Locating File Processing Vulnerabilities

Nuno Ferreira Neves
Fac. de Ciências da Univ. de Lisboa
nuno@di.fc.ul.pt

Abstract

An application is vulnerable to attacks if it exhibits incorrect behavior while it reads and processes the contents of a specially crafted malicious file. These vulnerabilities are often caused due to programming bugs in the routines that parse and utilize the fields of a file. In this paper, we propose a solution for the discovery of this kind of vulnerabilities, using an approach that is based on attack injection.

1. Introduction

Computer security is an important research subject due to our reliance on computer systems for the execution of our everyday life activities. In the near future, this dependency will tend to increase as more and more tasks will be done with the help of computers and through open networks (e.g., e-commerce, e-government, e-health). Current experience, however, shows that it is extremely difficult to build completely secure general purpose applications. For instance, the statistics published by CERT for the last two decades indicate an exponential grow on the number of incidents that have been reported [1].

An attack to be carried out in a successful manner, and to result in an intrusion, has to be able to explore a vulnerability in the computer system. These vulnerabilities are of different kinds, such as buffer overflow, format strings, information disclosure, race conditions, and (distributed) denial of service [2]. They can also be located in distinct components, ranging from the processor firmware to some library linked to an application. Many causes can explain why these flaws end up being inserted, for example an incorrect configuration parameter, an ill defined relation between components, or simply bad programming habits.

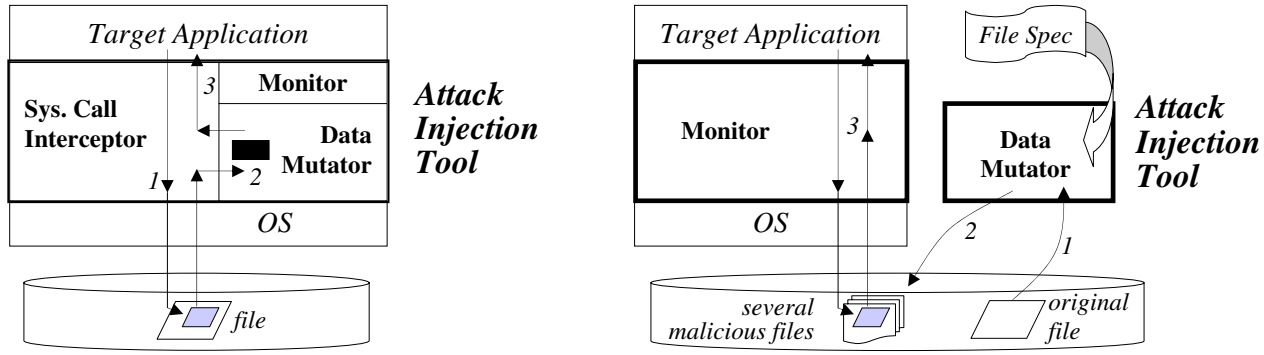
Traditionally, the most damaging vulnerabilities are the ones that can be exploited remotely (and that allow immediate access to an account with administrative privileges). An attack normally consists on the transmission of some erroneous data, for example, in a specially crafted packet that is sent by a malicious client to a target server application. The vulnerability is then activated when the server receives the packet and processes the data.

In the recent years, a new trend of remotely exploited vulnerabilities has been observed. In order to compromise an application, the adversary uses an indirect path to perform the attack. First, she or he generates a file of a given type (e.g., PDF) containing some bad data. Next, this file is posted in a web site waiting to be downloaded (or is distributed through email messages or IRC channels). And finally, the vulnerability is activated when the file is obtained by the target system, and is read by the flawed application.

Just during this first quarter of the year, several high profile *file processing vulnerabilities* have already been disclosed. They have appeared in a reasonable range of file types and applications, including Windows Meta Files (WMF) [8], Image files (TGA, TIFF, GIF, BMP) [5, 7], Playlist files (PLS or M3U) [9], Excel files (XLS) [6], and Flash files (SWF) [4]. Two worrying aspects of this kind of vulnerability are: it is difficult to prevent the attacks with network perimeter firewalls because malicious files come through usually open channels (e.g., web); and, they can be exploited even in the "more protected" machines of the internal networks.

In our research, we intend to develop tools that can help to automate the discovery of file processing bugs. The approach that will be followed is based on emulating the behavior of a malicious adversary when it tries to find a vulnerability, and that we have generically named as *attack injection* [3]. Basically, the tool automatically generates a large number of attacks that it directs against the target application. In parallel, it monitors the application, trying to observe incorrect behavior during the execution. Whenever such erroneous behavior is seen, it indicates with high probability that a vulnerability was activated by one of the attacks. At this point, and in order to determine exactly if a flaw exists, traditional debugging techniques can be employed to examine the application code and running environment.

This work was partially supported by the EC through project IST-2004-27513 (CRUTIAL), and by the FCT through projects POSC/EIA/61643/2004 (AJECT) and the Large-Scale Informatic Systems Laboratory (LASIGE).



a) Online mutation

b) Offline mutation

Figure 1. Attack injector with online and offline malicious file mutation.

2. The Attack Injection Tool

The *attack injection tool* is run in the same machine as the target application, and it is composed by two main modules, the Mutator and the Monitor (see Figure 1). The target application sees an execution environment that is equivalent to the normal case, with the exception that a specific file will contain malicious contents. For example, if one wants to test the GIF processing routines of an image viewer, then the GIF file that is read by the application will have some bad data.

The Mutator is in charge of generating the attacks. To accomplish this task, it modifies the contents of a valid file with some erroneous data, and then lets the target application read and process the data. In order to discover vulnerabilities, a large number of tests has to be carried out, each one corresponding to a malicious data insertion. The Mutator has to alter the various fields of a file in systematic a way, to ensure that all file processing activities of the target end up being examined.

We envision two possible ways to implement the Mutator. In the *online mutation*, the changes are done in real time, as the application reads the various parts of the file (see Figure 1 a)). The file related system calls of the local operating system (OS) will be intercepted by the tool, to allow the modification of the data that is read from the disk. In the *offline mutation* solution, the mutator receives as input a correct file, and then it produces a large set of erroneous files (see Figure 1 b)). Each one of these files has a specific change, and is used in a distinct test. To facilitate the generation of the data changes, the mutator can use a specification of the type of file that is being considered (e.g., the spec of a GIF file).

The Monitor determines if the attacks are successful. First, it sets up the experimental environment by initiating the application under the right conditions (e.g., ensures that the file system is reset and that the correct command line arguments are utilized). Second, it observes the execution of the application and looks for abnormal use of the system´s resources, trying to determine if errors are being produced. These errors would be a sign that a vulnerability was activated by the current attack. Third, it archives all relevant data that was collected, so that later an analysis can be performed.

Due to space limitations, it is not possible to describe in more detail the various mechanisms that are needed to successfully implement an efficient Mutator and Monitor. The interested reader can consult our paper on attack injection to get some ideas on these mechanisms [3].

References

[1] CERT Coordination Center. Statistics 1988-2005, Dec. 2005. http://www.cert.org/stats/.

[2] J. Koziol, D. Litchfield, D. Aitel, C. Anley, S. Eren, N. Mehta, and R. Hassel. *The Shellcoder's Handbook: Discovering and Exploiting Security Holes*. Wiley Publishing Inc, 2004.

[3] N. Neves, J. Antunes, M. Correia, P. Veríssimo, and R. Neves. Using attack injection to discover new vulnerabilities. In *Proc. of the Int. Conference on Dependable Systems and Networks*, July 2006.

[4] US-CERT. Adobe Macromedia Flash Products Contain Vulnerabilities. Technical Cyber Security Alert TA06-075A, Mar. 2006.

[5] US-CERT. Apple QuickTime Vulnerabilities. Technical Cyber Security Alert TA06-011A, Jan. 2006.

[6] US-CERT. Microsoft Office and Excel Vulnerabilities. Technical Cyber Security Alert TA06-073A, Mar. 2006.

[7] US-CERT. Microsoft Windows, Windows Media Player, and Internet Explorer Vulnerabilities. Technical Cyber Security Alert TA06-045A, Feb. 2006.

[8] US-CERT. Update for Microsoft Windows Metafile Vulnerability. Technical Cyber Security Alert TA06-005A, Jan. 2006.

[9] US-CERT. Winamp Playlist Buffer Overflow. Technical Cyber Security Alert TA06-032A, Feb. 2006.