An Automated Mechanism for Secure Input Handling

Jin-Cherng Lin
[1]The Dept. of Computer Sci & Eng, Tatung University Taipei 10451, Taiwan
jclin@ttu.edu.tw

Jan-Min Chen[1, 2]
[2]The Dept. of Information Management,Yu Da College of Business Miaoli 36143, Taiwan
ydjames@ydu.edu.tw

***Abstract*—Numbers of the programs are poorly written, lacking even the most basic security procedures for handling input data from users. The input validation vulnerability can be detected by many tools but few tools can fix the flaws automatically. The security gateway can used to protect vulnerable Web sites immediately but it may induce false recognition through impersonal rule. By means of hybrid analysis and injection test, the vulnerable Web pages can be listed. Only those in vulnerable list need to be checked completely, so as to mitigate the system load and false positives effectively. Moreover an algorithm based on multilevel strategy is proposed producing individual sanitizing rule automatically for every vulnerable injection point. To meet the aim of automated validation, the enhanced crawler, the testing framework and the meta-programs are integrated into a sanitizing mechanism after we analyze the data flow. According to the experimental results, the mechanism has been proved to be a more effective scheme than those traditional input handling methods for mitigating malicious injection.**

***Index Terms*—Injection attack, Bypass testing, Input validation, Security gateway**

I. INTRODUCTION

Injection attack is dangerous and ubiquitous, contributing enormously to some of the most elaborate Web hacks. It takes advantage of a weakness in the Web application design to insert intentionally some extra characters in input data, causing the inline application to send bogus data to the database, producing an error – or worse, the wrong data. The purpose of the injection attack is typically to bypass or modify the originally intended functionality of the program. It is popular in system hacking or cracking to gain information, privilege escalation or unauthorized access to a system. Many application's security vulnerabilities result from generic injection problems. Examples of such vulnerabilities are SQL injection, Shell injection and Cross Site Scripting. Some sites attempt to protect themselves by filtering malicious injection, but a surprising number of applications having no validation mechanisms still exist. Many tools have been developed to detect injection vulnerabilities, but hackers are still successfully exploiting applications. A possible reason is that most tools just inspect application's vulnerabilities, but few tools can automatically fix these vulnerabilities. An automatic revised tool for anti-malicious injection had been developed and verified its efficiency [1], but it needed source code to insert input validation function. If we can't modify the source code of the components, such method can't be used.

In this paper, we present an automated mechanism consisting of an enhanced crawler, a security gateway and an event driven security testing framework. Our system is like Web application firewalls (WAF) designed to protect web sites from attack and doesn't require modification of application source code. The enhanced crawler can be used to crawl completely to find all injection points and their relational parameters and then it can generate a grabbing list for later test. Through testing framework, we can find some server-side applications having injection vulnerabilities and then we can create a sanitizing web site consisting of the meta-programs on the security gateway. By means of an adjustable validation function included in meta-programs, malicious input data can be sanitized to avoid injection attack. Specially, the proper rules used by validation function can be automatically chosen to reduce false positives and false negatives.

This paper makes the following three main contributions. First, it pays attention to problem issue of false recognition through impersonal rule, and finds out the solution of producing individual sanitizing rule automatically for every vulnerable injection point to reduce errors (false positives and false negatives). Second, instead of modifying source code it introduces meta-programs as a sanitizing agent for validating the malicious injection. Thirdly, it integrates hybrid analyzer (information gathering), testing framework (attack launching) and meta-programs (prevention performing) into an automated sanitizing mechanism.

The paper is organized as follows: Section II describes the reason why the common injection

vulnerabilities have occurred and discusses related works about analyzing and fixing vulnerability. Section III surveys a number of secure input handling methods and presents our handling proposal. In section IV, we describe the technical details of finding injection flaws and verifying the defense method. In Section V, we describe how to choose the proper filtering rules and how to sanitize malicious injection. The system implementation is shown in section VI and its efficiency is evaluated in section VII. The last section summarizes conclusions.

II. Related Work

The injection vulnerabilities occur whenever an application takes user supplied data without first validating or encoding that content. The weakness induced by programmer having no security sense may allow attackers to execute script in the victim's browser which can hijack user sessions or to inject data as part of a command for tricking the interpreter into executing unintended commands. The hostile code, data or file may be included or executed if the filenames or files from user can be accepted without properly validating. In addition, attackers can use this weakness to steal sensitive data or launch more serious attacks. There are many types of injection flaws: SQL, LDAP, XPath, XSLT, XSS, XML, Shell injection and many more [28]. XSS, SQL injection and shell injection are common in web applications.

A. Vulnerability Analysis

Testing Web applications for security defects is now considered a necessary part of the development process. However, none of the traditional methods of automated security testing provides comprehensive security coverage and accurate results for Web applications. While static analysis (source code analysis) is capable of finding insecure programming practices that have potentially rendered the code vulnerable to malicious attacks, it can be limited by the types of languages that have been utilized in crafting the Web application and can only find potential vulnerabilities rather than actionable results. Another method called black-box testing is adopted to analyze Web applications externally without the aid of source code. It can eliminate language dependency and the need for parsing the source or binary code into an analyzable form and if unable to crawl completely the testing site, can provide a false sense of security by producing numerous "false negatives".

Static analysis can be used to analyze code of client-side or server-side Web application, for instance, JavaScript, Perl, ASP or PHP . However, this technique fails to adequately consider the runtime behavior of Web applications and we must get source code. Y.W. Huang, S.K. Huang, T.P. Lin, and C.H. Tsai have developed a tool called WebSSARI (Web application Security Analysis and Runtime Inspection) [3,4]. The tool can be successfully used for automated Web application security assessment. Afterward, Jovanovic, N., Kruegel, C. and Kirda, E present Pixy, the first open source tool for statically detecting XSS vulnerabilities in PHP 4 code by means of data flow analysis[5].

The tool adopting black-box analysis can perform an assessment very quickly and produce a useful report identifying vulnerable sites. Y.W. Huang, S.K. Huang, T.P. Lin, and C.H. Tsai also have developed a remote, black-box security testing tool for Web applications is also called the Web Application Vulnerability and Error Scanner (WAVES) [6]. It can be used to analyze the design of Web application security assessment mechanisms to identify poor coding practices that render Web applications vulnerable to attacks such as SQL injection and cross-site scripting. The Open Web Application Security Project (OWASP) has launched a WebScarab project [7]. The other available commercial scanners include SPI Dynamics' WebInspect , Kavados ScanDo and Sanctum's AppScan [8-10].

B. Fix Vulnerability

Protecting your site against malicious injection is like most security related topics – no one solution solves everything. Above approaches just focus on analysis, they seldom present efficient method to fix automatically program's vulnerabilities. Scott and Sharp [11] take the programmatic approach of specifying a security policy explicitly to provide a web application input validation mechanism (a rule-based security gateway) to protect against common application-level attacks. However, to enforce a security policy across a large web-application is difficult and adapt this mechanism requires that rules be defined for every single data entry point since they often contain some complex structures with little documents. Sanctum Inc. provides a plug-and-play tool called AppShield which adopts Security Gateway to inspect HTTP messages in an attempt to prevent application-level attacks, but it only can provide a limited degree of protection for existing websites with application level security problems [12]. Based on similar strategies, some advanced firewalls now also incorporate deep packet inspection technologies for filtering application-level traffic to provide immediate assurance [13]. However, using predicted behavior to produce general patterns without investigating the actual vulnerabilities may reduce compromise quality. We had proposed an advanced tool that can be used to revise injection vulnerabilities [1]. It can produce a proper input validation function based on the database server and the application framework, but it needs source code of application or component. An enhanced prototype had been proposed solving the problem when source code may not be viable to be modified. It adopted a security gateway to filter malicious input data in front of web server [27]. As the above tow methods that we had proposed both use single rule to constrain all input, some errors (false positives or false negatives) may be generated.

III. Secure Input Handling

Input handling is how an application, server or system handles the input supplied from clients. Secure input

handling is often required to prevent vulnerabilities related to Code injection, Directory traversal and similar vulnerabilities [2]. Input validation and encoding process are two common methods employed to handle input data for security.

A. Input validation

Input validation is a secure way verifying user input to ensure that input is safe prior to use. In general, it checks the user input based on a whitelist or a blacklist and if not, a proper action will be adopted. A whitelist is a list of some accepted items. Whitelist-based filtering ensures that all requests are denied unless specifically allowed. Generally speaking, the list may involve setting character sets, type, length, format, and range. A generic solution may not be easily implemented but we can know that is acceptable input data for application program in a localized way. It is much easier to validate data for known valid patterns but it may induce more false positive.

A blacklist is a list being denied known bad input. In general case, a configurable set of malicious characters is used to reject the input but it is an unrealistic idea assuming that all the variations of malicious injection had been known. While useful for applications that are already deployed and when you cannot afford to make significant changes, the "deny" approach is not as robust as the "allow" approach because bad data, such as patterns that can be used to identify common attacks, do not remain constant.

Valid data remains constant while the range of bad data may change over time. A negative security model (blacklist) allows more abundant input data than a positive security model (whitelist). That is to say, blacklist often can support more elastic input data than whitelist but it may induce more false negative.

B. Encoding process

Encoding process is another way sanitizing content. Any other characters could be possibly interpreted in an unexpected manner should be replaced with the alternative encoded representation. For example, HTML Encoding is popular character entity reference (e.g. <script> is encoded to <script>). Sanitizing is about making potentially malicious data safe. It can be helpful when the range of input that is allowed cannot guarantee that the input is safe. This includes anything from stripping a null from the end of a user-supplied string to escaping out values so they are treated as literals [7]. A common example of sanitizing input in Web applications is using URL encoding or HTML encoding to wrap data and treat it as literal text rather than executable script. HtmlEncode methods escape out HTML characters, and UrlEncode methods encode a URL so that it is a valid URI request.

IV. Input Validation Testing

The Input Validation Testing (IVT) technique has been traditionally developed to address the problem of statically analyzing input command syntax as defined in English textual interface and requirements specifications and then generating test cases for input validation testing [14]. We hope our programs should anticipate most classes of input errors and handle them gracefully. The programs should be tolerant of operator errors and be able to properly process both expected and unexpected input values. Input validation testing, then, is defined as choosing proper test case that attempt to show the presence or absence of specific errors pertaining to input data.

A. Enhanced crawler

Testing Web applications for security defects is the first step when we try to create an automatic defense mechanism for Web application injection attack. It aims to look through pages and pages of HTML code for small clues that may be used for a successful Web injection attack. The traditional methods used to find insecure programming practices that have potentially rendered the code vulnerable to malicious attacks have been simply described in subsection II.A. While static analysis may be limited by the types of languages that have been utilized in crafting the Web application and can only find the suspicious code snippets vulnerable to malicious attacks rather than actionable results. While black box testing techniques are beneficial because they eliminate language dependency and the need for source code, they are also limited by the fact that do not have access to the source code, and if unable to "guess" where entry points are located, can render deadly “false negatives” -- since a single unidentified link would nullify the tests conducted for all other links. Here the “entry point” means that the clues used to find next relevant pages for surfing the whole Web site.

Only an approach that combines the advantages of both static analysis and black box testing can be used to reach ideal analysis. This hybrid analysis approach can provide broad code coverage, identify all entry points of input to an application, track data as it moves through an application, and then validate the vulnerabilities it does find, ultimately resulting in more accurate results [15, 22]. Hybrid analysis method combines the coverage of static analysis with the accuracy of black box testing so we can find more injection points and vulnerabilities through it.

The hybrid analysis scheme is implemented as an enhanced crawler. It consists of Crawler and Analyzer and is showed in Fig. 1. The crawler here is different from the traditional one called general. It just downloads all pages of a target web site starting from a given URL and doesn’t consider the HTML pages outside the web site host. The crawler send HTTP request to a page, parses the HTML received, extracts all clues of entry points from it, and recursively performs the same action on each entry point. It surfs the Web site to create a nearly local mirrored copy of a Web site. The Analyzer parses the web pages having been traversed and then grabs only some HTML elements or attributes having implications of injection flaw. Its behavior is similar to static analysis. To achieve the goal of finding more injection points, we should know what HTML tags

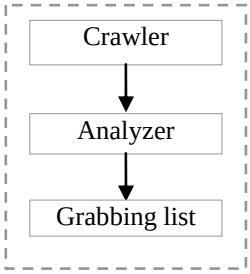

Figure 1. A diagrammatic view of enhanced crawler

delimit the information of injection point. For instance, HTML forms
constitute a crucial injection point to the Web application, from a security point of view. A simple substring search for the <form> tag reveals all the Web resources containing HTML forms. Along with looking for the presence of HTML forms, we also look at the fields passed within each form. Fig 2 shows the HTML elements that make up a form for example. An HTML form is identified by the <form>…</form> tags. All HTML tags embedded in the form tags are treated as part of the form. Among other HTML tags, the <input> tags comprise the input elements of the form. They allow the user to enter data in the HTML form displayed on the browser. The <form> tag implies the existence of entry point, and the <input> tag reveals input process. The <form action> attribute defines an associated server-side application designed to receive data from the various input elements of the form. At first, the Analyzer searches for <form> tags and extracts the URL of the server-side application from the <form action> attribute. We also have to look at the <form method> attribute to determine whether the data will be submitted as GET or POST method. Each input element must have a name, which is used by the server-side application for parsing parameters and their values. The second step is to grab parameters named ‘struser’ and ‘strpass’ from <input name> attribute. Upon completion of finding variables process, we can get a list as follows. A list of the grabbing result is presented in table I for example.

B. Injection Test

Injection testing is similar to bypass testing. It is dedicated to generate erroneous case for testing during run time so it belongs to black box testing. To validate the user’s data is necessary to ensure that the software receives data that will not cause the software to do bad things such as crash, corrupt the data store on the server,

```
<form method="post" action="verify.exe">
<input type="text" name="struser" />
<input type="password" name="strpass" />
<input type="submit" value="Login" />
</form>
```

Figure 2. The common HTML elements that make up a form.

TABLE I. A LIST OF THE GRABBING RESULT.

Result Field	Value
Action Program	verify.exe
Form Method	post
Parameter1	struser
Parameter2	strpass

or allow access to unauthorized users. General Web applications are client-server nature so input validation can be done on the client or the server. The client-side validation can give the user immediate feedback and prevent malformed requests from ever reaching your server but it is easy to be bypassed. Even if an experienced attacker may get around the client-side validation, the server-side validation is still effective. That is to say, server-side validation is safer than client-side. In our system, the injection test can do the best to bypass server-side validation. That the testing patterns are injected into the injection points of a tested web site is similar to the action of launching injection attack.

Another important thing we should consider is how to choose some proper testing patterns. Jeff Offutt had described a systematic approach to identify constrains among input parameters. This problem is common to all Web testing strategies as well as GUI testing strategies [21].

C. Testing Framework

The Input Validation Testing is conducted at a testing framework. Fig 3 shows its architecture of the testing framework and interactions between each component. The testing framework consists of Tester, Monitor, Verifier and Reporter. The Tester includes two main modules: test case generator and injector. The test case generator produced specific test cases in accordance with the obligation that is the result of hybrid analysis. The injector is responsible for injecting testing pattern. The Tester is responsible for injecting testing patterns. It is similar to general malicious injection tools, such as Web Scarab’s Parameter fuzzer, to expose incomplete parameter validation, leading to vulnerabilities like Cross Site Scripting (XSS) and SQL Injection [7]. Comparing with other injection tools, the Tester has one difference: it generates various testing patterns having expected output to enhance differentiability. The Verifier compares the actual output received by the Monitor with the expected output and judges whether both are similar or not [23, 24]. If the output of the evaluated system is different with the output of the Tester, it implies the sanitizing mechanism is effective, or else. The Reporter can record these results of each testing case and generate various reports. The effectiveness of the sanitizing mechanism can be evaluated in accordance with these reports.

V. SANITIZING MALICIOUS INJECTION

The methods having been used to inject malicious data into Web applications are common and can cause great losses so we focus our research on finding an effective sanitizing mechanism for malicious injection.

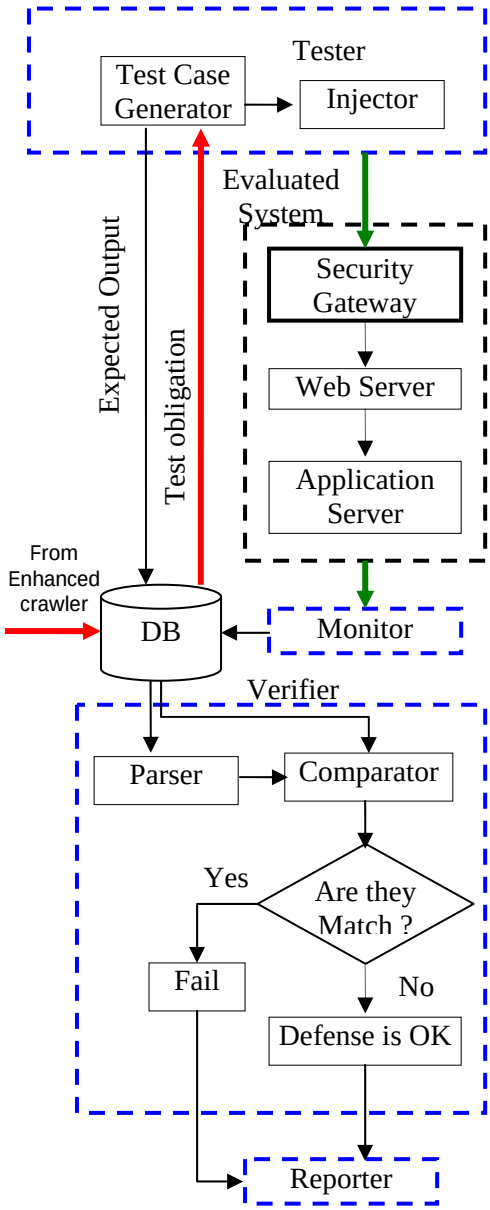

Figure 3. An architecture diagram of testing framework

Although the majority of web vulnerabilities are easy to understand and avoid, many web developers are not security-aware. The vulnerability was caused by the lack of input sanitization in the integrity of data received from the Web browser. Input validation is a the most efficient way verifying user input to ensure that input is safe prior to use but we always have no time to fix it immediately when we face a large number of vulnerable Web applications. A security gateway like Web application firewalls (WAF) is capable of preventing injection attack and does not require modification of application source code. We adopt the concept and propose an automated sanitizing mechanism for Web application injection attack.

A. Meta-program

How to perform input validation effectively is a critical task to prevent against injection attacks. In our sanitizing mechanism, the target can be achieved via meta-programs. Meta-programs will be automatically translated through a translator. After previous hybrid analysis and injection test performed in testing framework, we had gotten a vulnerable list that kept the name of Web applications having injection vulnerability. The list can be read as input of translator for generating proper meta-programs. For example, if a vulnerable server-side program named 'verify.exe' is kept in the list, and our translator can automatically generate a meta-program named 'verify.php'. To sanitize malicious data effectively, an adjustable validation function can be included in each meta-program. An input validation function and code snippet of the meta-program are presented in Fig 4 for instance. Note that the Italic words should be replaced by parameters grabbed at hybrid analysis step described in subsection IV.A. The meta-programs can be placed in security gateway to perform the task of input validation to replace the Web applications of poor validation ability.

The concept of meta-program is inspired by Model-view-controller (MVC) that is an architectural pattern used in software engineering. The MVC decouples data access and business logic from data presentation and user interaction by introducing an intermediate component: the controller. The controller receives and translates input to requests on the model or view. The input could appear as GET and POST HTTP requests in a Web application. Generally speaking, a controller often handles and validates the input and so does our meta-program.

B. Choosing proper validation rules

There is no specific answer for what implies valid input across applications, that is to say, there is no single validation rule to sanitize properly all input. As individual fields often require specific validation, choosing proper rules is a challenging issue and the primary burden of a solution falls on application developers. The concept inspires us to propose an automatic approach to make a decision of sanitizing rules. In an ideal scenario, the application checks for the most acceptable input for each entry point. As some existing Web applications that don't properly sanitize user input, we need an automatic approach to improve application's input validation strategy and reduce our burden on fixing these programs.

According to past experience and bypass testing result, we complete a rule-bank including some kinds of filter

(1) Input Validation Function:

```
function anti_injection($sql)
{  //injection patterns
$sql = preg_replace(sql_regcase("/(from|
slect|insert|delete|where|droptable|
show tables|#|\*|--|\\\\)/"),"",$sql);
$sql = trim($sql);
$sql = strip_tags($sql);
$sql = addslashes($sql);
return $sql; }
```

(2) meta-program (verify.php) snippet:

```
$struser= anti_injection($_POST["struser "]) ;
$strpass= anti_injection($_POST["strpass "]) ;
```

Figure 4. Validation function and meta-program.

rules. These rules are expressed as a string called the regular expression in computer field and classified carefully according to the input handling strategies described in section III. Every level has various graded sub-rules based on generic rule. Some graded sub-rules are shown in table II for example.

As individual fields require specific validation, each specific validation rules need an automatic mechanism to choose and test them. The pseudo-code used to select proper rules from rule-bank is shown in figure 5. To reduce false, we adopt a hybrid strategy discussed in subsection III.C for validation. In every level, the first rule filled in Null means "do nothing" and many common or specific rules are filled next for various sanitizing purpose. Every injection point should be injected all testing patterns and every rules in rule-bank should be used to validate in order. We can get many permutations from the rule sets of each level and represented as {element1, element2, element3}. The first element always comes from the rule sets of first level, the second and the third ones come separately from the second and the third level. After every bypass testing, we check whether the error was made or not. The errors (false positive or false negative) induced by the same permutation, such as {1, 2, 3}, were summed up separately after all the testing patterns having been injected. According to the result of injection testing, we can find a permutation having none of false negative and a minimum error for every injection points. If every permutation all have false negative and different permutations induce same error, the permutation having minimum false negative should be preferred. If all existent rules can't block a specific injection pattern sometimes, the pattern will be marked for modifying or producing a new rule by means of manual analysis. These procedures go round and round and specific rules will be organized to accord with the demand of individual fields. The choosing mechanism can not only accord with the demand which individual fields require specific validation but also reduce errors.

```
FOR each entry point (i)
  FOR each injection pattern (j)
      FOR each rule permutation (k)
     IF error happen THEN
      CASE false OF
            positive:  positivei,j,k ++
            negative:  negativei,j,k ++
      ENDCASE
     END IF
     IF all negativei,j,k <> 0 THEN
       //storing for analyzing manually
           store pattern (i, j)
        END IF
       END FOR
   END FOR
END FOR
  errorposi(i,k)=sum up positivei,j,k ,  per j
  errorneg(i,k)=sum up negativei,j,k ,  per j
  // select minimum false negative
  Set  neg  =  min(errorneg(i,1),  …,
 errorneg(i,k))
  // computing minimum error for every
     entry point
  // if ki is a element of minimum errorposi
     in Set neg
  error(i)=errorposi(i,ki)+ error neg(i,ki)
  // selecting permutation having minimum
    error for every entry point
  rule(i)= rule permutation (ki)
```

Figure5. The pseudo-code used to select proper rules from rule bank

VI. System Overview

The automated sanitizing system comprises a module of the hybrid analysis, a security gateway having adjustable validation functions and a framework of the injection test. Fig 6 presents its architecture and interactions between each component. The main components are marked with blue blocks. The data flows of the hybrid analysis module are expressed with the red arrows, the data flow of the translator is expressed with the brown arrow and the data flows of the injection test are shown with the green arrows. The enhanced crawler is aimed at getting a list keeping some vulnerable Web applications and their associated parameters. The security gateway is like Web application firewalls (WAF) designed to protect web sites from injection attacks. WAF solutions are capable of preventing attacks that network firewalls and intrusion detection systems can't, and they do not require modification of application source code [26]. It consists of two main components: the translator and meta-programs. In accordance with the result of the enhanced crawler, the translator can generate

TABLE II. Graded sub-rules in rule-bank

Level / Rule_no.	1	2	3
1	Null	Nul	Nul
2	[[:punct:] [:alnum:]]	[';-]*	addslashes()
3	^[a-zA-Z0-9]{1,40}$	[:anti_sql:]	Htmlspecial chars()

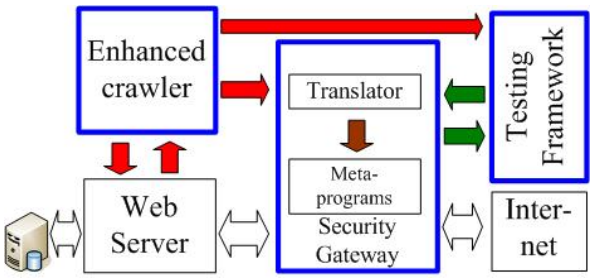

Figure 6. Architecture of the automated sanitizing system

corresponding meta-programs to create a sanitizing web site on security gateway. By means of an adjustable validation function included in meta-programs, malicious input can be sanitized to avoid injection attack. The testing framework is responsible for injecting test patterns and certifying the sanitizing mechanism. We can evaluate the effectiveness of the sanitizing system in accordance with the results of each testing case. The green arrows show these interactions between testing framework and security gateway.

VII. System Evaluation

Because the security solution is more difficult to be located in front of any Web site on internet for primary experiments so we design a experimental Web site consisting of some client-side and server-side Web pages and applications having injection vulnerability, so as to evaluate system's capability. We focus on the completeness of enhanced crawler and effectiveness of the meta-programs among these experiments. The completeness is an important issue in security assessment – that is, all links must be correctly identified for finding all entry points and injection points, and the effectiveness implies that all malicious patterns should be successfully sanitized. For assessing completeness accurately, the Web pages (client-side and server-side) on the experimental Web sites should have the characteristics such as static Web pages, dynamic Web pages, form submission and session management. Dynamic characteristic is the way it functions while any web page generated differently for each user、load occurrence、or specific variable values, not in its ability to generate a unique page with each page load. Form submission is a common method for submitting data in most Web applications. Some applications emphasize session management and they require cookies to assist navigation mechanisms and avoid bypassing authentication. Some traditional crawlers only use static parsing technique (lacking ability of form feed or script interpretation) can't achieve complete surfing if they crawl any Web sites having some characteristics described above. In order to achieve complete coverage, we test several commercial and open source crawlers, such as Google, Black Window, Teleport Pro, JoBo and so on [19-21]. We hope to achieve complete coverage by making use of hybrid strategy described in subsection IV.A.

For the sake of simple analysis, the experimental Web site has only 20 entry points, 20 injection pattern and 10 rules in each level. There are 5 normal patterns and 15 malicious patterns in all injection patterns. Thus, there are 4000 comparisons at most whenever the whiltelist or blacklist strategy is adopted and there are $4*10^5$ comparisons for hybrid strategy. While whitelist or blacklist method can use only one rule on every trial, the detection error of each injection point is represented as average. After initial test, various expected output has been recorded to compare with a succession of real result. For instance, table III shows the number of injection point and cooperative validation rule number. According to table III, different injection point can has various validation rule made up of some sub-rules. The errors include false positive and false negative. The false positive occurs when the normal input data is mistakenly blocked by the filtering rule. On the contrary, the false negative implies that the filtering rule can't sanitize the malicious injection. As different injection patterns and filtering rule–bank may cause different false rate, average false rat is more preferable. We continue to prove that the diversified validation rules organized by our automated mechanism can improve detection accuracy (reducing false positives or false negatives). Table IV reports the false rate relative to different validation strategy. As different injection patterns and sanitizing rule may cause different false rate, average of every round can make the result more preferable. It proves that the hybrid strategy gets the least average false rate than the blacklist or the whitelist strategy. The results of the experiment were encouraging and preliminarily showed the effectiveness of the automated sanitizing mechanism.

VIII. Conclusions

As the traditional security gateway needs to check every data flows completely and just use a generic sanitizing rule, it will induce more system load and false positives while performing the audits. In this paper, we present an integral sanitizing mechanism consisting of application-level security gateway, injection testing framework and enhanced crawler to reduce the error rate of detection. To mitigate the defects, the hybrid analysis module can cooperate with testing framework to perform injection test to generate a vulnerable list. Only the Web pages that are in a list need to be checked so the system load and false positives can mitigate effectively. Moreover, we propose a mechanism that can automatically create meta-programs and adjust the sanitizing rule of every vulnerable injection point. According to the experimental results, the sanitizing mechanism has been proved to be a more effective

TABLE III. Input variables and cooperative validation rules

Injection Point no.	Validation Rule no.			Errors
	Level 1	Level 2	Level 3	
1	5	2	1	1
2	5	5	2	2
3	3	1	l	1
4	2	1	1	1

TABLE IV. Average false rate gotten to different validation strategies.

Validation strategy	Total comparisons	Average False positives	Average False negatives
		Max=5	Max=15
Whitelist	4000	2.45	8.32
Blacklist	4000	4.35	1.22
Hybrid	$4*10^5$	2.26	0.65

scheme than those who use single strategy for mitigating malicious injection.

REFERENCES

[1] Jin-Cherng Lin and Jan-Min Chen, "An Automatic Revised Tool for Anti-malicious Injection", in Proceedings of The Sixth IEEE International Conference on Computer and Information Technology (CIT'06).

[2] Wikipedia, the free encyclopedia, "Secure input and output handling", http://en.wikipedia.org/wiki/ Secure input and output handling

[3] Y.W. Huang, S.K. Huang, T.P. Lin, C.H. Tsai, Securing Web application code by static analysis and runtime protection, in: Proceedings of the 13th International World Wide Web Conference, New York, May 17–22, 2004.

[4] Y.W. Huang, F. Yu, C. Hang, C.H. Tsai, D.T. Lee, S.Y. Kuo, Verifying Web applications using bounded model checking, in: Proceedings of the 2004 International Conference Dependable Systems and Networks (DSN2004), Florence, Italy, June 28–July 1, 2004.

[5] Jovanovic, N.; Kruegel, C.; Kirda, E.; "Pixy: a static analysis tool for detecting Web application vulnerabilities", In Proc. Of the 2006 IEEE Symposium on Security and privacy, 21-24 May 2006 Page(s):6 pp.

[6] Huang, Y. W., Huang, S. K., Lin, T. P., Tsai, C. H. "Web Application Security Assessment by Fault Injection and Behavior Monitoring." In Proc. 12th Int'l World Wide Web Conference, p.148-159, Budapest, Hungary, 2003.

[7] OWASP. "WebScarab Project." Available from http://www.owasp.org/webscarab/

[8] SPI Dynamics. "Web Application Security Assessment." SPI Dynamics Whitepaper, 2003.

[9] KaVaDo. "Application-Layer Security: InterDo 3.0." KaVaDo Whitepaper, 2003. Available from http://www.kavado.com/

[10] Sanctum Inc., Web Application Security Testing—AppScan3.5", Available from http://www.sanctuminc.com

[11] Scott, D., Sharp, R. "Abstracting Application-Level Web Security." In: The 11th International Conference on the World Wide Web (Honolulu, Hawaii, May 2002), 396-407.

[12] Sanctum Inc. AppShield. white paper. March 2003. Available from http://www.sanctuminc.com/.

[13] Dharmapurikar, S., Krishnamurthy, P., Sproull, T., and Lockwood, J. "Deep Packet Inspection Using Parallel Bloom Filters." In Proc. 11th Symp. High Performance Interconnects (HOTI' 03), p.44-51, Stanford, California, 2003.

[14] Hayes. J.H., Offutt. A.J., "Increased software reliability through input validation analysis and testing Software Reliability Engineering", 1999. Proceedings. 10th International Symposium on 1-4 Nov. 1999 Page(s):199 – 209

[15] Ounce lab,Inc, "The Top Web Application Vulnerabilities and How to Hunt Them Down at the Source" ,Available from: http://www.ouncelabs.com/dozen/ visit on 2006/2/05

[16] Yao-Wen Huang, Chung-hung Tsai, Tsung-Po Lin, Shih-Kun Huang, D. T. Lee, Sy-Yen Kuo "A Testing Framework for Web Application Security Assessment." Journal of Computer Networks, 48(5):739-761, Jun. 2005, Elsevier. 2

[17] Blighty Design, Inc., Sam spade, Available from: http://www.samspade.org.

[18] SoftbyteLabs, Black widow, Available from: http://www.softbytelabs.com

[19] Tennyson Maxwell Information Systems, Inc., Teleport Webspiders. Available from: http://www.tenmax.com/ teleport/home.htm.

[20] wget, Available from: http://www.gnu.org

[21] Offutt. J., Wu. Ye., Du. X.;,Huang. H., "Bypass testing of Web applications, Software Reliability Engineering", 2004. ISSRE 2004. 15th International Symposium on, 2-5 Nov. 2004 Page(s):187 – 197

[22] SPI Labs, "Hybrid Analysis- An Approach to Testing Web Application Security", Available from: http://www.spidynamics.com /assets/documents/hybrid_analysis.pdf visit on 2006/3/05

[23] Di Lucca G.A., Fasolino, A.R., Faralli, F. De Carlini U. , "Testing Web applications.; Software Maintenance", 2002. Proceedings. International Conference on 3-6 Oct. 2002 Page(s):310 – 319

[24] Elbaum. S., Rothermel. G., Karre. S., Fisher II. M. ,"Leveraging user-session data to support Web application testing", Software Engineering, IEEE Transactions on Volume 31, Issue 3, March 2005 Page(s):187 – 202

[25] J.D. Meier, Alex Mackman, Michael Dunner, Srinath Vasireddy, Ray Escamilla and Anandha Murukan , "Design Guidelines for Secure Web Applications", visit on 2007/3/15 http://msdn2.microsoft.com/en-us/library/aa302420.aspx.

[26] Web application security consortium, "Web Application Firewall Evaluation Criteria", http://www.webappsec.org/projects/wafec/

[27] Jin-Cherng Lin, Jan-Min Chen and Hsing-Kuo Wong, "An Automatic Meta-revised Mechanism for Anti-malicious Injection", in Proceedings of Network-Based Information Systems (NBiS) 2007, LNCS 4658, pp. 98-107, 2007

[28] Open Web Application Security Project. "The ten most critical Web application security vulnerabilities", http://www.owasp.org/images/ e/e8/OWASP _Top _10_ 2007.pdf

[29] Jin-Cherng Lin, Jan-Min Chen, Cheng-Hsiung Liu, "An Automatic Mechanism for Adjusting Validation Function", Proc. of the 4th International Symposium on Frontiers in Networking with Applications (FINA 2008)

Jin-Cherng Lin received the Ph.D. degree in Information Engineering and Computer Science from National Chiao-Tung University, Taiwan, in 1989. He is currently an associate professor in the Department of Computer Science and Engineering at Tatung University, Taiwan. His research interests include software testing and validation, software quality assurance, computer network management, and computer network security. His email is jclin@ttu.edu.tw.

Jan-Min Chen is a lecturer at the Yuda College of Business, Taiwan, with faculty appointments in department of information management. He received a Master degree in the Dept. of Computer Sci & Eng, Tatung University and a candidate for Ph.D. now. His current research interests include network security and web application security.