International Journal of Enterprise Computing and Business Systems

ISSN (Online) : 2230-8849

http://www.ijecbs.com

Vol. 2 Issue 1 January 2012

WEB APPLICATION VULNERABILITY DETECTION USING DYNAMIC ANALYSIS WITH PENETERATION TESTING

Sreenivasa Rao B [1]
Dept. of Computer Science & Engineering
CMJ University, Shillong, India

Kumar N [2]
Dept. of Computer Science & Engineering
ACE college of Engineering and Management
Agra, India

Abstract: The number of reported web application vulnerabilities is increasing dramatically. The most of the vulnerabilities result from improper input validation. This paper presents extensions to the Tainted Mode Model (TMM) which allows inter module vulnerabilities detection. Besides, this paper presents a new approach to vulnerability

International Journal of Enterprise Computing and Business Systems

ISSN (Online) : 2230-8849

http://www.ijecbs.com

Vol. 2 Issue 1 January 2012

analysis which incorporates advantages of penetration testing and dynamic analysis. This approach effectively utilizes the extended Tainted Mode Model.

Web Application Penetration Testing (WAPT) can be defined as a legally authorized, non-functional assessment of a given web application, carried out to identify loopholes or weaknesses, or known as vulnerabilities. These vulnerabilities, exploited by a malicious user (attacker/hacker), may affect the confidentiality, integrity and availability (CIA) of the web application and/or information distributed by it.

Keywords: Web Application Security, Vulnerability Analysis, Penetration Testing, Dynamic Analysis, Taint Analysis, Vulnerability Detection.

1. INTRODUCTION

Security vulnerabilities in web applications may result in stealing of confidential data, breaking of data integrity or affect web application availability. Thus the task of securing web applications is one of the most urgent for now. The most common way of securing web applications is searching and eliminating vulnerabilities therein. Examples of another ways of securing web application include safe development, implementing Intrusion Detection Systems (IDS) and/or protection systems, and web application firewalls. According to OWASP (open web application security project), the most efficient way of finding security vulnerabilities in web applications is manual code review. This technique is very time-consuming, requires expert skills, and is prone to overlooked errors. Therefore, we (as a security group) actively develop automated approaches to

finding security vulnerabilities. These approaches can be divided into two wide categories: black-box and white-box testing.

The first approach is based on web application analysis from the user side, assuming that source code of an application is not available. The idea is to submit various malicious patterns (for example SQL injection or cross-site scripting attacks) into web application forms and to analyze its output thereafter. If any application errors are observed an assumption of possible vulnerability is made. This approach does not give guarantee neither accuracy nor completeness of the obtained results. The second approach is based on web application analysis from the server side, with assumption that source code of the application is available. In this case dynamic or static analysis techniques can be applied.

The most common model of input validation vulnerabilities is the Tainted Mode model. This model was implemented both by means of static or dynamic analysis.

Another approach is input validation vulnerabilities is to model syntactic structure for sensitive operations arguments. The idea behind this is that the web application is susceptible to an injection attack, if syntactic structure for sensitive operation arguments depends on the user input. This approach was implemented by means of string analysis in static and it was applied to detect SQLI and XSS vulnerabilities. After all, this approach was implemented to detect injection attacks at runtime.

One of the main drawbacks of dynamic analysis is that it is performed on executed paths and does not give any guarantee about paths not covered during a given execution.

However, dynamic analysis having access to internals of web application execution process has the potential of being more precise.

In this paper we focus on the most common model of input validation vulnerabilities. The contributions of this paper are the following:

- Improve classical Tainted Mode Model so that inter-module data flows could be checked.
- We introduce a new approach to automatic penetration testing by leveraging it with information obtained from dynamic analysis. Thus:
  - More accuracy and precision is achieved due to widened scope of web application view (database scheme and contents, intra-module data flows);
  - Input validation routines can be tested for correctness.

II. TAINTED MODE MODEL (TMM)

Dynamic and static analysis uses Tainted Mode model for finding security vulnerabilities that cause improper input validation.

According to, following assumptions were made within Tainted Mode Model:

1. All data received from the client via HTTP-requests is untrustworthy (or tainted).
2. All data being local to the web application is trustworthy (or untainted).
3. Any untrustworthy data can be made trustworthy by special kinds of processing, named sanitization.

International Journal of Enterprise Computing and Business Systems

ISSN (Online) : 2230-8849

http://www.ijecbs.com

Vol. 2 Issue 1 January 2012

With these assumptions made, security vulnerability is defined as a violation of any of the following rules:

1. Untrustworthy (tainted) data should not be used in construction of HTTP responses. This prevents cross site scripting attacks.
2. Untrustworthy (tainted) data should not be saved to local storages. This prevents possible construction of HTTP responses from these sources in future.
3. Untrustworthy (tainted) data should not be used in system calls and in construction of commands to external services such as database, mail, LDAP, etc. This prevents most of injection attacks.
4. Untrustworthy (tainted) data should not be used in construction of commands that would be passed as input to interpreter. This prevents script code injection attacks.

III. DYNAMIC ANALYSIS

Dynamic analysis is the testing and evaluation of a program by executing data in real-time. The objective is to find security errors in a program while it is running. Dynamic analysis empowers to identify and remediate security issues in their running web applications before hackers can exploit them. Dynamic analysis inspects applications the same way a hacker would attack them – providing the most accurate and actionable vulnerability detection available.

A. Dynamic Analysis Testing

A Dynamic Analysis test communicates with a web application through the web front-end in order to identify potential security vulnerabilities and architectural weaknesses in the web application. Unlike source code scanners, a dynamic analysis program doesn't have

access to the source code and therefore detects vulnerabilities by actually performing attacks.

A Dynamic Analysis can facilitate the automated detection of security vulnerabilities within a web application. A Dynamic Analysis test is often required to comply with various regulatory requirements. Dynamic Analysis can look for a wide variety of vulnerabilities, including:

- Input/ Output validation: (Cross-site scripting, SQL Injection, etc.)
- Specific application problems
- Server configuration mistakes/errors/ version

B. Input Validation and Sanitization

Web applications typically work by first reading some input from the environment (either provided directly by a user or by another program), then processing this data, and finally outputting the results. As stated, the program locations where input enters the application are referred to as sources. The locations where this input is used are called sinks. Of course, sources often take data directly from potentially malicious users, and the application can make little (or no) assumptions about the values that are supplied. Unfortunately, many types of sinks cannot process arbitrary values, and security problems may arise when specially crafted input is passed to these sinks. We refer to these sinks as sensitive fields.

An example of a sensitive sink is a SQL function that accesses the database. When a malicious user is able to supply unrestricted input to this function, this might be able to modify the contents of the database in unintended ways or extract private information that is not supposed to access. This security problem is usually referred to as SQL

International Journal of Enterprise Computing and Business Systems

ISSN (Online) : 2230-8849

http://www.ijecbs.com

Vol. 2 Issue 1 January 2012

injection vulnerability. Another example of a sensitive sink is a function that sends some data back to the user. In this case, an attacker could leverage the possibility to send arbitrary data to a user to inject malicious JavaScript code, which is later executed by the browser that consumes the output. This problem is commonly known as XSS vulnerability.

To avoid security problems, an application has to ensure that all sensitive sinks receive arguments that are well formed, according to some specification that depends on the concrete type of the sink. Because input from potentially malicious users can assume arbitrary values, the program has to properly validate this input. Therefore, the application checks the input for values that violate the specification. When such invalid values are found, a programmer has two options. The first option is to abort further processing: the application stops to handle the request and returns an error code to signal incorrect input. The second option is to transform the input value such that the altered value conforms to the input specification and no longer poses a security threat when passed to a sensitive sink. We denote the process of transforming the input to a representation that is no longer dangerous as sanitization. Typically, sanitization involves the removal of (meta)-characters that have a special meaning in the context of the sink, escaping these characters, or truncating the length of the input.

IV. PENTETERATION TESTING

A penetration test, occasionally called as pentest, is a method of evaluating the security of a computer system or network by simulating an attack from malicious outsiders (who do not have an authorized means of accessing the organization's systems) and malicious insiders (who have some level of authorized access). The process involves an active analysis of the system for any potential vulnerabilities that could result from poor

International Journal of Enterprise Computing and Business Systems

ISSN (Online) : 2230-8849

http://www.ijecbs.com

Vol. 2 Issue 1 January 2012

or improper system configuration, either known and unknown hardware or software flaws, or operational weaknesses in process or technical countermeasures. This analysis is carried out from the position of a potential attacker and can involve active exploitation of security vulnerabilities.

Penetration testing approach is based on simulation of attacks against web applications. Currently, penetration testing is implemented as black box testing.

A vulnerability assessment simply identifies and reports noted vulnerabilities, whereas a penetration test attempts to exploit the vulnerabilities to determine whether unauthorized access or other malicious activity is possible. Penetration testing typically includes application security testing as well as controls and processes around the applications.

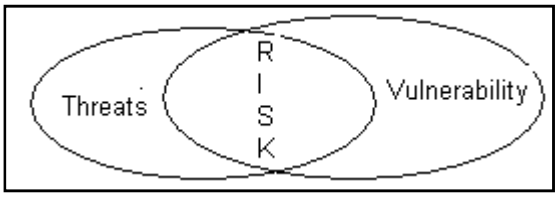

Figure 1: Visualization of Vulnerabilities in the application

International Journal of Enterprise Computing and Business Systems

ISSN (Online) : 2230-8849

http://www.ijecbs.com

Vol. 2 Issue 1 January 2012

The above diagram is useful to provide a visual perspective of how the assets in the organization are linked to risk. The left circle represents the universe of threats facing the organization. The right circle represents the vulnerabilities that have to be managed by virtue of the organizations infrastructure and network architecture configuration. There may only be a fraction of threats in the universe that present risk to the organization—as depicted by the intersection in this Venn diagram—and this is the set of threats in the local universe that we must manage.

Penetration tests are valuable for several reasons:

1. Determining the feasibility of a particular set of attack vectors.
2. Identifying higher-risk vulnerabilities that result from a combination of lower-risk vulnerabilities exploited in a particular sequence.
3. Identifying vulnerabilities that may be difficult or impossible to detect with automated network or application vulnerability scanning software.
4. Assessing the magnitude of potential business and operational impacts of successful attacks.
5. Testing the ability of network defenders to successfully detect and respond to the attacks.
6. Providing evidence to support increased investments in security personnel and technology.

A. Penetration Testing Methodology

Once the threats and vulnerabilities have been evaluated, design the penetration testing to address the risks identified throughout the environment.

International Journal of Enterprise Computing and Business Systems

ISSN (Online) : 2230-8849

http://www.ijecbs.com

Vol. 2 Issue 1 January 2012

The penetration testing should be appropriate for the complexity and size of an organization. All locations of sensitive data, all key applications that store, process, or transmit such data, all key network connections, and all key access points should be included. The penetration testing should attempt to exploit security vulnerabilities and weaknesses throughout the environment, attempting to penetrate both at the network level and key applications. The goal of penetration testing is to determine if unauthorized access to key systems and files can be achieved. If access is achieved, the vulnerability should be corrected and the penetration testing re-performed until the test is clean and no longer allows unauthorized access or other malicious activity.

B. Penetration Test Process

The Penetrator performs information discovery via a wide range of techniques that is, databases, scan utilities and more in order to gain as much information about the target system as possible. These discoveries often reveal sensitive information that can be used to perform specific attacks on a given machine.

International Journal of Enterprise Computing and Business Systems

ISSN (Online) : 2230-8849

http://www.ijecbs.com

Vol. 2 Issue 1 January 2012

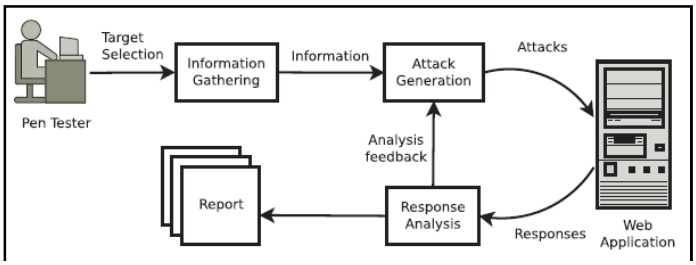

Figure 2: Penetration Testing Process

Vulnerability Identification: The vulnerability identification step is a very important phase in penetration testing. This allows the user to determine the weaknesses of the target system and where to launch the attacks.

Exploitation and Launching of Attacks: After the vulnerabilities are identified on the target system, it is then possible to launch the right exploits. The goal of launching exploits is to gain full access of the target system.

Denial of Service: A DOS (Denial of Service) test can be performed to test the stability of production systems in order to show if they can be crashed or not. When performing a penetration test of a preproduction system, it is important to test its stability and how easily can it be crashed. By doing this, its stability will be ensured once it is deployed into a real environment. It is important to perform DOS testing to ensure the safeness of certain systems. If an attacker takes down your system during busy or peak hours, both you and your customer can incur a significant financial loss.

Reporting: After the completion of the penetration test, it is important to get user-customized reporting suites for a technical and/or management overview. This

International Journal of Enterprise Computing and Business Systems

ISSN (Online) : 2230-8849

http://www.ijecbs.com

Vol. 2 Issue 1 January 2012

includes the executive summary, detailed recommendations to solve the identified vulnerabilities, and official security ID numbers for the vulnerabilities. The reports come in different formats such as html, pdf, and xml. Furthermore, all the reports are open to be modified as of the user's choice.

C. Managing the risks associated with Pen Test

Some of the key risks include the following while doing the penetration testing:

- The penetration test team may fail to identify significant vulnerabilities;
- Misunderstandings and miscommunications may result in the test objectives not being achieved;
- Testing activities may inadvertently trigger events or responses that may not have been anticipated or planned for (such as notifying law enforcement authorities);
- Sensitive security information may be disclosed, increasing the risk of the organization being vulnerable to external attacks.

Penetration testing is the most commonly applied mechanism used to gauge software security, but it's also the most commonly misapplied mechanism as well. By applying penetration testing at the unit and system level, driving test creation from risk analysis, and incorporating the results back into an organization's SDLC, an organization can avoid many common pitfalls. As a measurement tool, penetration testing is most powerful when fully integrated into the development process in such a way that findings can help improve design, implementation, and deployment practices.

International Journal of Enterprise Computing and Business Systems

ISSN (Online) : 2230-8849

http://www.ijecbs.com

Vol. 2 Issue 1 January 2012

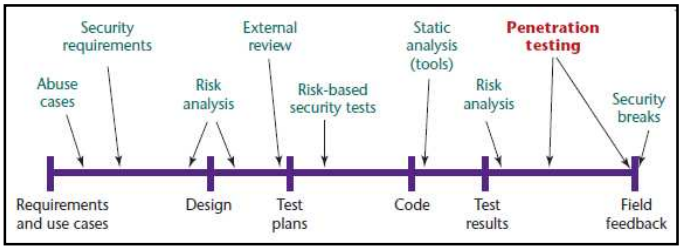

Figure 3: The software development life cycle (SDLC) with pen testing

V. CONCLUSION

This paper present an enhanced Tainted Mode Model that incorporates inter module data flows. We also introduced a new approach to automatic penetration testing by leveraging it with knowledge from dynamic analysis.

Number of reported web applications vulnerabilities is increasing dramatically. Most of them result from improper or none input validation by the web application. Most existing approaches are based on the Tainted Mode vulnerability model which cannot handle inter-module vulnerabilities.

Penetration testing is essential given the context of high operational risk in the financial services industry. Web-based and internal applications should be fully tested to ensure they do not provide an avenue of entry for attackers. Vulnerability management should be considered a priority given the sophisticated malware targeting client PCs inside the organization. Wireless vulnerabilities also add to the attack surface that can be exploited.

Penetration testing is the only legitimate means to identify residual risk that remains after code has been tested and operational and other threats have been minimized. To make

the most of penetration testing it is necessary to prioritize the effort. The penetration test should be scoped properly and should take advantage of the knowledge that the client organization has regarding exposures within their enterprise. And this information should be combined with the experience and insight of the penetration testing company.

The goal of penetration testing is to compromise a target system and ultimately steal information. Penetration testing is focused on finding security vulnerabilities in a target environment that could let an attacker penetrate the network or computer systems. A collaborative approach is recommended whereby the financial services organization and the penetration testing organization work together to more efficiently identify which exploits can be leveraged to steal information.

VI. REFERENCES

[1]. Halfond WGJ, Orso A. Improving test case generation for web applications using automated interface discovery. Proceedings of the Symposium on the Foundations of Software Engineering, Dubrovnik, Croatia, September 2007.

[2]. Halfond WGJ, Viegas J, Orso A. A classification of SQL-injection attacks and countermeasures. Proceedings of the International Symposium on Secure Software Engineering, Washington, DC, U.S.A., March 2006.

[3]. Christensen AS, Møller A, Schwartzbach MI. Precise analysis of string expressions. Proceedings of theInternational Static Analysis Symposium, San Diego, CA, U.S.A., June 2003; 1–18.

[4]. Halfond WGJ, Orso A, Manolios P. Using positive tainting and syntax-aware evaluation to counter SQL injection attacks. Proceedings of the Symposium on the Foundations of Software Engineering (FSE 2006), Portland, OR, U.S.A., November 2006; 175–185.

[5]. Halfond WGJ, Orso A, Manolios P. WASP: Protecting web applications using positive tainting and syntax-aware evaluation. Transactions on Software Engineering 2008; 34(1):65–81.

[6]. Halfond WGJ, Orso A. AMNESIA: Analysis and Monitoring for NEutralizing SQL-Injection Attacks. Proceedings of the International Conference on Automated Software Engineering, Long Beach, CA, U.S.A., November 2005; 174–183.

[7]. Halfond WGJ, Orso A. Combining static analysis and runtime monitoring to counter SQL-injection attacks. Proceedings of the International Workshop on Dynamic Analysis, St Louis, MO, U.S.A., 2005; 22–28.

[8]. Pietraszek T, Berghe CV. Defending against injection attacks through context-sensitive string evaluation. Proceedings of Recent Advances in Intrusion Detection, Seattle, WA, U.S.A., September 2005.

[9]. Gould C, Su Z, Devanbu P. Static checking of dynamically generated queries in database applications. Proceedings of the International Conference on Software Engineering, Edinburgh, Scotland, May 2004; 645–654.

[10]. Miller BP, Fredriksen L, So B. An empirical study of the reliability of UNIX utilities. Communications of the ACM 1990; **33**(12):32–44.

[11]. Sutton M, Greene A, Amini P. Fuzzing: Brute Force Vulnerability Discovery. Addison-Wesley: Reading, MA, 2007.

[12]. Thompson HH. Application penetration testing. Symposium Security and Privacy 2005; **3**(1):66–69.

[13]. Geer D, Harthorne J. Penetration testing: A duet. Proceedings of the Computer Security Applications Conference,Las Vegas, NV, U.S.A., December 2002; 185–195.

[14]. Arkin B, Stender S. McGraw G. Software penetration testing. IEEE Security and Privacy 2005; **3**(1):84–87.

[15]. Bishop M. About penetration testing. IEEE Security and Privacy 2007; **5**(6):84–87.

[16]. McAllister S, Kirda E, Kruegel C. Leveraging user interactions for in-depth testing of web applications. Proceedings of the International Symposium on Recent Advances in Intrusion Detection. Springer-Verlag: Berlin, Heidelberg, 2008; 191–210.

[17]. Martin M, Livshits B, Lam MS. Finding application errors and security flaws using PQL: A program query language. Proceeding of the Conference on Object Oriented Programming Systems Languages and Applications, San Diego, CA, U.S.A., October 2005; 365–383.

[18]. Huang YW, Yu F, Hang C, Tsai CH, Lee DT, Kuo SY. Securing web application code by static analysis and runtime protection. Proceedings of the International World Wide Web Conference, New York, NY, U.S.A., May2004; 40–52.

[19]. Wassermann G, Su Z. Sound and precise analysis of web applications for injection vulnerabilities. Proceedings of the Conference on Programming Language Design and Implementation. ACM: New York, NY, U.S.A., 2007; 32–41.

[20]. Wassermann G, Su Z. Static detection of cross-site scripting vulnerabilities. Proceedings of the International Conference on Software Engineering. ACM: New York, NY, U.S.A., 2008; 171–180.

[21]. Wassermann G, Yu D, Chander A, Dhurjati D, Inamura H, Su Z. Dynamic test input generation for web applications. Proceedings of the International Symposium on Software Testing and Analysis, Seattle, WA, U.S.A.,July 2008.

[22]. Su Z, Wassermann G. The essence of command injection attacks in web applications. Proceedings of the Symposium on Principles of Programming Languages, Charleston, SC, U.S.A., January 2006; 372–382.

[27]. Mesbah A, Bozdag E, van Deursen A. Crawling Ajax by inferring user interface state changes. In Proceedings of the International Conference on Web Engineering, Schwabe D, Curbera F, Dantzig P (eds.). IEEE Computer Society: Washington, DC, 2008; 122–134.

International Journal of Enterprise Computing and Business Systems

ISSN (Online) : 2230-8849

http://www.ijecbs.com

Vol. 2 Issue 1 January 2012

[28]. Mesbah A, van Deursen A. Invariant-based automatic testing of Ajax user interfaces. Proceedings of the 31st International Conference on Software Engineering (ICSE09) (Research Papers). IEEE Computer Society: Washington, DC, 2009;

[29]. http://www.OWASP.org

[30] C. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney,S. Wallace, V. J. Reddi and K. Hazelwood, "Pin: building customized program analysis tools with dynamic instru-mentation," Proc. of the ACM SIGPLAN Conf. on Programming Language Design and Implementation (PLDI), pp. 190–200, 2005.

[31] N. Nethercote and J. Seward, "Valgrind: a framework for heavyweight dynamic binary instrumentation," SIGPLAN Not., Vol. 42, No. 2, pp. 89–100, 2007.

[32] J. Newsome and D. Song, "Dynamic taint analysis for au-tomatic detection, analysis, and signature generation of ex-ploits on commodity software," Proc. of the 12th Sympo-sium on Network and Distributed System Security (NDSS),2005.

[33] F. Qin, C. Wang, Z. Li, H. Kim, Y. Zhou and Y. Wu,"LIFT: A Low-Overhead Practical Information Flow Track-ing System for Detecting Security Attacks," Proc. of the 39th Annual IEEE/ACM International Symposium on Mi-croarchitecture, pp. 135–148, 2006.