D.S. BHAVANI, G PRAVEEN BABU / International Journal of Engineering Research and Applications (IJERA) ISSN: 2248-9622 www.ijera.com
Vol. 1, Issue 4, pp. 2009-2012

REVIEW: IDENTIFYING THE BEHAVIOUR OF VULNERABITIES BY ATTACKING SERVER'S USING AJECT TOOL

D.S. BHAVANI[#1], G PRAVEEN BABU[*2]

[#1] *MTECH STUDENT, DEPARTMENT OF CSE, SIT, JNTUH, HYDERABAD, INDIA*
[*2] *ASSOCIATE PROFESSOR, DEPARTMENT OF CSE, SIT, JNTUH, HYDERABAD, INDIA*

ABSTRACT

Today's networked computer systems yield great value to businesses and governments, but also create risks. Vulnerabilities in computer systems lead to security and privacy risks. We are going to present how automatically we can detect the vulnerabilities in a software system which is similar to how hackers and security analysts do to discover vulnerabilities in network through this paper. This methodology is implemented in a tool called AJECT which uses a specification of the server's communication protocol and predefined test case generation algorithms to automatically create a large number of attacks. When the attack has been injected on to the network this tool monitors the server execution in the target system and responds to the client .When an unforeseen behaviour has been observed it is clear that the presence of vulnerability by a particular attack. This attack can then be used to reproduce the anomaly and to assist the removal of the error.

Keywords: **AJECT Tool, attack's, vulnerability, Network, server's.**

I.INTRODUCTION

Our reliance on computer systems for everyday life activities has increased over the years, as more and more tasks are accomplished with their help. Software evolution has provided us with applications with ever improving functionality. These enhancements however are achieved in most cases with larger and more complex projects, which require the coordination of several teams of people. Third party software is frequently utilized to speedup programming tasks, even tough in many cases it is poorly documented and supported. In the background, the ever present tradeoff between time to market and thorough testing puts pressure on the quality of the software. Experience has shown that some of the bugs result in security vulnerabilities that can be exploited by malicious adversaries. The existence of a vulnerability *presence* does not cause a security hazard, and in fact many times they can remain dormant for many years. An intrusion is only materialized when the right attack is discovered and applied to exploit that vulnerability. After an intrusion, the system might or might not fail, depending on the kind of capabilities it possesses to deal with errors introduced by the adversary. Sometimes the intrusion can be tolerated , but in the majority of the current systems, it leads almost immediately to the violation of one or more security properties .

Several tactics can be employed to improve the dependability of a system with respect to malicious faults. Of course, intrusions would never arise if all vulnerabilities could be eliminated. Vulnerability removal can be performed both during the development and operational phases. In the last case, besides helping to identify programming flaws which can later be corrected, it also assists the discovery of configuration errors and/or other similar problems. Intrusion prevention (e.g., vulnerability removal) has been advocated because it reduces the power of the attacker. In fact, even if the ultimate goal of zero vulnerabilities is never attained, vulnerability removal reduces the number of entry points into the system, making the life of the adversary increasingly harder (and ideally discouraging further attacks). Fig .1 shows a composite fault model which is used to demonstrate the vulnerability discovery methodology.

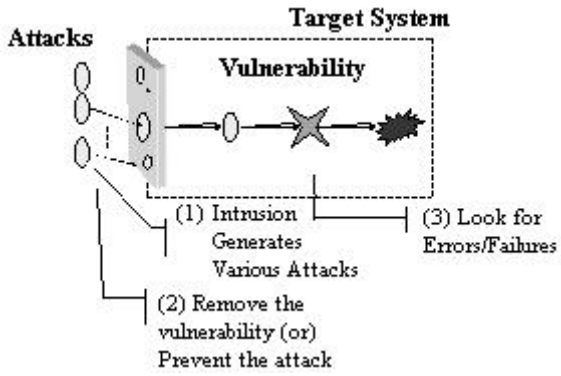

Fig .1 Composite fault model

II.OBJECTIVE OF PROPOSED SYSTEM

The objective is to manipulate get strings in a way that, you can print code directly as html into the page. Evaluating custom code for injection points. Retrieve pre-exploitation

reconnaissance for conditions. Compromise a users credentials using trust and redirects. Examine the actual exploitation of a vulnerable host. Examine the use of fuzzing tools for discovering vulnerabilities in applications. Examine how the output of a fuzzing tool might be used to develop software security exploits (case study). Describe the nature, types and associated methodologies of the various different classes of fuzzers. Briefly explain where fuzzers fit within the field of application security testing: i.e. who might use them, why they are used, and what value they over the Information Security industry, software developers, end-users, and attackers. Identify some of the limitations of, and problems with, fuzzing. Compare some of the available fuzzing tools and approaches available possibly using two or more types of fuzzer against a single target application with known vulnerabilities; examine the evolution of fuzzing tools, comment on the state of the art and the outlook for fuzzing; examine what metrics may be used to compare fuzzers. Comment on the influence of fuzzing on the information security and software development communities compare fuzzing with other forms of software security assurance - i.e. Common Criteria evaluations

III.BACKGROUND WORK

The paper describes a methodology and a tool for the discovery of vulnerabilities on services provided by network or local daemons, through the injection of attacks (i.e., malicious faults). This work has been influenced by several research areas, namely:

A) Fault Injection

It is an experimental approach for the verification of fault handling mechanisms (fault removal) and for the estimation of various parameters that characterize an operational system (fault forecasting), such as fault coverage and error latency . Traditionally, fault injection has been utilized to emulate several kinds of hardware faults, ranging from transient memory corruptions to permanent stuck-at faults. Three main techniques have been developed to inject these faults: hardware-based tools resort to additional hardware to actually introduce the faults in the system, in most cases through pin-level injection, but also through radiation and electromagnetic interference simulation models with different levels of abstraction, e.g., device and network, have been employed by simulation-based tools to study the behaviour of systems, starting from the early stages of design software-based tools insert errors in the various parts of a running system by executing specific fault injection code . The emulation of other types of faults has also been accomplished with fault injection techniques, for example, software and operator faults. Robustness testing mechanisms study the behaviour of a system in the presence of erroneous input conditions. Their origin comes both from the software testing and fault-injection communities, and they have been applied to various areas, for instance, POSIX APIs and device driver interfaces .

B) Vulnerability Scanners

These are tools whose purpose is the discovery of vulnerabilities in computer systems (in most cases network-connected machines). Several examples of these tools have been described in the literature, and currently there are some commercial products: COPS, Found Stone Enterprise, Internet Scanner, Nessus, and QualysGuard. They have a database of well known vulnerabilities, which should be updated periodically, and a set of attacks that allows their detection. The analysis of a system is usually performed in three steps:

Step 1: The scanner interacts with the target to obtain information about its execution environment (e.g., type of operating system, available services, etc).

Step 2: Then, this information is correlated with the data stored in the database, to determine which vulnerabilities have previously been observed in this type of system.

Step 3: later, the scanner performs the corresponding attacks and presents statistics about which ones were successful.

Even though these tools are extremely useful to improve the security of systems in production, they have the limitation that they are unable to uncover unknown vulnerabilities.

C) Static vulnerability analyzers

These look for potential vulnerabilities in the source code of the applications. Typically, these tools examine the source code for dangerous patterns that are usually associated with buffer overflows, and then they provide a listing of their locations. Next, the programmer only needs to go through the parts of the code for which there are warnings, to determine if an actual problem exists. More recently, this idea has been extended to the analysis of binary code. Static analysis has also been applied to other kinds of vulnerabilities, such as race conditions during the access of (temporary) files. In the past, a few experiments with these tools have been reported in the literature showing them as quite effective for locating programming problems. These tools however have the limitation of producing many false warnings, and skip some of the existing vulnerabilities.

D) Run-time prevention mechanisms

Change the run-time environment of programs with the objective of thwarting the exploitation of vulnerabilities. The idea here is that removing all bugs from a program is considered infeasible, which means that it is preferable to

contain the damages caused by their exploitation. Most of these techniques were developed to protect systems from buffer overflows. A few examples are: Stack Guard, Stack Shield, and Point-Guard are compiler-based tools that determine at runtime if a buffer overflow is about to occur, and stop the program execution before it can happen. A recent study compares the effectiveness of some of these techniques, showing that they are useful only to prevent a subset of the attacks .

IV.SYSTEM OVERVIEW

To detect and remove the vulnerabilities in this paper we describes a tool called *AJECT - Attack in-JECtion Tool* . AJECT calculates the behaviour of an antagonist by injecting attacks against a target system then it observes the execution of the system to determine if the attacks have caused a failure. In the positive case this indicates that the attack was successful, which reveals the existence of a vulnerability. After the identification of a, vulnerability we can use any traditional debugging techniques to inspect the application code and running environment and to find out the source of the vulnerability and allow its successive removal.

The existing version of AJECT mainly targets network server applications, although it can also be utilized with most local daemons. Considering the perspective of security to check for vulnerability we chose servers and off course they are most relevant components that need protection. Generally a hacker can intrude into a network only through server and then he uses all the facilities of a server like she or he immediately gains access to a local account, which can then be used as a launch pad for further attacks. The tool treats the server as a black box it does not need the source code of the server to perform the attacks. AJECT has to obtain a specification of the protocol implemented by the target server in order to generate smart attacks.

To demonstrate the usefulness of our approach, we have conducted a number of experimental attacks like Special char attack, Assignment attacks, and Query attack. The main intention was to show that AJECT could automatically discover a number of different vulnerabilities, which were described in bug tracking sites by various people. This experiment managed to prove that AJECT could detect many vulnerabilities and even able to discover a new vulnerability that was previously unknown to the security community.

As in Fig .2 in the architecture of AJECT there are four basic entities namely the *Target System*, the *Target Protocol Specification*, the *Attack Injector* and the *Monitor*. The Target System entity corresponds to the system which we want to test and the remaining three entities are the main components of AJECT. The target application and its execution environment are present in Target System which indeed have middleware libraries ,operating system and hardware configuration. Using client program the target application service can be invoked remotely. In addition to that it can also be a local daemon supporting a given task of the operating system. In both the cases, well-known protocol is used by the target application to communicate with the clients, and by transmitting malicious packets, these clients can carry out attacks to the server. If the packets are not correctly processed, the target can suffer various Kinds of errors with distinct consequences, ranging, for instance, from a slow down to a crash. The A graphical interface is provided to target Protocol Specification component for the specification of the communication protocol used by the target application. The Attack Injector is responsible for the generation and implementation of the attacks, and for receiving the responses returned by the target. It also performs some analysis on the information acquired during the attack, to determine if vulnerability was exposed.

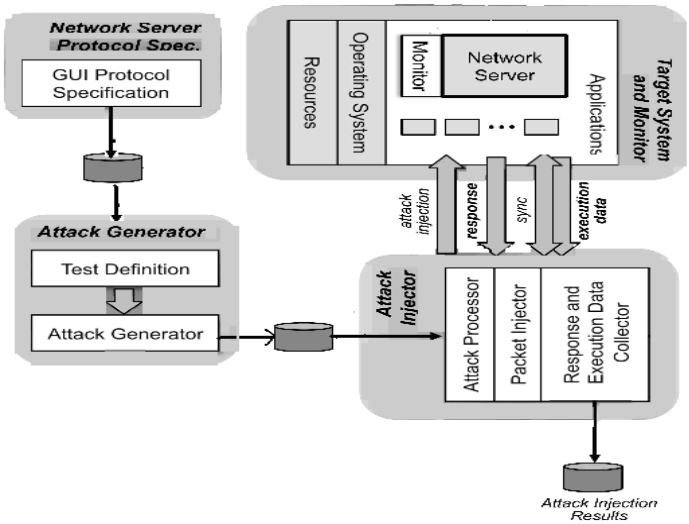

Fig .2 Architecture of AJECT

The main purpose of the Monitor is to examine and collect data about the target system execution, which requires a careful synchronization with the Injector.

V.CONCLUSION

In this paper we present a tool which is used to discovery and remove the vulnerabilities in server applications. AJECT simulates the behaviour of a malevolent challenger by injecting different kinds of attacks against the target server. In the process of simulating it collects various information of the server by observing the application. And that information which is collected is analyzed later to determine incorrect execution of the server, which is a strong indication that vulnerability exists. To demonstrate the usefulness of this approach, a considerable number of experiments were carried out with several attacks like Special char attack, Assignment attacks, and Query attack. These experiments indicate that AJECT could be utilized to locate and remove a significant number of distinct types of vulnerabilities.

REFERENCES

[1] B.P. Miller, L. Fredriksen, and B. So, "An Empirical Study of the Reliability of UNIX Utilities," Comm. ACM, vol. 33, no. 12, pp. 32-44, 1990.

[2] P. Oehlert, "Violating Assumptions with Fuzzing," IEEE Security and Privacy, vol. 3, no. 2, pp. 58-62, http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1423963, Mar./Apr. 2005.

[3] Univ. of Oulu, "PROTOS—Security Testing of Protocol Implementations," http://www.ee.oulu.fi/research/ouspg/protos/, 1999-2003.

[4] M. Sutton, "FileFuzz," http://labs.idefense.com/labs-software.php?show=3, Sept. 2005.

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

[6] Tenable Network Security, "Nessus Vulnerability Scanner," http://www.nessus.org, 2008.

[7] Saint Corp., "SAINT Network Vulnerability Scanner," http://www.saintcorporation.com, 2008.

[8] Qualys, Inc., "QualysGuard Enterprise," http://www.qualys.com, 2008.

[9] D. Wagner, J.S. Foster, E.A. Brewer, and A. Aiken, "A First StepTowards Automated Detection of Buffer Overrun Vulnerabilities,"Proc. Network and Distributed System Security Symp., Feb. 2000.

[10] E. Haugh and M. Bishop, "Testing C Programs for BufferOverflow Vulnerabilities," Proc. Symp. Networked and Distributed System Security, pp. 123-130, Feb. 2003.

[11] J. Dura˜es and H. Madeira, "A Methodology for the Automated Identification of Buffer Overflow Vulnerabilities in Executable Software without Source-Code," Proc. Second Latin-Am. Symp.Dependable Computing, Oct. 2005.

[12] M. Bishop and M. Dilger, "Checking for Race Conditions in File Accesses," Computing Systems, vol. 9, no. 2, pp. 131-152, Spring 1996.

[13] Wilander and M. Kamkar, "A Comparison of Publicly Available Tools for Dynamic Buffer Overflow Prevention," Proc. Network and Distributed System Security Symp., pp. 149-162, Feb. 2003.

[14] Microsoft, Corp., "A Detailed Description of the Data Execution Prevention (DEP) Feature in Windows XP Service Pack 2, Windows XP Tablet PC Edition 2005, and Windows Server 2003," http://support.microsoft.com/kb/875352, Sept. 2006.

AUTHORS

D S Bhavani received Bachelor's degree in Computer science and Engineering from JNTUH, Pursuing M.Tech in Computer Science and Engineering from JNTUH. She is a research scholar in field of Information Security. She can be reached at E-Mail: bavani.satya@gmail.comm

G. Praveen Babu is presently working as an Associate Professor of Computer Science & Engineering, at School of Information Technology, JNT University.He has a teaching experience of more than 9 years.Presently, pursuing Ph. D. In the area of Computer Networks.Areas of interest are computer Networks, Network Security, Analysis of Algorithms, Artificial Intelligence and Software Engineering Additionally, In-charge of the Placement and Training cell at School of Information Technology, JNTU, Hyderabad. He can be reached at Email : pravbob@jntuh.ac.in.