West Chester University

CSC 471

Dr. Si Chen

Spring 2024 Lab 2

Submitted by

Sean Berlin

3/5/24

1. Introduction:

The purpose of this lab is for students to explore the concepts of stack and stack frame. By utilizing beginner-friendly reverse engineering tutorials, students gain hands-on experience exploring code analysis concepts in a controlled environment. Throughout the exercises, the aim is to learn about the complexities of executables. Specifically, focusing on understanding the Stack and Stack Frame, maneuvering through binary executables with OllyDbg, and dissecting the mechanisms behind function calls, parameters, and return values. Furthermore, by developing strategies for binary modification, students will gain practical skills working with the underlying mechanics of executable files and gain the necessary critical thinking skills needed for effective problem-solving in cybersecurity.

2. Analysis and Results:

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

The CPU register used to store the return value is the EAX (Extended Accumulator) register. This register is used for holding function return values since it's specifically designated for arithmetic operations. As a result, it often serves as a place to store return values. In this case, rtcMsgBox() stores its return value in EAX because it's a convention followed by many calling conventions, including those used by the Visual Basic runtime library, MSVBVM50.

Question 2: What is the meaning of ”PUSH EBP, MOV EBP, ESP”?

The assembly instruction means a new stack frame is trying to be created. Specifically, the current EBP value is being saved to the stack. The ‘PUSH EBP’ pushes the value of the EBP, extended base pointer, register onto the top of the stack. Furthermore, the ‘MOV EBP, ESP instruction moves the current value of the ESP (Extended Stack Pointer) register into the EBP register. This establishes a new base pointer for the current function by setting it to the current top of the stack. Ultimately, these instructions set up the stack frame.

Question 3: Please explain why changing the instruction at 0x402C17 from ”PUSH EBP” to ”RETN 4” removes the Nag screen.

The assembly instruction ultimately removes the stack frame. First, changing the instruction from "PUSH EBP" to "RETN 4" at memory address 0x402C17 alters the program's execution flow. "PUSH EBP" is used at the beginning of the function to save the previous base pointer, whereas "RETN 4" returns from a function and cleans up the stack. This change skips over the code responsible for displaying a Nag screen because it prematurely exits the function. To add, the "4" in "RETN 4" indicates that after popping the return address, the stack pointer (ESP) will be incremented by 4 bytes.

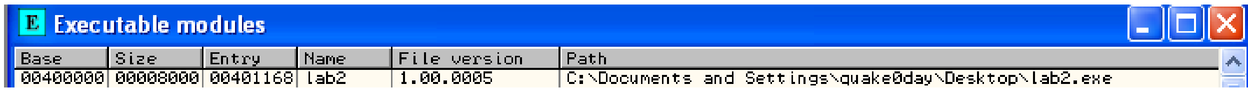
Question 4: What is the hardcoded registration code found near the vbaStrCmp function call? Hint: Look for a string that is compared against the user input.

The hardcoded registration code is “I’mlena151”. It was located at address 00402A2A as shown in *Figure 5*.

Analysis:

The attempts taken to remove the Nag screen were done by modifying the program’s execution flow and identifying the specific function call that triggers the Nag screen. The first step taken was to use ollydbg’s “Search for - All intermodular calls” function. This served to help locate the rtcMsgBox function calls memory address. It was revealed the base address was 00400000, so this served as a general range for finding the rtcMsgBox function. Observe *Figure 1* for this demonstration. This served as a universal step for the following two attempts at removing the Nag screen. The first attempt was to modify the binary file at the memory address 0x402CFE. The instruction was modified to ”ADD ESP,14”, effectively skipping over the call to the Nag screen function. Two "NOP" instructions are added to fill in the space left by the shortened instruction in order to preserve the integrity of the code flow. This change is shown in *Figure 2*. Yet, this modification caused errors as the replaced function needs to return a value of 1. Next, the second attempt took a similar approach by changing the instruction at 0x402C17 from ”PUSH EBP” to ”RETN 4”. The explanation for this instruction change is provided in the above “Question 4”. The code edit demonstration is shown in *Figure 3*, along with the absent Nag screen when the “Nag?” Button Event is clicked. To summarize, this successfully removed the Nag screen. The next challenge was identifying and validating the correct registration code. The first step taken for this exercise was to find “_ _vbaStrCmp” function Address from Intermodular Calls as shown in *Figure 4*. Next, by examining the instructions around this memory address, 00402A2F, the correct hardcoded registration code was found in the vicinity of the comparison function call. This is demonstrated in *Figure 5*. Lastly, the correct registration code, “I’mlena151”, was used as input to demonstrate its validity, and this can be seen in *Figure 6*.

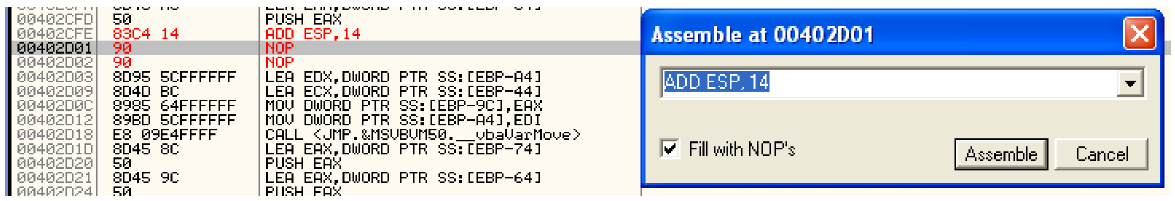
Screenshot of Executable Modules



Base	Size	Entry	Name	File version	Path
00400000	00008000	00401168	lab2	1.00.0005	C:\Documents and Settings\quake0day\Desktop\lab2.exe

Figure 1

Screenshot of Modified Disassessembled Code using RETN 4



```
00402CFD  50              PUSH EAX
00402CFE  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  8985 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 EAX,DWORD PTR SS:[EBP-74]
00402D20  50              PUSH EAX
00402D21  8D45 9C         LEA EAX,DWORD PTR SS:[EBP-64]
```

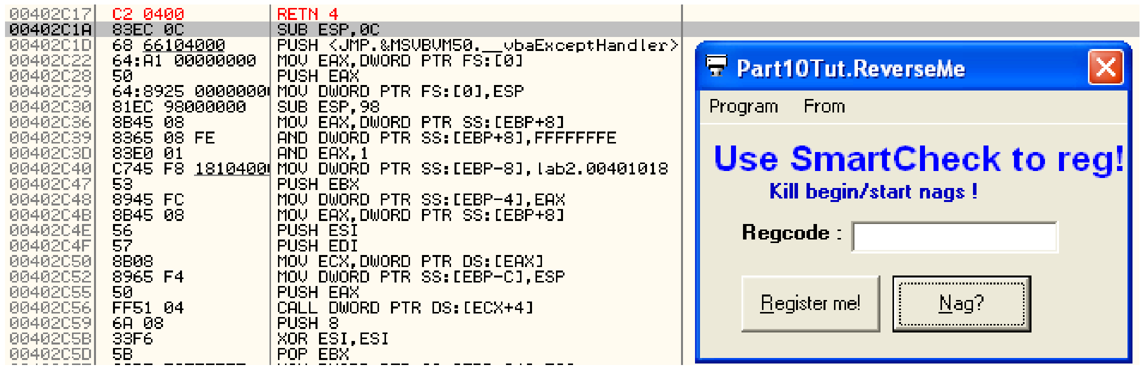
Assemble at 00402D01

ADD ESP, 14

[✓] Fill with NOP's

Figure 2

Screenshot of Modified Disassessembled Code using RETN 4



```
00402C17  C2 0400          RETN 4
00402C1A  83EC 0C          SUB ESP,0C
00402C1D  68 66104000      PUSH <JMP.&MSVBVM50.__vbaExceptHandler>
00402C22  64:A1 00000000   MOV EAX,DWORD PTR FS:[0]
00402C28  50               PUSH EAX
00402C29  64:8925 0000000  MOV DWORD PTR FS:[0],ESP
00402C30  81EC 98000000    SUB ESP,98
00402C36  8B45 08          MOV EAX,DWORD PTR SS:[EBP+8]
00402C39  8365 08 FE       AND DWORD PTR SS:[EBP+8],FFFFFFFE
00402C3D  83E0 01          AND EAX,1
00402C40  C745 F8 1810400  MOV DWORD PTR SS:[EBP-8],lab2.00401018
00402C47  53               PUSH EBX
00402C48  8945 FC          MOV DWORD PTR SS:[EBP-4],EAX
00402C4B  8B45 08          MOV EAX,DWORD PTR SS:[EBP+8]
00402C4E  56               PUSH ESI
00402C4F  57               PUSH EDI
00402C50  8B08             MOV ECX,DWORD PTR DS:[EAX]
00402C52  8965 F4          MOV DWORD PTR SS:[EBP-C],ESP
00402C55  50               PUSH EAX
00402C56  FF51 04          CALL DWORD PTR DS:[ECX+4]
00402C59  6A 08            PUSH 8
00402C5B  33F6             XOR ESI,ESI
00402C5D  5B               POP EBX
```

Part10Tut.ReverseMe

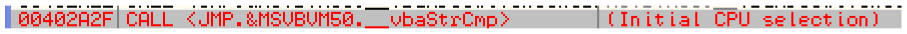
Use SmartCheck to reg!

Kill begin/start nags !

Regcode :

Figure 3

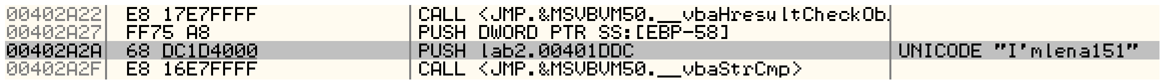
Screenshot of " _ _vbaStrCmp" function Address from Intermodular Calls



```
00402A2F  CALL <JMP.&MSVBVM50.__vbaStrCmp>   (Initial CPU selection)
```

Figure 4

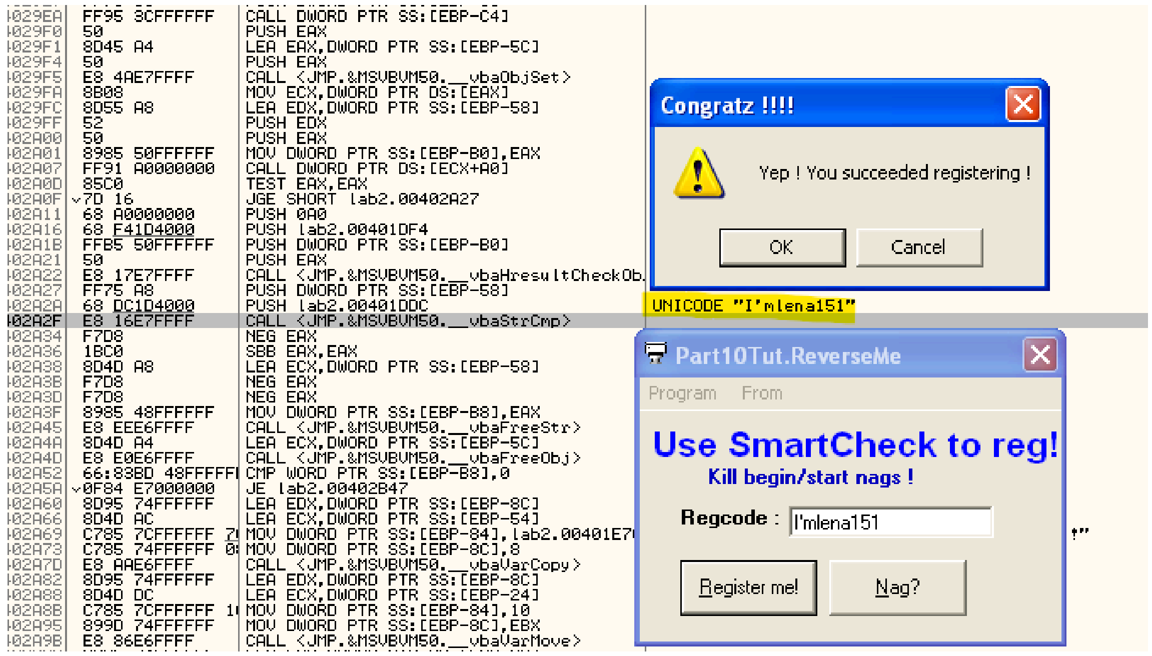
Screenshot of Hardcoded Registration Code



```
00402A22  E8 17E7FFFF   CALL <JMP.&MSVBVM50.__vbaHresultCheckOb
00402A27  FF75 A8       PUSH DWORD PTR SS:[EBP-58]
00402A2A  68 DC1D4000   PUSH lab2.00401DDC                       UNICODE "I'mlena151"
00402A2F  E8 16E7FFFF   CALL <JMP.&MSVBVM50.__vbaStrCmp>
```

Figure 5

Screenshot of Successful Registration Code



```
402029EA  FF95 3CFFFFFF   CALL DWORD PTR SS:[EBP-C4]
402029F0  50              PUSH EAX
402029F1  8D45 A4         LEA EAX,DWORD PTR SS:[EBP-5C]
402029F4  50              PUSH EAX
402029F5  E8 4AE7FFFF     CALL <JMP.&MSVBVM50.__vbaObjSet>
402029FA  8B08            MOV ECX,DWORD PTR DS:[EAX]
402029FC  8D55 A8         LEA EDX,DWORD PTR SS:[EBP-58]
402029FF  52              PUSH EDX
40202A00  50              PUSH EAX
40202A01  8985 50FFFFFF   MOV DWORD PTR SS:[EBP-B0],EAX
40202A07  FF91 A0000000   CALL DWORD PTR DS:[ECX+A0]
40202A0D  85C0            TEST EAX,EAX
40202A0F ˅7D 16           JGE SHORT lab2.00402A27
40202A11  68 A0000000     PUSH 0A0
40202A16  68 F41D4000     PUSH lab2.00401DF4
40202A1B  FFB5 50FFFFFF   PUSH DWORD PTR SS:[EBP-B0]
40202A21  50              PUSH EAX
40202A22  E8 17E7FFFF     CALL <JMP.&MSVBVM50.__vbaHresultCheckOb
40202A27  FF75 A8         PUSH DWORD PTR SS:[EBP-58]
40202A2A  68 DC1D4000     PUSH lab2.00401DDC
40202A2F  E8 16E7FFFF     CALL <JMP.&MSVBVM50.__vbaStrCmp>
40202A34  F7D8            NEG EAX
40202A36  1BC0            SBB EAX,EAX
40202A38  8D4D A8         LEA ECX,DWORD PTR SS:[EBP-58]
40202A3B  F7D8            NEG EAX
40202A3D  F7D8            NEG EAX
40202A3F  8985 48FFFFFF   MOV DWORD PTR SS:[EBP-B8],EAX
40202A45  E8 EEE6FFFF     CALL <JMP.&MSVBVM50.__vbaFreeStr>
40202A4A  8D4D A4         LEA ECX,DWORD PTR SS:[EBP-5C]
40202A4D  E8 E0E6FFFF     CALL <JMP.&MSVBVM50.__vbaFreeObj>
40202A52  66:83BD 48FFFFF CMP WORD PTR SS:[EBP-B8],0
40202A5A ˅0F84 E7000000   JE lab2.00402B47
40202A60  8D95 74FFFFFF   LEA EDX,DWORD PTR SS:[EBP-8C]
40202A66  8D4D AC         LEA ECX,DWORD PTR SS:[EBP-54]
40202A69  C785 7CFFFFFF 7 MOV DWORD PTR SS:[EBP-84],lab2.00401E7
40202A73  C785 74FFFFFF 0 MOV DWORD PTR SS:[EBP-8C],8
40202A7D  E8 AAE6FFFF     CALL <JMP.&MSVBVM50.__vbaVarCopy>
40202A82  8D95 74FFFFFF   LEA EDX,DWORD PTR SS:[EBP-8C]
40202A88  8D4D DC         LEA ECX,DWORD PTR SS:[EBP-24]
40202A8B  C785 7CFFFFFF 1 MOV DWORD PTR SS:[EBP-84],10
40202A95  899D 74FFFFFF   MOV DWORD PTR SS:[EBP-8C],EBX
40202A9B  E8 86E6FFFF     CALL <JMP.&MSVBVM50.__vbaVarMove>
```

UNICODE "I'mlena151"

Congratz !!!!

Yep ! You succeeded registering !

Use SmartCheck to reg!

Kill begin/start nags !

Regcode : I'mlena151

Figure 6

3. Discussion and Conclusion:

Generated by ChatGPT- In conclusion, the objectives outlined in the introduction were effectively met through the hands-on exploration and analysis conducted in this lab. By utilizing reverse engineering tutorials and tools like OllyDbg, students gained practical experience in code analysis and binary manipulation, contributing to a deeper understanding of software internals. The lab successfully facilitated comprehension of complex concepts such as the Stack and Stack Frame, as well as the role of function calls, parameters, and return values in software functionality. Throughout the analysis and experimentation, several key observations were made. The identification of the EAX register as the storage location for the return value of the rtcMsgBox() function aligns with established conventions in calling conventions, highlighting the significance of register usage in function return operations. Similarly, the modification of assembly instructions at memory addresses 0x402CFE and 0x402C17 effectively demonstrated the impact of altering program execution flow, resulting in the removal of the Nag screen. These modifications underscored the importance of understanding assembly language and its implications for program behavior. Furthermore, the process of identifying the hardcoded registration code near the vbaStrCmp function call showcased the practical application of code analysis techniques in real-world scenarios. By locating and validating the registration code, students honed their skills in identifying critical program components and understanding their significance within the context of program functionality. Overall, this lab served as a valuable learning experience, bridging theoretical knowledge with hands-on application.

By exploring the intricacies of executables and mastering techniques for code analysis and modification, students gained essential skills for problem-solving in the field of cybersecurity. Moving forward, continued practice and exploration in reverse engineering will be crucial for further enhancing proficiency in software analysis and security assessment.