International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 1 Issue 6, August - 2012

Vulnerability Discovery with Attack Injection

Subbarao Gogulamudi ^{1}***M.Tech II year,***
Department of Computer Science & Engg,
Nova College of Engineering & Technology,
Affiliated to JNTU Kakinada,

Ch.Raj Jacob
Head of the Department, CSE
Nova College of Engineering & Technology
Affiliated to JNTU Kakinada,

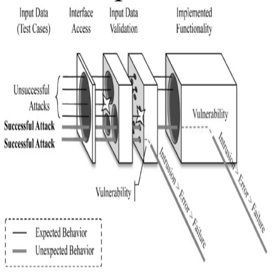
Abstract—The increasing reliance put on networked computer systems demands higher levels of dependability. This is even more relevant as new threats and forms of attack are constantly being revealed, compromising the security of systems. This paper addresses this problem by presenting an attack injection methodology for the automatic discovery of vulnerabilities in software components. The proposed methodology, implemented in AJECT, follows an approach similar to hackers and security analysts to discover vulnerabilities in network-connected servers. AJECT uses a specification of the server's communication protocol and predefined test case generation algorithms to automatically create a large number of attacks. 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. The observation of an unexpected behavior suggests the presence of a vulnerability that was triggered by some particular attack (or group of attacks). This attack can then be used to reproduce the anomaly and to assist the removal of the error. To assess the usefulness of this approach, several attack injection campaigns were performed with 16 publicly available POP and IMAP servers. The results show that AJECT could effectively be used to locate vulnerabilities, even on well-known servers tested throughout the years.

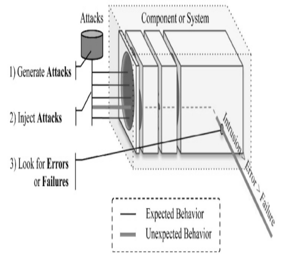
Index Terms—Testing and debugging, software engineering, test design, testing tools, experimental evaluation, fault injection, attack injection.

1 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. 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 soft-ware. Everyday, 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 behavior 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 behavior 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.

To demonstrate the usefulness of our approach, we have conducted 58 attack injection experiments with 16 e-mail servers running POP and IMAP services. The main objective was to investigate if AJECT could automatically discover previously unknown vulnerabilities in fully devel-oped and up-to-date server applications. Although the number and type of target applications was not exhaustive, they are nevertheless a representative sample of the universe of the network servers. Our evaluation confirmed that AJECT could find different classes of vulnerabilities in five of the servers, and assist the developers in their removal by providing the test cases, that is, the attack/ vulnerability/intrusion syndromes. These experiments also lead to other interesting conclusions. For instance, we confirmed the expectation that complex protocols are much more prone to vulnerabilities than simpler ones since all detected vulnerabilities were related to the IMAP protocol.





Additionally, based on the 16 e-mail servers, we found that closed source applications appear to have a higher predis-position to contain vulnerabilities (none of the open source servers was found vulnerable whereas 42 percent of the closed source servers had problems).

2 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 con-ventional test cases do not cover all of the obscure and unexpected usage scenarios. Hence, vulnerabilities are typically found 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.

3 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 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 Fig. 2.

4 THE ATTACK INJECTION TOOL

The attack injection methodology can be applied to any type of component that we wish to search for vulnerabilities. Several implementations of this methodology could be created to evaluate different types of targets, from simple COTS components to entire systems. In this study, we decided to focus on network servers because, from a security point of view, this is probably the most interesting class of target systems. First, these large and often complex applications are designed to sustain long periods of uninterrupted operation and are usually accessible through the Internet. Second, an intrusion in a network server usually has a significant impact since a corruption on the server may compromise the security of all clients (e.g., if the adversary gets a root shell). Consequently, network servers are a highly coveted target by malicious hackers. The Attack inJECtion Tool (AJECT) is a vulnerability detection tool that implements the proposed methodology. Its architecture and main components can be seen in Fig. 3. The architecture was developed to achieve automatic injection of attacks independently of the target server's implementation. Furthermore, it was built to be flexible regarding the classes of vulnerabilities that can be dis-covered and the method used to monitor the target system.

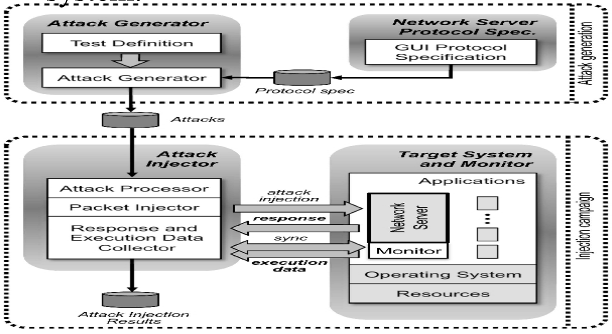


Fig. 3. The architecture of the AJECT tool.

4.1 Attack Generation Phase

The purpose of attack generation is to create a series of attacks that can be injected in the target system. The design of the tool does not require the source code of the server to be available to devise the attacks. This allows AJECT to support a larger number of target systems, such as commercial servers. Instead, the tool employs a specification of the communication protocol of the server, which, in practice, characterizes the server's external interface to the clients. Therefore, by exploring the input space defined by the protocol, it is possible to exercise much of the intended functionality of the target, i.e., the parts of the code that are executed when processing the clients' requests. In contrast to the source code, which is often inaccessible, communication protocols tend to be reasonably well documented, at least for standard servers (e.g., the Internet protocols produced by IETF).

4.1.1 Delimiter Test Definition

This specific type of test creates messages with illegal or missing delimiters of a field. For example, on text-based protocols, each field is delimited by a space character and, usually at the end of the messages, there are carriage return and line feed characters. The attack generation algorithm cycles through all message specifications of the protocol and generates copies of each message with small variations of their delimiters, such as messages without one of the delimiters or messages with some delimiters replaced by a predefined set of illegal delimiter characters. Note that, with the exception of the delimiters, all generated messages will contain only legal data taken from the specification. Moreover, one can use any custom data to experiment with as illegal delimiters.

4.1.2 Syntax Test Definition

This kind of test generates attacks that infringe on the syntax of the protocol. The currently implemented syntax violations consist on the addition, elimination, or reordering of each field of a correct message. Note that, as with the previous algorithm, the field specifications are kept un-changed, i.e., they only hold valid values. Like all other test definitions, after generating new message specifications (i.e., variations from the original ones), each specification will result in several test cases, each one trying a different combination of possible field data.

As an example, consider a message containing two different fields (e.g., a command with one parameter) represented as [A] [B]. Below are depicted some of the variations of the original message specification from which test cases are going to be created:

- [A] (removed field [B]),
- [B] [B] (duplicated field [B]), and
- [B] [A] (swapped fields).

4.1.3 Value Test Definition

This test determines if the server can cope with messages with bad data. For this purpose, a mechanism is used to derive illegal data from the message specification, in particular, from each field's specified legal data. Ideally, one would like to experiment with all possible illegal values; however, this proves to be unfeasible when dealing with a large number of messages and fields with arbitrary textual content. To overcome such an impossibility, a heuristic method was conceived to reduce the number values that have to be tried (see Pseudocode 1). The algorithm has the following structure: All states and message types of the protocol are traversed, maximizing the protocol space; then each test case is generated based on one message type. This algorithm differs from the others because it systematically populates each field with wrong values, instead of only resorting to the legal values.

Creating illegal words, however, is a much more complex problem because there are an infinite number of character combinations, making such an exhaustive approach impossible. Our objective was to design a method to derive potentially desirable illegal words, i.e., words that are usually seen in exploits, such as large strings or strange characters (see Pseudo code 2, which is called in the under-lined line of Pseudo code 1). Basically, this method produces illegal words by combining several tokens taken from two special input files. One file holds malicious tokens or known expressions, collected from the exploit community, piously defined by the operator of the tool (see Fig. 7). AJECT expands the special keyword $(PAYLOAD) with each line taken from another file with payload data. This payload file could be populated with already generated random data, long strings, strange characters, known

```
TestValue
 Input: $Protocol$ ← specification of the network protocol used
        by the server
 Output: $Attacks$

 $S$ ← set of all states of the $Protocol$ specification
 foreach State $s \in S$ do
   $M$ ← set of message specifications of $s$
   $Transition_s$ ← set of ordered network packets necessary
         to reach $s$
   $P$ ← $\phi$

   foreach MessageSpecification $m \in M$ do
     foreach FieldSpecification $f \in m$ do
       if $f$ is type Numbers then
         $f'$ ← all boundary values, plus some intermediary
              illegal values, from $f$ specification
       elseif $f$ is type Words then
         $f'$ ← combin. of predefined malicious tokens

       $m'$ ← copy of $m$ replacing $f$ with $f'$
       $P$ ← $P \cup$ {set of all network packets based on $m'$
            specification}

   foreach $attack\_packet \in P$ do
     $attack$ ← $Transition_s \cup \{attack\_packet\}$
     $Attacks$ ← $Attacks \cup \{attack\}$

 return $Attacks$
```

usernames, and so on. The resulting data combinations from both files are used to define the illegal word fields.

5 EXPERIMENTAL FRAMEWORK

This section provides the details about the laboratory conditions in which the experimental evaluation took place. It includes a description of the network server protocols that were specified and tried with AJECT and the tested configuration.

5.1 Network Server Protocols

The experimental evaluation was designed to assess the advantages of using attack injection to discover vulnerabilities in real applications. For that purpose, we chose fully developed and commonly used standard protocols, POP3 [4] and IMAP4Rev1 [5], instead of obscure or immature proto-cols that are typically implemented in applications with reduced levels of testing and utilization. Therefore, finding bugs in our targets will usually be hard because applications have gone through several revisions, where all vulnerabilities had an opportunity to be removed. Additionally, the selected protocols are not overly complex, leading to much simpler and less error-prone implementations.

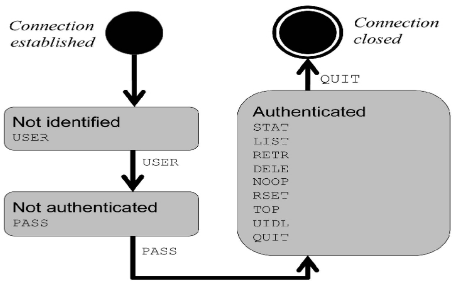


Fig. 5. Finite state machine of the protocol POP3

5.1.1 POP Protocol

The Post Office Protocol (POP) is a widely used protocol for e-mail retrieval. It was designed to allow users without a permanent connection to remotely view and manipulate messages. POP3 servers listen on port number 110 for incoming connections, and use a reliable data stream (TCP) to ensure the transfer of commands, responses, and message data All interactions between the client and the server are in the form of text strings that end with Carriage Return and Line Feed (CRLF) characters. Client messages are case-insensitive commands, with three or four letters long, followed by the respective parameters. Server replies are prefixed with +OK or -ERR, indicating a successful unsuccessful command completion.

7. AUTHORS

G.Subbarao[1] received B.Sc degree in Physics from Acharya Nagarjuna University, Guntur, in 2006, received M.C.A. Degree from JNTU, Kakinada, in 2009, He is currently pursuing **M.Tech** in Computer Science & Engineering at Nova College of Engineering &Technology, Veagavarum which is affiliated under **JNTU** Kakinada. He published one National Level Conference Paper and his areas of interests are **Networking & Data Warehousing, Software Engineering, Operating systems.**

Mr.Ch.RajaJacob[2], well known Author and excellent teacher Received M.C.A and **M.Tech (CSE)** from Acharya Nagarjuna university is workings as Associate Professor and **HOD,**Department of **MCA**, **M.Tech** Computer Science engineering, Nova College of Engineering and Technology, He is An active member of ISTE. He has 7Years of teaching experience in various Engineering colleges. To his credit Couple of publications both national and International conferences /journals. His area of Interest includes Data Warehouse and Data Mining, information security, flavors of Unix Operating systems and other advances in computer Applications

8 CONCLUSION

Our evaluation confirmed that AJECT could detect different classes of vulnerabilities in e-mail servers and assist the developers in their removal by providing the required test cases. The 16 servers chosen for the experiments were fully patched and up-to-date applications and most of them had gone through many revisions, making them challenging targets. In any case, AJECT successfully discovered vulnerabilities in five servers, which corresponded to 42 percent of all tested commercial applications.

REFERENCES

[1] P. Verissimo, N. Neves, C. Cachin, J. Poritz, D. Powell, Y. Deswarte, R. Stroud, and I. Welch, “Intrusion-Tolerant Middle-ware: 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, http://www. ietf.org/rfc/rfc1939.txt, May 1996.

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