**CSC435 - Computer Security**

**Lab 2 - Buffer Overflow**

**For Dr. Ayman Tajjedine**

**Group 7:**

Roula Ghaleb

Abdallah Tourbah

Louay Zeineldien

Omar Shatila



## Table of Contents

[**Table of Contents 2**](#_vzxxuhf3fcmb)

[**Part 1: General Questions 3**](#_mh0elo58wtwk)

[1. What is buffer overflow? On which parts of memory does it work? Explain how or when it works on each? (Omar) 3](#_bfp08s8c1iiw)

[2. What happen when you run an executable? Explain the steps of a process and how they are saved in memory and pushed into stack/heap… (Omar) 4](#_s9xix2qht7vx)

[3. List the functions in C that are vulnerable to buffer overflow. What are their safe substitutes? (Omar) 5](#_ykgw8fyufefu)

[4. Name the different buffer overflow protection algorithms, and give a brief explanation of each. (Louay) 6](#_e1f1s4evvzpt)

[5. What does this string (“%x:%x:%s”) do? How can this be used for memory leaking? (Louay) 7](#_hooef87et5xz)

[**Part 2: Vulnerable C 7**](#_gg8wd6exxtg6)

[1. Explain the program and how does it work? (Louay) 8](#_ll1hnskjp6de)

[2. Where are the variables stored? (Omar) 8](#_z6lyz5f7u3ei)

[3. What exactly is the type of attack the program is vulnerable to? (Omar) 8](#_u97ropewff4y)

[4. In which line is the code vulnerable to buffer overflow? (Roula) 9](#_g8ir8ner6nka)

[A. In A, the code is vulnerable in at line 5: 9](#_qli8ib2opit)

[5. Explain how to achieve buffer overflow? (Abdallah) 10](#_jvzlmbgm6980)

[6. If you want to use safe coding, how can you make this code immune to buffer overflow (just by adding some code)? 10](#_a9064you24a9)

[A.(Abdallah) 10](#_nlmwv8um4w9t)

[B. (Roula) 10](#_41bw4iqgj8he)

[C. (Roula) 11](#_h56ecf4bzk3k)

[**Part 3: Lab – Vulnerability 11**](#_30nds9nqqnlr)

[**Setup Windows Server 2008 (Roula) 11**](#_ynn0wnlrqlwz)

[After pressing on default website, and then view virtual directories, we add a virtual directory which is one we created called photo.png made using MS paint and placed it in documents. We gave it a virtual name of ‘test’. 14](#_f84xs9g7de3a)

[Research a vulnerability related to Buffer overflow and DoS. (Omar) 17](#_oaqnjawow3fx)

[What is the name of this vulnerability? How does it work? Explain. (Omar) 17](#_p6i3bmn05rul)

[Use “wget” or “curl” or just your “browser”, forge the request and perform this attack on the server. (Abdallah and Roula) 18](#_r59b9t1j3676)

[What is the result of this attack on the server? (Abdallah) 19](#_fcscm5icymix)

[How can we remediate the vulnerability without patching? (Roula) 20](#_7rha4e8zpool)

[**References: 20**](#_ynlu4nrwb544)

## 

## Part 1: General Questions

### What is buffer overflow? On which parts of memory does it work? Explain how or when it works on each? (Omar)

Buffer overflow is a type of software vulnerability that occurs when a program writes more data to a buffer than it can hold, causing the excess data to overflow into adjacent memory locations. This can cause unexpected behavior, crashes, or even allow an attacker to execute malicious code on the system.

Buffer overflow attacks typically target either the stack or the heap memory regions of a program.

Stack-based buffer overflows occur when a program stores data on the stack without proper bounds checking, allowing an attacker to overwrite the return address or other critical data on the stack. When the function returns, the overwritten return address causes the program to jump to a memory address controlled by the attacker, executing arbitrary code.

Heap-based buffer overflows occur when a program dynamically allocates memory on the heap without proper bounds checking, allowing an attacker to overwrite adjacent heap memory or heap metadata. This can lead to heap corruption, memory leaks, or even arbitrary code execution.

In both cases, the attacker typically needs to craft a specific input that causes the buffer overflow and triggers the desired behavior. This can be done through a variety of techniques, such as input validation bypass, integer overflow, or format string vulnerabilities.

To prevent buffer overflow vulnerabilities, programmers should use safe coding practices such as bounds checking, input validation, and memory allocation techniques that minimize the risk of buffer overflows. Additionally, using programming languages that provide built-in memory safety features, such as Rust or Swift, can help prevent buffer overflow vulnerabilities.

### What happen when you run an executable? Explain the steps of a process and how they are saved in memory and pushed into stack/heap… (Omar)

When you run an executable, the operating system loads the program into memory and creates a process for it to run within. The process is a container that provides a virtual address space for the program to use, along with various system resources such as file handles and network sockets.

The steps involved in the execution of a program can vary depending on the programming language, operating system, and hardware platform, but in general, they follow a common set of principles:

**1.** The program starts executing from the entry point, which is typically the ‘main’ function in C/C++ programs or the ‘\_\_start’ function in assembly programs.

**2.** The operating system reserves memory for the program and loads the executable file into memory, along with any shared libraries that the program depends on.

**3.** The ‘main’ function initializes the program and sets up any data structures or resources that it needs. This might include allocating memory on the heap, opening files, or initializing network connections.

**4.** The program executes its main logic, which might involve calling various functions or methods to perform specific tasks. Each function call creates a new stack frame on the stack, which includes the function's arguments, local variables, and return address.

**5.** The program may also use the heap to allocate memory dynamically during execution, using functions like ‘malloc’. The heap is a separate region of memory from the stack, and is typically managed by the program's runtime system or memory allocator.

**6.** As the program executes, it may modify the values of its variables, either on the stack, heap, or in global/static data segments. These changes are reflected in the program's virtual address space, and are saved back to disk when the program exits.

**7.** When the program finishes executing, it returns from the ‘main’ function and exits. At this point, the operating system releases any resources that were allocated for the program, and marks the memory used by the program as available for other processes to use.

Overall, the process of executing a program involves a complex interplay of system resources, memory management, and program logic. Understanding these concepts is essential for building reliable, efficient, and secure software.

### List the functions in C that are vulnerable to buffer overflow. What are their safe substitutes? (Omar)

| ***Unsafe functions*** | ***Safe functions*** |
| --- | --- |
| gets (char \*str) | **fgets (**char \*str, int n, FILE \*stream) |
| sprintf (char \*str, char \*format, ...) | **snprintf (char \*str, size\_t size, const char \*format, …);** |
| strcat (char \*dest, char \*src) | **size\_t strlcat (char \*dst, const char \*src, size\_t size);** |
| strcpy (char \*dest, char \*src) | **size\_t strlcpy (char \*dst, const char \*src, size\_t size);** |
| vsprintf (char \*str, char \*fmt, va\_list ap) | **vsnprintf (char \*restrict buffer, size\_t bufsz, const char \*restrict format, va\_list vlist);** |

### Name the different buffer overflow protection algorithms, and give a brief explanation of each. (Louay)

There exist mainly two types of defenses against buffer overflow. The first one makes the program resistant to attacks at compile time, while the second detects attacks and aborts them at run time. Algorithms that provide compile time defense include OpenBSD for safe coding techniques, Libsafe which is a safe library, and random canary values that detect buffer overflows and provide stack protection. Other algorithms include Stackshield or Return Address Defender (RAD) which compare the return address at program entry with the return address at exit, and abort the program if they are different.

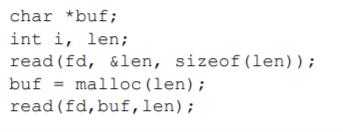
Runtime defenses include Address Space Randomization which randomizes the locations of stack, heap buffers, and libraries, and guard pages which are placed between critical memory regions, like in the memory management unit (MMU) as illegal addresses.

### What does this string (“%x:%x:%s”) do? How can this be used for memory leaking? (Louay)

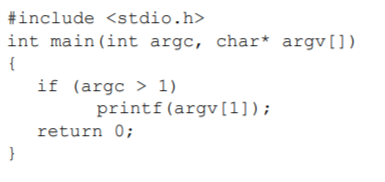
%x means that the specified value should be in hexadecimal which contains numerical and characters values however %s specifies that the format should be a string but now using print format function with %x it will lead to attacks from the stack the program will read values from the stack and with the %s it will read whatever information that is not in the stack memory and that will lead to memory leakage using the printf function.

## Part 2: Vulnerable C

A.



B.



C.

### Explain the program and how does it work? (Louay)

1. The program takes a character and integer to count and a buffer array of characters of length 5 after the first print it will enter the while loop and keeps executing until it finds a character that is /n which means new line. If it doesn’t find the new line character, it will keep reading and increasing the index from 0 until the character read is a new line after exiting the while loop it will take as character of last index \0 and finally prints the buffer.
2. The program declares a character pointer buffer and initializes i and the length, and then it will read from the file descriptor array of length len and store it in the length with the &len of size length. Then it will allocate len in memory and store it under buf character and finally read from file descriptor of length of and of bytes len.
3. The program takes as inputs an integer argc and an array of characters argv from command, then it will check if the agrc value of command is higher than 1. If so, the printf(argv[1] will execute and will print the format being the command array of index 1 not the command value. At the end, zero is returned.

### Where are the variables stored? (Omar)

1. The variables 'ch', 'buffer', and 'i' are stored in the program's stack memory
2. The variables 'buf', 'i', and 'len' are stored in the program's stack memory. The 'buf' variable is a pointer to a memory location in the heap memory allocated by the 'malloc()' function.
3. The variables 'argc' and 'argv' are stored in the program's stack memory.

### What exactly is the type of attack the program is vulnerable to? (Omar)

1. The 'getchar()' function used in the program does not limit the number of characters that can be entered into the 'buffer' array, which can lead to a buffer overflow vulnerability. If the user enters more than 5 characters, the extra characters will overwrite the adjacent memory locations in the stack, which can cause the program to behave unexpectedly or even crash. An attacker can exploit this vulnerability to execute arbitrary code by overwriting the return address of the 'foo()' function or by injecting malicious code into the buffer.
2. The program is vulnerable to a heap buffer overflow attack. The 'read()' function reads 'len' bytes of data from the file descriptor 'fd' and stores it in the memory location pointed to by 'buf'. If the length of the data in the file is greater than the size of the allocated memory block, then the 'read()' function can write beyond the bounds of the allocated memory block, causing a heap buffer overflow. An attacker can exploit this vulnerability to execute arbitrary code, modify data structures, or crash the program by overwriting adjacent heap memory locations.
3. The program is vulnerable to a buffer overflow attack. When the 'printf()' function is called, the user-supplied argument 'argv[1]' is used as the format string without any validation. An attacker can craft a malicious format string argument that can read arbitrary memory or modify the program's execution by using format specifiers, such as '%n', '%s', or '%x'.

### In which line is the code vulnerable to buffer overflow? (Roula)

### A. In A, the code is vulnerable in at line 5:

while ((ch = getchar()) != ’\n’) buffer[i++] = ch;

We specified that the size of the buffer array is 5 in line 1: char buffer[5], this means that a user may purposefully or by accident enter an input larger than 5 and overwrite more memory locations than what was intended in the code.

B. In B, the code is vulnerable at line 5:

read(fd,buf,len);

Buffer overflow may occur if the fd is greater in size than the buf.

C. In C, the code is vulnerable at line 5:

printf(argv[1]);

We have an overflow here since the printf() function requires a formatted string as an argument, but here we are taking an argv[1] which is a command line argument. This means that a malicious code may be inserted in the argument and then passed as the formatted string.

### Explain how to achieve buffer overflow? (Abdallah)

Buffer overflow happens when the buffer’s fixed length gets bypassed by the program independently, especially when the amount of data exceeds the length of the array. For example in A, the buffer length is 5 however the buffer keeps adding up until the character entered is « \n » so it is definitely gonna break the fixed length with the index keep adding up if the user doesn’t enter the condition in the while loop which will generate buffer overflow message. Or in B,where the issue comes if the file descriptor array length bypasses the buffer array length. Or in C where the argv array where the user can possibly enter a numerical array based on which command the user entered since it is printf type so all these codes could lead to buffer overflow exceeding the buffer length.

### If you want to use safe coding, how can you make this code immune to buffer overflow (just by adding some code)?

### A.(Abdallah)

The problem with this code is that the buffer’s fixed length is 5 and that is bypassed during the loop, so we can change the condition in the while loop to while((ch=getchar()!=‘/n’&&i<buffer.length) and that will solve the buffer overflow issue.

#### B. (Roula)

In general, to fix this code we need to make sure that the size of the file descriptor is smaller than that of the buf. There are several ways to approach this such as finding out the input of the fd and not allowing the program to continue if it is greater than the size of buf

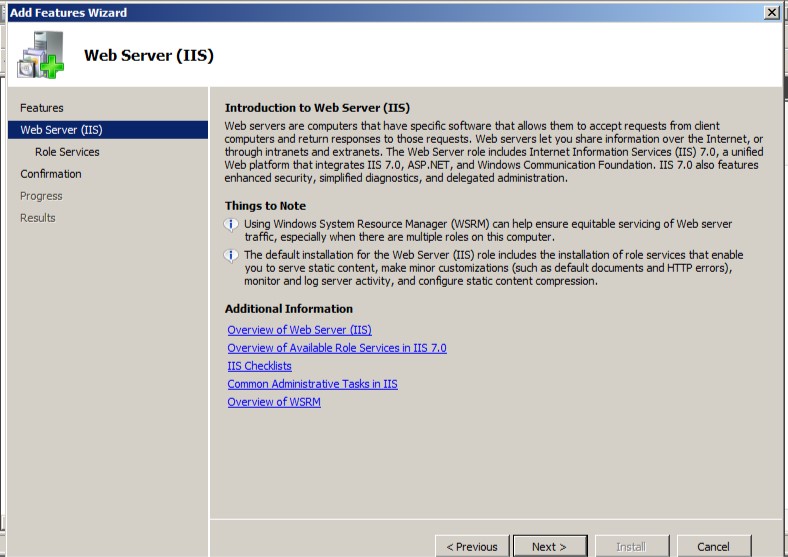
#### C. (Roula)

To fix this issue, we use printf("%s", argv[1]), or even determine the field width to be more specific, for example, “%12s”. This determines the type of argument to be interpreted by printf which should be a string(helps avoid string format attack) and also limits its length.

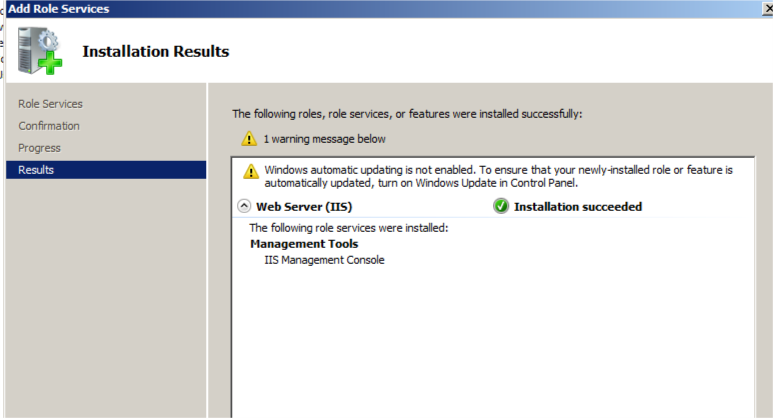
## Part 3: Lab – Vulnerability

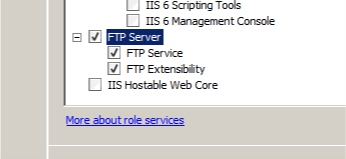
## Setup Windows Server 2008 (Roula)

After trying for several days to use windows 10 Home and Windows 10 Pro via VirtualBox to access hyper-v manager, we were met with nothing but errors so we opted to install Windows 8 instead. A major error was the bios was legacy and it needs UEFI for hyper-V to work properly, and switching these using Virtual Box resulted in a lot of errors. After installing and running the ISO, we enabled IIS

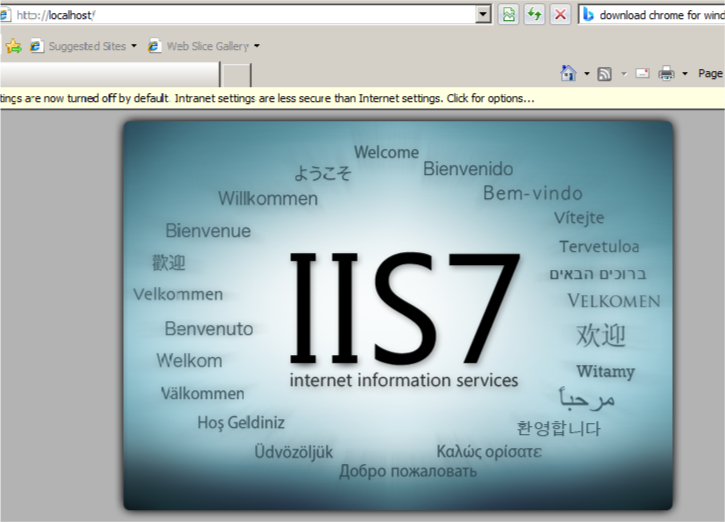


Then we installed the IIS manager

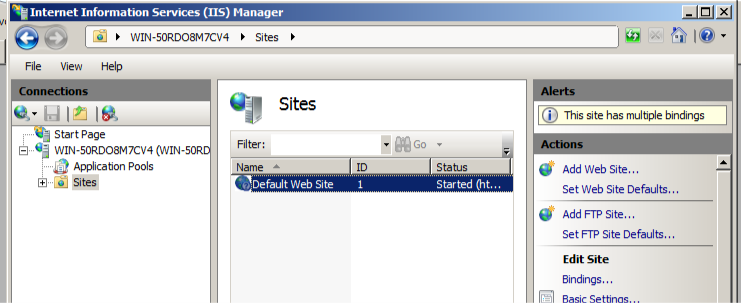


Then we installed FTP server:

And here is ISS running by using http://localhost/



### 

now , inside IIS Manager, we need to access default website to add a virtual path:

### 

### After pressing on default website, and then view virtual directories, we add a virtual directory which is one we created called photo.png made using MS paint and placed it in documents. We gave it a virtual name of ‘test’.

### 

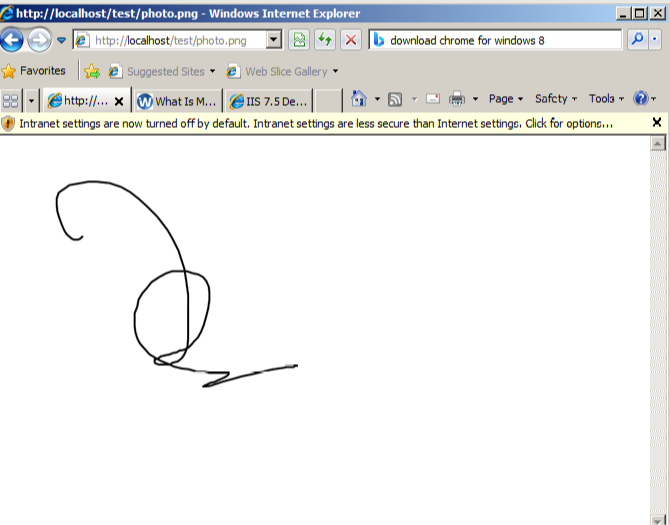
### 

### 

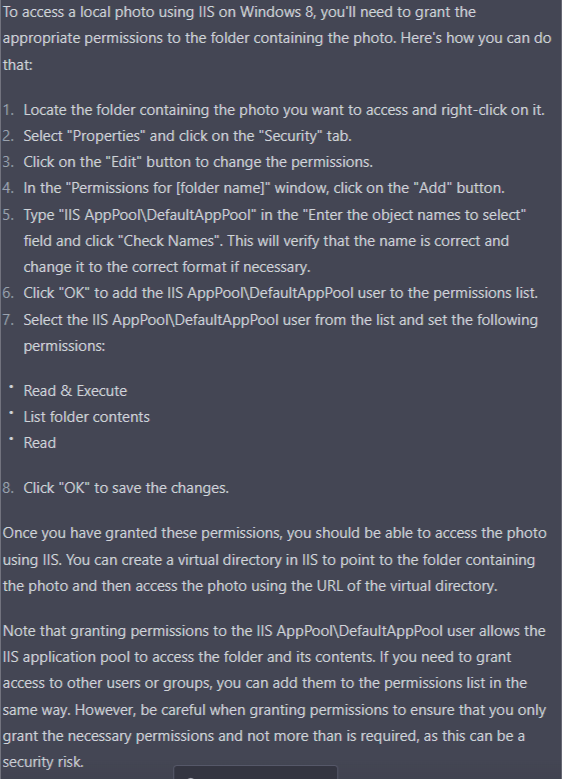
### 

Editing permissions:

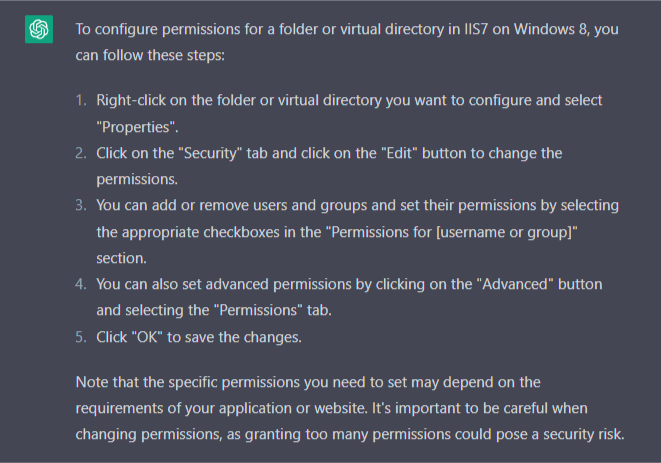
### 

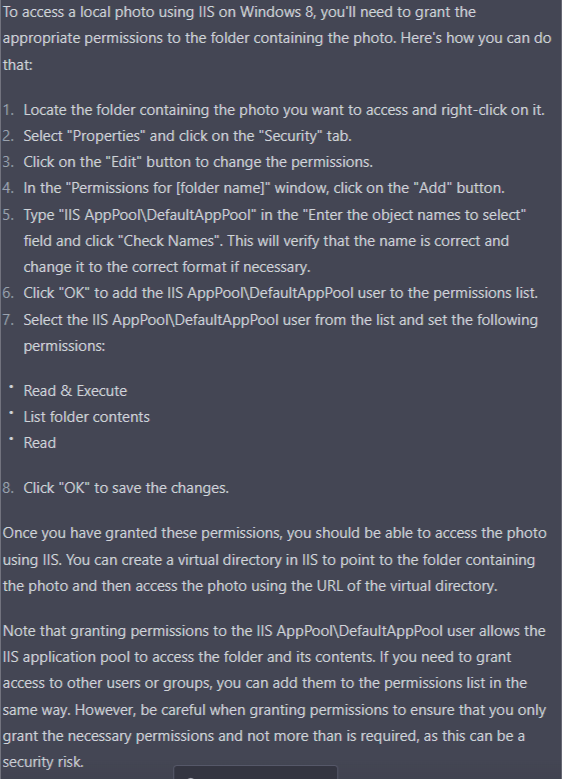
It is working:

Note: some information we got from ChatGPT to help fix our errors:



### 





### 

### Research a vulnerability related to Buffer overflow and DoS. (Omar)

This vulnerability allowed malicious hackers to send a specially crafted HTTP request to a target server running Windows, causing a buffer overflow in the HTTP.sys driver.

### What is the name of this vulnerability? How does it work? Explain. (Omar)

The vulnerability that meets the criteria mentioned in the question is known as "HTTP.sys Remote Code Execution Vulnerability" or "MS15-034."

HTTP.sys is a kernel-mode driver used by Windows operating systems to handle HTTP traffic. It processes HTTP requests and forwards them to the appropriate application. In April 2015, researchers discovered a vulnerability in HTTP.sys that allowed remote attackers to execute arbitrary code on the affected server.

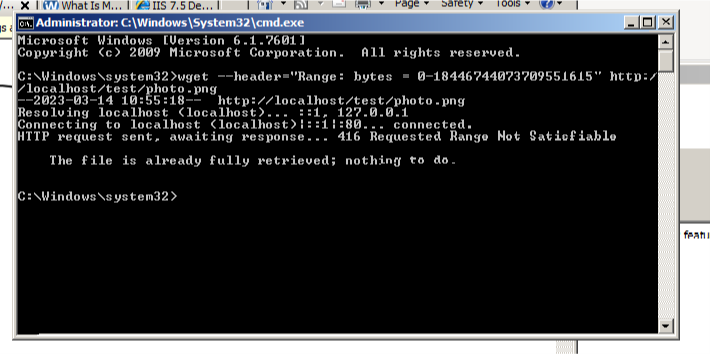
The vulnerability occurs due to a buffer overflow in the HTTP.sys driver. When the driver receives a specially crafted HTTP request with a large Range header, it fails to validate the length of the header, leading to a buffer overflow. This buffer overflow can be exploited by attackers to execute arbitrary code in the kernel mode, which can lead to a denial of service (DoS) attack or complete compromise of the affected system.

An attacker can exploit this vulnerability by sending a specially crafted HTTP request with a Range header to the targeted server. The Range header should contain a large value in the "Range" field, such as "Range: bytes=0-18446744073709551615." The server will attempt to process the request and fail, causing a buffer overflow in the HTTP.sys driver.

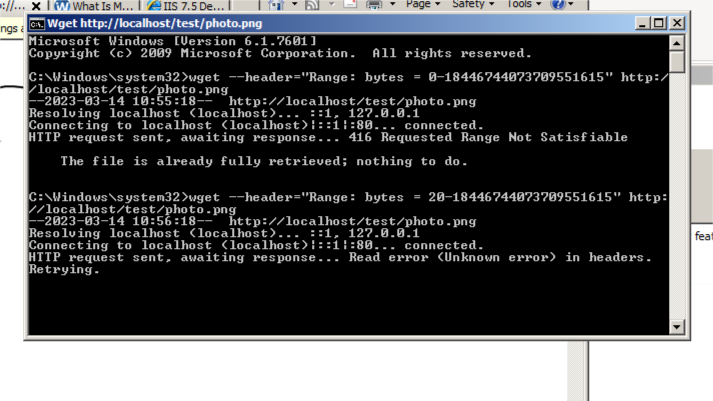
If exploited successfully, an attacker can take complete control of the affected system and perform various malicious activities, including installing malware, stealing sensitive data, or using the system to launch further attacks.

### Use “wget” or “curl” or just your “browser”, forge the request and perform this attack o n the server. (Abdallah and Roula)

To start off, we downloaded a wget.exe file to be able to use the command in the cmd. We were not able to get curl to work as it needed a newer version of .NET framework. From our research via a link provided in the references, we are able to find this vulnerability.



From what we learned, this error (Requested range not satisfied) shows that the server is vulnerable.

Now, we increase the number from 0, for example, we took 20: 

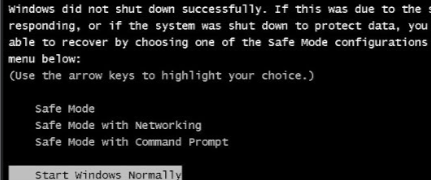
hg

Here, we can see a read error:

After that, the computer crashes:

There was a blue screen we were not able to capture a photo of properly as it was disappearing too quickly and the virtual machine was rebooting.

After that we get this windows error page:



### What is the result of this attack on the server? (Abdallah)

This is a Denial of Service or a DOS attack since the system restarted itself.

### How can we remediate the vulnerability without patching? (Roula)

2 simple ways to escape this vulnerability is to either disable IIS kernel caching or install the Microsoft patch called KB3042553.

## 

## References:

Dave Stork. (2015, April 18). IIS exploit can reboot your windows server; install patch KB3042553 ASAP. Dave Stork's IMHO. Retrieved March 2023, from <https://dirteam.com/dave/2015/04/18/iis-exploit-can-reboot-your-windows-2008-server-r2-and-up-install-patch-kb3042553-asap/>

*Ping of death ddos attack | cloudflare*. (n.d.). Retrieved March 2023, from <https://www.cloudflare.com/learning/ddos/ping-of-death-ddos-attack/>