RIT | Rochester Institute of Technology of Dubai

Department of Electrical Engineering and Computing

Computing Security

CSEC 202 Reverse Engineering Fundamentals

Spring 2024

Sections: 600, 601, and 602

Homework Assignment #2:

Advanced Static Analysis

Release Date: February 25, 2024

Due Date: March 11, 2024 – 11:59:59 p.m. (GST= GMT+4)

Instructor: Emad AbuKhousa (eakcad@rit.edu)

Homework Assignment #2:

Important notes

Please pay close attention to the following essential guidelines and expectations:

- **Submission Deadline:** There will be **no extensions** under any circumstances. This assignment must be submitted before the midterm to allow enough time for review. Ensure you manage your time effectively to meet this deadline.
- **Instructor Support Window: The** instructor will not provide support 3 days before the assignment's deadline. It's crucial to start early to maximize the opportunity for receiving guidance and support. Procrastination may limit your ability to seek help.
- **Support Availability**: Support for this assignment will be available only during office hours. Make sure to bring up any questions or concerns during these designated times to receive assistance.
- **Communication Channels:** WhatsApp is not considered an official channel for office hours or assignment support. Please use the designated communication methods provided by your instructor for all correspondence related to this assignment.

Academic Integrity:

- Zero Tolerance for Plagiarism: Plagiarism and paraphrasing of any kind will be strictly penalized. All submissions will be thoroughly checked for originality. **We have robust measures in place to detect malpractice, so do not underestimate the system's ability to identify cheating.**

- Consequences of Academic Misconduct: Being caught in acts of academic dishonesty, such as plagiarism, will result in a score of zero for this assignment. Further disciplinary actions may follow according to the academic conduct policies. It is imperative that you adhere to the highest standards of academic integrity in your work.

These guidelines are put in place to ensure a fair and conducive learning environment for all students. Adhering to these rules is not only about avoiding penalties but also about cultivating professionalism, responsibility, and integrity in your academic pursuits.

Remember:

Start working on your assignment early to navigate through potential challenges with ample time for adjustments.

Objectives:

This assignment is designed to enhance your hands-on experience with advanced static analysis, a crucial phase in the comprehensive process of reverse engineering. Through this in-depth analysis, you will gain valuable insights into the structure of binary code, including its functions and the detailed behaviors that are not immediately apparent during execution.

Please note: The binary provided for this assignment is intended solely for educational purposes. You are strongly advised against executing it directly on your primary system. Instead, conduct your analysis within a controlled environment, such as a sandbox, to ensure safety and integrity. You bear full responsibility for any consequences resulting from a failure to take the appropriate precautions during your analysis.

Rubric:

Homework Assignment #2 Rubric	
Criteria	**Pts**
Two Detailed Reports (one for individual tasks and one for group tasks): **The group report should be submitted by only one member of the team.** a. Craft a well-organized, step-by-step report detailing your entire analysis process. b. Include clear explanations for each step, accompanied by relevant screenshots. c. Discuss the result and purpose of each tool or command you employed, demonstrating your understanding. d. Format your report professionally in a standard Word document using consistent styles and headings. e. Include a cover page listing the assignment details, team members, their contributions, and this rubric for reference.	10%
Individual tasks [30%] ▪ Individual Work: This portion stresses the importance of independent work to apply reverse engineering and assembly programming skills. ▪ No Group Work: Emphasizes the prohibition of collaboration. ▪ Originality: Requires submissions to be entirely original, underlining the importance of academic integrity. ▪ Growth: Designed for both assessment and personal development within cybersecurity.	
Question 1: Quick wins with Disassembly Challenges a) Quick win 1 [1%] b) Quick win 2 [1%] c) Quick win 3 [1%] d) Quick win 4 [1%] e) Quick win 5 [1%] f) Quick win 6 [3%] g) **Quick win 7 [2%]**	10%

Question 2 [10%] a) Linux-function analysis [2%] b) Factorial calculation in assembly [3%] c) Machine state analysis and tasks [5%]	
Question 3: Reverse Engineer this code [10%] The task involves reverse-engineering a given assembly code into C code, including several sub-tasks like providing C code, compilation commands, execution instructions, and preparation for demonstration, each essential for the overall 10% weight. ▪ Present the reverse-engineered C code in its entirety, ensuring it maintains the original program's structure and functions. ▪ Use meaningful names for variables, labels, and functions to enhance readability and understandability. ▪ The code should be organized in a clear and logical manner, reflecting the structure and logic of the assembly code as closely as possible. ▪ Embed comments within your C code to explain each line's purpose and its correlation to the assembly instructions, ensuring the code is commented out properly for clarity. ▪ More details of the rubric are listed in the section. ▪ Additional Rubric Details: More details about the rubric requirements are listed in the subsequent sections.	10%
Part 2 Individuals or Groups **This part of the assignment can be done individually or in a team of 2.** **[60%]**	
Question 4: Assembly Language Program The task description includes multiple sub-tasks like prompting for integer inputs, input validation, user choice of operation, result display, division by zero handling, and loop with exit condition, culminating in a session summary. detailed description of the assembly code in the first part and take into account the following: a) The code should be commented out properly. b) Use meaningful names for variables, labels and functions. c) The code should be organized in a proper way so that it can be read easily. **d) The code prompts for input and gives output with clear messages?** **e) Additional Rubric Details: More details about the rubric requirements are listed in the subsequent sections.**	20%

Question 5: Reverse engineering a binary code using IDA disassembler. This task involves using IDA to reverse engineer a provided executable, with an emphasis on writing equivalent C code and annotating the process. The entire task is worth 15%. Detailed description of the C code you will provide for the third part and take into account the following: a) The code should be commented out properly. b) Use meaningful names for variables, labels and functions. c) The code should be organized in a proper way so that it can be read easily. d) The code prompts for input and gives output with clear messages? e) The code must reflect to the extent possible the same structure and logic of the assemble code. f) Provide screen shots of the assemble code you will get by the disassembler with comments and name of various part that you have changed. g) Additional Rubric Details: More details about the rubric requirements are listed in the subsequent sections.	10%
Question 6 **"Us vs. the CAT" round 2** [1] WinMain Address Identification [2%] [2] API GetComputerNameA Usage [2%] [3] Locating the String \cmd.exe [2%] [4] Sleep Function Analysis [4%] [5] Encoded URL Recovery [14%] [6] Command Reading Loop [2%] [7] Response to "list /d" Command [2%] [8] Response to "pidrun" Command [2%]	30%
Question 7 (Optional Bonus) This task involves analyzing the "greencat-2" binary to answer specific investigation questions regarding the potential compromise of an eCommerce start-up's financial records.	15%

Note:

This assignment demands individual contributions from each team member. Collaboration and discussion are encouraged, but it is imperative that each member conducts their own analysis and significantly contributes to the final report, showcasing their personal comprehension. Individual efforts will be distinctly evaluated, reflecting in your final grade. Therefore, the final grades among team members may vary.

Good Luck with the GreenCat

Part 1 [30%]: individual tasks

Question 1: Quick wins with Disassembly Challenges [10 %]:

Reverse engineer the disassembled outputs of simple C code snippets provided below. Your objective is to analyze the disassembled instructions to comprehend the functionalities of the code snippets and then translate them back into their high-level language equivalents (C pseudocode). Utilize your understanding of assembly language to interpret the operations performed by each instruction. The disassembled outputs of simple C code snippets you're tasked with reverse-engineering are designed to be quick wins: straightforward, direct, and firmly rooted in the textbook references and slide materials of this course.

a) **Quick win 1 [1 %]:**

```
1. mov dword ptr [ebp-4], 5  ; Line 1
2. mov dword ptr [ebp-8], 10 ; Line 2
3. mov eax, dword ptr [ebp-4] ; Line 3
4. add eax, dword ptr [ebp-8] ; Line 4
5. mov dword ptr [ebp-12], eax ; Line 5
```

b) **Quick win 2 [1 %]:**

```
1. mov dword ptr [ebp-4], 5
2. mov dword ptr [ebp-8], 10
3. mov dword ptr [ebp-12], 15
4. mov eax, [ebp-4]
5. add eax, [ebp-8]
6. add eax, [ebp-12]
7. mov [ebp-16], eax
8. mov eax, [ebp-4]
9. sub eax, [ebp-8]
10. sub eax, [ebp-12]
11. mov [ebp-20], eax
12. mov eax, [ebp-4]
13. imul eax, [ebp-8]
14. imul eax, [ebp-12]
15. mov [ebp-24], eax
```

c) **Quick Win 3 [1%]**

1. mov dword ptr [ebp-4], 0
2. mov dword ptr [ebp-8], 5
3. mov eax, [ebp-4]
4. cmp eax, [ebp-8]
5. jge loc_40200C
6. add eax, 1
7. mov [ebp-4], eax
8. jmp loc_402014

16. loc_40200C:

9. sub eax, 1
10. mov [ebp-4], eax

17. loc_402014:

11. mov ecx, [ebp-4]
12. xor ecx, [ebp-8]
13. mov [ebp-4], ecx

d) **Quick win 4 [1 %]:**

1. mov dword ptr [ebp-4], 0
2. mov dword ptr [ebp-8], 5
3. mov eax, [ebp-4]
4. cmp eax, [ebp-8]
5. jge loc_40200C

6. add eax, 1
7. mov [ebp-4], eax
8. jmp loc_402014

loc_40200C:

9. sub eax, 1
10. mov [ebp-4], eax

loc_402014:

11. mov ecx, [ebp-4]
12. xor ecx, [ebp-8]
13. mov [ebp-4], ecx

e) **Quick win 5 [1 %]:**

1. mov dword ptr [ebp-8], 3
2. mov dword ptr [ebp-4], 10
1. loc_403020:
2. mov eax, [ebp-8]
3. cmp eax, dword ptr [ebp-4]
4. jge short loc_40303F
5. imul eax, eax, 2
6. mov [ebp-8], eax
7. add dword ptr [ebp-8], 1
8. jmp short loc_403020
9. loc_40303F:

f) **Quick win 6 [3%]:**

```
        push ebp
        mov ebp, esp
        sub esp, 20h
        mov dword ptr [ebp-20h], 4
        mov dword ptr [ebp-1Ch], 5
        mov dword ptr [ebp-18h], 6
        mov dword ptr [ebp-14h], 7
        mov dword ptr [ebp-10h], 0

loc_402000:
        cmp dword ptr [ebp-10h], 4
        jge loc_40202E

        mov eax, [ebp-10h]
        shl eax, 2
        add eax, ebp
        sub eax, 20h
        mov ecx, [eax]
        call DoubleValue
        mov [eax], ecx
        add dword ptr [ebp-10h], 1
        jmp loc_402000


loc_40202E:

        xor eax, eax
        mov esp, ebp
        pop ebp
        ret
```

DoubleValue:

```
push ebp
mov ebp, esp
mov ecx, [ebp+8]
shl ecx, 1
mov esp, ebp
pop ebp

ret
```

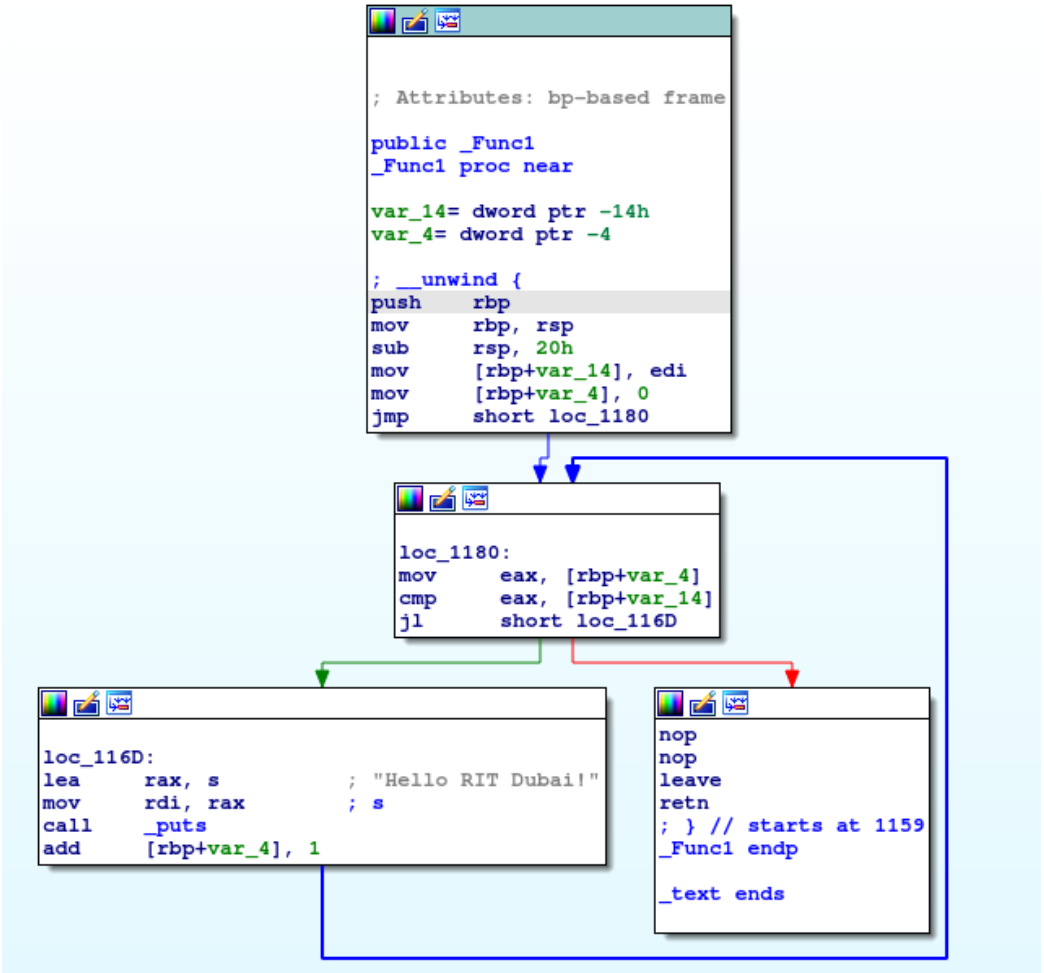
g) **Quick win 7 [2%]:**

```
; Attributes: bp-based frame

; int __fastcall main(int argc, const char **argv, const char **envp)
public main
main proc near

var_4= dword ptr -4

; __unwind {
push    rbp
mov     rbp, rsp
sub     rsp, 10h
mov     [rbp+var_4], 5
mov     eax, [rbp+var_4]
mov     edi, eax
call    _Func1
mov     eax, 0
leave
retn
; } // starts at 1139
main endp
```



Question 2 [10%]:

a) What does this Linux-function do? AT&T assembly language syntax is used here. [2%]

```
main:
        pushq   %rbp
        movq    %rsp, %rbp
        movl    $2, %edi
        call    sleep
        popq    %rbp
        ret
```

b) Consider the following x86-32 assembly code running on an Intel processor, designed to calculate the factorial of a number initially stored in the EAX register. Your task is to execute this code, trace the value changes, and write comments for each instruction explaining its purpose. Additionally, determine the resulting value in the EAX register after the code has executed, assuming the initial value in EAX is 5 [3%]:

```
section .text
global _start

_start:
        mov eax, 5
        mov ebx, 1
        mov ecx, eax

factorial_loop:
        mul ebx
        inc ebx
        dec ecx
        jnz factorial_loop

        mov eax, 1
        int 0x80
```

C) Based on the machine state provided in the following tables and utilizing Intel x86-32 syntax, complete the following tasks and answer the questions below. Assume that the machine state includes a variety of memory addresses and register values. [5%]

Registers	
Name	Value
ESP	0x100
EAX	0x100
EDX	0x2
EBP	0x10C
ECP	0x0

Stack Memory	
Address	Value
0x100	0XABCD
0x104	0xDEAD
0x108	0x1313
0x10C	0x0808
0x110	0xBEEF

Tasks and Questions:

[1] Move Immediate to Register:
Execute mov eax, 42. What is the new value of EAX after this instruction?

[2] Move from Memory to Register:
Execute mov ebx, [0x104]. Given the stack content, what value is now in EBX

[3] Subtract Immediate from Register:
Execute sub eax, 10 after the initial mov. What is the value of EAX after this operation?

[4] Bitwise AND Operation:
Execute and edx, 0xFF. What is the resulting value in EDX?

[5] Compare Two Registers:
Perform cmp eax, edx after the previous operations. Does this comparison set any flags?

[6] Test Bits of a Register:
Perform test eax, eax. What flags are set after this operation?

[7] Stack Operations:
What value is pushed onto the stack with push eax?
After executing pop eax, what value is in EAX considering the initial stack state?

[8] Array Element Access:
Assuming EBX holds an array base address, what is the value at the third element accessed with mov ecx, [ebx+8]?

[9] Complex Memory Operation:
Using base-plus-index addressing, what value is moved to ECX with
mov ecx, [eax + edx*4] after initializing EAX and EDX with the given values?

[10] Accessing Values Relative to EBP:
If you execute mov eax, [ebp-8], what value is loaded into EAX?
Similarly, what value is loaded into EDX with mov edx, [ebp+4]?

Question 3: Reverse Engineer this code [10]

Reverse engineer the following program by providing and equivalent C code. The C code should maintain the same structures. In your report, please include the following components:

[1] The C Code: Present the reverse-engineered C code in its entirety. Ensure that your code maintains the original program's structure and/or functions. Use meaningful names for variables, labels, and functions. Embed comments within your C code to explain each line's purpose and its correlation to the assembly instructions.

[2] Compilation Commands: Detail the commands used to compile your C code. For example, if you are using GCC (GNU Compiler Collection), your command might look like this:

gcc -o myProgram myProgram.c

Replace myProgram with the name of your executable and myProgram.c with the name of your C source file. Explain any flags used during compilation and their purpose.

[3] Execution Instructions: Provide instructions on how to execute your compiled program within a virtual machine (VM). This might simply be:

./myProgram

Ensure these instructions are clear and easy to follow, as they will be used to verify the functionality of your code.

[4] Demonstration Preparation: Be prepared to demonstrate the working code on your VM. This includes having the necessary environment set up and being familiar with the execution process. Your ability to showcase the working code will be essential for receiving full credit.

[5] For submission, in addition to your individual report, please upload the source code file to MyCourses. Place it under the designated assignment section. It is imperative that your submission is both timely and conforms to the specific guidelines outlined for this assignment.

[6] Remember, this is an individual task. The expectation is that you complete this assignment independently to demonstrate your proficiency in reverse engineering and C programming. The compilation and execution process, along with the code itself, should reflect your understanding and ability to apply course concepts practically.

```
section .data
msg1    db "z is zero and x = y.", 0
msg2    db "z is non-zero and x = y.", 0
msg3    db "z zero and x != y.", 0
msg4    db "z non-zero and x != y.", 0
format  db "%s", 0

section .text
global main
extern printf
```

```
main:
        push ebp
        mov ebp, esp
        sub esp, 16

        mov dword [ebp-12], 0 ; var_C = 0
        mov dword [ebp-16], 1 ; var_10 = 1
        mov dword [ebp-20], 2 ; var_14 = 2

        mov edx, [ebp-12]
        cmp edx, [ebp-16]
        jnz Label_LOC1

        cmp dword [ebp-20], 0
        jnz Label_LOC2

        lea eax, [msg1]
        push eax
        push format
        call printf
        add esp, 8
        jmp end
```

```
Label_LOC1:
        cmp dword [ebp-20], 0
        jnz Label_LOC3


        lea eax, [msg3]
        push eax
        push format
        call printf
        add esp, 8
        jmp end
```

```
Label_LOC2:
        lea eax, [msg2]
```

	push eax push format call printf add esp, 8 jmp end
Label_LOC3:	lea eax, [msg4] push eax push format call printf add esp, 8
end:	 mov esp, ebp pop ebp ret

Part 2 [60%]: Individuals or groups of 2 tasks

Question 4: Assembly Language Program [20]

Objective:

This programming assignment is designed to refine your assembly language skills by guiding you through the creation of a program that dynamically interacts with user inputs. You will craft a program that solicits integer inputs from users, validates these inputs, and performs operations based on user-selected criteria. The program's execution will be governed by conditional logic, looping constructs, and function calls, concluding only when a predetermined exit condition is encountered.

Note: use scanf ad printf for input and output.

Task Description: Develop an assembly program that accomplishes the following objectives:

Detailed Task Description:

[1] Prompt for Integer Inputs: The program should continuously request two integer values from the user, ensuring it can perform multiple calculations as needed.

[2] Input Validation: Validate the provided inputs to confirm they are integers. If the inputs are invalid, prompt the user again for correct inputs.

[3] User Choice of Operation: After validating the inputs, ask the user to choose an arithmetic operation (e.g., 1 for addition, 2 for subtraction, 3 for multiplication, 4 for division), facilitating user-driven decision-making.

[4] Immediate Result Display and Re-prompting:

a. Conditional Statements and Function Calls: Employ conditional statements to determine the operation selected by the user. Call the appropriate function to perform the arithmetic operation, passing the user-provided integers as arguments.
b. Design a Printing Function: Create a function specifically for outputting the results of the operation in a formatted and readable manner. After displaying the results, the program should prompt the user to enter new integers and select a new operation, thereby maintaining a continuous interaction loop.

[5] Division by Zero Handling: For division operations, verify that the second input is non-zero before proceeding. If zero is detected, display an error message indicating that division by zero is not permitted, and prompt the user for a new set of inputs. This step is critical for robust error handling and maintaining a smooth user experience.

[6] Loop with Exit Condition: Implement a loop that allows the program to continuously interact with the user until a sentinel value (-666) is entered, signaling the user's desire to end the session. This mechanism ensures the program can handle repetitive user interactions efficiently.
[7] Session Summary upon Exit: When the user decides to close the program, display a summary showing the total number of operations performed during the session. This summary provides valuable feedback and a sense of completion to the user.
[8] Array Usage (Optional): Consider using arrays to enhance the program's data management capabilities, such as storing the results of operations or tracking the count of operations performed. Arrays introduce more complex data structures into the program, enriching the learning experience.

Demonstration and Submission Instructions:

[1] Compilation and Linking: Provide detailed instructions in your README on how to compile, assemble, or link your program, specifying any required tools or dependencies.
[2] Classroom Demonstration: Be prepared to compile/assemble/link and run your program in class. This demonstration will form part of your evaluation and is essential for receiving full marks. You should be ready to explain your code, the choices you made, and how it meets the assignment's objectives.
[3] Submission: Submit the .asm (or .s) source code file and the README. The README should include your assembly and execution instructions, a brief description of your program, and any additional features or functionalities.

Evaluation Criteria:

[1] Implementation [4%]: Correct execution of input handling, arithmetic operations, and program termination based on the sentinel value.
[2] Use of Assembly Constructs [4%]: Effective application of loops, conditions, and function calls to manage program flow.
[3] Error Handling [4%]: Implementation of input validation and error prevention strategies.
[4] Code Organization and Documentation [4%]: Well-organized code with extensive comments explaining the functionality.
[5] Classroom Demonstration [4%]

Question 5: Reverse engineering a binary code using IDA disassembler [10 points]

Use IDA to Reverse engineering the executable **HW2** posted on the Course's GitHub account and MyCourses. Write the equivalent C code. Give Screen shots of the IDA interface that shows various parts of the program with your comments and modified names.

https://github.com/RITDubaiCSEC202/Homework2/blob/main/HW2

Question 6 [30]:

"Us vs. the CAT" Round 2

Congratulations on your remarkable achievement! You have successfully navigated and conquered phase one of the GreenCAT challenge in the "**Us vs. The Car: Basic Static Assignment**." This feat not only showcases your adeptness in static analysis but also marks the beginning of an exciting journey. Congratulations! You have just been named the newest reverse engineering intern the prestigious Dubai Silicon Oasis National Laboratory (DSONL). Your demonstrated skills and dedication have earned you this opportunity to further hone your abilities and contribute to cutting-edge projects alongside some of the brightest minds in the field. Welcome to the team, and we look forward to witnessing your continued growth and success in the realm of reverse engineering.

You arrive at work at 7:00 am on your first day, finding your newly vacuumed cubicle, eagerly anticipating the exciting malware reverse engineering tasks ahead. "What will they assign me on my first day?" you wonder. Could it be Stuxnet? Perhaps some malware targeting nuclear weapons!

Your boss arrives around 11:00 am. He notices you and mentions, "*We just got that fancy new version of the IDA tool, but it doesn't show us the control flow the way we prefer*."

"*Click on Graph View!*" you suggest, but it's evident he has never used IDA before.

"*Use it to pinpoint where this GreenCat is connecting*," he instructs.

"*I've already discovered the URL using VirusTotal*!" you assert.

"Additionally," he continues, *"search for the hidden URL and determine how it's concealed, given that you didn't find it during your strings analysis. Your instructor at RIT Dubai criticized your reliance on dynamic analysis tools. We need you to reverse engineer and uncover the URL."*

"*This will be a long summer*" you think as you begin looking up the IDA SDK docs.

Tasks:

[1] What is the address of WinMain? [2%]

[2] How many times the API GetComputerNameA is used and where in which functions and what's the call address? [2%]

[3] Use the Strings window to locate the string \cmd.exe in the disassembly. Where is it located? [2%]

[4] At text:00402A38 , there is a call to Sleep (an API function that takes one parameter containing the number of milliseconds to sleep). Looking backward through the code, how long will the program sleep if this code executes? [4%]

[5] Have you recovered the encoded URL of greencat-2's command and control (C&C) server? [14%]

- Hint1: What is pushed onto the stack before making this call? Investigate the details of this string, including its position and size. Examine how it is applied within the main function. Is there a loop involved? Analyze the activities being conducted
- Hint 2: What is the purpose of the memset() function?
- Hint 3: Consult the instruction manual for shifting operations.

[6] Have you found the loop where greencat-2 reads commands from the C&C server? [2%]

[7] Do you know what greencat-2 does if the C&C server sends it a "list /d" command? [2%]

[8] Do you know what greencat-2 does if the C&C server sends it a "pidrun" command? [2%]

Question 7 (Optional Bonus) [15%]:

The following is based on true events from 2014, which occurred in a small start-up company in Monroe, CT.

A new small to medium sized eCommerce start-up based in Monroe, CT has recently begun to notice anomalies in their financial records. They have also recently received a number of customer complaints saying that invoices received from the company often bill more services than the customers asked for, and they are often directed to send check payments to a PO Box that does not look legitimate.

Upon reviewing their most recent set of invoices (saved as plain text files), the accounting department notices that the information in the invoices is incorrect, as reported by the customer complaints. The CEO orders an immediate investigation of the three accountants' computers, which had only recently been purchased last month from a bankrupt internet cafe popular for LAN parties.

As a growing company, they only employ a small team of six part-time IT support professionals. hey undertake an initial check of the systems, and they find a suspicious process running on one of the computers which seems to be connected to a McAfee service with a large amount of data being sent outside the company firewall. At this point, they do not feel that they have the expertise to carry out a full-scale malware/forensic investigation, and becoming nervous of a possible attack, the IT team scribbles down the name of the suspicious process ("greencat-2") and quickly shuts down all three accounting department computers and unplugs their power cords.

As there is increased competition in the start-up eCommerce domain, the company is anxious to ensure that their systems have not been compromised. The CEO quickly employs a cyber forensic investigator (YOU) to determine what malicious activity has taken place, where it originated from, and if the company is to blame (i.e., could be sued) for any damages.

Unfortunately, because the accounting department computers contain sensitive customer financial information, the CEO will only allow you access to the suspicious greencat-2 binary. By investigating only this binary, can you determine:

[1] Could greencat-2 or disgruntled accounting employees be responsible for manipulating the company's financial records and customer invoices?

[2] Was greencat-2 targeting the startup eCommerce company all along, or could it have come from somewhere else?

[3] What could explain the "large amount of data being sent outside the company firewall" that the company's IT observed? What functionality in greencat-2 could cause a large amount of data to be sent out over the network?

Part 2 Instructions:

(1) Download webc2-greencat-2.7z
https://github.com/RITDubaiCSEC202/Homework2/blob/main/webc2-greencat-2.7z

(2) Move this zip file up to the VM, unzip it using the following 7z command. The 7z file is password protected. The password is "infected" (no quotes).

```
7z e webc2-greencat-2.7z
```

(3) Load the executable into IDA. This malware is a standard 32-bit PE executable.
(4) Starting from the WinMain(...) function located at ********, follow the control flow in the disassembly. Imagine that you are a recursive-descent disassembler --- starting from the entry of each function and going until the return of that function.
(5) Answer question 6
(6) Optional: Provide answers to the 3 investigation questions of question 7. MAX of 5 sentences per answer. 5 bonus points per correct answer.