Hill Valley SECURITY

Penetration Testing Report

From:
Hill Valley Security
Corporate Office
1955 Lyon Avenue
Providence, RI
02908

For:
Lisa Cuddy, CIO
Gregory House, IT Systems Manager
Dewey, Cheatum, & Howe Law Firm

By: Justin Leonard

Table of Contents

Executive Summary

General Summary

My company has been hired to perform penetration tests on the systems and networks of your firm relating to the use of the main organizational application your company uses, DCH Law. Based on the penetration testing plans and documentation we have previously submitted, we have compiled our results and recommendations for your company. The penetration tests were directly focused on the main vulnerability we found relating to buffer overflows on your Windows applications and systems. A buffer is simply a space in temporary memory where data can be stored on a system. When the data in a buffer exceeds the amount of space allocated for it, this can lead to system crashes or unauthorized access. We used both static testing techniques analyzing the source code of your application and dynamic testing techniques analyzing the allocated memory for your application while the process is running. Based on our research, we have found that your application is vulnerable to buffer overflow exploits.

Scope of Work

The penetration testing of your systems includes all workstations and networks that use the DCH Law application that connects to the databases and web applications for external users that are stored on your servers. We used the application as well as the source code for further research when performing the tests.

Project Objectives

The objective of this testing was to analyze the security posture of the DCH Law application relating to the access from various clients and find the source of random application and system crashes that were occurring. This is imperative to the security and constant availability of your services to both your associates and clients.

Assumption

We assume that both employees and external users can have access to the application, and that the unauthorized access of a threat actor could cause this vulnerability to be exploited.

Summary of Findings

Based on our research, we have found that there numerous insecure programming techniques in shared libraries of your application that make it vulnerable to a buffer overflow attack. The input received inside the application itself is validated and error corrected, but this is not true for loading and saving settings and data files. The application saves individual user settings in various compressed and uncompressed files in addition to the storage of other stored data files. If the contents of these files are modified by an external source where the new data exceeds the predefined buffer space for the specified file or data, a buffer overflow can occur which could cause application and system crashing. There is no input validation nor error handling for this data. A threat actor can discover the layout of your program’s memory and use that to exploit your program. This could cause crashes to the system as well as more complex malicious payloads being executed without a victim user’s knowledge. This could compromise a user and lead to further privilege escalation and other exploitations on your system.

Summary of Recommendations

We recommend a few security measures to ensure the integrity and persistence of your services. Firstly, a complete overhaul of your application’s source code and libraries using secure programming techniques. There are multiple locations in this application project where a buffer overflow could occur due to incorrect data handling. Having insecure source code in software can lead to numerous vulnerabilities. You should also encrypt and store your settings and data files. Secondly, we recommend more extensive and comprehensive software testing to be performed on this application. Constant and well rounded testing techniques can help eliminate these vulnerabilities. Reviewing the source code for undetected bugs as well as run time debugging can show you if a buffer overflow will or has occurred, and help you find the source. Finally, we recommend upgrading the operating systems on your workstations. We have found that a large number of your workstations are running on Windows 7 which has ended support on January 14, 2020. This operating system can be susceptible to a lot of security flaws and lead to buffer overflows on your application.

Methodology

Vulnerability Information

Primary memory is one of the most important parts of a system. The Central Processing Unit (CPU) uses memory as a temporary storage space for processes and data. Processes use virtual memory addresses that get mapped to the physical ones on the component itself. Windows processes get a virtual address assigned to it, and all commands and data are stored with it (Bowden). The main places in memory where data is stored for a process is the heap for global data and the stack for local data. Functions in a program are continuously run throughout it’s execution. When a function is called, it gets a frame on the stack. A stack frame is simply allocated space for the arguments, variables, return address, etc. of the function. A pointer is a piece of data that holds a memory address and points to another place in memory. The basic anatomy of the stack starts with the extended stack pointer at the top, buffer space, the extended base pointer, and the extended instruction pointer, also known as the return address (VetSec). The buffer space and return address are usually the highest points of interest. Buffer space holds the data for buffers. Buffers are temporary storage regions to hold data. A buffer overflow occurs when the amount of data being stored in a buffer exceeds it’s allocated space (Imperva). This can lead to application failure or access to other addresses in memory. This is a huge vulnerability. There are buffer overflows that can happen in both the heap and stack, but the main vulnerability of this application comes from stack based buffer overflows on the Windows systems. The majority of buffer overflows usually happen within the stack. These can be caused by erroneous input, bad error handling, or another fault in the application. Buffer overflows can lead to crashes because the data input into the buffer that overflows out of the allocated memory has to go in other memory addresses. These memory addresses right after the buffer can lead to the return address after buffer space. This can be advantageous for exploitation. An attacker can change the return address to point to malicious code and further exploit a system (VetSec). Modifying this data can lead to crashes in addition to the application catching the error (Bowden).

Planning

When planning to test the application for buffer overflows, we found all sources where data is entering the application from an external source. We used a debugger and hex data editor to analyze the process of your application and to see where and when a crash would occur, as well as what memory the data could overwrite. The debugger used was X64dbg which is software used to debug Windows applications (Bowden). We can see where data in memory is placed as well as any overflows that would occur. We noticed that user input directly into the application would not cause a crash nor buffer overflows. All input was validated and data above a certain size could not be entered to their respective locations. Upon deeper inspection into the application’s file layout, we discovered that there were numerous compressed and uncompressed user settings files in addition to binary data files that were storing data that would be input into the application. When we edited the files directly and used the program, crashes would begin to occur from a buffer overflow.

Exploitation

With the information gathered from planning, we successfully were able to exploit the buffer overflow vulnerability found on your application. First, we located a user settings file that gets accessed when the user logs into the system. A script was created to edit a data point for the user’s preferred user interface layout. This data point internally is stored as a small number, and we modified it to have thousands of bytes. Upon loading the application and logging in with the modified settings file, the application crashed. Using the debugger, we were able to see that a buffer overflow had occurred. We then used the debugger to find the exact offset between the start of the buffer and the extended instruction pointer. This can be used to further exploit a system. Next, we used Metasploit, a well known penetration testing environment, conjunction with MSFvenom, which is a tool within Metasploit to output different types of shell code from Metasploit in addition to a payload. We can use MSFvenom to create our payload for the return address using the offset we gained earlier (Bowden). The payload will be a command for the application to execute instead of returning. This payload can be used to continue the exploitation of the workstation and system. With this exploitation, we can successfully initiate a buffer overflow on your application in order to either crash it or further exploit the system. This buffer overflow exploit on your application can be a gateway to infiltrate the system and exploit other vulnerabilities in order to gain access and elevate control. After the buffer overflow occurs on the application, without proper handling, this can make your system very vulnerable to attacks based on the exploitation described.

Reporting

Based on our planning and exploitation of your application DCH Law, we can see that you are at high risk for application or system crashes with the chance of a security breach. The assets that could be affected by this exploitation are the availability of the company's services to their employees and end users, crashing of the application, systems, or networks, and vulnerability to unauthorized access to devices and data in addition to further exploitation of the system by threat actors.

The buffer overflow vulnerability is a very general and well known problem. Because the nature of the vulnerability makes it relatively easy to find and exploit, the risk for this is very high and the likelihood that it could be exploited is also high.

Detailed Findings

The buffer overflow vulnerability is well known and can happen a lot on Windows systems, especially on the unsupported Windows 7 workstations used in your systems. Memory is an important function of a computer. Data is constantly being buffered, written, and read on a system. Faults in software can lead this data to overflow out of it's allocated memory causing a myriad of problems. This can range from crashing the application to taking advantage of the extended instruction pointer after the buffer space to point to malicious code and further exploit a system.

There are a lot of different methods that one can use to test for buffer overflows. Some of these buffer overflow testing methods include:

- Testing the values of all user input that can be entered into the application
- Using debuggers and memory editors to deeply analyze the allocated memory for the process. This can show you exactly where and why a crash occurred, along with the data and process state for analysis.
- Using static software testing techniques to look over the source code in order to see where code in the program could lead to buffer overflows.

One can exploit a buffer overflow in an application by first finding the source of the buffer overflow. Where ever the data input into an application can be overflowed causing errors is the initial place to start. You can exploit this by automating an initiator for this input so that the input data is far greater than the allocated space causing a crash. You can further exploit this vulnerability by using a debugger to find the offset between the buffer and the extended instruction pointer. By modifying the value of the extended instruction pointer to point to a malicious payload, you can use other exploits to gain unauthorized access to a system and continue your exploitation (Bowden).

My recommendations for preventing this threat would be to first and foremost make sure that the code for your application is secure. Have advanced input validation and error checking. Numerous other coding techniques include Address Space Randomization (ASLR) which is the process of randomly changing the address locations of data regions in order to prevent buffer overflow from going into predefined addresses, Data execution prevention which prevents execution of memory in certain arbitrary non executable regions, and Structured Exception Handler Overwrite Protection (SEHOP) which prevents malicious code from overriding the application's built in exception handling (Imperva). Having well made and secure code can be a large factor in preventing these vulnerabilities. In addition, it is important to upgrade all of your systems to one of the newest and well supported operating systems while keeping them up to date. These methods of testing and prevention will help you further secure your systems and applications for the future.

Conclusion

After initial penetration testing and research, we have come to the conclusion that your application DCH Law as well as your system infrastructure could be at risk for future crashes or failure and possible security breaches. We recommend immediate action to correct the faults in the source code of your software, create new comprehensive software testing guidelines for the future, encrypt and securely store your user settings and application data files, and upgrade along with keeping up to date all workstations and systems in your workplace. Following these steps will aid you in correcting the buffer overflow vulnerabilities that your application currently possesses. Fixing this vulnerability and having routine penetration tests to make sure it will not happen again will protect your system from future exploitation. We thank you for your time and look forward to working with you again in the future.

References

[01] Bowden, Andy. "The Basics of Exploit DEVELOPMENT 1: Win32 Buffer Overflows." Coalfire.com, Jan. 2020, www.coalfire.com/the-coalfire-blog/january-2020/the-basics-of-exploit-development-1.

[02] "Memory Management: Frame Allocation." Microsoft Docs, 4 Nov. 2016, docs.microsoft.com/en-us/cpp/mfc/memory-management-frame-allocation?view=msvc-160.

[03] The Redscan Marketing and Communications team. "Windows Buffer Overflow Attacks Pt. 1." Redscan, 25 Mar. 2020, www.redscan.com/news/windows-buffer-overflow-attacks-pt-1/.

[04] VetSec Webmaster. "32-Bit Windows Buffer Overflows Made Easy." VetSec, Inc - A Non-Profit Helping Veterans Enter Careers in Cybersecurity, 10 Sept. 2018, veteransec.com/2018/09/10/32-bit-windows-buffer-overflows-made-easy/.

[05] "What Is a Buffer OVERFLOW: Attack Types and PREVENTION Methods: Imperva." Imperva, www.imperva.com/learn/application-security/buffer-overflow/.

Tools

WinHex: Computer Forensics & Data Recovery Software, Hex Editor & Disk Editor

https://x-ways.net/winhex/

x64dbg: An open-source x64/x32 debugger for Windows

https://x64dbg.com/#start

Metasploit: The world's most used penetration testing framework

https://www.metasploit.com/

MSFvenom

https://www.offensive-security.com/metasploit-unleashed/msfvenom/