International Journal of Modern Engineering Research (IJMER)
www.ijmer.com Vol.1, Issue1, pp-127-133 ISSN: 2249-6645

STATIC ANALYSIS TOOL FOR DETECTING WEB APPLICATION VULNERABILITIES

L. VENKATA SATYANARAYANA,
2/2 M.TECH CSE, DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING,
ADITYA INSTITUTE OF TECHNOLOGY AND MANAGEMENT, TEKKALI,
ANDHRA PRADESH, INDIA.

M.V.B.CHANDRA SEKHAR,
ASSOCIATE PROFESSOR,
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING,
ADITYA INSTITUTE OF TECHNOLOGY AND MANAGEMENT, TEKKALI,
ANDHRA PRADESH, INDIA.

Abstract— **Over the past few years, injection vulnerabilities have become the primary target for remote exploits. SQL injection, command injection, and cross-site scripting are some of the popular attacks that exploit these vulnerabilities. Many web applications written in ASP suffer from injection vulnerabilities, and static analysis makes it possible to track down these vulnerabilities before they are exposed on the web. In this paper we propose a new technique to detect XSS attacks and SQL injection vulnerabilities based on taint analysis, It tracks various kinds of external input, tags taint types, constructing control flow graph is constructed based on the use of data flow analysis of the relevant information, taint data propagate to various kinds of vulnerability functions, and detect the XSS or SQL Injection vulnerability in web application's source code. Results show the benefits of the tool in identifying potential security vulnerabilities.**

***Index Terms*— W3; SQL Injection; XSS.**

I. INTRODUCTION

Web applications are growing more and more popular as both the availability and speed of the internet increase. Many web servers nowadays are equipped with some sort of scripting environment for the deployment of dynamic web applications. However, most public web hosting services do not enforce any kind of quality assurance on the applications that they run, which can leave a web server open to attacks from the outside[1].

Poorly written web applications are highly vulnerable to attacks because of their easy accessibility on the internet. One careless line of program code could potentially bring down an entire computer network. The reasons for the increase of threats in Web application [2,3] could be divided into two main parts: On one hand, software are developing in too large a scale together with the expanding complexity and extensibility of software while flaws still exist in their source codes; On the other hand, This is probably due to ease of detection and exploitation of web vulnerabilities, combined with the proliferation of low-grade software applications. At the moment, the overflow of Web application programs and Plug-in lead to the result that much of the code is alpha or beta, written by inexperienced programmers with easy-to learn languages such as ASP (Active Server Pages).

Such software is often rife with easy-to-find vulnerabilities, even malicious hideaway back door. For instance, injection threats exist in the early version of Eweb editor and the fckeditor. Security problem in software refers to threats incurred because of the flaws in software research, designation, programming, testing and implementation [4]. They are taken use of by attackers so as to change the function of the software from original intention of the software designers. As a typical Web application attacks, the most popular is the SQL injection and XSS, because the most basic data manipulations for these vulnerabilities are very simple to perform, e.g. '' for SQL injection and '<script> alert('hi') </script>' for XSS [5]. This makes it easy for beginning researchers to quickly test large amounts of software.

II. RELATED WORK

Static analysis security tools attempt to find security vulnerabilities without executing the software by scanning the source code for known potentially security-compromising functions[6]. They then perform analyses to try to determine if, indeed, a function call could be maliciously attacked. These

tools cannot guarantee to find all security vulnerabilities in a program and often report many false positives (those potential vulnerabilities reported by a tool which are not actual vulnerabilities).

In 2005 and 2006, XSS was number 1, and SQL injection was number 2 [7]. In 2009 around domestic college entrance examination online enrollment, the domestic university's admission websites suffer the threat by "Trojan horse" attack, when the user visits the page that attacked by XSS, the user's sensitive data are stolen. These explained that strengthens the Web application security the work to be urgent. ITS4 was one of the first available static security analysis tools to search C source code looking for potentially dangerous function calls [8]. ITS4 performs limited analysis to determine how risky a function call is and, for every problem reported provides suggestions how to mitigate the security vulnerability. RATS is similar to ITS4 in its approach but performs additional analysis to attempt to reduce the number of false positives reported [9]. Unlike ITS4, however, RATS performs analysis to discover Time Of Check, Time Of Use race conditions.

Splint (Secure Programming Lint) is an improvement over another static security analysis tool, Lint [10] that does additional analysis on potential security vulnerabilities beyond both ITS4 and RATS. Other tools perform different analysis techniques to try and discover a different type of security vulnerability or eliminate a different type of false positives. For example, BOON [11] performs analysis focusing primarily on the detection of the buffer overflow security vulnerability whereas Flaw Finder [12] uses a vulnerability database as does ITS4 and RATS. Thus, different tools often produce different sets of results.

III. System Architecture

A web application, as the name implies, is a computer application that is accessed through a web-based user interface. This is typically implemented through a client-server setup, with the server running an HTTP server software package (such as Apache or Microsoft IIS) capable of generating dynamic web pages, while the client communicates with the server through a web browser (such as Microsoft Internet Explorer or Mozilla Firefox). The working of such a web application is roughly sketched in Figure 1.

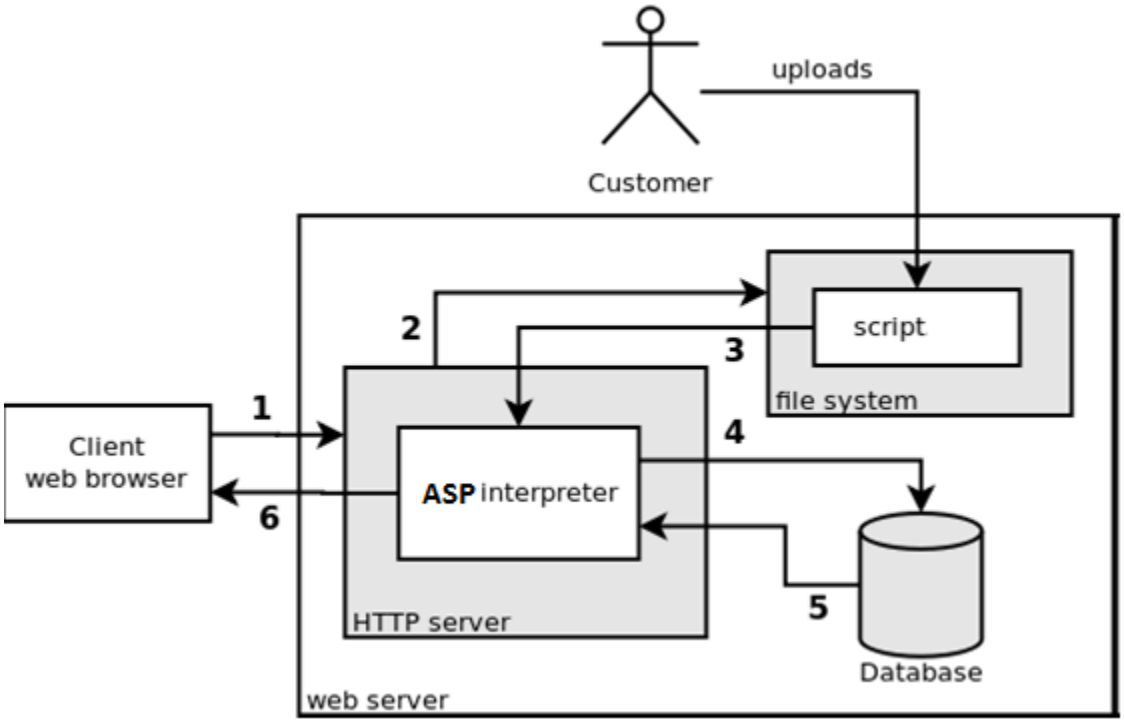


Fig 1 System architecture

Whenever the client interacts with the server, communication takes place in the form of HTTP requests. The client sends a request (typically a GET or POST request) to the server, along with a series of parameters (step 1 in Fig. 1). The HTTP server recognizes that this is a request for a dynamic web page, in this case a page that is to be generated by a ASP script. It fetches the corresponding ASP script from the web server's file system (step 2) and sends it off to be processed by the integrated ASP interpreter (step 3). The ASP interpreter then executes the ASP script, making use of external

resources whenever necessary. External resources can for example be a database (steps 4 and 5), but also the file system or an API exposed by the operating system. The executed ASP script typically produces output in the form of an HTML page, which is sent back to the client and displayed in the web browser (step 6).

The most prevalent and most exploited vulnerabilities in web applications are cross-site scripting (XSS) and SQL injection (SQLI). According to a top ten composed by the Open Web Application Security Project (OWASP), XSS and SQLI were the top two most serious web application security flaws for both 2007 and 2010 [13]. According to this list, the top five of security flaws has not changed over the past three years.

Vulnerabilities occurring in real-world applications are much more complicated and subtle than the examples appearing in this section. In many cases, user data is utilized in applications in ways that appear to be safe on the surface. However, due to complex interactions that is difficult to predict, unsafe data can still slip through in specific edge cases. Such vulnerabilities are hard to spot, even when using professional coding standards, careful code reviewing, and extensive testing.

A. Cross-Site Scripting

Cross-site scripting (XSS) is a type of vulnerability that allows attackers to inject unauthorized code into a web page, which is interpreted and executed by the user's web browser. XSS has been the number one web application vulnerability for many years, and according to White Hat Security, has been responsible for 66% of all website attacks in 2009 [14].

Web pages can include dynamic code written in JavaScript to allow the web page's content to be altered within the web browser as the user interacts with it. Normally, a web browser will only execute JavaScript code that originates from the same domain as the web page itself, and that code is only executed within a self-contained sandbox environment. This is the so-called Same Origin Policy [15]. This policy prevents attackers from making web browsers execute un-trusted code from an arbitrary location.

```
1  <html>
2  <body>
3   <%
4   ‘Re t r i e v e the user ' s name from a
5  form
6   name = Request.Response('name' )
7   // Pr int the user ' s name back to them
8   Response.write "Hello there , $name!
9  How are you doing?"
   %>
   </body>
   </html>
```

Fig 2. Example of an XSS vulnerability

B. SQL Injection

SQL injection is a taint-style vulnerability, whereby an unsafe call to a database is abused to perform operations on the database that were not intended by the programmer. White Hat Security's report for 2009 lists SQL injection as responsible for 18% of all web attacks, but mentions that they are under-represented in this list because SQL injection flaws can be difficult to detect in scans [14].

SQL, or Structured Query Language, is a computer language specially designed to store and retrieve data in a database. Most database systems (e.g. MySQL, Oracle, Microsoft SQL Server, SQLite) use a dialect of SQL as a method to interact with the contents of the database. Scripting languages such as PHP o_er an interface for programmers to dynamically construct and execute SQL queries on a database from within their program.

It is common practice for programmers to construct dynamic database queries by means of string concatenation. Listing 2 shows an example of an SQL query that is constructed and executed from a PHP script using this method. The variable $username is copied directly from the input supplied by the user and is pasted into the SQL query without modifications. Although this query is constructed and executed on the server and users can not directly see the vulnerable code, it is possible through a few assumptions and educated guesses to find out that there is a database table called users and that the entered user name

The test method of common SQL Injection attack is the use “ ' ”, “ union ”, “ --; ” and so on key words, in test dynamic SQL sentence in program whether to exist injection vulnerability. For example, consider the login page of a web application that expects a user-name and the corresponding password. When the credentials are submitted, they are inserted within a query template such as the following:

“select * from admin where username =’” + request.form(“username”) + “‘ and Password = ‘” + request.form(“passwd”)+“’”

Instead of a valid user name, the malicious user sets the “username” variable to the string:’ or 1=1; - -’, causing the Vbscript to submit the following SQL query to the database:

“select * from admin where username = ‘’ or 1=1 ; - -‘‘ and Password = ‘ any_passwd’ ”

Therefore, the password value is irrelevant and may be set to any character string. The result set of the query contains at least one record, since the “where” clause evaluates to true. If the application

identifies a valid user by testing whether the result set is non-empty, the attacker can bypass the security check.

IV. Static Analysis Methods

Detection software security vulnerabilities are mainly dynamic analysis, formal method validation and static analysis. Static analysis is divided as type inference, data flow analysis and constraints analysis [1,16,17].

A. Type Inference

Type system of programming language concludes type definition and rules for type equivalence, type inclusiveness and type dedication. Type dedication is to derive the types of variables and methods within a program automatically so as to determine whether or not their visit meet these type rules. This kind of dedication could be used to examine the bug in types and conduct necessary type transmission with proper operations. It boasts the characteristics of simplicity and high efficiency which makes it perfect for quick detection of security threats in software. Now it is mainly applied in detection of format string vulnerability, OS kernel vulnerable pointer use.

B. Data-Flow Analysis

Data-flow analysis is used in the process programming, which collect semantic information from programs and then define and use the variables with algebraic approach. It is used in program optimization, program validation, debugging, parallel, Vectorization and serial program environment. Its realization makes use of the pair "variable definition-quoting".

C. Constraint Analysis

Constraint analysis divides program analysis into constraint generation and constraint solution. The former constructs variable type with constraint generation rules or analyses constraint system among statuses. While the later solve such constraint systems. Constraint system is comprised of equation constraint, set constraint and incorporate constraint. In the first kind, only equation exists between constraint objects. Set constraint takes program variables as a set of values, whose evaluation is regarded as conclusion relation between set expressions? While the last constraint concludes equation constraint part and set constraint part.

D. The Comparison among three Main methods

The three main methods mentioned are all explain the abstract semantics of programs and construct mathematic models based on the program property, with which they determine the property of the program. In comparison, constraint analysis boasts the greatest ability in detection while the lowest speed of that, which makes it fit for security examination of software, data-flow analysis has relatively high speed and remarkable ability of detection which is appropriate in static analysis which should take control flow information and requires only simple operation among variable properties; when it comes to type dedication, it has the poorest ability and the fastest speed in examination and suits for security test in finite property domain and unrelated control flow.

The website that issued news or BBS forum is one kind of web application. Analyzing the logic characteristics of its services, it is not complex to find out that the process of dataflow could be give a summaries: data input (parameters) → data service processing (web server)→result output (HTML).

Based on principles of XSS threats and SQL injection threats, we could see vulnerability is mainly generated from the sanitation process of input data. Thus sanitation process of all the input data would neglect such vulnerability of taint data (outside client input data). In this paper, we would examine the code with data flow analysis; the function framework of system is shown in fig 2.

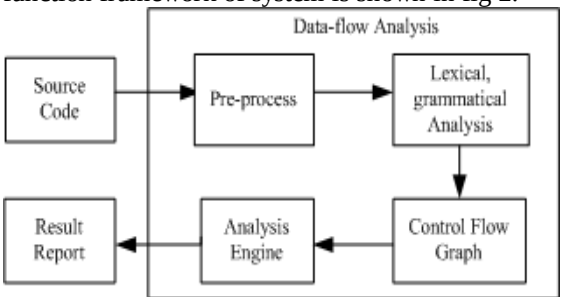
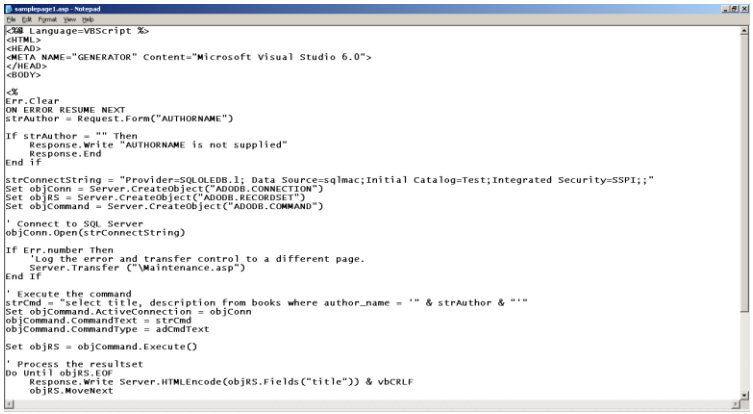


Fig 3. Function Framework of Code Review System

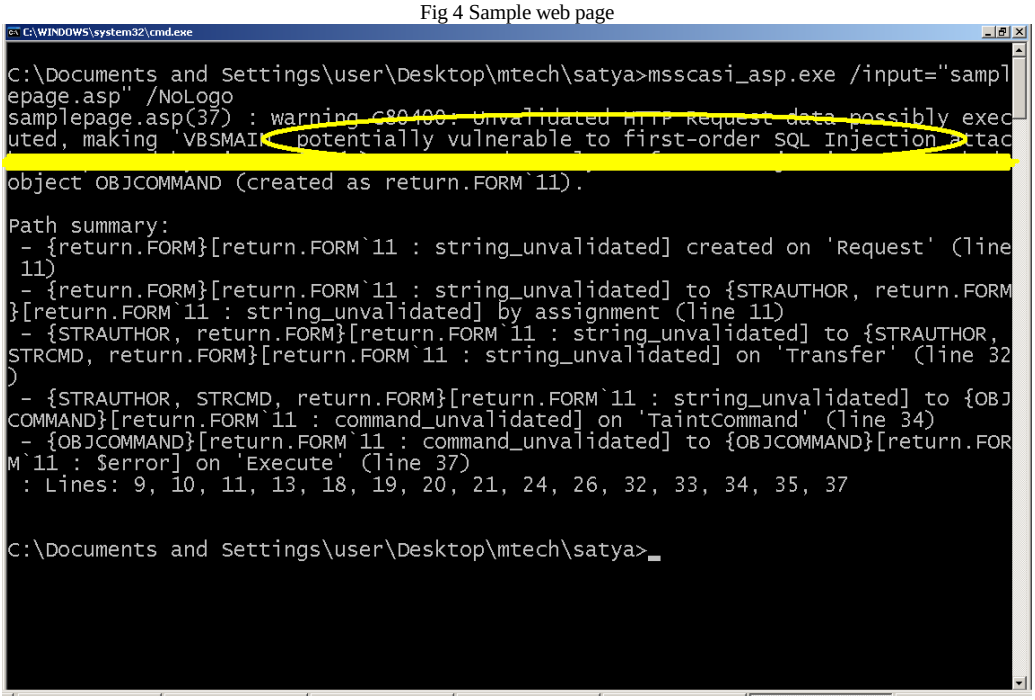
V. Results

To test the validity of our approach, we select three open source program written by ASP. These software are commonly used a source codes of tools in network application layer and are representative. Test environment: Intel XEON CPU: 3.0GHz, 1GB cache, Windows 2003 Server , IIS6.0. Then we conduct penetrating examination in Acunetix Web with the results shown in Fig 4. We develop a tools named as ASPWC. The number of XSS reported by Acunetix Web tools is success of the XSS attack test. Possible SQL Injection vulnerability through data that is read from the Request object without any input validation. These warnings are very likely bugs that must be fixed. Sample Web Page is shown in Fig 4



```
<%@ Language=VBScript %>
<HTML>
<HEAD>
<META NAME="GENERATOR" Content="Microsoft Visual Studio 6.0">
</HEAD>
<BODY>

<%
Err.Clear
ON ERROR RESUME NEXT
strAuthor = Request.Form("AUTHORNAME")

If strAuthor = "" Then
    Response.Write "AUTHORNAME is not supplied"
    Response.End
End If

strConnectString = "Provider=SQLOLEDB.1; Data Source=sqlmac;Initial Catalog=Test;Integrated Security=SSPI;;"
Set objConn = Server.CreateObject("ADODB.CONNECTION")
Set objRS = Server.CreateObject("ADODB.RECORDSET")
Set objCommand = Server.CreateObject("ADODB.COMMAND")

' Connect to SQL Server
objConn.Open(strConnectString)

If Err.number Then
    'Log the error and transfer control to a different page.
    Server.Transfer ("\Maintenance.asp")
End If

' Execute the command
strCmd = "select title, description from books where author_name = '" & strAuthor & "'"
Set objCommand.ActiveConnection = objConn
objCommand.CommandText = strCmd
objCommand.CommandType = adCmdText

Set objRS = objCommand.Execute()

' Process the resultset
Do Until objRS.EOF
    Response.Write Server.HTMLEncode(objRS.Fields("title")) & vbCRLF
    objRS.MoveNext
```

Fig 4 Sample web page



```
C:\WINDOWS\system32\cmd.exe

C:\Documents and Settings\user\Desktop\mtech\satya>msscasi_asp.exe /input="sampl
epage.asp" /NoLogo
samplepage.asp(37) : warning c80400: Unvalidated HTTP Request data possibly exec
uted, making 'VBSMAIN' potentially vulnerable to first-order SQL Injection attac
k
object OBJCOMMAND (created as return.FORM`11).

Path summary:
 - {return.FORM}[return.FORM`11 : string_unvalidated] created on 'Request' (line
 11)
 - {return.FORM}[return.FORM`11 : string_unvalidated] to {STRAUTHOR, return.FORM
}[return.FORM`11 : string_unvalidated] by assignment (line 11)
 - {STRAUTHOR, return.FORM}[return.FORM`11 : string_unvalidated] to {STRAUTHOR,
STRCMD, return.FORM}[return.FORM`11 : string_unvalidated] on 'Transfer' (line 32
)
 - {STRAUTHOR, STRCMD, return.FORM}[return.FORM`11 : string_unvalidated] to {OBJ
COMMAND}[return.FORM`11 : command_unvalidated] on 'TaintCommand' (line 34)
 - {OBJCOMMAND}[return.FORM`11 : command_unvalidated] to {OBJCOMMAND}[return.FOR
M`11 : $error] on 'Execute' (line 37)
 : Lines: 9, 10, 11, 13, 18, 19, 20, 21, 24, 26, 32, 33, 34, 35, 37


C:\Documents and Settings\user\Desktop\mtech\satya>
```

Fig 5 Output from our tool

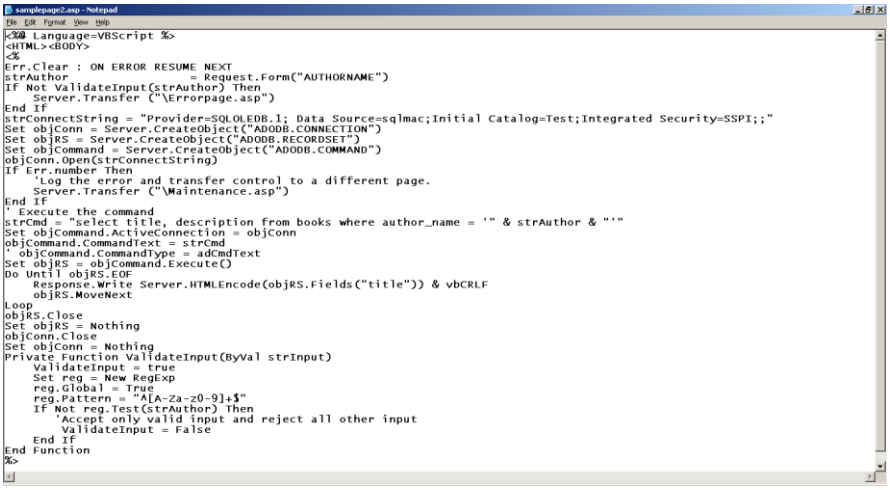
Comments: strAuthor is assigned a value from Request.QueryString("AUTHORNAME") on line number 11 and is eventually used in the construction of dynamic SQL and executed through OBJCOMMAND on line number 37

Use parameterized SQL query to mitigate the SQL Injection identified by the tool

Table 1 Sample Code

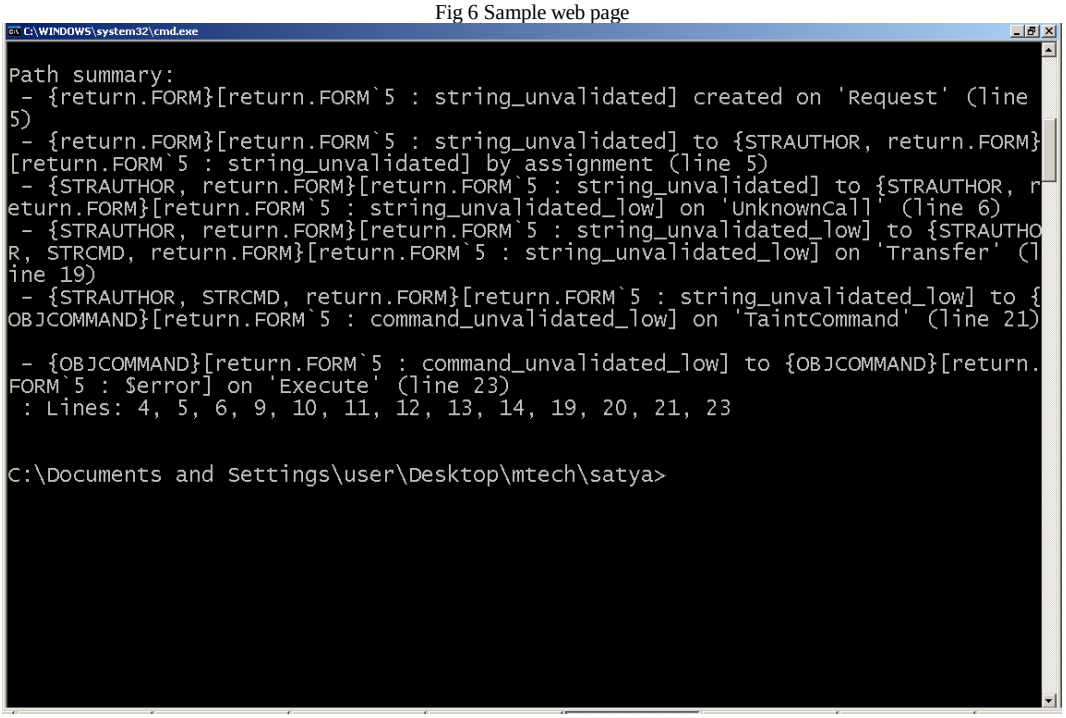
```
' Execute the command
strCmd = "select title, description from books where author_name = ?"
Set objCommand.ActiveConnection = objConn
objCommand.CommandText = strCmd
objCommand.CommandType = adCmdText
Set param1 = objCommand.CreateParameter ("author", adWChar, adParamInput, 50)
param1.value = strAuthor
objCommand.Parameters.Append param1
```

Possible SQL Injection vulnerability through data that is read from the Request object where the input is passed through some unknown function calls that might perform data validation. If there is no data validation done inside the function call, then these are likely bugs else these are likely false positives



```
samplepage2.asp - Notepad
File  Edit  Format  View  Help
<%@ Language=VBScript %>
<HTML><BODY>
<%
Err.Clear : ON ERROR RESUME NEXT
strAuthor           = Request.Form("AUTHORNAME")
If Not ValidateInput(strAuthor) Then
    Server.Transfer ("\Errorpage.asp")
End If
strConnectString = "Provider=SQLOLEDB.1; Data Source=sqlmac;Initial Catalog=Test;Integrated Security=SSPI;;"
Set objConn = Server.CreateObject("ADODB.CONNECTION")
Set objRS = Server.CreateObject("ADODB.RECORDSET")
Set objCommand = Server.CreateObject("ADODB.COMMAND")
objConn.Open(strConnectString)
If Err.number Then
    'Log the error and transfer control to a different page.
    Server.Transfer ("\Maintenance.asp")
End If
' Execute the command
strCmd = "select title, description from books where author_name = '" & strAuthor & "'"
Set objCommand.ActiveConnection = objConn
objCommand.CommandText = strCmd
' objCommand.CommandType = adCmdText
Set objRS = objCommand.Execute()
Do Until objRS.EOF
    Response.Write Server.HTMLEncode(objRS.Fields("title")) & vbCRLF
    objRS.MoveNext
Loop
objRS.Close
Set objRS = Nothing
objConn.Close
Set objConn = Nothing
Private Function ValidateInput(ByVal strInput)
    ValidateInput = true
    Set reg = New RegExp
    reg.Global = True
    reg.Pattern = "^[A-Za-z0-9]+$"
    If Not reg.Test(strAuthor) Then
       'Accept only valid input and reject all other input
       ValidateInput = False
    End If
End Function
%>
```

Fig 6 Sample web page



```
C:\WINDOWS\system32\cmd.exe

Path summary:
 - {return.FORM}[return.FORM`5 : string_unvalidated] created on 'Request' (line
5)
 - {return.FORM}[return.FORM`5 : string_unvalidated] to {STRAUTHOR, return.FORM}
[return.FORM`5 : string_unvalidated] by assignment (line 5)
 - {STRAUTHOR, return.FORM}[return.FORM`5 : string_unvalidated] to {STRAUTHOR, r
eturn.FORM}[return.FORM`5 : string_unvalidated_low] on 'UnknownCall' (line 6)
 - {STRAUTHOR, return.FORM}[return.FORM`5 : string_unvalidated_low] to {STRAUTHO
R, STRCMD, return.FORM}[return.FORM`5 : string_unvalidated_low] on 'Transfer' (l
ine 19)
 - {STRAUTHOR, STRCMD, return.FORM}[return.FORM`5 : string_unvalidated_low] to {
OBJCOMMAND}[return.FORM`5 : command_unvalidated_low] on 'TaintCommand' (line 21)

 - {OBJCOMMAND}[return.FORM`5 : command_unvalidated_low] to {OBJCOMMAND}[return.
FORM`5 : $error] on 'Execute' (line 23)
 : Lines: 4, 5, 6, 9, 10, 11, 12, 13, 14, 19, 20, 21, 23


C:\Documents and Settings\user\Desktop\mtech\satya>
```

Fig 7 Output from our tool

The experimental results analysis: experimental data can be seen from the above, based on control flow graph; data-flow analysis of the vulnerability detection algorithm can be effectively used to detect XSS, SQL injection vulnerabilities which exist in the source code. The blacklist is applied to check the input data in OK3W and Leichinews program. They have a common function to check all input string. The programs produce a certain false positive. Despite the weaknesses found in the report contains false positives, reporting the total number of articles, or less, from the relatively small number of reports of these find the true vulnerabilities have been greatly reduced the workload.

VI. Conclusions

Software needs to be secure in order to allow parties of different trust levels to interact with each other, without risking that un-trusted parties exploit critical parts of the software. Web applications are mostly susceptible to input validation vulnerabilities, also known as taint-style vulnerabilities, the most common of which are cross-site scripting and SQL injection. Because web applications are made to be easily accessible through the internet, they are particularly exposed to attacks.

Static analysis allows programmers to look for security problems in their source code, without the need to execute it. Static analysis tools will always produce false positives and false negatives, because they need to make trade-offs between accuracy and speed, and because program analysis is inherently un-decidable.

This tool has manifests its usefulness in examining the web sites based on ASP of the virtual host computer in a high school. Despite the fact that fault rate remains high now, we would use data-flow analysis and add rules to detect sensitive information so as to yield higher accuracy of examination in source code and lower false positive amount within an acceptable bound.

References

[1] Brian Chess, Jacob West (2007). “Secure Programming with Static Analysis". Addison-Wesley. ISBN 978-0321424778.

[2] Yao-Wen Huang, Shih-Kun Huang, Tsung-Po Lin, Chung-Hung Tsai. \Web Application Security Assessment by Fault Injection and Behavior Monitoring". In Proceedings of the Twelfth International Conference on World Wide Web (WWW2003), pages 148-159, May 21-25, Budapest, Hungary, 2003.

[3] Yao-Wen Huang, Fang Yu, Christian Hang, Chung-Hung Tsai, D.T. Lee, Sy-Yen Kuo. Verifying Web Applications Using Bounded Model Checking". In Proceedings of the 2004 International Conference on Dependable Systems and Networks (DSN2004), pages 199-208, Florence, Italy, Jun 28-Jul 1, 2004.
[4] Gray McGraw. Software Security. IEEE Secruity & Privacy[J]. March-April 2004,2(2):80-83
[5] V. Haldar, D. Chandra, and M. Franz. Dynamic Taint Propagation for Java. In Twenty-First Annual Computer Security Applications Conference (ACSAC), 2005.
[6] Yao-Wen Huang, Fang Yu, Christian Hang, Chung-Hung Tsai, D.T. Lee, Sy-Yen Kuo. Securing Web Application Code by Static Analysis and Runtime Protection". In Proceedings of the Thirteenth International World Wide Web Conference (WWW2004), pages 40-52,New York, May 17-22, 2004.
[7] Steve Christey, Robert A. Martin. Vulnerability Type Distributions in CVE.[EB/OL].2007:V1.1, (2007 -5-22).
[8] Viega, J., Bloch, J. T., Kohno, T. and McGraw, G. ITS4: A Static Vulnerability Scanner for C and C++ Code. In Proc. 16th Computer Security Applications Conferences pp. 257- 266, New Orleans, LA, 2000.
[9] RATS: Rough Auditing Tool for Security. http://www.securesoftware.com/resources/tools.html.
[10] Splint: Secure Programming Lint. http://www.splint.org/.
[11] Wagner, D. BOON: Buffer Overrun Detection. http://www.cs.berkeley.edu/~daw/boon/.
[12] FlawFinder Home Page. http://www.dwheeler.com/flawfinder/
[13] OWASP: Top Ten Project. http://www.owasp.org/index.php/OWASP_Top_Ten
[14] The WhiteHat Website Security Statistics Report - 8th Edition - Fall 2009.http://www.whitehatsec.com/home/resource/stats.html
[15] David Scott, Richard Sharp. “Abstracting application-level web security”. In Proceedings of the 11th international conference on World Wide Web (WWW2002), pages 396-407, Honolulu, Hawaii, May 17-22, 2002.
[16] Xia Yiming. Security Vulnerability Detection Study Based on Static Analysis[J]. Computer Science, 2006, 33(10): 279-283.
[17] Paul Biggar, David Gregg. “Static analysis of dynamic scripting languages". August, 2009