Electrical Engineering & Computing Sciences
Computing Security

CSEC 202 Reverse Engineering Fundamentals

Spring 2024| Sections: 600, 601, and 602

Quiz 2

Defusing a Binary Bomb

Part 1: Home-Based Exam (60%)
Part 2: In-Class Exam (40%)

Student Name				**Duration**	**48 hours + 60 minutes**
Student ID		**Section**		**Exam Date**	**May/1 - 2/2024**

Instructions

- **Part 1: Home-Based Exam (60%)** For the home-based portion, **students will have 48 hours** to defuse an electronic bomb simulation. You may use GDB, the internet, and all available resources, including AI-based LLM portals, to aid in solving the problem. This segment is designed to allow you to apply a wide range of tools and knowledge in a practical, open-resource environment. It's essential that each student submits a unique solution; while collaboration in the form of discussion is allowed, directly sharing answers is considered cheating and is strictly prohibited.

- **Part 2: In-Class Exam (40%)** The in-class exam builds directly on the skills and strategies developed during the home-based exam. Students will be given 1 hour to defuse a similar but not identical electronic bomb simulation. This will test your ability to apply the knowledge gained in a controlled, timed setting without the use of external resources.

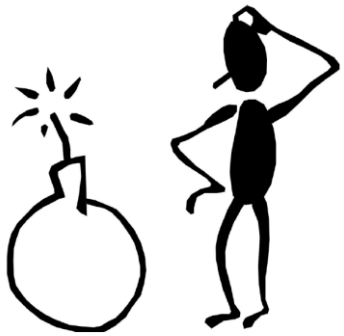
Please ensure that your submissions are entirely your own work. The uniqueness of each solution will be verified, and academic integrity will be strictly enforced to maintain fairness and evaluate each student's genuine capabilities.

All the best!

Instructor: Emad AbuKhousa (eakcad@rit.edu)

Overview:

The Binary Bomb involves a hands-on, practical exercise designed to enhance students' understanding of assembly language, debugging, and reverse engineering. This bomb, originally published by Carnegie Mellon University (CMU) team, challenges students to defuse a "binary bomb" by providing the correct strings to avoid triggering the bomb. It's a test of skill, patience, and understanding of low-level programming.



Objectives:

- Develop proficiency in using debuggers, specifically GDB.
- Gain practical experience in disassembling and reverse-engineering binary files.
- Enhance problem-solving skills in a high-stakes environment with immediate feedback (the bomb "exploding").
- Understand the mechanics behind program execution at the machine level and the consequences of incorrect inputs.

Exam Structure:

Part 1: Take-Home Exam (60%)

Students will download their binary bomb and work to defuse it over a period of 48 hours. This part of the exam allows the use of GDB, the internet, AI-based tools, and any other resources they find pertinent. The primary task is to disassemble the bomb, understand its phases, and input the correct strings to prevent it from exploding. Students are encouraged to document their process, strategies, and the tools they use, as this will form part of their exam submission.

Instructions:

- Download your binary bomb via the provided link.
- Use tools like GDB, ida, and/or objdump to analyze and understand the binary.
- Each phase will require a specific input; deduce these inputs to defuse the bomb.
- Maintain a detailed log of your methods, insights, and the reasoning behind each solved phase.

Part 2: In-Class Exam (40%)

Leveraging the skills and techniques from the home-based exam, students will face a similar but not identical bomb in a controlled, in-class setting. This exam will test their ability to apply their learned skills under time pressure, with only one hour to defuse the bomb.

Instructions:

- Attend the exam session with all necessary preparations.
- Apply the debugging and reverse engineering skills acquired from the home exam.
- Defuse the new binary bomb within the allotted time without prior access to the binary.

Submission and Grading:

- Students are permitted to attempt defusing the bomb multiple times during the homephase without any penalties.
- For the in-class exam, only one defusal attempt per student per phase is allowed. Penalties will be applied for incorrect attempts during this phase.
- **For the in-class exam, each explosion (incorrect answer) will deduct points, so precision is crucial.**
- In-class performance will be assessed based on the ability to apply techniques effectively under exam conditions.

Submission Instructions:

Detailed documentation of the take-home part is required and will be part of the grading.

Please submit two files for your assignment:

1. **psol.txt** - This file should contain the solutions to the task. Follow these guidelines to ensure correct formatting:
    - Place your answer for each phase on a separate line; the first line for phase 1, the second line for phase 2, and so on.
    - Start the file with your answer for phase 1—do not include your name or any other personal information at the top.
    - Avoid adding any comments or numbering to your answers.
    - Ensure each answer is followed by a newline character, which you can usually add by pressing [Enter] or [Return] on your keyboard.

2. **YourSection_YourName_Report.pdf** - This should be a concise report summarizing the steps you took to defuse each stage of the bomb. Describe the techniques and strategies you employed for each phase, highlighting any challenges you encountered and how you overcame them.

Additional Notes:

- The take-home part of the exam encourages open resource usage, but all submissions must be individually completed. Plagiarism will be strictly penalized.
- For the in-class part, no external resources will be permitted.
- Access the Exam Materials: Download your binary bomb here:

https://github.com/RITDubaiCSEC202/DrEvil

Prepare diligently, and remember, each step you take towards understanding the bomb not only helps you defuse it but also significantly bolsters your debugging and reverse engineering skills. Good luck!

Part 1 [60%]: Home-Based Exam (60%)

1. Introduction

The infamous **Dr. Evil** has placed a series of "binary bombs" on our class servers. A binary bomb is a program divided into several phases. Each phase requires you to input a specific string. If the input is correct, the phase is defused, and the program moves to the next phase. However, if the input is incorrect, the bomb "explodes" by displaying "BOOM!!!" and then terminates. The entire bomb is considered defused only when all phases have been successfully completed.

Your mission, which you have no choice but to accept, is to defuse your bomb before the due date. Good luck, and welcome to the bomb squad!

Step 1: Get Your Bomb

To begin, access your personalized binary bomb by navigating to the following URL in your web browser:

https://github.com/RITDubaiCSEC202/DrEvil

To copy the repository from GitHub to your local machine, you'll use the git clone command. Here is how you can clone the repository from the provided URL:

```
git clone https://github.com/RITDubaiCSEC202/DrEvil.git
```

This command will create a new directory named DrEvil in your current working directory, and it will download the contents of the repository into this directory. If you wish to clone the repository into a different directory, you can specify the directory name at the end of the command like this:

```
git clone https://github.com/RITDubaiCSEC202/DrEvil.git MyCustomDirectory
```

This will clone the repository into a directory named MyCustomDirectory.

This directory contains several important files:

- README: Details about the bomb.
- bomb: The executable binary of the bomb.
- bomb.c: The source file featuring the main routine and a greeting from Dr. Evil.
- psol.txt: The solution file

Step 2: Defuse Your Bomb

Your primary task in this lab is to defuse your bomb, which must be done using a Linux VM. There's a persistent rumor that the bomb will only explode if run on non-approved machines due to Dr. Evil's additional security measures.

To assist in defusing the bomb, utilize a variety of tools and refer to the hints provided in the documentation. The most effective approach is to use your preferred debugger to navigate through the bomb's disassembled binary.

The scoring for the exam is as follows:

- The first four phases award **8** points each.
- The more challenging fifth and sixth phases award **14** points each, making the maximum achievable score **60** points.
- **The bomb includes a secret phase. Successfully defusing this phase will award students an additional 20 credits that can be applied during the classroom exam**.

As you progress through each phase, your growing expertise will help mitigate the increasing difficulty. However, do not underestimate the final phase; start early to ensure sufficient preparation.

The bomb will not react to empty input lines. If you prefer not to re-enter solutions repeatedly for phases you've already completed, you can run the bomb with a command-line argument like so:

```
linux> ./bomb psol.txt
```

This command directs the bomb to read inputs from **psol.txt** until EOF is reached, after which it will switch to standard input.

To prevent unintended detonations, you'll need to master the use of debugging tools to step through the assembly code, set breakpoints, and inspect register and memory states. Developing proficiency with these debugging techniques is a valuable skill that will benefit you greatly in your future endeavors.

Exam Hints and Strategy for Defusing the Binary Bomb

Don't brute force:

There are multiple strategies for defusing your bomb. One method involves closely examining the bomb's code without ever executing the program, allowing you to understand its exact workings. This approach is quite insightful but can be challenging. Alternatively, running the bomb under a debugger lets you observe its behavior step-by-step, using this real-time data to effectively defuse it. This latter method is often the quickest way to handle the bomb.

However, we urge you to refrain from using *brute force methods*! Attempting to guess the correct strings by trying every possible key combination is not advisable for several reasons. We have not specified the length of the strings

or their possible characters. Assuming each string is under 80 characters and consists only of letters, you'd be facing **26^{80}** potential combinations per phase. This vast number of guesses would be impractical to process and unlikely to yield the correct answers before the deadline.

Basic Strategy:

The most effective strategy for tackling the binary bomb is to use a debugger like GDB or LLDB. Experience shows that almost no students succeed without one. On the department Unix machines, GDB is readily available.

Analyzing the Bomb:

Start by running **objdump -t bomb** to reveal the symbols in the executable, which includes names of all methods and variables. Pay special attention to method names that sound critical or relevant to the bomb's phases.

To safely navigate through the bomb without setting it off, you must become proficient in single-stepping through the assembly code and setting breakpoints effectively. Additionally, learning to inspect registers and memory states is crucial.

Tools Tools Tools:

There are numerous tools available to help you understand how programs function and diagnose issues when they malfunction. Below is a list of some tools that you might find useful in analyzing your bomb, along with hints on how to effectively use them:

GDB (GNU Debugger)

GDB is a command-line debugger tool available on nearly every platform. It allows you to trace through a program line by line, examine memory and registers, look at both the source code and assembly code (note: we are not providing the source code for most of your bomb), set breakpoints, set memory watch points, and write scripts.

To prevent your bomb from detonating due to incorrect inputs, mastering the setting of breakpoints is crucial. For additional help, type “help” at the GDB command prompt, or consult “man gdb” or “info gdb” in Unix. Many users also prefer running GDB under gdb-mode in Emacs.

Objdump

objdump -t: This command will print the bomb’s symbol table, which includes the names of all functions and global variables in the bomb, the names of all the functions the bomb calls, and their addresses. Reviewing the function names might provide insights.

objdump -d: Use this to disassemble all of the code in the bomb, or just specific functions. Reading the assembler code can be very enlightening about the bomb's functionality. Keep in mind that calls to system-level functions may not be straightforward; for example, a call to sscanf might appear cryptically and require deeper analysis within GDB.

Strings

This utility displays all the printable strings within your bomb, which can be crucial for identifying key data elements and understanding what outputs or messages your program might produce.

IDA (Interactive DisAssembler)

IDA is a powerful disassembler and debugger that is not limited to simple disassembly. It offers a graphical user interface and the ability to interactively explore the binary in a more intuitive way than traditional command-line tools. IDA supports a wide range of programming languages and has advanced features like the ability to identify and analyze code patterns, making it invaluable for reverse engineering complex binaries. This tool is particularly useful for making sense of obfuscated or complicated assembly code that might be part of your bomb.

For each tool mentioned, it's important to become familiar with their more advanced features as you progress through defusing the bomb. Each offers a unique approach to understanding and interacting with your code, and effective use of these tools can greatly enhance your ability to manage the challenges presented by the binary bomb lab.

Bomb Usage Tips:

The bomb will ignore blank input lines. If you want to avoid retyping solutions for already defused phases, you can run the bomb with a command-line argument like this:

```
linux> ./bomb psol.txt
```

This command tells the bomb to read inputs from **psol.txt** until it reaches EOF, after which it will switch to standard input.

Examining the Executable:

Use **objdump -d** to disassemble the bomb's code. You can disassemble the entire program or specific functions to understand their operations. For Intel syntax in the disassembly, use **objdump -M intel -d**.

The **strings** command is also useful as it displays all printable strings within the bomb, which might give clues or direct strings used in the bomb's checks.

Using GDB Effectively:

If working on a department Unix machine, consider loading the latest GDB version for enhanced functionality.

Set a breakpoint at a suspected method, and use commands like step (step through source code) and stepi (step through assembly instructions) to meticulously go through the functions:

```
(gdb) b methodName
(gdb) run
(gdb) stepi
```

```
(gdb) info locals
(gdb) info registers
```

Keep an eye on suspicious functions that might be checking your inputs, and use disas methodName to view the assembly instructions of specific methods.

Additional GDB Commands:

- x/s $someRegister to display strings stored in registers.
- print expr to evaluate expressions.
- call (void) puts (0x...) to output strings.
- disas methodName to view disassembled code.

On Interpreting Disassembly: Familiarize yourself with the x86-64 calling conventions and pay attention to the standard library function names. Avoid disassembling standard library functions unnecessarily, and instead, consult their documentation for functionality details.

HINT:

If you're still having trouble figuring out what your bomb is doing, here are some hints for what to think about at each stage (phase):

1. Comparison
2. Loops
3. Switch statements
4. Recursion
5. Pointers and arrays
6. Linked lists
7. Secret phase (extra credit)

An End-of-Semester Bonus Hint:

You can watch online walkthroughs to defuse all the phases. However, keep in mind that the solutions shown may not match yours exactly, as each student's bomb and the classroom bomb will differ. The good news is that the underlying concepts are consistent across all versions. Once you master the techniques from one, you can adapt them to any others. Enjoy the learning experience.

Computer Systems Bomblab Phase 1 Walkthrough: https://www.youtube.com/watch?v=VsxCGI65AjA

Look for the other phases.

Remember:

Handling a binary bomb requires a blend of careful inspection, strategic debugging, and understanding of assembly code. Your goal is to learn the debugger's capabilities deeply, which will be invaluable for your Cybersecurity career. Good luck on your mission to defuse the bomb!

References:

https://csapp.cs.cmu.edu/3e/labs.html

https://www.cs.cmu.edu/afs/cs/academic/class/15213-f22/www/recitations/rec04_slides.pdf

https://heather.cs.ucdavis.edu/matloff/public_html/UnixAndC/CLanguage/Debug.html (Guide to Faster, Less Frustrating Debugging)