Vulnerability Scanning for Buffer Overflow

Aishwarya Iyer
Graduate student
aish112@cs.nmt.edu

Lorie M. Liebrock
Assistant Professor
liebrock@cs.nmt.edu

Computer Science Department
New Mexico Tech, New Mexico

Abstract

With new applications being launched each day, the number vulnerabilities to attack increases. Attackers find new ways to attack any application they come across. Some attacks are successful because the application is vulnerable due to the vulnerabilities in files assisting in the proper functioning of the application. Although the area of attacks is vast, buffer overflow attacks form the basis for most attacks. Hence it would be a great boon to the Software Industry if their products were made secure from buffer overflow attacks. This paper introduces a tool, which is used to locate vulnerable files, which in turn have been the root cause for buffer overflow.

1. Introduction

Buffer overflow attacks comprise of over 50% of the attacks. If buffer overflow could be avoided, then most of the attacks based on buffer overflow become ineffective.

The focus of this paper is how to detect and avoid buffer overflow attacks, while finding out which file and function allowed the attack.

A few tools, which have been previously implemented, namely, StackGuard, Libsafe, and Janus have been used as references for the tool implemented in this paper.

2. S-tool

2.1: First Stage of the S-tool – Running the application: The application under test is run under normal circumstances. At the same time, a record is made of the stack trace through the entire execution of the application. There are many different ways in which a stack trace could be made for an application, for example:

a. *When the position of the return address of a function is required:* When a module/ function is started, its return address and its local variables are stored in the stack. In .Net[1], the VarPtr of a local variable returns the address of the location of the variable on the stack. Once the address of a local variable is obtained, it is possible to obtain the return address of that function. This can be done by trial and error. The distance of the return address from the local variable can be tracked.

b. *When all the contents of the stack are required:* Using the StackTrace function in .Net, the entire contents of the stack can be obtained. The entire image of a stack can be obtained by collecting StackFrame objects in each subroutine and function.

c. *When only the control flow through the application is required:* Using the StackFrame objects, the module name and line number from where the module is called can be obtained.

Since it is sufficient that the control flow through the application is known (to see if it is vulnerable to an attack), the method in *c* is followed in S-tool.

2.2: Second Stage - Run the application again and apply an attack: The application is executed again, but with the injection of a buffer overflow attack.

2.3: Compare the two stack traces: When comparing the two stack traces, several cases must be considered:

Case 1: *When the stack traces are similar*

The stack traces are said to be similar when both traces follow the same execution path. If this happens, the attack was not successful.

Case 2: *When the stack traces are different*

Stack traces are different if the execution paths are different or if the return from any function does not happen at the same time in both traces. From this, there are more sub-cases to consider:

1. When the execution paths are different in the two cases, it could be concluded that the attack was a success and that the return address on the stack has been modified. In other words, when the return from a function does not happen, then it means that the attack was successful.
2. When the return from a function is delayed, two more cases result.

a. The delay in the return of the function was likely due to an exception/error handler being executed. For example, consider a situation where the attack code inputs a long string so that the stack would overflow. If the application reads and possibly stores elsewhere the extra characters, but does not overflow the declared storage, the application avoids the attack. Therefore, the application is secure against this particular attack.

b. The delay could also have been caused by an exceptionally advanced attack which will record the return address, overwrite it and insert the return address elsewhere and hence allowing execution of the function to continue. This sort of attack has not yet been witnessed.

The difference between whether the change in execution path is due to an attack or an exception handler can be seen on the stack traces where the method names and line numbers would be displayed.

Now that it is known whether an attack has occurred or not, the next step is to trace the vulnerable area, in case of an attack.

2.4: Locate the vulnerable area in the Application: As its name implies, the StackTrace object keeps track of all the procedures that are waiting for the current one to complete. The StackTrace object can also get information such as the caller of the routine, File name, Line number, etc., which would be useful in locating the vulnerable area in the program. Using the location where the two stack traces differ and the method name and line number at that particular point, the vulnerable area can be located.

Once the vulnerable area has been identified, the programmer(s) can repair that portion of code and the tester(s) can repeat the above process again with different kinds of Stack Overflow attacks. This process goes on until the application is considered to be secure enough.

3. Implementation

The .Net environment was chosen to implement the S-tool. It is a menu-based tool and has the following menu items

- File
- Attacks
- Demonstration
- Compare
- Documentation

Some of the menu items are briefed below.

3.1: Attacks

This menu consists of the different types of attacks that could be made on an application in a particular language. As an example, one of the attacks discussed is as follows:

String Copy - strcpy attack

The vulnerability in strcpy function is the most common form of buffer overflow attack. Defining strings of different lengths and copying the longer string into the shorter one will result in an overflow.

The stack trace is invoked at the beginning and at the end of every module to keep track of what stage the execution is at. The stack trace code is actually written in a separate header file and is invoked when necessary.

3.2: Compare

The user must click on the *compare* menu. The user is then asked which two files (containing the stack traces) are to be compared. When given the file names along with the parent folder, the Compare executable compares the file contents and displays the result. The result will contain the module name and the line number where the vulnerability lies in the application.

4. Results

Three simple programs were written to test the concept of the S-tool and again the string copy function result is discussed.

The string copy application is run from the tool and execution is as follows:

1. The user is prompted for the text file name where the stack trace is to be stored.
2. The stack trace through the program is displayed.
3. The user is asked for the string input.
4. When a short string is entered, the rest of the execution takes place as planned and the stack trace is stored in a file.

The next step is to run the application again but an attack should now be made. The application execution is the same as described previously, but a string of length greater than the declared buffer is given as input. The resultant stack trace is stored in another text file. The comparison is then done to show where the vulnerability lies.

In general, the vulnerability lies between the location indicated by the tool and the previous stack trace call location. If this is a large section of code, the user may want to insert more stack trace calls and rerun the tool.

5. Conclusions and Future Work

The vulnerable area of an application code can be detected by S-tool and hence the applications can be made more secure. S-tool is most useful to application developers for testing the software security.

An extension to this work could be to do the scanning of vulnerabilities for other kinds of known attacks like input validation, SQL injection, et cetera.

5. References

1. Darin Haggins Emulate VB.NET Error Handling http://www.fawcette.com/archives/premier/mgznarch/vbpj/2001/09sep01/bb0109/bb0109.asp