A Software Engineering Approach for Vulnerability Analysis

D. Rakesh[1], R. Vignesh[2]
Department of Computer Science and Engineering
Sri Sairam Engineering College, Anna University
Chennai, India

{[1]drakesh,[2]vigneshrr}@live.com

***Abstract*- Due to the increasing dependency on networked computer system, it is important to make a network reliable and dependent. This is even more relevant as new threats of attack are constantly being revealed, compromising the security of systems. This paper addresses this problem by presenting an attack injection methodology for the automatic discovery of vulnerabilities in software components. The proposed methodology, implemented in XDoS & TCP/IP, follows an approach similar to hackers and security analysts to discover vulnerabilities in network-connected servers. To assess the usefulness of this approach, several attack injections are made in POP and IMAP servers. XDetector uses a specification of the server's communication protocol. Then, while it injects these attacks through the network, it monitors the execution of the server in the target system and the responses returned to the clients. If any abnormality is detected, then the corresponding client's connection is terminated by the XDetector to prevent any damage to the server and the faulty client can be made secure using traditional debugging tools.**

***Keywords*- Software Engineering, Attack Injection, Testing and Debugging, XDoS, POP & IMAP.**

I. INTRODUCTION

Reliance on computer systems for everyday life activities has increased over the years, as more and more tasks are accomplished with their help. The advancements in software development have provided us with an increasing number of useful applications with an ever improving functionality. These enhancements, however, are achieved in most cases with larger and more complex projects, which require the coordination of several teams. Third party software, such as COTS components, is frequently utilized to speed up development, even though in many cases it is poorly documented and supported. In the background, the ever-present trade-off between thorough testing and time to deployment affects the quality of the software. These factors, allied to the current development and testing methodologies, have proven to be inadequate and insufficient to construct dependable software. Every day, new vulnerabilities are found in what was previously believed to be secure applications, unlocking new risks and security hazards that can be exploited by malicious adversaries.

The paper describes an attack injection methodology that can be used for vulnerability detection and removal. It mimics the behaviour of an adversary by injecting attacks against a target system while inspecting its execution to determine if any of the attacks has caused a failure. The observation of some abnormal behaviour indicates that an attack was successful in triggering an existing flaw. After the identification of the problem, traditional debugging techniques can be employed, for instance, by examining the application's control flow while processing the offending attacks, to locate the origin of the vulnerability and to proceed with its elimination. It is implemented by using an XML Denial Of Service (XDoS) attack in a common network using TCP/IP. To demonstrate the usefulness of our approach, 58 attack injection experiments with 16 e-mail servers running POP and IMAP services have been conducted.

II. USING ATTACKS TO FIND VULNERABILITIES

Vulnerabilities are usually caused by subtle anomalies that only emerge in such unusual circumstances that were not even contemplated in test design. They tend to elude the traditional software testing methods, mainly because conventional test cases mostly do not cover all of the obscure and unexpected usage scenarios. Hence, vulnerability is typically found

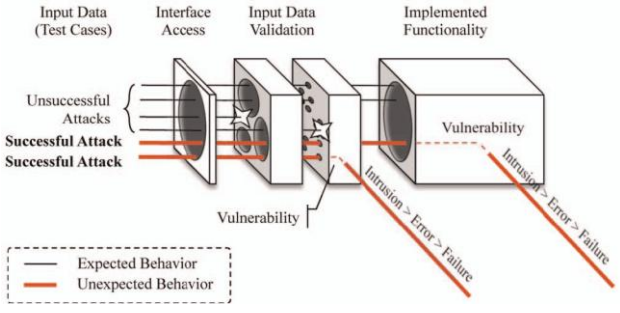

Fig 1a- Attack Injection Methodology

either by accident or by attackers or special tiger teams (also called penetration testers) who perform thorough security audits. The typical process of manually searching for new vulnerabilities is often slow and tedious specifically, the source code must be carefully scrutinized for security flaws or the application has to be exhaustively experimented with several kinds of input (e.g., unusual and random data, or more elaborate input based on previously known exploits) looking for problems during its execution.

Fig.1a shows a model of a component with existing vulnerabilities. Boxes in the figure represent the different modules or software layers that compose the component, with the holes symbolizing access being allowed (as intended by the developers or inadvertently through some vulnerability). Lines depict the interaction between the various layers. The same rationale can be applied recursively to any abstraction level of a component, from the smallest subcomponent to more complex and larger systems, terms component and system can be used interchangeably.

The external access to the component is provided through a known Interface Access, which receives the input arriving, for instance, in network packets or disk files, and eventually returns some output. Whether the component is a simple function that performs a specific task or a complex system, its intended functionality is, or should be, protected by Input Data Validation layers. These additional layers of control logic are supposed to regulate the interaction with the component, allowing it to execute the service specification only when the appropriate circumstances are present (e.g., if the client messages are in compliance with the protocol specification or if the procedure parameters are within some bounds). In order to achieve this goal, these layers are responsible for the parsing and validation of the arriving data. The purpose of a component is defined by its implemented functionality. This last layer corresponds to the implementation of the service specification of the component, i.e., it is the sequence of instructions that controls its behaviour to accomplish some well-defined objective, such as responding to client requests according to some standard network protocol. By accessing the interface, an adversary may persistently

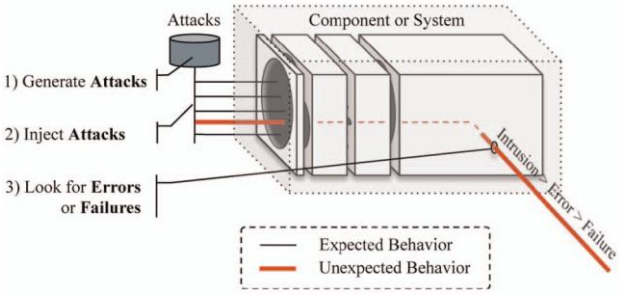

Fig 1b- Attack Injection Methodology

look for vulnerabilities by stressing the component with a dependable system should continue to operate correctly, even in the presence of these faults, i.e., it should keep executing in accordance with the service specification. However, if one of these attacks causes an abnormal behaviour of the component, it suggests the presence of vulnerability somewhere on the execution path of its processing logic. Vulnerabilities are faults caused by design, configuration, or implementation mistakes, susceptible to being exploited by an attack to perform some unintended and usually illegal activity. The component, failing to properly process the offending attack, enables the attacker to access the component in a way unpredicted by the designers or developers, causing an intrusion. This further step toward failure is normally succeeded by the production of an erroneous state in the system (e.g., a root shell). Consequently, if nothing is done to handle the error (e.g., prevent the execution of commands in the root shell), the system will fail.

III. THE ATTACK INJECTION METHODOLOGY

The attack injection methodology adapts and extends classical fault injection techniques to look for security vulnerabilities. The methodology can be a useful asset in increasing the dependability of computer systems because it addresses the discovery of this elusive class of faults. An attack injection tool implementing the methodology mimics the behaviour of an external adversary that systematically attacks a component, hereafter referred to as the target system, while monitoring its behaviour. An illustration of the main actions that need to be performed by such a tool is represented in Fig. 1b.

First, several attacks are generated in order to fully evaluate the target system's intended functionality (step 1). Restrictions apply on absence of vulnerabilities; the attacks have to be exhaustive and should look for as many classes of flaws as possible. It is expected that the majority of the attacks are deflected by the input data validation mechanisms, but others will be allowed to proceed further along the execution path, testing deeper into the component. Each attack is a single test case that

exercises some part of the target system, and the quality of these tests determines the coverage of the detectable vulnerabilities. Ideally, one would like to build test cases that would not only exercise all reachable computer instructions but also try them with every possible instance of input.

This goal, however, is unfeasible for most systems due to the amount of effort necessary to generate the various combinations of input data and then to execute them. The effort can be decreased by resorting to the analysis of the source code, and by manually creating good test cases. This approach requires a great deal of experience and acuteness from the test designers, and even then, some vulnerability can be missed altogether. In addition, source code might be unavailable because it is common practice to reuse general purpose components developed by third parties.

To overcome these limitations and to automate the process of discovering vulnerabilities, this paper proposes a method of generating a large number of test cases from a specification of the component's interface. The tool should then carry out the attacks (step 2) while monitoring how the state of the component is evolving, looking for any unexpected behaviour (step 3). Depending on its monitoring capabilities, the tool could examine the target system's outputs, its allocated system resources, or even the last system calls it executed. Whenever an error or failure is observed, it indicates that a new vulnerability has potentially been discovered. For instance, a vulnerability is likely to exist in the target system if it crashes during (or after) the injection of an attack—this attack at least compromises the availability of the system.

Likewise, if what is observed is the abnormal creation of a large file, this can eventually lead to disk exhaustion and subsequent denial-of-service, so it should be further investigated. The collected evidence provides useful information about the location of the vulnerability and supports its subsequent removal. System calls and the component responses, along with the offending attack, can identify the protocol state and the execution path to find the flaw more accurately. If locating and removing the vulnerability is unfeasible or a more immediate action is required, for instance, if the target system is a COTS component or a fundamental business-related application, the attack description could be used to take preventive actions, such as adding new firewall rules or IDS filters. By blocking similar attacks, the vulnerability can no longer be exploited, thus improving the system's dependability.

IV. MODULES

The proposed methodology implemented in XDoS and TCP/IP, follows an approach similar to actors and security analysis to discover vulnerability in network connected server. XDetector uses a specification of the server's communication protocol. Then, while it injects these attacks through network monitors the execution of the server in the target system and the response is returned to the client. It remains passive when there is no fault but it terminates the connection of the corresponding client with the server.

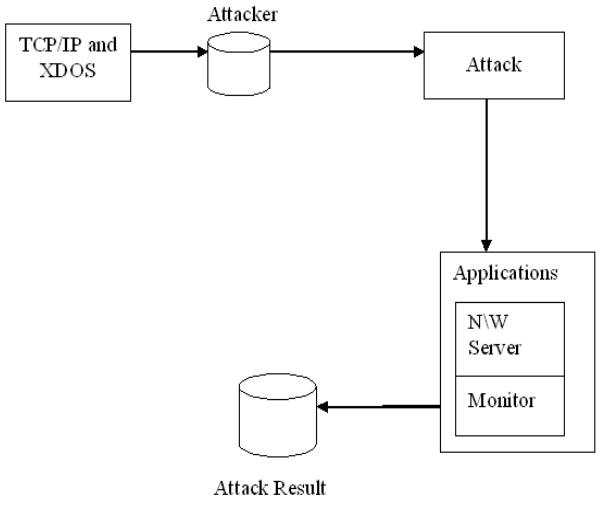

Fig 2-Architecture Diagram

A.XDoS and TCP/IP

The mainstay of the proposed framework is to create an Open Grid Services Architecture (OGSA) by employing Service Oriented Traceback Architecture (SOTA) in Conjunction with a filter defense system (XDetector) for an effective defense against XDoS. In fact, headlines about these new attacks can be seen in the coming days and months. These new attacks have been referred by researchers as XDoS (XML based DoS).The attacker would chose this new form of Denial of Service attack due to its simpler and devastating form against Web services. Each Client which is connected with the server is provided with an interface id, which uniquely identifies each client.

The Open Grid Services Architecture (OGSA) describes architecture for a service-oriented grid computing environment for business and scientific use, developed within the Global Grid Forum (GGF). OGSA is based on several other Web service technologies, notably WSDL and SOAP, but it aims to be largely agnostic in relation to the transport-level handling of data. Simple Object Access Protocol (SOAP) is defined to be a protocol specification for exchanging structured information in the implementation of Web Services in computer. Service Oriented Trace back Architecture (SOTA) provides a framework to be able to identify the source of an attack. This is accomplished by deploying

our defense system at distributed routers, in order to examine the incoming SOAP messages and place our own SOAP header. By this method, the new SOAP header information can be used to trace back through the network to determine the source of the attack. According to our experimental performance evaluations, SOTA is found to be quite scalable, simple and quite effective at identifying the source.

Once, the server identifies the attack. It checks first, whether were it comes from based on a ID information and it sends the response to the previous router. Over there it checks which source owns this ID based on TCP/IP to prevent the attack.

B. Attack Injector

The attack injection methodology adapts and extends classical fault injection techniques to look for security vulnerabilities. The methodology can be a useful asset in increasing the dependability of computer systems because it addresses the discovery of this elusive class of faults. An attack injection tool implementing the methodology mimics the behavior of an external adversary that systematically attacks a component, hereafter referred to as the target system, while monitoring its behavior. An illustration of the main actions that need to be performed by such a tool is represented in figure 2.

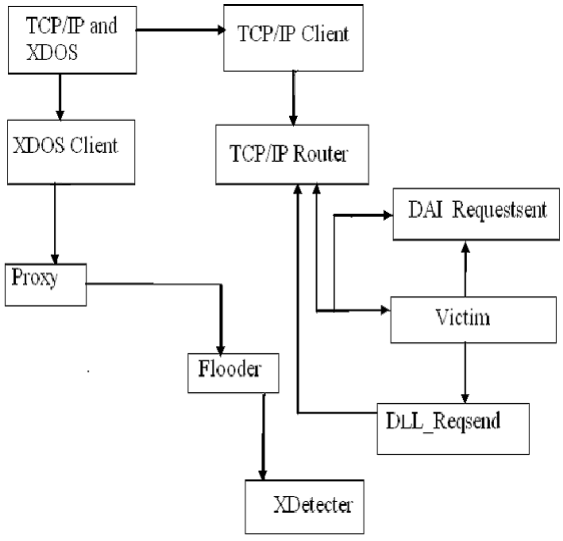

Fig 3-Transition Diagram

C. Target System and Monitor

The Target System is the entire software and hardware components that comprise the target application and its execution environment, including the operating system, the software libraries, and the system resources. The Network Server is typically a service that can be queried remotely from client programs. The target application uses a well-known protocol to communicate with the clients, and these clients can carry out attacks by transmitting erroneous packets. If the packets are not correctly processed, the target can suffer various kinds of errors with distinct consequences, ranging, for instance, from a slowdown to a crash. The Network Server Protocol Specification is a graphical user interface component that supports the specification of the communication protocol used by the server.

This specification is utilized by the Attack to produce a large number of test cases. The Attack Injector is responsible for the actual execution of the attacks by transmitting malicious packets to the server. It also receives the responses returned by the target and the remote execution profile collected by the Monitor. Some analysis on the information acquired during the attack is also performed to determine if vulnerability was exposed.

V. CONCLUSION

A methodology and a tool for the discovery of vulnerabilities in server applications is presented, which are based on the behavior of malicious adversaries. XDetector has been used to detect XDoS vulnerabilities in common network protocols such as TCP/IP. It detects vulnerabilities in clients and disconnects the corresponding client from the server to protect it against any attacks. The transition diagram for the same is shown in figure 3.

REFERENCES

[1] P. Verissimo, N. Neves, C. Cachin, J. Poritz, D. Powell, Y.Deswarte, R. Stroud, and I. Welch, "Intrusion-Tolerant Middleware:The Road to Automatic Security," IEEE Security and Privacy,vol. 4, no. 4, pp. 54-62, July/Aug. 1996.

[2] B. Beizer, Software Testing Techniques, second ed. Van Nostrand Reinhold, 1990.

[3] N. Neves, J. Antunes, M. Correia, P. Verissimo, and R. Neves,"Using Attack Injection to Discover New Vulnerabilities," Proc. Int'l Conf. Dependable Systems and Networks, June 2006.

[4] J. Myers and M. Rose, "Post Office Protocol Version 3," RFC 1939 (Standard), updated by RFCs 1957, 2449, May 1996.

[5] M. Crispin, "Internet Message Access Protocol Version 4rev1,"Internet Eng. Task Force, RFC 3501, Mar. 2003.

[6] J. Arlat, A. Costes, Y. Crouzet, J.-C. Laprie, and D. Powell, "Fault Injection and Dependability Evaluation of Fault-Tolerant Systems," IEEE Trans. Computers, vol. 42, no. 8, pp. 913-923, Aug. 1993.

[7] M.-C. Hsueh and T.K. Tsai, "Fault Injection Techniques and Tools," Computer, vol. 30, no. 4, pp. 75-82, Apr. 1997.

[8] J. Carreira, H. Madeira, and J.G. Silva, "Xception: Software Fault Injection and Monitoring in Processor Functional Units," Proc. Int'l Working Conf. Dependable Computing for Critical Applications,pp. 135-149, Jan. 1995.

[9] T.K. Tsai and R.K. Iyer, "Measuring Fault Tolerance with the FTAPE Fault Injection Tool," Proc. Int'l Conf. Modeling

Techniques and Tools for Computer Performance Evaluation, pp. 26-40, 1995.

[10] J. Christmansson and R. Chillarege, "Generation of an Error Set That Emulates Software Faults," Proc. Int'l Symp. Fault-Tolerant Computing, pp. 304-313, June 1996.

[11] J. Dura˜es and H. Madeira, "Definition of Software Fault Emulation Operators: A Field Data Study," Proc. Int'l Conf. Dependable Systems and Networks, pp. 105-114, June 2003