Anti-malicious Injection Based on Meta-programs

Jin-Cherng Lin[1,*] and Jan-Min Chen[1, 2]

[1]Department of Computer Science and Engineering

Tatung University

Taipei City 10451, Taiwan

jclin@ttu.edu.tw

[2]Department of Information Management

Yu Da College of Business

Miaoli 36143, Taiwan

ydjames@ydu.edu.tw

Received Date: 24 October 2007; Revised Date: 20 December 2007; Accepted Date: 10 January 2008

Abstract. Injection attack is a technique to bypass or modify the originally intended functionality of the program by injecting codes into a computer program or system. 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 Script injection (Cross Site Scripting). Some applications attempt to protect themselves by filtering malicious input data, but it may not be viable to modify the source of such components (either because the code was shipped in binary form or because the license agreement is prohibitive). We have tried to develop a defense mechanism that can automatically generate meta-programs on security gateway to filter malicious injection. The security gateway is allocated in front of application server to eliminate malicious injection vulnerabilities. To verify the efficiency of the mechanism, we create the web sites made up of some Web applications that often contain third-party vulnerable components shipped in binary form. According to the result of these experiments, our defense mechanism has proved itself efficiency.

Keywords: Black box testing, Malicious injection, Input validation, Security gateway.

1 Introduction

"Unvalidated Input" is Top One critical Web application security vulnerabilities according to OWASP Top 10 2004 [1]. Most importantly, Top 1-5 critical Web application security vulnerabilities caused by unchecked input according to OWASP Top 10 2007 [2]. Unvalidated Input may lead hacker to inject code to bypass or modify the originally intended functionality of the program to gain information, Privilege escalation or unauthorized access to a system. Many tools have been developed to detect injection vulnerabilities but hackers are still successfully exploiting applications. A possible reason is that most tools just scan application's vulnerabilities, but few tools can automatically revise these vulnerabilities. An advanced tool producing a proper input validation function depending on the database server and the application framework had been developed and verified its efficiency [3], but it needed source code to insert input validation function. If we can't find or modify the source codes of such components, above method can't be used.

In this paper, we present an advanced proposal adopting the concept of application level security gateway and more effectively resolving the problem than similar gateways or proxies. Our system mainly consists of hybrid analysis framework, meta-programs and testing framework. Hybrid analysis framework can be used to find all entry points, perform penetration testing and generate a list of weak Web application. In accordance with the list, meta-programs can be translated to create a sanitizing web site on security gateway to filter malicious input data. The testing framework is responsible for performing bypass test and certifying the defense mechanism. We can evaluate the defense capability of our mechanism in accordance with the results of each testing cases. We integrate the detection of injection vulnerability and generic sanitizing methods into an automatic defense mechanism. It can be used to protect a surprising number of applications having no adjustable validation mechanisms against malicious injection attack.

* Correspondence author

The remainder of the paper is structured as follows: Section 2 surveys a number of web application vulnerabilities and discusses related works have been proposed. In Section 3, we describe the technical details of our defense mechanism. The system implementation is discussed in Section 4. The efficiency is evaluated in Section 5, finally, Section 6 concludes.

2 Related Works

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. Static analysis can be used to analyze Web application code, for instance, ASP or PHP scripts. 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) [4,5]. The tool can be successfully used for automated Web application security assessment. Recently, 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[6].

Another method called a black-box approach is adopted to analyze Web applications externally without the aid of source code. A black-box security analysis tool 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 have developed a remote, black-box security testing tool for Web applications is also called the Web Application Vulnerability and Error Scanner (WAVES) [7]. 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) [8] has launched a WebScarab project. Two other available commercial scanners include SPI Dynamics' WebInspect [9] and Kavado's ScanDo [10].

Above approaches just focus on analysis, they seldom propose any efficient method to automatically fix 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 [13] technologies for filtering application-level traffic to provide immediate assurance. However, using predicted behavior to produce general patterns without investigating the actual vulnerabilities may reduce compromise quality. We had proposed an advanced tool producing a proper input validation function depending on the database server and the application framework [3], but we need source code of program or component to revise injection vulnerabilities. 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 [14].

3 Anti-malicious Injection

The methods having been used to inject malicious data into Web applications are common and can cause great losses, we focus our research on this topic. Although the majority of web vulnerabilities are easy to understand and avoid, many web developers are not security-aware. 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 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.

Black box testing analyzes the application from the actual view of the user (and also a potential hacker), and not the developer of the application. Wherever there is an opportunity to inject malicious user input, the black box testing application injects it and analyzes the response to take further action. Black box testing techniques are beneficial because they eliminate language dependency and the need for parsing the source or binary code into an analyzable form. They are also limited by the fact that do not have access to the source code, and if

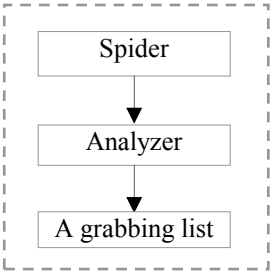

Fig. 1. A diagrammatic view of hybrid analysis framework

unable to "guess" where some pages or files are located, can provide a false sense of security by producing numerous “false negatives”.

Only an approach that combines the strengths of both source code analysis and black box testing can be used to produce secure Web applications. This hybrid analysis approach can provide broader code coverage, identify all data entry points, track data as it moves through an application, and then validate the vulnerabilities it does find, ultimately resulting in more accurate results [15]. Hybrid analysis method combines the depth of source code analysis with the accuracy of black box testing.

We try to develop an advanced proposal adopting a concept of application-level security gateway. Our system mainly consists of hybrid analysis framework, meta-programs and testing framework. Hybrid analysis framework can be used to find all entry points, perform penetration testing and generate a list of Web application vulnerability. In accordance with the vulnerability list, meta-programs can be translated to create a sanitizing web site on security gateway to sanitize malicious input data. The testing framework is responsible for taking bypass test and certifying the defense mechanism.

3.1 Hybrid Analysis Framework

Our hybrid analysis framework consists of Spider and Analyzer and shows in Fig. 1. Spiders (also known as a web crawler or ant) go out on the web and collect information. The way a typical spider (like google) works is by looking at one page and finding the relevant information. It then follows all the links in that page, collecting relevant information in each following page, and so on. Our spider is different from the traditional one called general. It just downloads all pages of a target web site starting from a given URL. Each page found within the site host is downloaded and the HTML documents outside the web site host are not considered. The pages of a site are obtained by sending the associated requests to the Web server. To improve complete spider, the ways that HTML pages reveal the existence of other pages or entry points have been mentioned, and came up with the following list [16]:

1. Traditional HTML anchors.
   Ex: Google
2. Framesets.
   Ex: <frame src = ‘‘f1.htm’’ >
3. Meta refresh redirections.
   Ex: <meta http-equiv = ‘‘refresh’’ content = ‘‘0; URL = http://www.google.com’’ >
4. Client-side image maps.
   Ex: <area shape = ‘‘rect’’ href = ‘‘http://www. google.com’’ >
5. Form submissions.
6. Javascript variable anchors.
   Ex: document.write(“\”+LangDir+”index.htm”);

Our spider grabs only some variables included in web page having been traversed. To achieve the goal, we should know what HTML tags delimit the information we want to find. The common HTML elements that make up a form show in Fig. 2. The <Form> tag implies the existence of entry pointer, and the <input> tag reveal input process. Our spider scans the HTML for <form> tags and extracts the destination URL 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 parameters. The second step was to grab parameters named ‘struser’ and ‘strpass’ from the input name attribute. Upon completion of finding variables process, we can get a list as follow. A list of the grabbing results presents in Table 1.

Table 1. A list of the grabbing result.

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

3.2 Bypass testing

Bypass testing is dedicated to generate erroneous test case. 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, 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 gives the user immediate feedback and prevents malformed requests from ever reaching your server. But even if an experienced attacker gets around the client-side validation, the server-side validation is still efficient. Generally speaking, server-side validation is for security. In our system, bypass testing technique just only do the best of ability to bypass server side checking. We generate some bypass testing patterns and thereby inject them to the injection points of a tested web site.

Another important thing we should consider is how to choose some proper bypass 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 [17]. Some general factors that should be considered to generate bypass testing patterns to test the Web application are as follows:

1. Illegal characters: some specific characters that can be problems for Web applications and their polymorphous encoding values to represent same characters.
2. Numeric and length limits: limiting all numbers and the length of input parameters to the minimum and maximum allowed values.
3. URLs: URLs (and more generally, URIs) should be checked to ensure that they have a valid form and the destination exists.
4. Malicious patterns: the signatures have been used to malicious injection.

A test obligation is a defect or potential software problem that is detected by means of the hybrid analysis [15]. If a defect is found, it is stored in the test obligation database. Some bypass testing patterns generated for each test obligation and their respectively expected output are recorded in the table used to record all the test cases that are generated. Our bypass testing will be conducted at value and parameter levels according to the table. By means of automatic recognition of failure and expected output, we can evaluate input validation capability of a program.

3.3 Testing Framework

The Input Validation Testing is conducted at a testing framework. Fig. 3 shows the architecture of the testing framework and interactions between each component. It consists of Tester, Monitor, Verifier and Reporter. The Tester is responsible for bypass testing tasks. It is similar to general malicious injection tools, such as Web Scarab's Parameter fuzzer [8], to expose incomplete parameter validation, leading to vulnerabilities like Cross Site Scripting (XSS) and SQL Injection. Comparing with other injection tools, our Tester has one difference: it generates various testing patterns having expected bypass output. The Verifier compares the actual output received by the Monitor with expected output and judges whether both output are similar or not. If output of the evaluated system does not match that of the Tester, it implies the defense mechanism is certified. The Reporter can record these results of each testing case and generate various reports and then we can evaluate the defense capability of our mechanism in accordance with these reports.

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

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

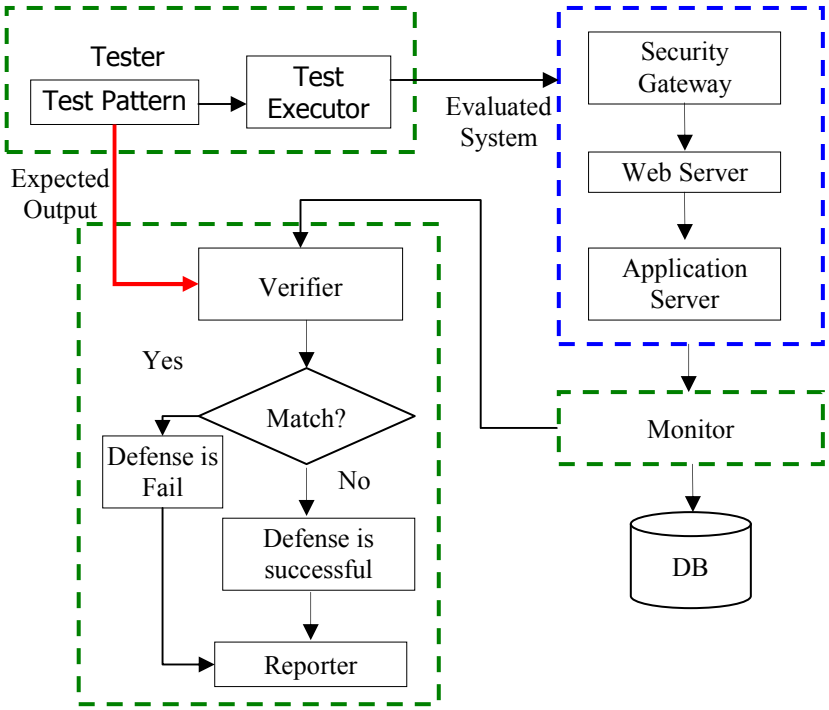

Fig. 3. Architecture of the testing framework

3.4 Application-level Security Gateway

Fig. 4 shows those tasks performed by the Security Gateway on receipt of an HTTP request. First, the URL is extracted from the HTTP header. Having identified a valid URL, the security gateway proceeds to check the names of all parameters and cookies passed in the HTTP request. Errors are generated if any of the parameters present precisely match those specified in deny_list (included some characters have been proved to high risky attack patterns). If any violations occur at this stage then a descriptive error message is returned to the client.

After previous black box testing, we can create an entry pointer list having written all records needed to be validated. Once we are sure that the HTTP message contains a valid combination of cookies and GET / POST parameters, we'll search entry pointer list. If we can find a record from the list, we'll continue to validate input data. Otherwise the redirection mechanism will be applied. Next the proper meta-programs that will be described in subsection 3.5 are applied to validate input data. Finally, if all results of the validation are evaluated to true then the redirection mechanism will be used to process tasks forwarding HTTP request to the web-server and HTTP responses returned from the web-server. Fig. 5 shows how the security gateway works.

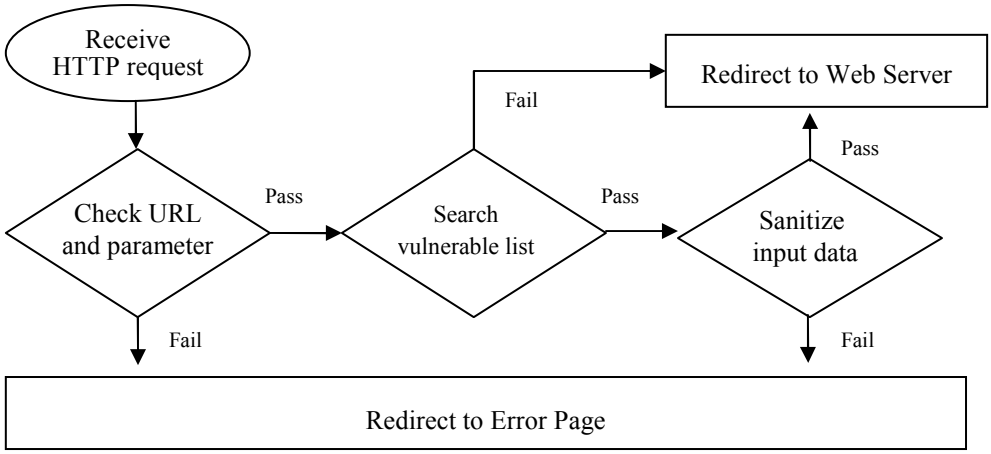

Fig. 4. Tasks performed by the Security Gateway.

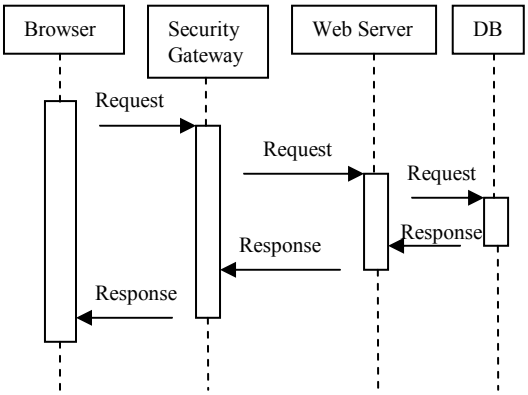

Fig. 5. How the security gateway works

3.5 Meta-programs

Meta-programs will be automatically translated via a translator. An obligation database described in subsection 3.2 keeps associated data about those vulnerable Web applications in Web server. The database 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 obligation database, our translator can automatically generate a meta-program named 'verify.php'. To effectively sanitize malicious data, an adjustable validation function is included in each meta-program. An input validation function and code snippet of the meta-program are presented in Fig. 6 for instance. Note that the Italic words should be replaced by parameters grabbed at hybrid analysis step described in subsection 3.1. The meta-programs can be placed at security gateway. By means of these, those that may not be viable to modify their vulnerable source codes can be protected from avoiding injection attack.

The concept of meta-program is induced 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.

```
(1) Input Validation Function:

function anti_injection($sql)

{  //injection patterns

$sql =
preg_replace(sql_regcase("/(from|select|insert|delete|where|drop
table|show tables|#|\*|--|\\\\)/"),"",$sql);

$sql = trim($sql);

$sql = strip_tags($sql);

$sql = addslashes($sql);

return $sql; }

(2) meta program (verify.php) segment:

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

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

Fig. 6. Validation function and meta- program.

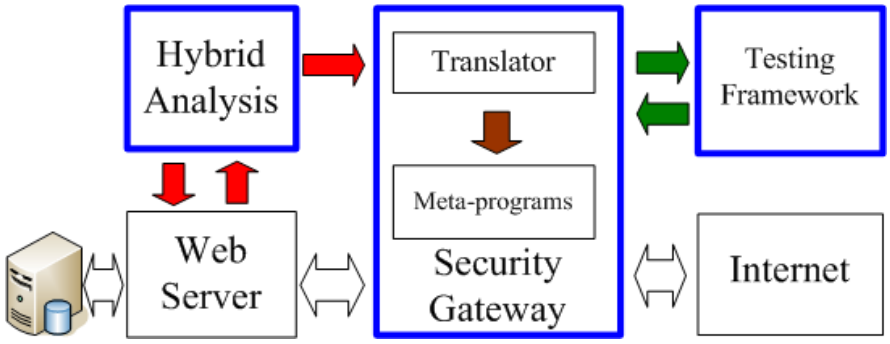

Fig.7. Architecture of our defense system.

4. System Overview

We have described the technical details of our system in Section 3. Fig. 7 presents its architecture and interactions between each component. The main components are marked with blue blocks. Hybrid analysis procedures are expressed with the red arrows, translator is expressed with the brown arrow and bypass testing procedures are shown with the green arrows. Hybrid analysis is aimed at getting a list keeping some Web application programs having injection vulnerabilities in Web server and their associated input variables. Above-mentioned interactions are expressed with the red arrows. In accordance with the result of hybrid analysis, the translator can generate corresponding meta-programs to create a sanitizing web site on security gateway. For example, if there is a vulnerable server-side program named '*verify.exe*' in a vulnerable list, the translator can automatically produce a meta-program named '*verify.php*'. By means of an adjustable validation function included in meta- programs, malicious input data can be sanitized to avoid injection attack. The testing framework is responsible for taking bypass test and certifying the defense mechanism. We can evaluate the defense capability of our mechanism in accordance with the results of each testing case. The green arrows show these interactions between testing framework and security gateway.

5 System Evaluation

We try to design a website consisting of some form feed web pages having some security problems, so as to evaluate the system's defense capability. We focus on the completeness of hybrid analysis and effectiveness of the meta-programs. The completeness is an important issue in security assessment – that is, all entry points must be correctly identified. The effectiveness implies that all attack patterns must be successfully filtered. The Web sites including various characteristics have been implemented for assessing completeness and shown in Table 2. Form submission is a primary method of input data in most Web applications. Many pages within Web applications currently contain such dynamic content such as Javascript and DHTML and some applications emphasize session management that requires the use of cookies to assist navigation mechanisms. Some traditional crawlers which use static parsing and lack form feed or script interpretation abilities can't achieve complete surfing if they crawl any Web sites having above features except static characteristic. For example, We can use GNU's *wget* to create a mirrored copy of a Web site and then perform searches on it with *grep*. The method just can efficiently crawl static Web pages. In order to achieve complete coverage, we test several commercial and open source crawlers, such as Google, Black Window [18], Teleport Pro [19], JoBo [20] for crawling and analyzing Web sites. We hope to achieve complete coverage by making use of integration of various tools.

Table 2. Some sample sites have various characteristics.

Web Site	Static	Dynamic content	Session control	Form submission
Site 1	✓			✓
Site 2	✓	✓		✓
Site 3	✓	✓	✓	✓

For the sake of simple analysis, each experimental Web site only includes 20 entry points and we only use 10 different attack patterns for each entry points. By means of our testing framework, the Spider can correctly find entry point. Analyzer can also parser web page, so as to grab variables for function's arguments. The validation functions that are dependent on various script languages have correct arguments and are included in meta-programs to validate input data. Among our experiments, the security gateway can always successfully avoid malicious injection attack by means of meta-programs. The results of the experiment were encouraging and preliminarily showed the effectiveness of meta-programs. Because our security gateway must be in front of web server, our mechanism cannot be experimented on a large number of internet websites.

6 Conclusions and Future Works

Many web servers have vulnerabilities that can result in unauthorized access to a system by means of submitting malicious requests. Given even a tiny hole in a web-application code, an experienced hacker can break into most commercial websites. The risk of poor application security must exceed our imagination. To effectively audit and manage software risk, a software security assurance (SSA) strategy has been developed and implemented. In the rush to bring new software products to market, few companies test their products from a security perspective, yet users rely on these products every day. Many tools just scan Web application vulnerabilities, but few tools can automatically fix these problems. We had developed a tool producing a proper input validation function that is dependent on the database server and the application framework, but it needed source code for inserting input validation functions to revise injection vulnerabilities. In this paper, we present a prototype making use of meta-programs in security gateway to solve the problem as we can't or don't know how to modify the source code.

Unless a web application has a strong, centralized mechanism for validating all input from HTTP requests (and any other sources), vulnerabilities based on malicious input are very likely to exist. It is our contribution that efficiently combines hybrid analysis, meta-programs with security gateway to eliminate malicious injection problem. Facing a great number of weak Web sites having some unknown hidden web pages, we need an efficient tool which can comprehensively identify all data entry points. A single unidentified link would nullify the total protection from injection attack. Hackers constantly find new vulnerabilities to attack, so an older validation function may lose efficiency. As to that, we just enhance validation function. Our mechanism is an easy and effective way to fix a large number of programs. In the future, we are going to combine with honeypot to find new attack about malicious injection. We hope to present an automatic mechanism that can be employed to achieve complete coverage and adjust sanitizing rules to improve completeness and effectiveness.

References

[1] Open Web Application Security Project, "The ten most critical Web application security vulnerabilities", http://umn.dl.sourceforge.net/ sourceforge/owasp/OWASPTopTen2004.pdf, visit on 2005/10/05

[2] 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, visit on 2007/12/05

[3] J.- C. Lin and J.- M. Chen, "An Automatic Revised Tool for Anti-malicious Injection", *Proceedings of The Sixth IEEE International Conference on Computer and Information Technology (CIT'06)*, 2006.

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

[5] Y.W. Huang, F. Yu, C. Hang, C.H. Tsai, D.T. Lee, S.Y. Kuo, "Verifying Web applications using bounded model checking", *Proceedings of The 2004 International Conference Dependable Systems and Networks (DSN2004)*, Florence, Italy, June 2004.

[6] N. Jovanovic, C. Kruegel, E. Kirda, "Pixy: a static analysis tool for detecting Web application vulnerabilities", *Proceedings of The 2006 IEEE Symposium on Security and privacy*, p.6, May 2006.

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

[8] OWASP, “WebScarab Project.” http://www.owasp.org/webscarab/, visit on 2005/10/08

[9] SPI Dynamics, “Web Application Security Assessment.” *SPI Dynamics Whitepaper*, 2003.

[10] KaVaDo, “Application-Layer Security: InterDo 3.0”, *KaVaDo Whitepaper 2003*, Available from http://www.kavado.com/

[11] D. Scott, R. Sharp, “Abstracting Application-Level Web Security”, *Proceedings of The 11th International Conference on the World Wide Web*, Honolulu, Hawaii, pp.396-407, May 2002.

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

[13] S. Dharmapurikar, P. Krishnamurthy, T. Sproull, and J. Lockwood, “Deep Packet Inspection Using Parallel Bloom Filters”, *Proceedings of The 11th Symposium on High Performance Interconnects (HOTI’03)*, Stanford, California, pp.44-51, 2003.

[14] J.- C. Lin, J.- M. Chen and H.- K. Wong, “An Automatic Meta-revised Mechanism for Anti-malicious Injection”, *Proceedings of Network-Based Information Systems (NBiS) 2007*, LNCS 4658, pp. 98-107, 2007

[15] 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

[16] D. - J. Chen, C. - C. Hwang, S. - K. Huang, and D. T. K. Chen, "A Testing Framework for Web Application Security Assessment." *Journal of Computer Networks*, Vol.48, No.5, pp.739-761, June 2005.

[17] J. Offutt., Y. Wu., X. Du, H. Huang, “Bypass testing of Web applications, Software Reliability Engineering”, *Proceedings of The 15th International Symposium on ISSRE 2004*, pp.187-197, Nov. 2004.

[18] SoftByte Labs., Inc., Black window, http://www.softbytelabs.com/us/bw/index.html

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

[20] Verschiedenes infos, JoBo, http://www.matuschek.net/jobo/