FyLasso Antivulnerability

William B. Kimball (wk@fylasso.com)

April 2007

Abstract

FyLasso Antivulnerability is a software utility, developed for Windows XP and Windows Server 2003, which protects from 0-days, worms and malicious hackers. FyLasso uses a periodically updated Attack Vectors Database to search your computer for potential software vulnerabilities. Every potential vulnerability found is protected from exploitation using FyLasso's Proactive Real-time Defense System (PRDS). The purpose of this paper is to provide a technical explanation of FyLasso and PRDS.

1 What is the problem being solved?

The software running on todays computer systems contain bugs. These software bugs are errors during programming or design of an application [2]. Some types of bugs, called software vulnerabilities, are exploited by worms and malicious hackers. Software vulnerabilities allow arbitrary code execution to occur resulting in the primary reason why computers are insecure. The following sections explain how FyLasso Antivulnerability (FyLasso) detects potential software vulnerabilities and prevents arbitrary code execution from occuring thus protecting computers from attack.

2 The Attack Vectors Database

FyLasso organizes software bugs into attack vectors. Each attack vector contains a unique set of adjacent assembly instructions, consisting of a variety of buffer overruns, buffer underruns and integer overflows. One of the most commonly attacked bug used to penetrate into computer systems is the buffer overflow [7, 10]. Listing 1 contains three examples of common buffer overflow attack vectors.

Listing 1: Example Buffer Overflow Attack Vectors

```
004030DC        mov     ch,byte[eax]
004030DE        mov     byte[ebx],ch
004030E0        inc     eax
004030E1        inc     ebx
004030E2        test    ch,ch
004030E4        jnz     004030DC

004012F0        rep     movsd
004012F2        mov     ecx,eax
004012F4        and     ecx,3
004012F7        rep     movsb

00400087        mov     dl,byte[eax]
00400089        mov     byte[ecx+eax],dl
0040008C        inc     eax
0040008D        test    dl,dl
0040008F        jne     00400087
```

2.1 How many attack vectors are there?

The Attack Vectors Database increases monontonically. As techniques emerge to target new software bugs, FyLasso adds new attack vectors to protect them. The Limited Edition of FyLasso Antivulnerability scans for potential vulnerabilities using 300 attack vectors while the Professional Edition detects and protects using over 1,000. The Professional Edition also has the capability to update the Attack Vectors Database every day. By default FyLasso - Professional Edition is configured to automatically update at 3:00 a.m. every Monday morning.

3 Detecting Potential Software Vulnerabilities

FyLasso's attack vectors are used to scan for potential software vulnerabtilities. Every file containing code (module) should be scanned to provide the maximum protection for your computer. The most critical modules are those currently being used by running applications. FyLasso allows you to automatically send every running-processes modules to its Scheduler. The Scheduler manages the list of modules ready to be scanned and when to start scanning them. By default the Scheduler is configured to automatically start scanning at 3:05 a.m. every morning, but also has the capability to scan the running-processes modules in realtime.

FyLasso creates a seperate file to store the potential vulnerabilities it finds for each module. These files are saved in FyLasso's Data folder. The SHA-1 hash of each module is used as the file name to ensure that the module protected at runtime was the actual module scanned prior to its protection. If a new application is installed or updated the SHA-1 hash for one or more modules changes and FyLasso sends those modules back to the Scheduler to be rescanned.

During scanning FyLasso verifies that each module is using the Windows Portable Executable (PE) file format. If a module is not then it is discarded, otherwise the base address and size of each module's code sections are determined. Each code section is scanned using every attack vector. When a potential vulnerability is found the offset relative to the base address of the current code section is saved in the module's file. The address of the potential vulnerability will be recomputed using the new base address, by the PRDS, at runtime. Saving the offsets allows for section relocation and Address-Space Layout Randomization (ASLR) within the address space of the protected process.

4 The Proactive Real-time Defense System

FyLasso's PRDS redirects every potential vulnerability, using interprocedural control-flow obscuring, to a dynamically allocated memory location. From the signature of the attack vector and the state of the address space at the time of execution the PRDS determines writable memory ranges with respect to the potential vulnerability being executed.

FyLasso stops both attacker-defined code and code located in existing modules (return-into-libc attacks) from being executed by checking if memory outside the writable memory ranges are modified. This check occurs immediately after the potential vulnerability executes. If a modification is detected then the user is notified and the process is safely terminated.

4.1 Example Buffer Overflow Attack Vector

In the case of a buffer overflow attack vector the memory surrounding the destination buffer is sanity checked. Before the potential vulnerability is executed, four bytes adjacent to the end of the destination buffer are saved and a randomly-generated four byte guard overwrites its location. Immediately after the potential vulenerability executes the guard is checked if it was modified. If the value of the guard is the same then the original four bytes that were overwritten are restored, and normal execution continues. If the guard was modified then the user is notified and the process safely terminates.

Consider a stack-based buffer overflow. If the buffers address is located within the current stack frame then the closest saved base pointer will be used for the location of the guard. This will protect the saved base pointer, saved return address, function pointers as function parameters, longjmp buffers as function parameters and exception handlers from being overwritten. Every stack frame at a higher memory location than the current stack frame will also be protected. The saved base pointer in the frame below the buffer will be used in the case of a buffer underrun.

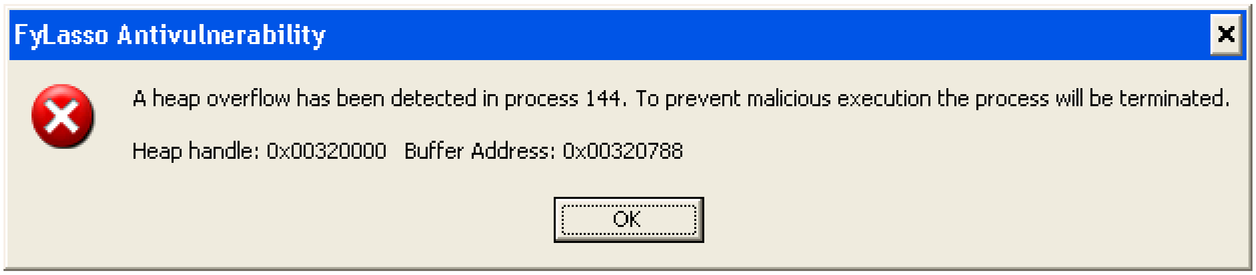
If the destination buffer is located in a heap section then the guard is located at the address following the block that is allocated for the buffer. Windows XP SP2 added a sanity check on the forward and backward links when the block is removed from the freelist along with a one byte integrity check. One way to bypass this mechanism is by manipulating the use of the lookaside list [3, 6]. Since the sanity check only occurs when the block is freed other function pointers or allocated vtables can still be overwritten and used to control execution. FyLasso removes the attack windows in the above scenarios by checking the integrity of the four byte heap-guard *immediately* after the potential vulnerability is executed. Listing 2 is an example of a heap buffer overflow.

Listing 2: Example Heap Buffer Overflow

```
#include <iostream>
using namespace std;

int main(int argc, char **argv) {

        //Allocate the source and destination buffers.
        char *szSourceBuff = (char*)malloc(200);
        char *szDestBuff = (char*)malloc(20);

        //Get input from the user.
        cin >> szSourceBuff;

        //Copy the buffer without checking the size
        //of the destination buffer.
        strcpy(szDestBuff, szSourceBuff);

        return 0;
}
```

If the size of *szSourceBuff* in Listing 2, including the null byte, is greater than twenty bytes then a message box with the process id, heap handle, and the buffer's address is displayed to the user. Figure 1 shows the notification to the user when an overflow occurs in the example from Listing 2. Supplying additional information to the user would potentially break FyLasso's protection mechanism. Therefore, a debugger should be used at the time of notification to obtain additional information.

Figure 1: Heap Overflow Notification



A heap overflow has been detected in process 144. To prevent malicious execution the process will be terminated.

Heap handle: 0x00320000   Buffer Address: 0x00320788

5 Conclusion

FyLasso Antivulnerability is an easy to use, easy to install, next-generation application security utility. Designed to integrate into existing *Layered Defense* solutions, FyLasso prevents worms and malicious hackers from penetrating your computer systems. If you're looking for an affordable way to increase security on your network then start by trying the free Limited Edition from www.fylasso.com. We thank you for your interest in FyLasso. For further questions or inquiries please contact support@fylasso.com.

References

[1] P. Silberman, R. Johnson, ”A Comparison of Buffer Overflow Prevention Implementations and Weaknesses” *IDEFENSE*, August 2004.

[2] Wikipedia: The Free Encyclopedia ”Software Bugs” August 2004.

[3] A. Alexander, ”Defeating Microsoft Windows XP SP2 Heap Protection and DEP Bypass” *Positive Technologies*, January 2005.

[4] C. Cowan et al, ”Stackguard: Automatic adaptive detection and prevention of buffer-overflow attacks, in Proceedings of the 7th USENIX Security Symposium”, San Antonio, TX, January 1998.

[5] G. Hunt, D. Brubacher, ”Detours: Binary Interception of Win32 Functions”, Microsoft Research, Redmond Washington, July 1999.

[6] D. Litchfield, ”Defeating the Stack Based Buffer Overflow Prevention Mechanism of Microsoft Windows 2003 Server”, Next Generation Security Software, September 2003.

[7] P. FAYOLLE, V. GLAUME, ”A Buffer Overflow Study Attacks and Defenses”, Networks and Distributed Systems, 2002.

[8] G. Hoglund, G. McGraw, ”Exploiting Software: How to Break Code”, Addison-Wesley, February 2004.

[9] K. Lhee, S. Chapin, ”Buffer Overflow and Format String Overflow Vulnerabilities”, Center for Systems Assurance, Syracuse University, Syracuse, NY, 2002.

[10] Wikipedia: The Free Encyclopedia ”Buffer Overflow” April 2007.

[11] D. Litchfield, ”Buffer Underruns, DEP, ASLR and improving the Exploitation Prevention Mechanisms (XPMs) on the Windows platform”, Next Generation Security Software, September 2005.

[12] D. Larochelle, D. Evans, ”Statically Detecting Likely Buffer Overflow Vulnerabilities”, University of Virginia, Department of Computer Science, August 2001.