**STRUCTURE EXCEPTION HANDLING**

**EXCEPTIONS:**

In computer programming, an exception refers to an anomalous or exceptional event that occurs during the execution of a program and disrupts its normal flow. Exceptions are typically situations that deviate from the expected or intended behavior of a program. When an exception occurs, the normal flow of the program is interrupted, and the control is transferred to an exception-handling mechanism.

**SEH:**

SEH stands for Structured Exception Handling, and it is a mechanism used in Windows operating systems to manage exceptions (unusual events or errors) in a structured way. SEH is part of the Windows API and is designed to provide a structured and uniform approach for handling exceptions in both user-mode and kernel-mode programs.

Here are some key points about Structured Exception Handling in Windows:

* **Exception Handling Structure**: SEH uses a data structure known as an exception handling frame to manage the handling of exceptions. This frame contains information about the exception and a chain of exception handlers.
* **Exception Handling Process**: When an exception occurs (such as a divide-by-zero or access violation), the operating system uses SEH to locate and invoke the appropriate exception handler. This handler can be at the application level, operating system level, or even at the kernel level.
* **Try, Except, and Finally Blocks**: SEH is implemented in the Windows programming environment using try, except, and finally blocks. The code inside the try block is monitored for exceptions, and if an exception occurs, control is transferred to the associated except block. The finally block is executed regardless of whether an exception occurred or not.

CODE:

|  |
| --- |
| \_\_try {  // Code in the try block  }  \_\_except (EXCEPTION\_EXECUTE\_HANDLER) {  // Exception handling code in the except block  }  \_\_finally {  // Code in the finally block  } |

SEH:

1. Exception Handling in Software: When an exception occurs in a program (an abnormal event like division by zero or accessing invalid memory), the system needs to handle it gracefully to prevent crashes. In many programming languages, including those that run on operating systems like Windows, there is a mechanism for exception handling.

2. Kernel Search for Exception Handler: When an exception is raised, the system looks for an exception handler in the function where the exception occurred. If the function doesn't have a handler, the system will then search the call stack for an appropriate handler.

3. Call Stack Exhaustion and System Crash: If the system exhausts the call stack (i.e., it searches through all the function calls made up to that point) and still doesn't find a suitable exception handler, the system will crash. A crash in this context usually means that the program or system becomes unresponsive or terminates abruptly.

4. Buffer Overflow and STATUS\_ACCESS\_VIOLATION: Buffer overflow is a type of software vulnerability where more data is written to a buffer than it can hold, leading to the overwrite of adjacent memory. This can result in unexpected behavior, including crashes. In the context of Windows operating systems, a buffer overflow can lead to a STATUS\_ACCESS\_VIOLATION error, indicating that there has been an attempt to access memory that "does not belong" to the program.

5. User Space vs. Kernel Space: Operating systems typically divide memory into user space and kernel space. User space is where user applications run, while kernel space is reserved for the core functions of the operating system. If an access violation occurs in user space, it may be caught by an exception handler. However, if the violation happens in kernel space, it is more severe and often cannot be caught. This is because kernel space is critical for the functioning of the operating system, and errors there can lead to system instability.

[SEH Exploit | Structured Exception Handler Overwrite (rapid7.com)](https://www.rapid7.com/resources/structured-exception-handler-overwrite-explained/)

**RETURN ORIENTED PROGRAMMING**

ROP stands for Return-Oriented Programming, and it is a computer security exploitation technique commonly used by attackers to take control of a system. ROP attacks typically target vulnerabilities that allow arbitrary code execution but do not permit the direct execution of injected code.

In a ROP attack on Windows or any other platform, an attacker leverages existing snippets of code called "gadgets" within the target application or its libraries. These gadgets are short sequences of machine code that end with a return instruction (ret). By chaining these gadgets together, an attacker can execute arbitrary commands without injecting new code. Instead, they manipulate the control flow by returning to existing code locations.

This technique is effective because it bypasses modern security mechanisms like Data Execution Prevention (DEP), which aims to prevent the execution of data in specific regions of memory. ROP attacks exploit the fact that the existing code in a program is already marked as executable.

To defend against ROP attacks, various security measures have been implemented, such as Control Flow Guard (CFG) and Address Space Layout Randomization (ASLR). These features make it more challenging for attackers to predict or manipulate the control flow and the memory layout of a program.

Arbitrary code execution refers to the ability of an attacker to run arbitrary or unauthorized code on a target system. When an attacker achieves arbitrary code execution, they gain control over the execution of the software and can execute any instructions or commands of their choosing.

**EXAMPLE:**

Suppose there's a vulnerable C function in a program that reads user input into a buffer without proper bounds checking, leading to a buffer overflow:

|  |
| --- |
| **void vulnerableFunction(char \*input) {**  **char buffer[100];**  **strcpy(buffer, input); // Vulnerability: No bounds checking on input**  **}** |

Now, an attacker wants to exploit this vulnerability using ROP to execute arbitrary code. They might craft a payload that includes a series of gadgets already present in the program's code or linked libraries. Each gadget ends with a ret instruction, allowing the attacker to chain them together:

|  |
| --- |
| **; Gadget 1: Load address of system function into a register**  **pop rdi ; gadget instruction to pop the next value into rdi (address of system)**  **address\_of\_system ; address of the system function**  **ret ; return to the next gadget**  **; Gadget 2: Load address of "/bin/sh" string into a register**  **pop rsi ; gadget instruction to pop the next value into rsi (address of "/bin/sh")**  **address\_of\_bin\_sh ; address of the "/bin/sh" string**  **ret ; return to the next gadget**  **; Gadget 3: Call the system function**  **call rdi ; gadget instruction to call the function at the address in rdi (system function)**  **; The payload would then be a series of these gadgets chained together** |

In this simplified example, the attacker would overflow the buffer with addresses of these gadgets, and when the vulnerable function returns, it starts executing the chain of gadgets. Ultimately, the attacker achieves arbitrary code execution by manipulating the control flow of the program using the existing code snippets (gadgets).

**ROP CHAINS:**

A ROP (Return-Oriented Programming) chain is a series of carefully crafted instructions that an attacker assembles by linking together existing code snippets, known as gadgets, in a program's memory. These gadgets are sequences of machine code that typically end with a return instruction (ret). By chaining these gadgets together, an attacker can manipulate the control flow of a compromised program to execute arbitrary code.

The ROP chain takes advantage of the fact that the attacker cannot directly inject new code into the program's memory due to security mechanisms like Data Execution Prevention (DEP). Instead, the attacker leverages the existing code by redirecting the program's execution flow to sequences of gadgets that achieve the desired malicious outcome.

Creating a full-fledged ROP example requires a detailed analysis of the target program and the identification of specific gadgets, which can be complex.

**C CODE:**

|  |
| --- |
| **#include <stdio.h>**  **void vulnerableFunction(char \*input) {**  **char buffer[100];**  **strcpy(buffer, input); // Vulnerability: No bounds checking on input**  **}**  **int main(int argc, char \*argv[]) {**  **if (argc != 2) {**  **printf("Usage: %s <input>\n", argv[0]);**  **return 1;**  **}**  **vulnerableFunction(argv[1]);**  **return 0;**  **}** |

**DEP**

In the context of Windows operating systems, DEP stands for Data Execution Prevention. DEP is a security feature designed to prevent certain types of malicious code, such as viruses and other exploits, from running in certain regions of memory. It helps protect against buffer overflow attacks by marking specific areas of memory as non-executable, making it more difficult for attackers to execute code from these regions.

DEP operates at both the hardware and software levels. At the hardware level, it requires a processor with support for the NX (No Execute) or XD (eXecute Disable) bit. Most modern processors have this capability. At the software level, DEP is implemented by the operating system.

There are two types of DEP:

1. Hardware-enforced DEP: This type of DEP relies on hardware support and works by marking specific areas of memory as non-executable.

2. Software-enforced DEP: This type of DEP can work on older hardware that may not have hardware support for DEP. It uses software techniques to monitor and prevent code execution in specific memory regions.

* **DEP Activity: Enable on your computer.**
* **What are DLLs?**

**USER ACCOUNT CONTROL**

**Microsoft:** [User Account Control - Windows Security | Microsoft Learn](https://learn.microsoft.com/en-us/windows/security/application-security/application-control/user-account-control/)

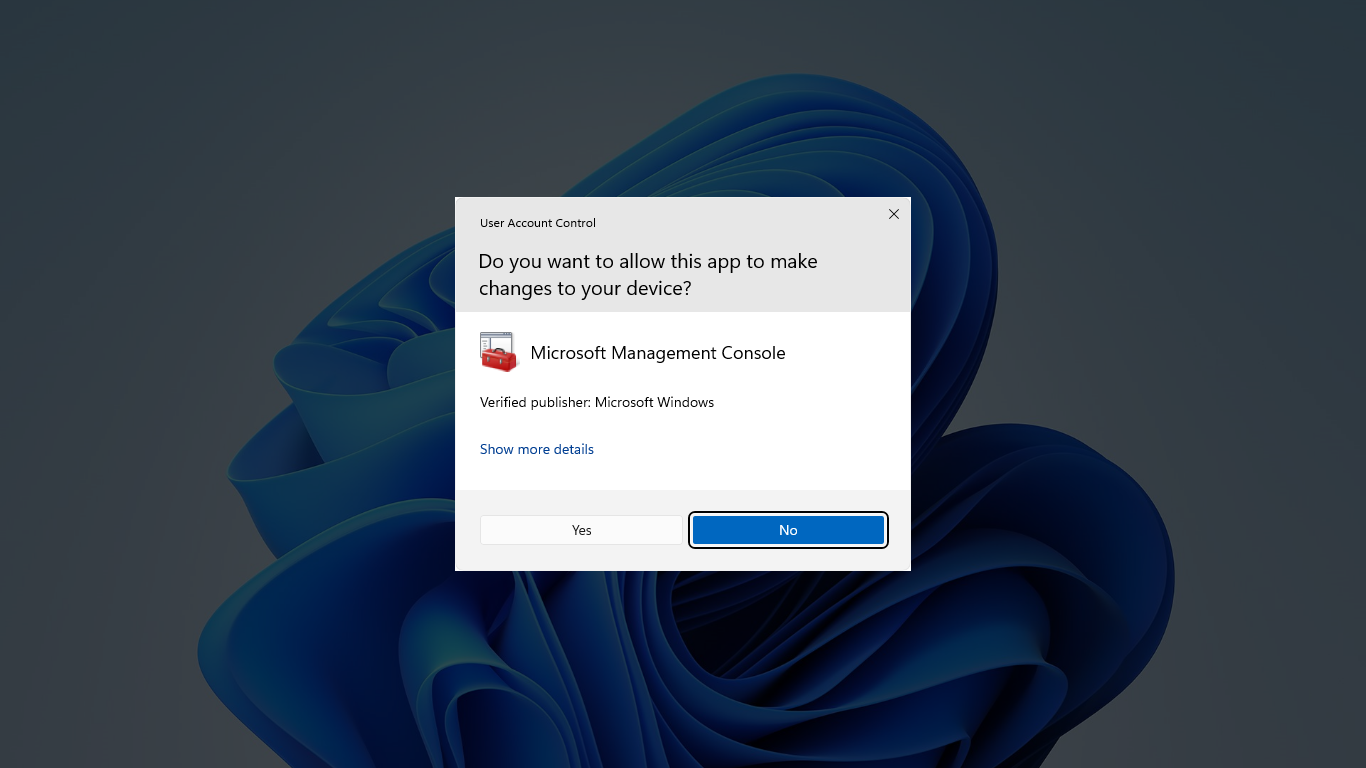
User Account Control (UAC) is a Windows security feature designed to protect the operating system from unauthorized changes. When changes to the system require administrator-level permission, UAC notifies the user, giving the opportunity to approve or deny the change. UAC improves the security of Windows devices by limiting the access that malicious code has to execute with administrator privileges. UAC empowers users to make informed decisions about actions that may affect the stability and security of their device.

**BENEFITS OF UAC:**

UAC allows all users to sign in their devices using a standard user account. Processes launched using a standard user token may perform tasks using access rights granted to a standard user. For instance, Windows Explorer automatically inherits standard user level permissions. Any applications that are started using Windows Explorer (for example, by opening a shortcut) also run with the standard set of user permissions.

When a user tries to perform an action that requires administrative privileges, UAC triggers a *consent prompt*. The prompt notifies the user that a change is about to occur, asking for their permission to proceed:

* If the user approves the change, the action is performed with the highest available privilege
* If the user doesn't approve the change, the action isn't performed and the application that requested the change is prevented from running



For Windows 10:

1. Open Settings:

• Click on the Start menu and select "Settings" (gear icon).

2. Go to "Accounts":

• In the Settings window, click on "Accounts."

3. Select "Sign-in options":

• In the left sidebar, select "Sign-in options."

4. Scroll down and locate "User Account Control":

• You can find the UAC settings under the "Privacy" section.

5. Adjust UAC settings:

• Use the slider to adjust the level of User Account Control. Moving it to the topmost position enhances security by prompting for consent on more actions, while lowering it reduces the frequency of prompts.

## UAC process and interactions

With UAC, each application that requires the administrator access token must prompt the end user for consent. The only exception is the relationship that exists between parent and child processes. Child processes inherit the user's access token from the parent process. Both the parent and child processes, however, must have the same integrity level.

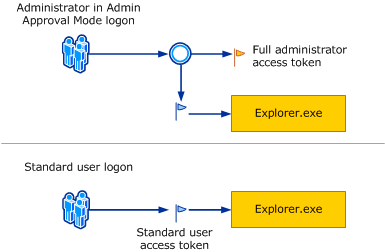
Windows protects processes by marking their integrity levels. Integrity levels are measurements of trust:

* A high integrity application is one that performs tasks that modify system data, such as a disk partitioning application
* A low integrity application is one that performs tasks that could potentially compromise the operating system, like as a Web browser

Applications with lower integrity levels can't modify data in applications with higher integrity levels. When a standard user attempts to run an app that requires an administrator access token, UAC requires that the user provides valid administrator credentials.

To better understand how this process works, let's take a closer look at the Windows sign in process.

**SIGN IN PROCESS:**



By default, both standard and administrator users access resources and execute apps in the security context of a standard user.  
When a user signs in, the system creates an access token for that user. The access token contains information about the level of access that the user is granted, including specific security identifiers (SIDs) and Windows privileges.

When an administrator logs on, two separate access tokens are created for the user: a *standard user access token* and an *administrator access token*. The standard user access token:

* Contains the same user-specific information as the administrator access token, but the administrative Windows privileges and SIDs are removed
* It's used to start applications that don't perform administrative tasks (standard user apps)
* It's used to display the desktop by executing the process *explorer.exe*. Explorer.exe is the parent process from which all other user-initiated processes inherit their access token. As a result, all apps run as a standard user unless a user provides consent or credentials to approve an app to use a full administrative access token

**Please refer to MS website.**

[How User Account Control works - Windows Security | Microsoft Learn](https://learn.microsoft.com/en-us/windows/security/application-security/application-control/user-account-control/how-it-works)s

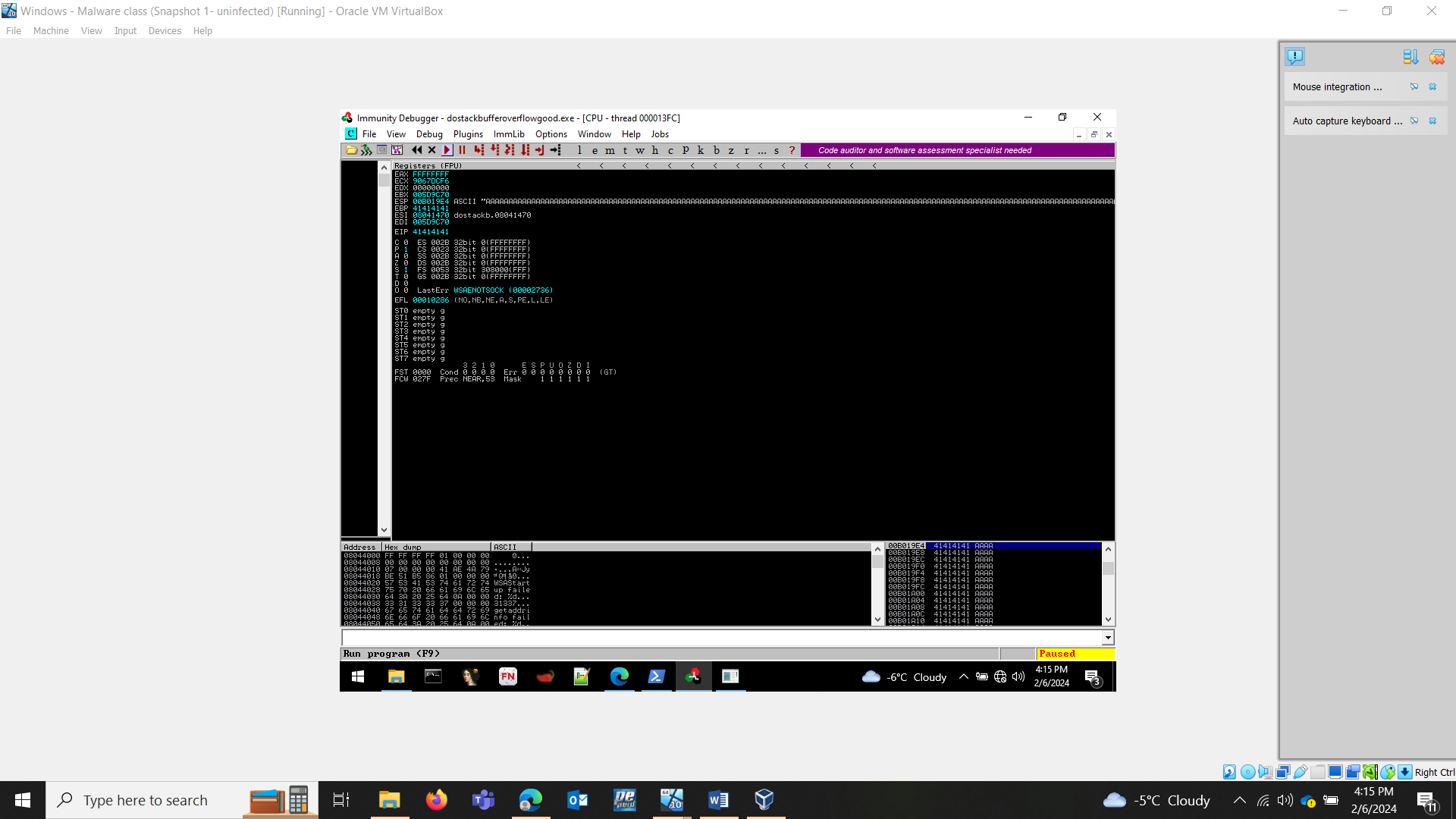
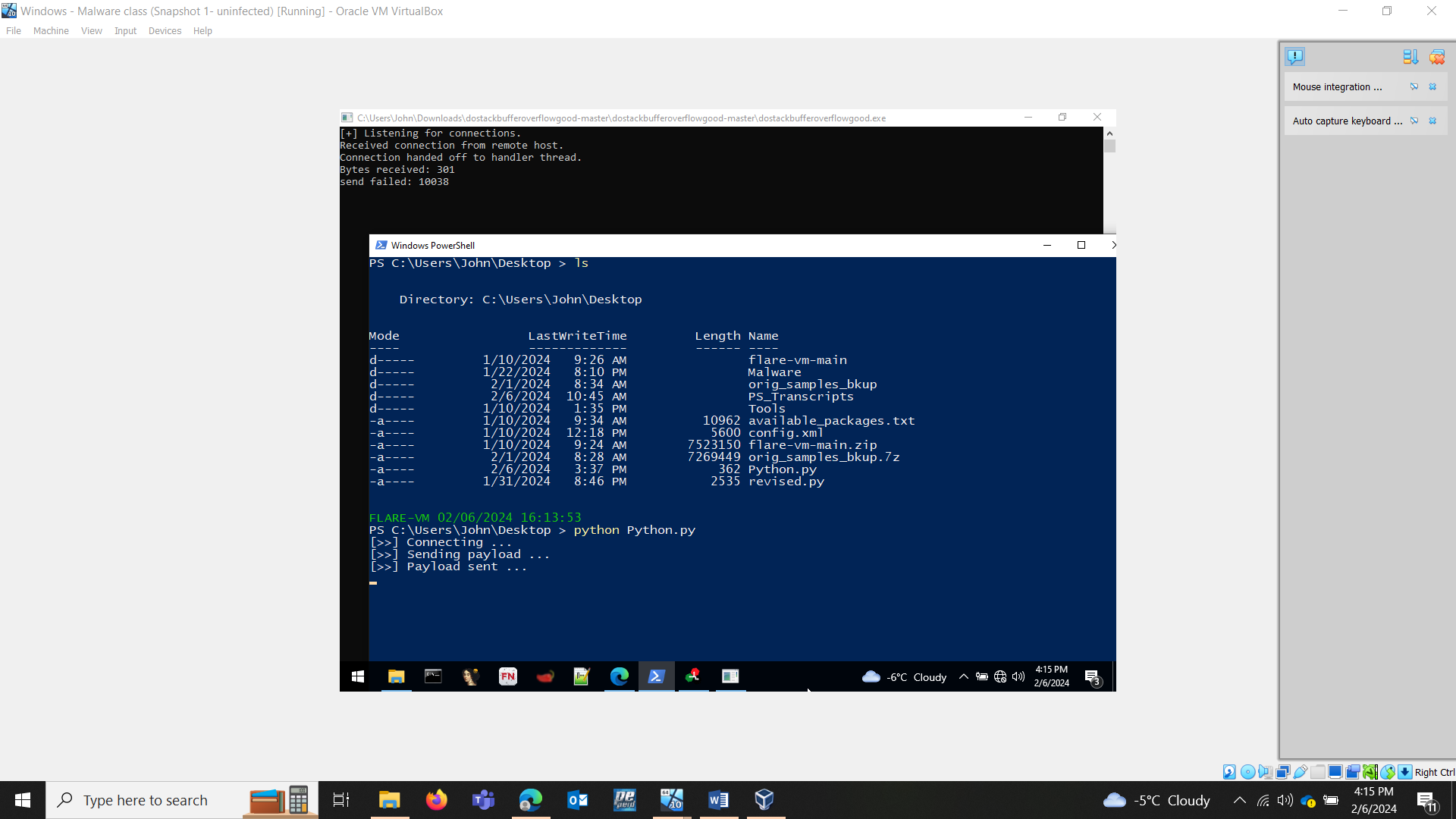
**LAB HELP:**

**Windows 10**

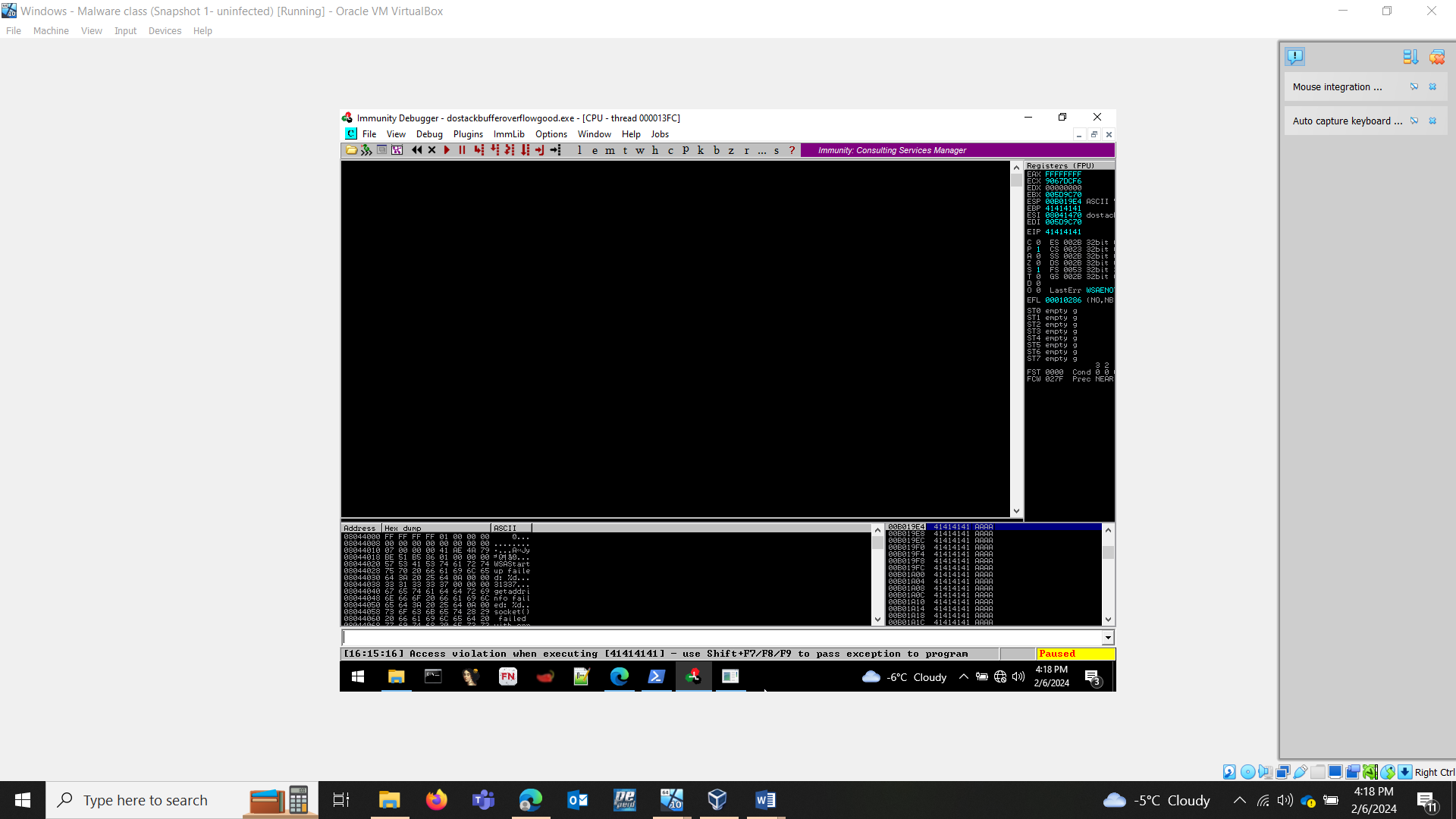
**SCRIPT:**

|  |
| --- |
| import socket  ip\_addr = "127.0.0.1"  port= 31337  s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  payload = "A" \* 300  print('[>>] Connecting ...' )  s.connect((ip\_addr, port))  print('[>>] Sending payload ... ')  s.send((payload +'\n').encode())  print('[>>] Payload sent ... ')  resp = s.recv(1024)  print(resp.decode('utf-8'))  s.close() |

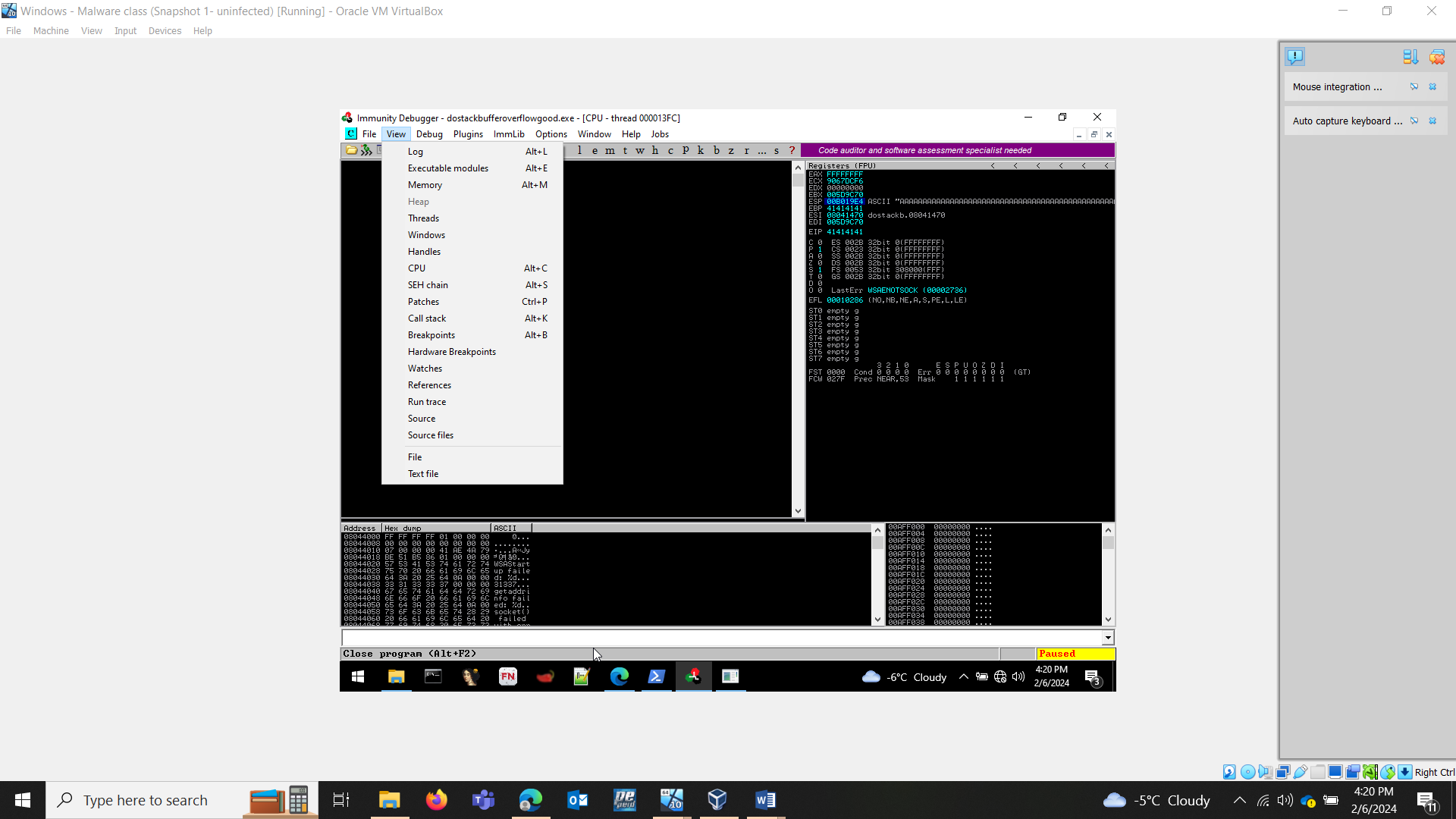
**Intentional Vulnerable Application:** [GitHub - justinsteven/dostackbufferoverflowgood](https://github.com/justinsteven/dostackbufferoverflowgood)

****

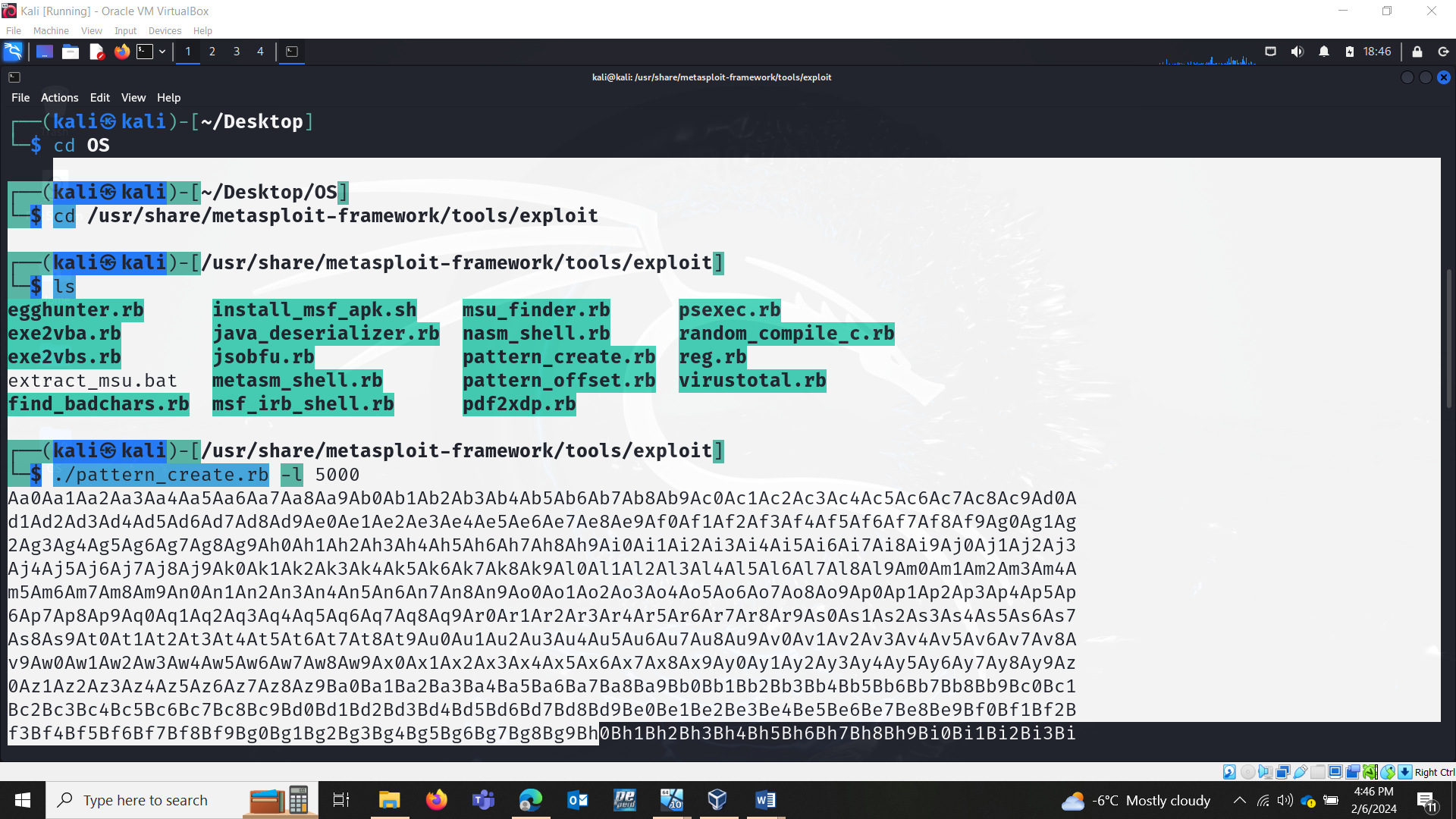
**Access violation when reading 4141418D**

****

**Buffer overflow:**

****

**CREATING PATTERN:**



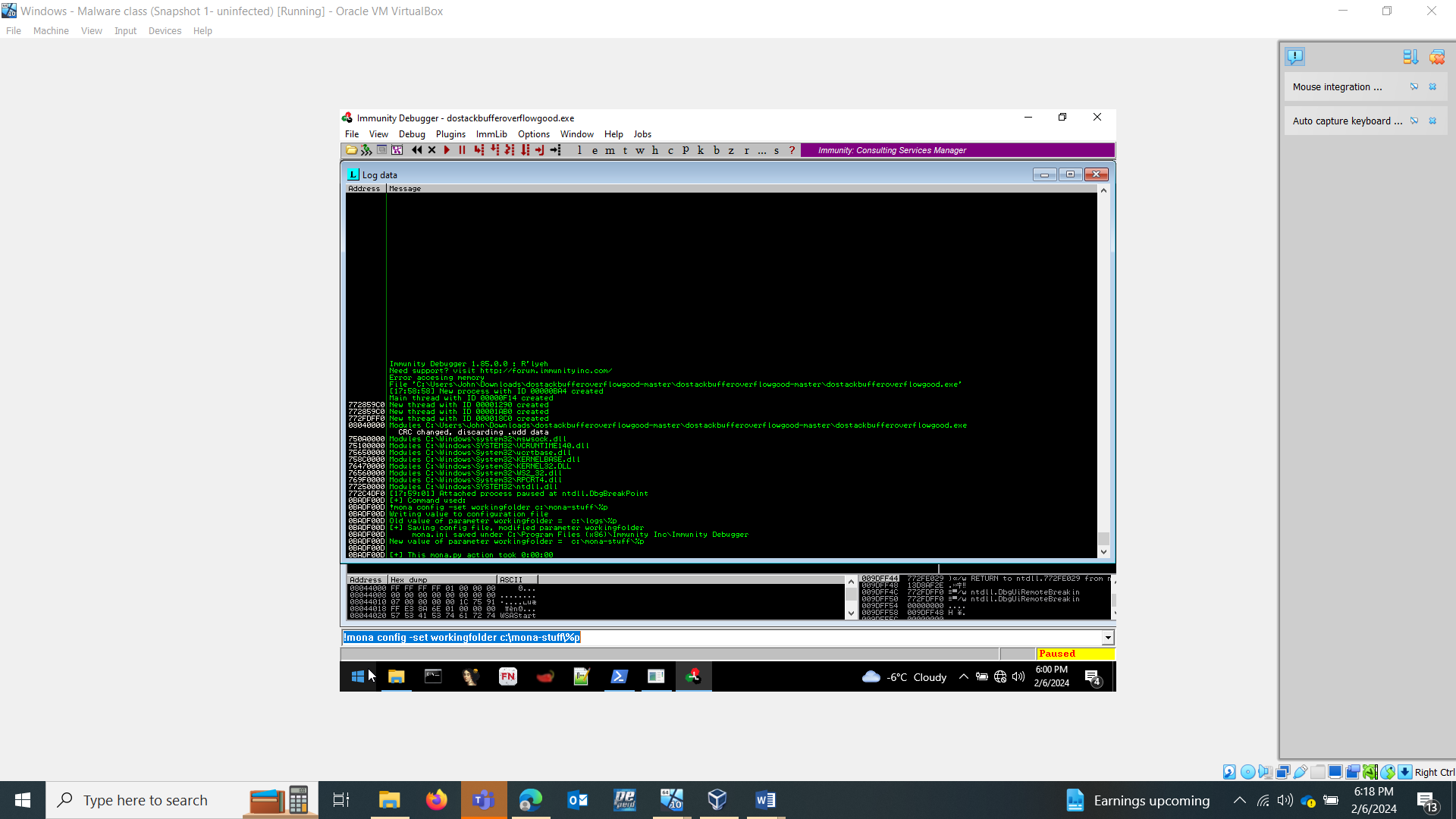
**MONA.PY**

### What is mona.py?

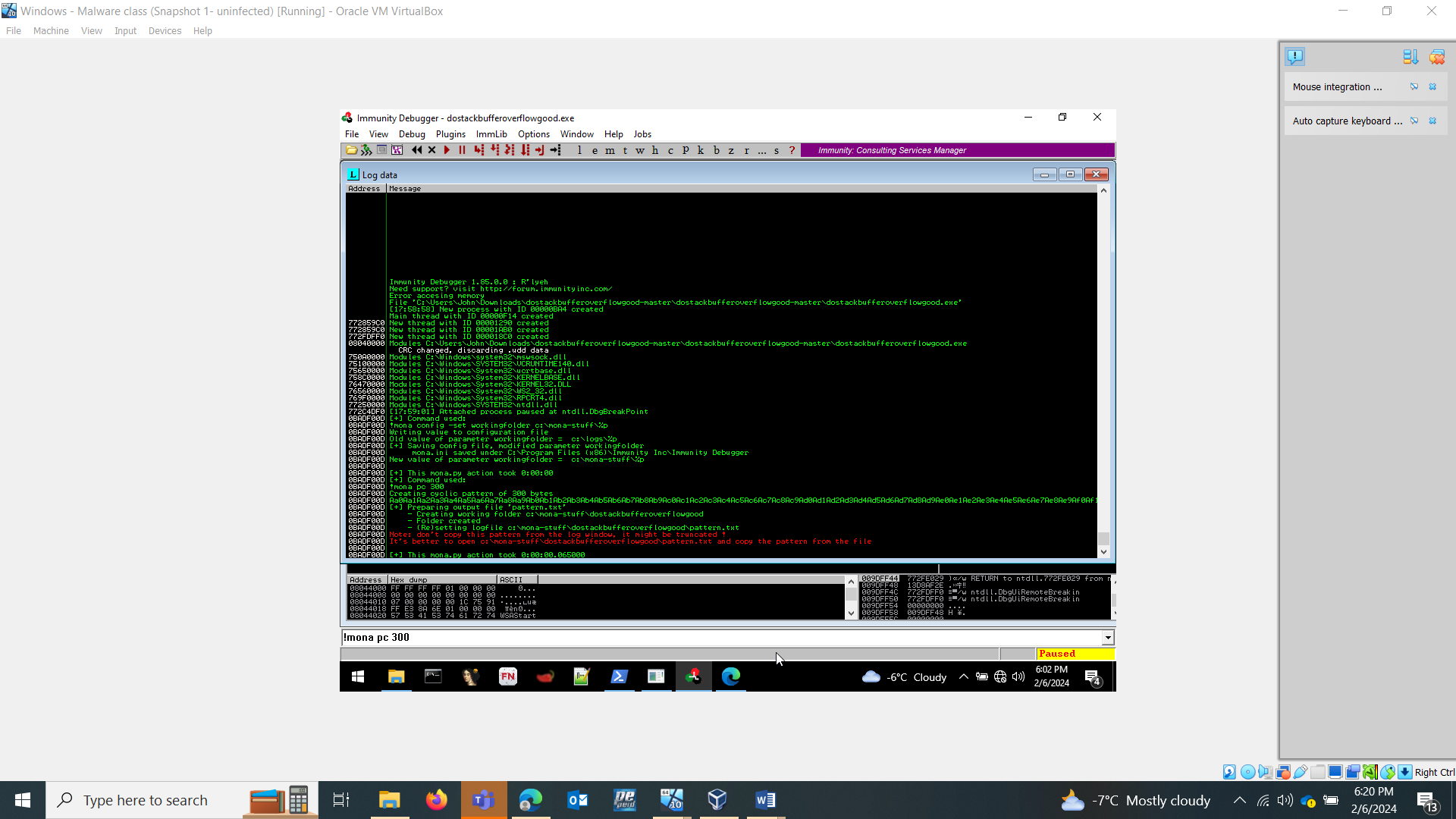
Mona.py is a python script that can be used to automate and speed up specific searches while developing exploits (typically for the Windows platform). It runs on Immunity Debugger and WinDBG, and requires python 2.7. Although it runs in WinDBG x64, the majority of its features were written specifically for 32bit processes.

Inside the installation directory, look for a folder named "pycommands" or "pycommands3" (for Python 3 scripts). This is where you can place Python scripts that will be automatically loaded as commands when Immunity Debugger starts. MOVE YOUR mona.py file to PYCOMMAND directory.

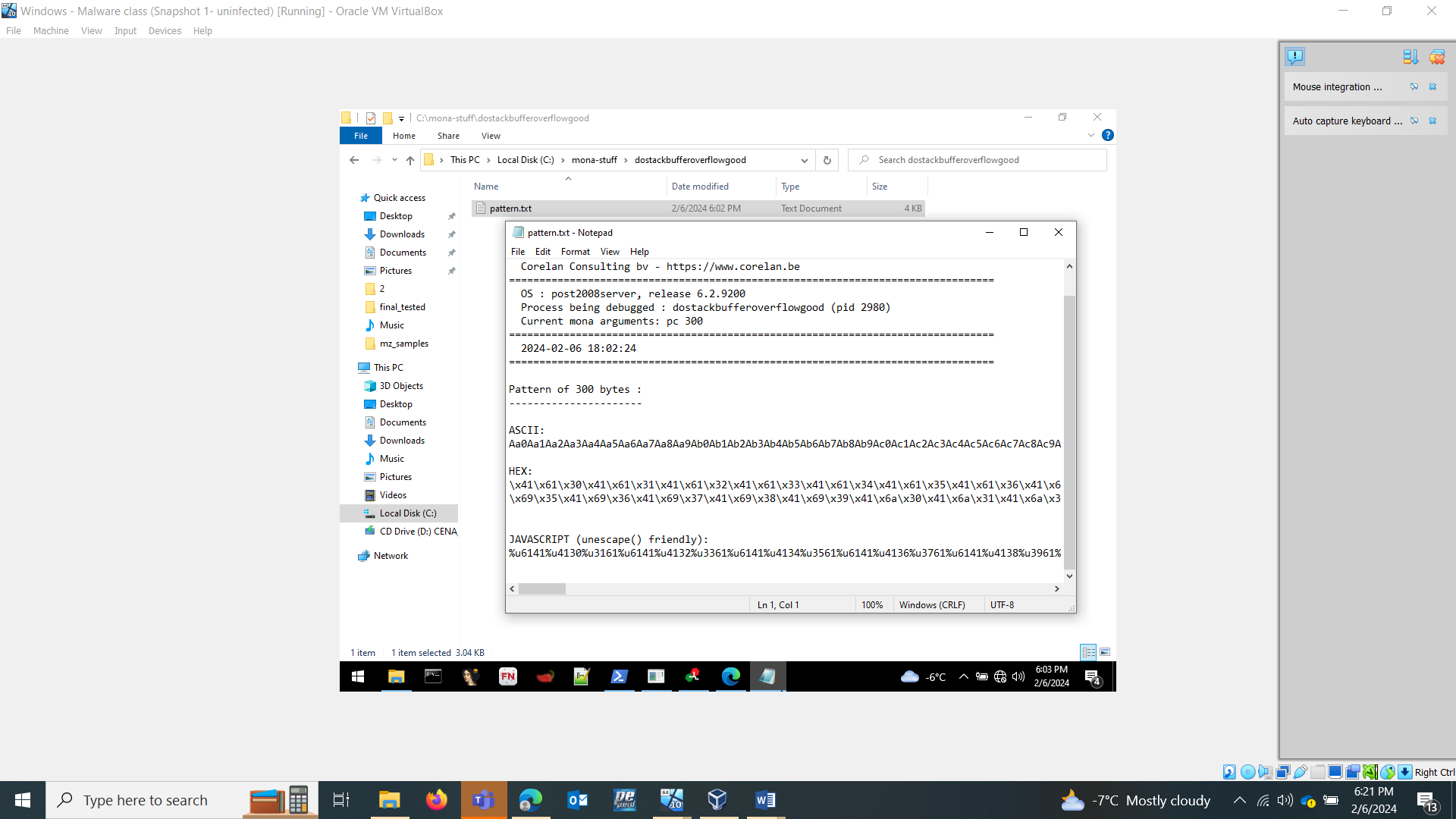
CREATING WORKING DIRECTORY WITH MONA:

****

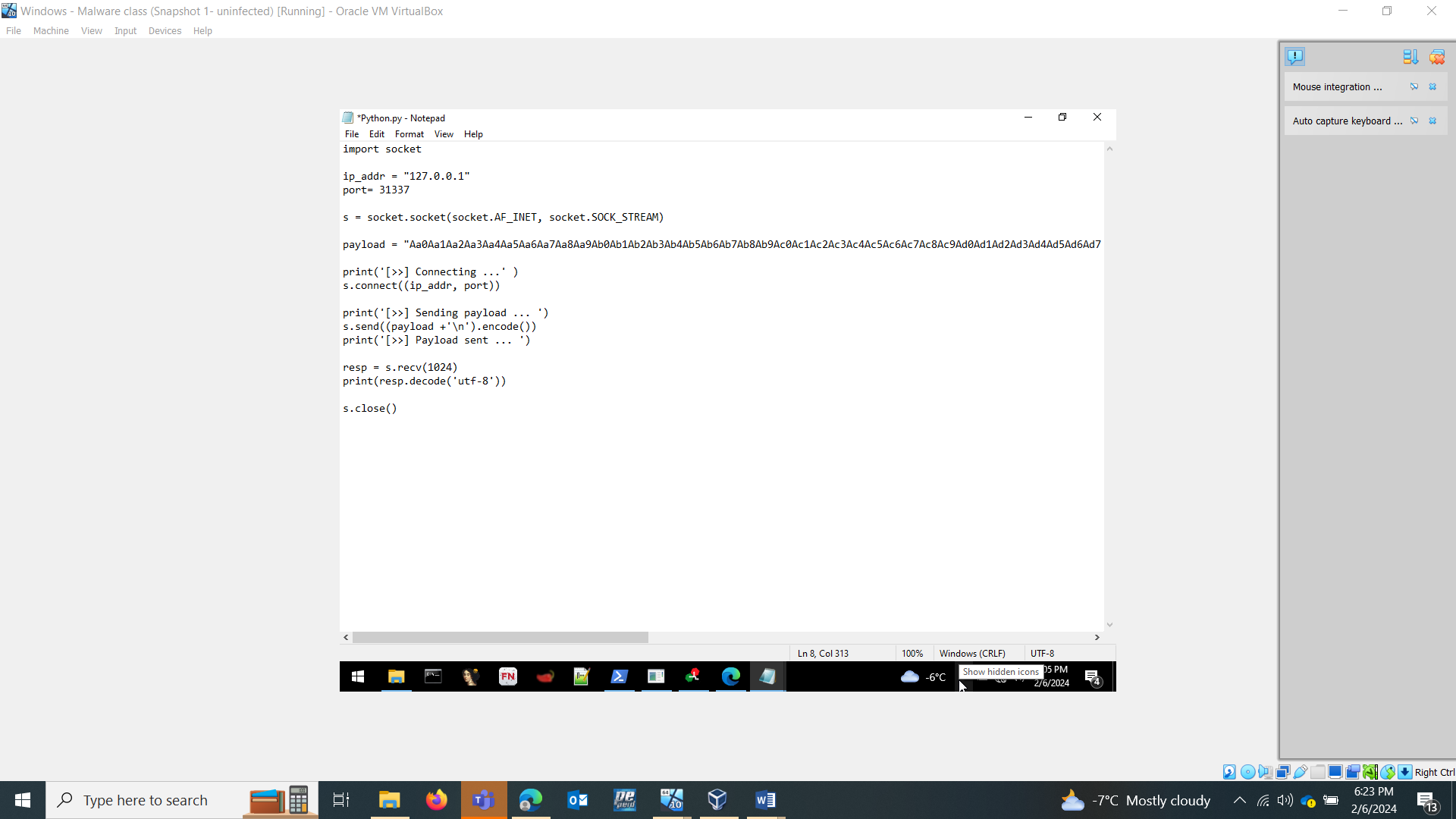
**CREATING A PAYLOAD:**

****

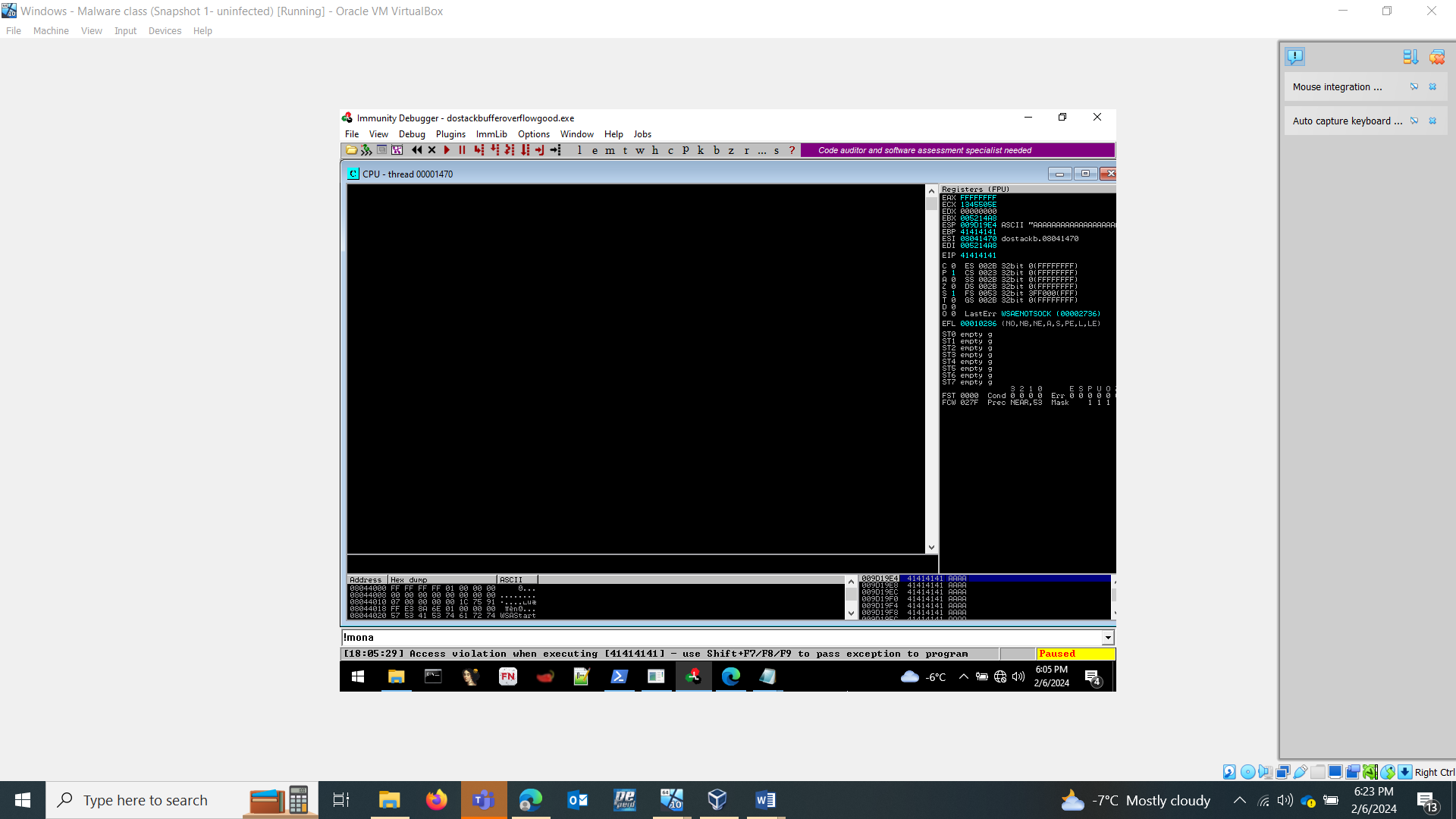
**It will get saved in your working directory.**

****

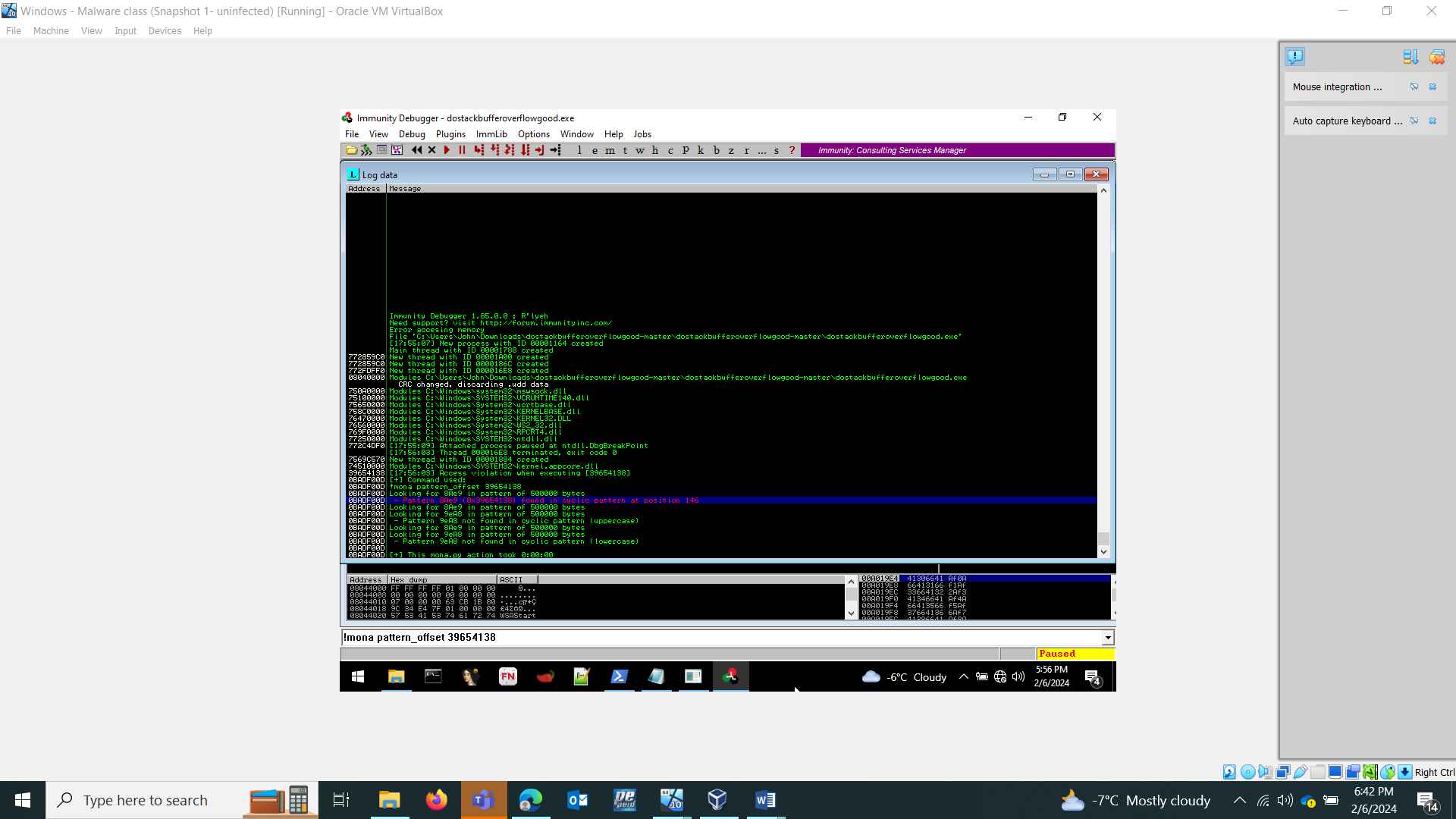
**Including the payload:**

****

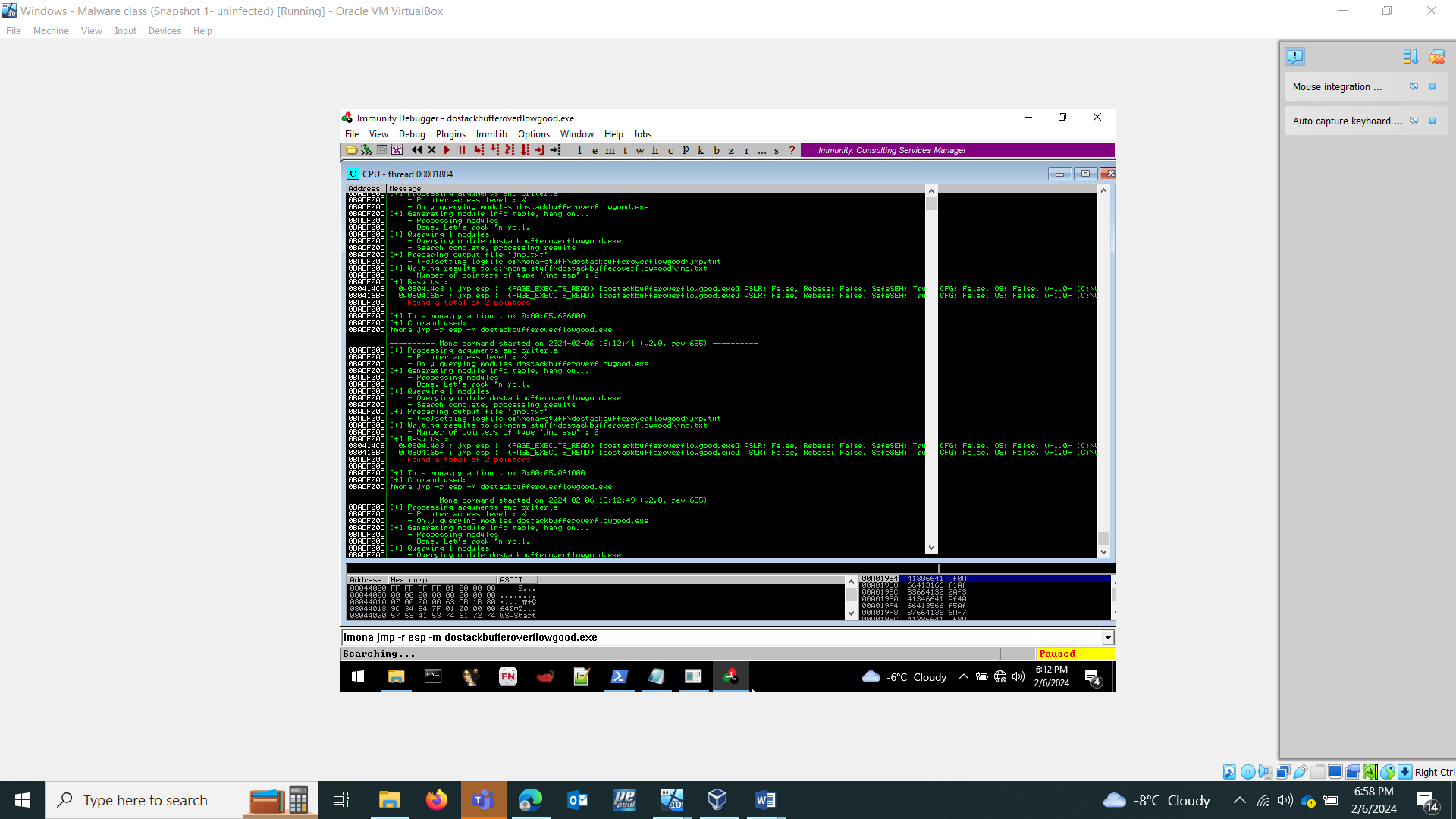
**RESULT:**

****

**CALCULATING OFFSET:**

****

**JMP address:**

****

**Right click on it and pull up the address.**

**EXPOSURE:**

[Creating Win32 ROP Chains - The Human Machine Interface (h0mbre.github.io)](https://h0mbre.github.io/Creating_Win32_ROP_Chains/)

[SEH Exploit | Structured Exception Handler Overwrite (rapid7.com)](https://www.rapid7.com/resources/structured-exception-handler-overwrite-explained/)