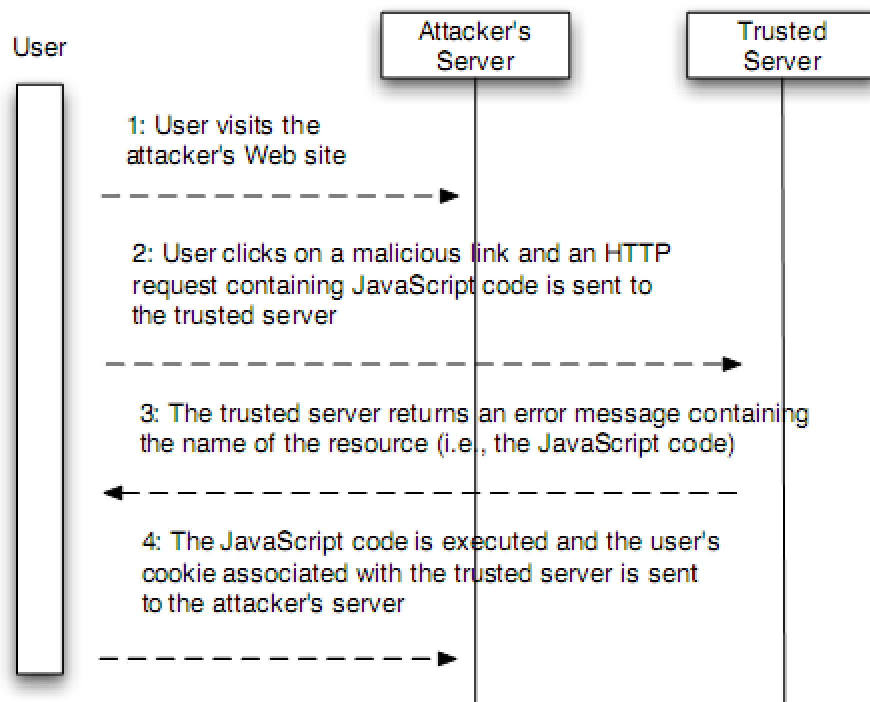

Fig 3: Cross Site Scripting Scenario

XSS is of two types. They are.

3.2.1 First-order XSS.

A first-order XSS (also known as Type 1, or reflected, XSS) vulnerability results from the application inserting part of the user's input in the

next HTML page that it renders. The attacker uses social engineering to convince a victim to click on a (disguised) URL that contains malicious HTML/JavaScript code. The user's browser then displays HTML and executes JavaScript that was part of the attacker-crafted malicious URL. This can result in stealing of browser cookies and other sensitive user data. To prevent first-order XSS attacks, users need to check link anchors before clicking on them, and applications need to reject or modify input values that may contain script code.

3.2.3 Second-order XSS

A second-order XSS (also known as persistent, stored, or Type 2 XSS) vulnerability results from the application storing (part of) the attacker's input in a database, and then later inserting it in an HTML page that is displayed to multiple victim users (e.g., in an online bulletin board application). It is harder to prevent second-order XSS than first-order XSS, because applications need to reject or sanitize input values that may contain script code and are displayed in HTML output, and need to use different techniques to reject or sanitize input values that may contain SQL code and are used in database commands.

4. PROPOSED WORK

This paper proposed an approach called IPAAS that automatically integrates robust, input parameter validation into web application. IPAAS refers to Input PArameter Analysis System which is one of the best solution for detecting and reducing XSS and SQL injection attacks.

IPAAS mainly performs three functions. They are

(i) extracting the parameters for a web application
(ii) learning type for each parameter by applying a combination of machine learning over training data and a simple static analysis of the application and
(iii) automatically appliying robust validators for each parameter to the web application with respect to the inferred types

4.1 IPAAS Architecture

These three main functions of IPAAS are decomposed into three phases in the prevention of vulnerabilities. The three phases are named as

1) parameter extraction phase,
2) analysis and training phase, and
3) runtime validation.

The architecture of IPAAS including three phases listed above are shown in the following fig 4 representing the IPAAS architecture.

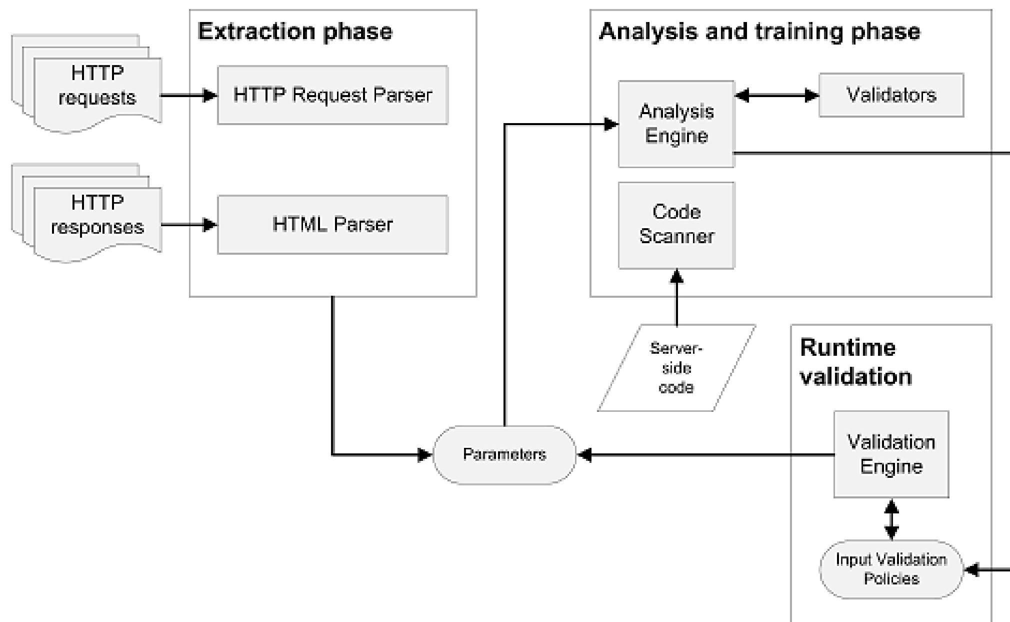

Fig 4: IPAAS architecture

As shown in the Fig 2 an IPAAS consists of a proxy server that intercepts HTTP messages generated during application testing and is given to extraction phase. Input parameters are classified during an analysis phase according to one of a set of possible types. After sufficient data has been observed, IPAAS derives an input validation policy based on the types learned for each application input parameter. This policy is automatically enforced in the third phase at runtime by rewriting the application.

4.1.1 Parameter Extraction

Parameter extraction refers to data collection step. During testing a proxy server intercepts the HTTP requests that are exchanged between web client and application. For each HTTP requests all the observed parameters in training phase are parsed into key-value pairs and are stored in database. Each HTTP response containing HTML document is processed by HTML parser and extracts links and forms during the testing. The key-value pairs that are extracted in the case of requests are also generated in the case of links for each string that are extracted.

4.1.2 Parameter analysis and training

The each parameter that is extracted during the first phase is labled with a data type based on the key-values observed on that parameter. This process is performed by applying set of validators to test inputs. Validators are functions that check whether a value meets particular set of constraints or not. IPAAS applies set of validators, which checks that an input belongs to one set of types.

IPAAS determines the type of parameter in two sub phases. In the first phase, learning the types based on the values that are recorded for each parameter. Next in the second sub phase learning types are augmented based on the values extracted from HTML documents.

In the first sub phase i.e learning, the analysis begins by retrieving all the resource paths that were observed during application testing. For each path, the it retrieves the unique set of parameters and the complete set of values for each of those parameters observed during the extraction phase. Each parameter is assigned an integer score vector of length equal to the number of possible validators.

The actual type learning phase beginss by passing each value of a parameter to every possible type validator. If a validator accepts a value, the corresponding entry in that parameter's score vector is incremented by one. In the case that no validator accepts a value, then the analysis engine assigns the *free-text* type to the parameter and stops processing its values. After all values for a parameter have been processed, then the score vector is used to select a type and a validator. generally, the type with the highest score in the vector is selected. If there is a tie, then the most restrictive type is assigned; this corresponds to the ordering given in Table I.

<table>
<tr><th>Type</th><th>Validator</th></tr>
<tr><td>boolean</td><td>(0|1)|(true|false)|(yes|no)</td></tr>
<tr><td>integer</td><td>(+|-)?[0-9]+</td></tr>
<tr><td>float</td><td>(+|-)?[0-9]+(\+.[0-9]+)?</td></tr>
<tr><td>URL</td><td>RFC 2396, RFC 2732</td></tr>
<tr><td>token</td><td>static set of string literals</td></tr>
<tr><td>word</td><td>[0-9a-zA-Z@_-:]+</td></tr>
<tr><td>words</td><td>[0-9a-zA-Z@_-: \r\n\t]+</td></tr>
<tr><td>free-text</td><td>none</td></tr>
</table>

Table I
IPAAS TYPES AND THEIR VALIDATORS.

The second sub phase that is augmenting the values uses the information that is extracted from the HTML documents. In this phase initially a check is performed to determine whether the parameter is associated with HTML *textarea* element or not. If it is associated with that the parameter is assigned the *free text* type. Otherwise it checks whether the parameter belongs to input element that is one of the check box or radio button or select list. In this case the observed set of values are assigned to parameter.

The analysis engine then derives the input validation policies for each parameter. For each resource the path is linked to corresponding application source file. Then the resource parameters are grouped by input parameter and are serialized as a part of input validation policy. Finally it is written to disk.

The learning sub phase described above is augmented by static analysis. The static analysis is performed in order to determine or find parameters and application resources that are missed during the training phase. Static analysis tools check the security of web applications often employ data flow analysis to track the use of program inputs. The goal of these systems is to identify program paths between the location where an input enters the application and a location where this input is used. Once such a program path is identified, the tool checks whether the programmer has properly sanitized the input on its way from the source to the sensitive sink(location where input is used).

4.1.3 Runtime enforcement

After the completion of first two phases a set of input validation policies for each input parameter is achieved. This runtime enforcement occurs during the deployment. During runtime IPAAS intercepts the incoming requests and checks each against the validation policy for that parameter. If the parameters that are present in the HTTP request does not meet the validation policy then IPAAS drops that specific request. If that request meets the input validation policy then it continues its execution.

There may be some situations where a specific HTTP request may contain some parameters that where not observed during the learning sub phase or during the static analysis. At this time the request may either simply dropped or the request may be accepted and the parameter is marked as valid. It can be used in further learning phases to refresh application input validation policies.

4.2 IPAAS limitations

The implementation of IPAAS has some limitations. They are as follows.

1) Type learning can fail when HTTP requests contain custom query strings. In this case the parameter extraction phase may not be able to assign key-value pairs for parameters.

2) The implementation of static analysis is complex process and is rudimentary.

5. CONCLUSION

Web applications have become important part of daily lives of millions of users for their personal and corporate work. Unfortunately due to their wider usage by people for business transaction these web applications are highly prone to different types attacks. Among these attacks the XSS attack and SQL attacks are paid most attention by the researchers due to their vulnerability. Current techniques to prevent these attacks mainly focus on output sanitization which is overhead and has lack of precision.

In this paper an alternative to output sanitization that is automated input validation (IPAAS) is presented for preventing XSS and SQL attacks. This approach improves the secure development of web applications by performing parameter extraction and type learning methods and by applying robust input validators at runtime.

REFERENCES

[1] Theodoor Scholte, William Robertson, Davide Balzarotti, Engin Kirda: Preventing Input Validation Vulnerabilities in Web Applications through Automated Type Analysis, *IEEE transaction on computer softeware and application conference,* july 2012.

[2] Meixing Le, AngelosStavrou, Brent ByungHoon Kang," Double Guard: Detecting Intrusions in Multitier Web Applications", *IEEE Transactions on dependable and secure computing, vol. 9, no. 4,* July/august 2012.

[3] Arisholma, E., Briand, L. C., and Johannessen, E. B. 2010. A systematic and comprehensive investigation of methods to build and evaluate fault prediction models. *Journal of Systems and Software*, 83, 1, 217.

[4] Jovanovic, N., Kruegel, C., and Kirda, E. 2006. Pixy: a static analysis tool for detecting web application vulnerabilities. In *Proceedings of the IEEE Symposium on Security and Privacy*. 258-263.

[5] Kieun, A., Guo, P. J., Jayaraman, K., and Ernst, M. D. 2009. Automatic creation of SQL injection and cross-site scripting attacks. In *Proceedings of the 31st International Conference on Software Engineering*. 199-209.

[6] Martin, M. and Lam, M. S. 2008. Automatic generation of XSS and SQL injection attacks with goal-directed model checking. In *Proceedings of the 17th USENIX Security Symposium*. 31-43.

[7] Shar, L. K. and Tan, H. B. K. 2012. Mining input sanitization patterns for predicting SQLI and XSS vulnerabilities. In *Proceedings of the 34th International Conference on Software Engineering*. 1293-1296.
[8] Weinberger, P. Saxena, D. Akhawe, M. Finifter, R. Shin, and D. Song. An Empirical Analysis of XSS Sanitization in Web Application Frameworks. Technical report, UC Berkeley, 2011.
[9] Y. Kosuga, k. Kono, m. Hanaoka, m. Hishiyama, and y. Takahama. Sania: syntactic and semantic analysis for automated testing against sql injection. In acsac, pages 107–117. Ieee computer society, 2007