

CSC 471 Spring 2025 Lab 2

Dr. Si Chen

Stack and Stack Frame

The goals of this lab are to:

- Understand the concepts of **Stack** and **Stack Frame**.
- Navigate and analyze binary executables using **OlllyDbg**.
- Understand the role of **function calls**, **parameters**, and **return values** in software functionality.
- Develop strategies for binary modification that achieve desired outcomes without introducing errors or instability.

Introduction

This lab is inspired by a series of reverse engineering tutorials aimed at beginners, showcasing the practical application of code analysis techniques. The tutorials, popular for their approachable format and comprehensive coverage, serve as a valuable resource for those interested in delving into the realm of software reverse engineering. Through this lab, participants will gain hands-on experience with binary manipulation, enhancing their understanding of software internals. The primary focus is to apply learned concepts in a controlled environment, reinforcing theoretical knowledge with practical skills. This approach not only demystifies the underlying mechanics of executable files but also equips learners with the tools necessary for effective problem-solving in the field of cybersecurity.

Background

The lab exercises are structured around a challenge involving a binary file, **lab2.exe**, that presents a "Nag screen" upon execution. The task is to remove this screen by modifying the binary, using **OlllyDbg**, a popular debugger for reverse engineering. This scenario mimics real-world reverse engineering tasks, where understanding the software's flow and manipulating its execution are essential skills. By engaging with this exercise, learners will navigate the complexities of binary analysis, gaining insights into the software's structure and behavior, and the impact of assembly-level modifications.

The **lab2.exe** for debugging is crafted in Visual Basic. Before diving into debugging, it's useful to grasp the features of Visual Basic files.

0.0.1 VB-Specific Engine

Visual Basic files leverage a VB-specific engine called **MSVBVM50.DLL** (Microsoft Visual Basic Virtual Machine 5.0), also known as The Thunder Runtime Engine. For instance, to display a message box, VB code necessitates calling the **MsgBox** function. Actually, the VB editor genuinely invokes the **rtcMsgBox** function within **MSVBVM50.DLL**, which consequently functions by calling the **MessageBoxW** function (Win32 API) inside **user32.dll** (this can also be directly invoked in VB code).

0.0.2 Native Code and P-Code

Based on the compilation options employed, VB files may be compiled into native code (N-Code) and P-Code. Native code generally utilizes IA-32 instructions more decipherable by debuggers, whereas P-Code is an interpreter language utilizing self-parsing instructions (bytecode) through a virtual machine implemented by the VB engine. Accurate parsing of VB's P-Code necessitates analysis of the VB engine and emulation implementation.

0.0.3 Event Handlers

Visual Basic is extensively used for developing GUI programs, making its IDE interface ideally suited for GUI programming. VB programs operate on an event-driven model employed by the Windows operating system, meaning that user code does not exist in functions like **main()** or **WinMain()**; instead, user code is found within various event handlers.

0.0.4 Undocumented Structures

VB utilizes various structures to store information (such as Dialog, Control, Form, Module, Function, etc.) within the file. Since Microsoft has not officially disclosed these structures, debugging VB files can become somewhat more challenging.

Objectives and Targets

Download **lab2.exe** to your Windows XP VM and run it to display a Nag screen, as shown in Figure 1. Your task is to completely remove the Nag screen by modifying the binary program using OllyDbg.

Experiment Setup

1. Start Windows XP in VirtualBox.
2. Inside Windows XP, download or copy **lab2.exe** to a folder, accessible via:

<https://www.cs.wcupa.edu/schen/malware25/download/lab2.exe>

3. Open **lab2.exe** with Ollydbg.

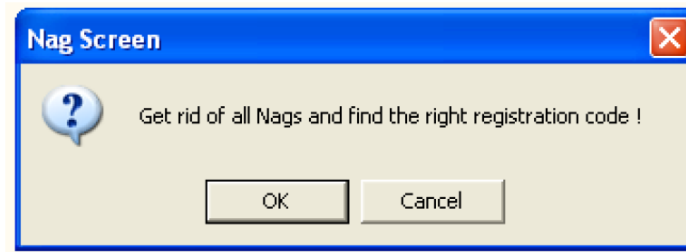


Figure 1: Nag Screen of lab2

4. Review the following questions in "Lab Exercise" and include your answers in your report.

Lab Exercise: Remove the Nag screen

Step 1: Analysis and Patching Process

The goal of this exercise is to eliminate the Nag screen by modifying the program's execution flow. This involves identifying the specific function call that triggers the Nag screen, which, through analysis with `OlllyDbg`, is found to be a call to the `rtcMsgBox` function from the Visual Basic runtime library, `MSVBVM50`.

Step 2: Identifying and Modifying the Function Call

Utilizing `OlllyDbg`'s "Search for - All intermodular calls" feature helps locate the `rtcMsgBox` function calls. Setting breakpoints on these calls allows us to pause execution right at the critical moment before the Nag screen is displayed, offering a precise location for modification.

Upon analyzing the call at `0x402CFE`, Dr. Chen decides to modify the "CALL XXXX" instruction to "ADD ESP,14", effectively skipping over the call to the Nag screen function. To maintain the integrity of the code flow, two "NOP" instructions are added to fill in the gap left by the shortened instruction. This strategic patch removes the Nag screen without altering the program's core functionality.

However, this modification caused errors because `rtcMsgBox()` needs to return a value of 1 to indicate successful display, which his modification did not account for.

Question: Which CPU register is used to store the return value (1) of the function `rtcMsgBox()`? Why?

Dr. Chen found another way to "hack" this program by changing the instruction at `0x402C17` from "PUSH EBP" to "RETN 4", successfully removing the Nag screen.

Question: What is the meaning of "PUSH EBP, MOV EBP, ESP"?

```

00402CB5 . 8D95 7CFFFFFF LEA EDX,DWORD PTR SS:[EBP-84]
00402CB8 . 8D4D CC       LEA ECX,DWORD PTR SS:[EBP-34]
00402CBB . C745 84 781F44 MOV DWORD PTR SS:[EBP-7C],1ab3.00401F78
00402CC5 . 899D 7CFFFFFF MOV DWORD PTR SS:[EBP-84],EBX
00402CCB . E8 5CE4FFFF CALL <JMP.&MSVBVM50.__vbaVarCopy>
00402CD0 . 6A 00       PUSH 00
00402CD2 . B9 04000200 MOV ECX,00020004
00402CD7 . 50         POP EAX
00402CD8 . 894D 94       MOV DWORD PTR SS:[EBP-6C],ECX
00402CDB . 8945 8C       MOV DWORD PTR SS:[EBP-74],EAX
00402CDE . 8945 9C       MOV DWORD PTR SS:[EBP-64],EAX
00402CE1 . 8D45 8C       LEA ECX,DWORD PTR SS:[EBP-74]
00402CE4 . 894D A4       MOV DWORD PTR SS:[EBP-5C],ECX
00402CE7 . 50         PUSH EAX
00402CE8 . 8D45 9C       LEA ECX,DWORD PTR SS:[EBP-64]
00402CEB . 50         PUSH EAX
00402CEC . 8D45 CC       LEA ECX,DWORD PTR SS:[EBP-34]
00402CEF . 50         PUSH EAX
00402CF0 . 8D45 DC       LEA ECX,DWORD PTR SS:[EBP-24]
00402CF3 . 50         PUSH EAX
00402CF4 . E8 21E4FFFF CALL <JMP.&MSVBVM50.__vbaI4Var>
00402CF9 . 50         PUSH EAX
00402CFA . 8D45 AC       LEA ECX,DWORD PTR SS:[EBP-54]
00402CFD . 50         PUSH EAX
00402CFE . E8 1DE4FFFF CALL <JMP.&MSVBVM50.#595>
00402D03 . 8D95 5CFFFFFF LEA EDX,DWORD PTR SS:[EBP-A4]
00402D09 . 8D4D BC       LEA ECX,DWORD PTR SS:[EBP-44]
00402D0C . 89B5 64FFFFFF MOV DWORD PTR SS:[EBP-9C],EAX
00402D12 . 89BD 5CFFFFFF MOV DWORD PTR SS:[EBP-A4],EDI
00402D18 . E8 09E4FFFF CALL <JMP.&MSVBVM50.__vbaVarMove>
00402D1D . 8D45 8C       LEA ECX,DWORD PTR SS:[EBP-74]
00402D20 . 50         PUSH EAX
00402D21 . 8D45 9C       LEA ECX,DWORD PTR SS:[EBP-64]
00402D24 . 50         PUSH EAX
00402D25 . 6A 02       PUSH 2
00402D27 . E8 E8E3FFFF CALL <JMP.&MSVBVM50.__vbaFreeVarList>
00402D2C . 83C4 0C       ADD ESP,0C
00402D2F . 8D45 BC       LEA ECX,DWORD PTR SS:[EBP-44]
00402D32 . C745 84 010000 MOV DWORD PTR SS:[EBP-7C],1
00402D39 . C7B5 7CFFFFFF MOV DWORD PTR SS:[EBP-84],0003
00402D43 . 50         PUSH EAX
00402D44 . 8D85 7CFFFFFF LEA ECX,DWORD PTR SS:[EBP-84]
00402D4B . 50         PUSH EAX
00402D4B . E8 BEE3FFFF CALL <JMP.&MSVBVM50.__vbaVarTstEq>
00402D50 . 66 85C0      TEST AX,AX
00402D53 . 75 05       JNZ SHORT 1ab3.00402D5A
00402D55 . E8 AEE3FFFF CALL <JMP.&MSVBVM50.__vbaEnd>
00402D5B . 8975 FC     MOV DWORD PTR SS:[EBP-4],ESI
00402D5D . 68 98D40000 PUSH 1ab3.00402D58
00402D62 . FR 13      JMP SHORT 1ab3.00402D77

```

Figure 2: Disassembled code showing the call to rtcMsgBox at 0x402CFE

```

00402CE7 . 50         PUSH EAX
00402CE8 . 8D45 9C       LEA ECX,DWORD PTR SS:[EBP-64]
00402CEB . 50         PUSH EAX
00402CEC . 8D45 CC       LEA ECX,DWORD PTR SS:[EBP-34]
00402CEF . 50         PUSH EAX
00402CF0 . 8D45 DC       LEA ECX,DWORD PTR SS:[EBP-24]
00402CF3 . 50         PUSH EAX
00402CF4 . E8 21E4FFFF CALL <JMP.&MSVBVM50.__vbaI4Var>
00402CF9 . 50         PUSH EAX
00402CFA . 8D45 AC       LEA ECX,DWORD PTR SS:[EBP-54]
00402CFD . 50         PUSH EAX
00402C1F . 83C4 14      ADD ESP,14
00402D01 . 90         NOP
00402D02 . 90         NOP
00402D03 . 8D95 5CFFFFFF LEA EDX,DWORD PTR SS:[EBP-A4]
00402D09 . 8D4D BC       LEA ECX,DWORD PTR SS:[EBP-44]
00402D0C . 89B5 64FFFFFF MOV DWORD PTR SS:[EBP-9C],EAX
00402D12 . 89BD 5CFFFFFF MOV DWORD PTR SS:[EBP-A4],EDI
00402D18 . E8 09E4FFFF CALL <JMP.&MSVBVM50.__vbaVarMove>
00402D1D . 8D45 8C       LEA ECX,DWORD PTR SS:[EBP-74]
00402D20 . 50         PUSH EAX
00402D21 . 8D45 9C       LEA ECX,DWORD PTR SS:[EBP-64]
00402D24 . 50         PUSH EAX
00402D25 . 6A 02       PUSH 2
00402D27 . E8 E8E3FFFF CALL <JMP.&MSVBVM50.__vbaFreeVarList>
00402D2C . 83C4 0C       ADD ESP,0C
00402D2F . 8D45 BC       LEA ECX,DWORD PTR SS:[EBP-44]
00402D32 . C745 84 010000 MOV DWORD PTR SS:[EBP-7C],1
00402D39 . C7B5 7CFFFFFF MOV DWORD PTR SS:[EBP-84],0003
00402D43 . 50         PUSH EAX
00402D44 . 8D85 7CFFFFFF LEA ECX,DWORD PTR SS:[EBP-84]
00402D4B . 50         PUSH EAX
00402D4B . E8 BEE3FFFF CALL <JMP.&MSVBVM50.__vbaVarTstEq>
00402D50 . 66 85C0      TEST AX,AX
00402D53 . 75 05       JNZ SHORT 1ab3.00402D5A
00402D55 . E8 AEE3FFFF CALL <JMP.&MSVBVM50.__vbaEnd>
00402D5B . 8975 FC     MOV DWORD PTR SS:[EBP-4],ESI
00402D5D . 68 98D40000 PUSH 1ab3.00402D58
00402D62 . FR 13      JMP SHORT 1ab3.00402D77

```

Figure 3: Modified disassembled code of lab2

Question: Please explain why changing the instruction at 0x402C17 from "PUSH EBP" to "RETN 4" removes the Nag screen.

Bonus: Finding the Registration Code (extra 2 points)

In this exercise, you will explore the process of identifying and validating a registration code for lab2.exe using OllyDbg. The goal is to understand how conditional checks and string comparisons are performed in assembly language and how to manipulate these checks to discover the correct registration code.

```

00402C09 . 5F          POP     EDI
00402C0A . 5E          POP     ESI
00402C0B . 64:890D 000000 MOV    DWORD PTR FS:[0],ECX
00402C0C . 5B          POP     EBX
00402C0D . C9          LEAVE
00402C0E . C2 0400    RETN    4
00402C0F . C2 0400    RETN    4
00402C10 . 83EC 0C     SUB     ESP,0C
00402C11 . 68 66104000 PUSH    <JMP.&MSVBM50..._vbaExceptionHandler> SE handler installation
00402C12 . 64:A1 00000000 MOV    EAX,DWORD PTR FS:[0]
00402C13 . 50          PUSH    EAX
00402C14 . 64:8925 000000 MOV    DWORD PTR FS:[0],ESP
00402C15 . 81EC 98000000 SUB     ESP,98
00402C16 . 8B45 08     MOV    EAX,DWORD PTR SS:[EBP+8]
00402C17 . 8B45 08 FE  AND    DWORD PTR SS:[EBP+8],0FFFFFFF
00402C18 . 83E0 01     AND    EAX,1
00402C19 . C745 F8 181040 MOV    DWORD PTR SS:[EBP+8],1ab3.00401018
00402C1A . 53          PUSH    EBX
00402C1B . 8945 FC     MOV    DWORD PTR SS:[EBP+4],EAX
00402C1C . 8B45 08     MOV    EAX,DWORD PTR SS:[EBP+8]
00402C1D . 56          PUSH    ESI
00402C1E . 57          PUSH    EDI
00402C1F . 8B08        MOV    ECX,DWORD PTR DS:[EAX]
00402C20 . 8965 F4     MOV    DWORD PTR SS:[EBP+C],ESP
00402C21 . 50          PUSH    EAX
00402C22 . FF51 04     CALL   DWORD PTR DS:[ECX+4]
00402C23 . 6A 08       PUSH    8
00402C24 . 33F6        XOR     ESI,ESI
00402C25 . 5B          POP     EBX
00402C26 . 89B5 7CFFFF MOV    DWORD PTR SS:[EBP-84],ESI
00402C27 . 8D95 7CFFFF LEA     EDI,DWORD PTR SS:[EBP-84]
00402C28 . 8D4D AC     LEA     ECX,DWORD PTR SS:[EBP-54]
00402C29 . 8975 DC     MOV    DWORD PTR SS:[EBP-24],ESI
00402C2A . 8975 CC     MOV    DWORD PTR SS:[EBP-34],ESI
00402C2B . 8975 BC     MOV    DWORD PTR SS:[EBP-44],ESI
00402C2C . 8975 AC     MOV    DWORD PTR SS:[EBP-54],ESI
00402C2D . 8975 9C     MOV    DWORD PTR SS:[EBP-64],ESI
00402C2E . 8975 8C     MOV    DWORD PTR SS:[EBP-74],ESI
00402C2F . 89B5 5CFFFF MOV    DWORD PTR SS:[EBP-84],ESI

```

Figure 4: Another method to modify the disassembled code of lab2

The program uses a string comparison function to compare the entered registration code against the correct one. This function is typically named `__vbaStrCmp` in Visual Basic compiled programs. Locate the call to the `__vbaStrCmp` function related to the registration code check. Think the following question – what are the two strings being compared by this function? Upon finding the string comparison, you’ll notice the hardcoded correct registration code in the vicinity of the comparison function call.

Question: What is the hardcoded registration code found near the `__vbaStrCmp` function call? Hint: Look for a string that is compared against the user input.

Hint

Check the lecture slides and video – Class 5, 6 Stack and Stack Frame, and Class3 & 4 for IA32 CPU register and X86 ASM basics.

Submission

- The lab due date is available on our course website. Late submissions will not be accepted.
- Submit your assignment directly to D2L.
- Include a detailed project report in PDF format describing your process, including screenshots of the final result.
- No copy or cheating is tolerated. If your work is based on others’, give clear attribution, or you will fail this course.