



# Linux Academy

## Kali Linux Deep Dive

### *Study Guide*

Reverse Engineering

Ermin Kreponic

[ermin@linuxacademy.com](mailto:ermin@linuxacademy.com)

January 15, 2019

# Contents

|  |          |
|--|----------|
| <b>Reverse Engineering</b>                                   | <b>1</b> |
| Objectives . . . . .   | 1        |
| Introduction to Reverse Engineering . . . . .                | 1        |
| Steps For Reverse Engineering a Program . . . . .            | 1        |
| Compiling, Running, and Configuring a Keylogger . . . . .    | 2        |
| Keep in mind . . . . .                                       | 2        |
| Decompiling a Keylogger . . . . .                            | 3        |
| Overview of the Process of Decompiling a Keylogger . . . . . | 3        |
| Buffer Overflow . . . . .                                    | 5        |
| How Buffer Overflow Works . . . . .                          | 5        |
| How Buffer Overflow is Exploited . . . . .                   | 5        |
| Famous Buffer Overflow Attacks . . . . .                     | 6        |
| Common Terms . . . . .                                       | 6        |

# Reverse Engineering

## Objectives

The Reverse Engineering module contains several chapters dedicated to explaining the basics of reverse engineering, both theoretical and practical. It starts with getting you familiar with what reverse engineering really is, why it is needed, what kind of tools are necessary to perform it, and the steps one has to go through in order to reverse engineer a program.

## Introduction to Reverse Engineering

Reverse engineering is the process of taking a program apart and discovering how it works in order to replicate it or identify and exploit its vulnerabilities. It involves recreating the program's source code from its binary code. Reverse engineering serves several different purposes. It is used for fixing bugs, improving performance, recreating the original source code if it is lost or inaccessible, understanding how a specific portion of a code works, and analyzing malware.

- Hex Dumper
  - Software that allows you to view the content of a file in hexadecimal format rather than in binary number system.
- Disassembler
  - Software that converts machine language into assembly language.
- Debugger
  - Software that searches for and corrects any errors (bugs) in other programs.

## Steps For Reverse Engineering a Program

- Define the objective
  - First, we figure out why we are reverse engineering something. Let's say we have a web server and we don't know the source code of the server. We are trying to reverse engineer a portion of it where there is, for example, a socket through which network communication flows. We would like to understand how it works. Once we get that information, we can use it to cause a certain behavior we want. For example, a buffer overflow. So, defining our objective is important so we know what to look for and do.
- Observe the process
  - We observe what is happening and try to get an overview of the situation.
- Disassemble the program
  - Run the program through a disassembler.
- Analyze the disassembled parts
  - From the disassembled parts, we extract all of the information we need to perform our objective.

## Compiling, Running, and Configuring a Keylogger

In this part of the module we go through the process of compiling, running, and configuring a keylogger that we treat as surveillance oriented malware. We go through this whole process so that later we can learn the reverse procedure. Our goal is to edit the keylogger signature, set up the keylogger, learn how to read the output we are given by the decrypt file, as well as go through the processes of finding a keylogger on our own machine, isolating it, and finding information on the attacker.

### Keep in mind

It is important to alter the keylogger signature because it decreases the likelihood of antivirus software matching a known type of malware. The more times you change it the better. If you change it enough, the hashes shouldn't match. If you only make a few small changes, it may be matched due to multiple created variations of signatures that already exist.

Decrypt functions need to be the exact reversal of an encrypt function. Whatever was done in an encrypt function must be undone in a decrypt function.

After you have a working decrypt file on your computer, the decryption of it is easily performed in a command-line shell:

- For Windows Powershell, it would be something like this:

```
cd .\Desktop\decrypt\bin\Debug\  
.\decrypt.exe .\log
```

What to look for when trying to detect a suspicious task in your task manager:

- The malicious file will probably be named like another file that is stored on your computer, disguised as a Runtime Broker process or some other program. It is good to check the running processes, find some you are not sure about, and try to recollect why that process would be running on your computer.
- You can check the details and see the Username, PID, Status, Memory and Description of the process to see if anything is off.
- You can check the properties to see some general information like location of the file and when it was created. In the details there will be a more detailed description telling us the type, product name, product version, copyright, size, and date modified. You can compare the data of two processes that have the same name and see if one looks more reliable than the other.
- Another thing we can check in the properties is if there are any digital signatures.
- You can open up the file location and see if the location is suspicious. For example, why would something called Notepad run from the **System32** folder?
- Compare the file hashes of two processes. Sometimes the difference can be just because of the version, but sometimes it might signify that something is wrong.

## Decompiling a Keylogger

### Overview of the Process of Decompiling a Keylogger

In order to perform reverse engineering, several tools are needed:

- IDA is the best tool for reverse engineering
  - It uses the Hex-Rays decompiler
  - The free version is restricted but great for beginners

There is another decompiler called Snowman we can use. It is always good to have more than one decompiler, so you can compare it with another decompiler's result in order to have a better idea of what is going on. What is usually done is that parts of the program are selected, and as such are run in Snowman to test out individual segments.

We need IDA 32-bit because the keylogger is 32-bit. The keylogger is 32-bit because we want it to work on as many systems as possible.

Once we insert the file into IDA, we will see the decompiled program written in assembly language. Some assembly instructions for the x86 intel instruction we might see are: **lea**, **and**, **or**, **push**, **mov**, **call**, and **sub**.

**Recommendation:** Look up the x86 Intel instruction set on the Intel official page, since they have very detailed documentation of every instruction and what it does!

Apart from viewing it in assembly language (which is correct, legitimate code), we can view the code in a more readable, yet less trustworthy auto-generated pseudo code format. This view is where we will try to figure out the code and make more sense of it.

The code will seem confusing at first. There will be a lot of compiler added code because of the way the compiler processes the code. Functions are also tricky, since the first value that seems to be the first parameter of a function is actually the return value.

When decrypting, the code is read from bottom to top, since it is the opposite process of encrypting!

If we recognize some general purpose algorithms like **Base64**, or any other algorithms that are already known, you don't have to reverse engineer them, but rather just write them out at any point it is needed.

Most of the process now is trying to figure out the code. We start with finding variable declarations and rename the variables to names that are more logical and clear. From there you then move on to the functionalities, which are harder to detect.

While figuring out the programs pseudocode, it is important to write your own pseudocode and try to form the structure of the function.

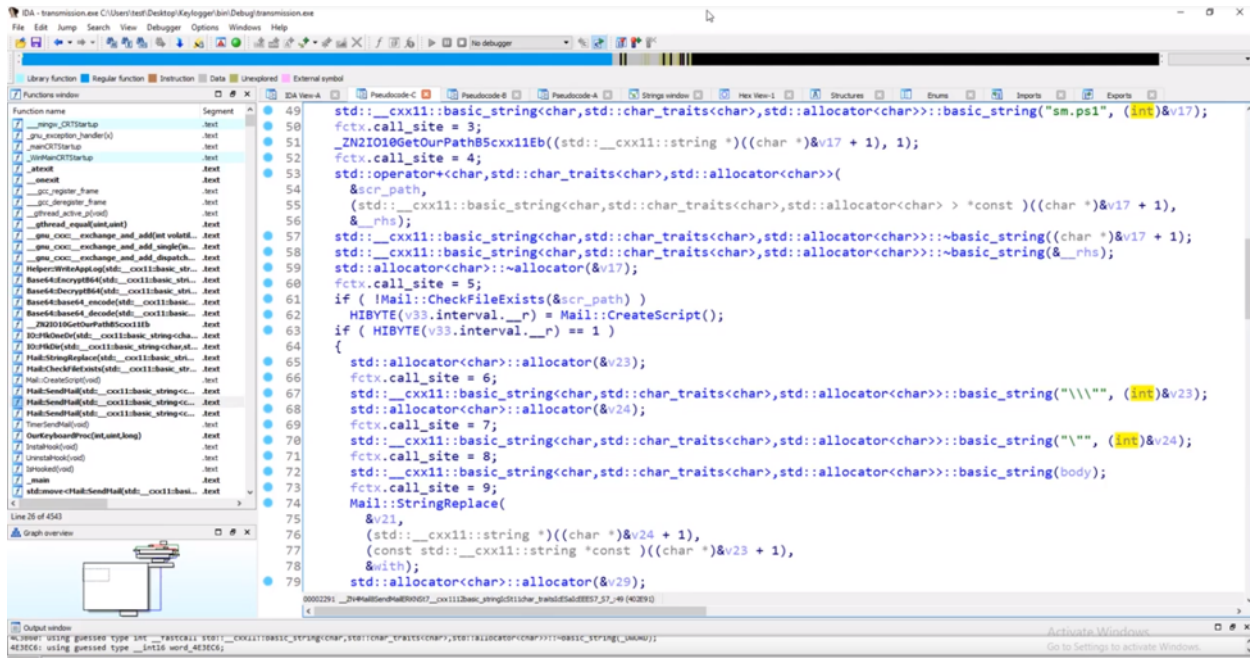
Reference points in the code can help you with orientation and chunking up the program into smaller pieces.

Note that your reversed code will probably never be exactly the same as the pre-compiled (original) code, but it will be close enough to perform all needed functionalities.

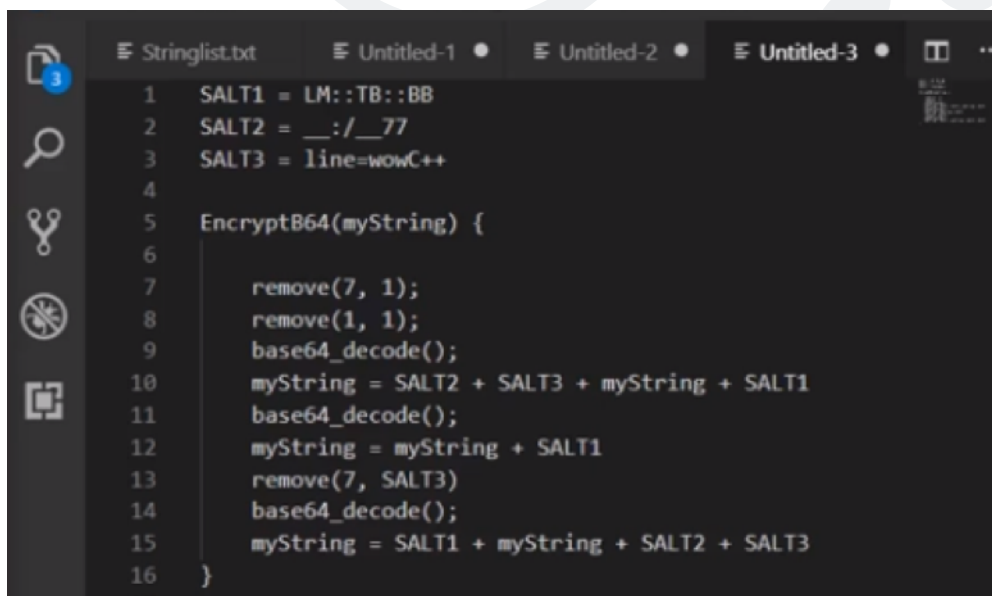
After completing the pseudocode, the next step is writing the decryption function in a high level language and checking if it works properly.

Lastly, the files can be decrypted through a command-line Powershell.

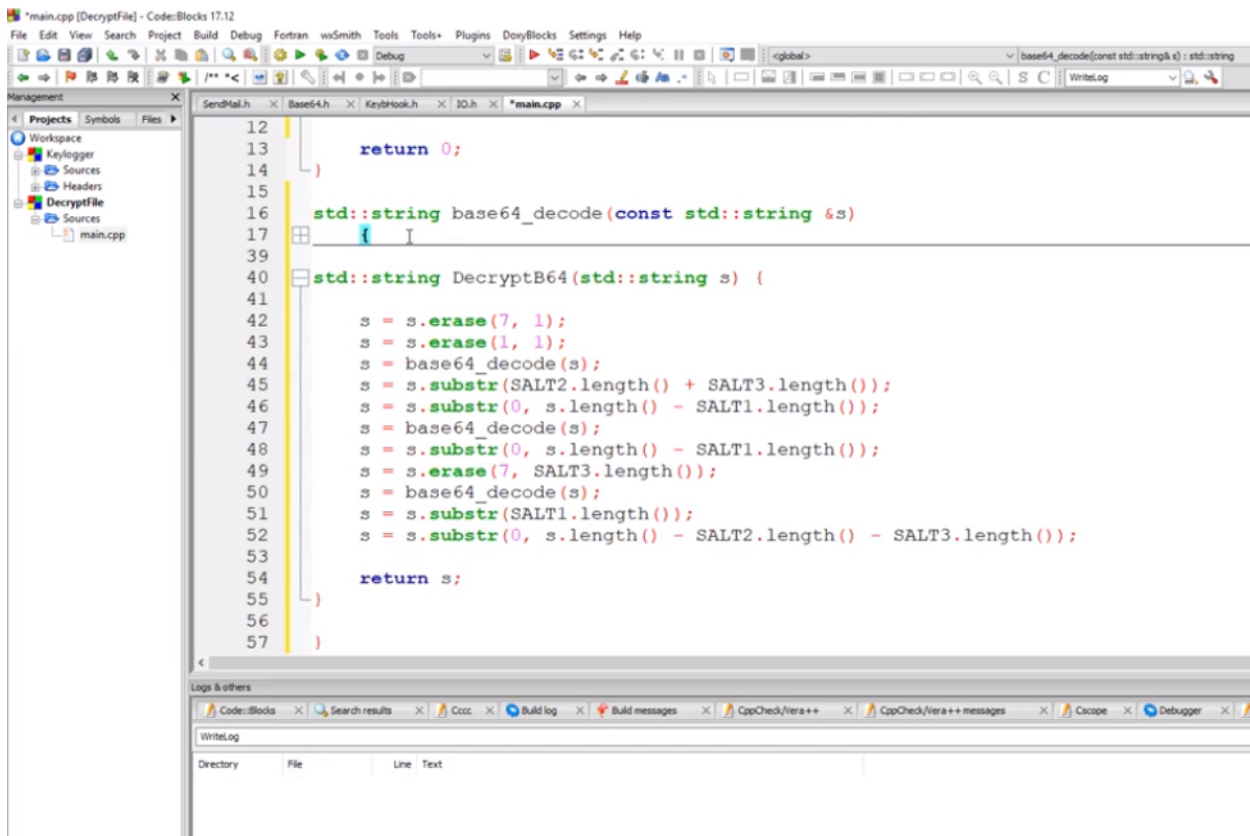
We go from computer generated code:



To creating our own pseudocode:



Then, finally an implemented version in Code Blocks:



```
*main.cpp [DecryptFile] - Code::Blocks 17.12
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DovyBlocks Settings Help
Management
Workspace
  Keylogger
  Sources
  Headers
  DecryptFile
  Sources
  main.cpp
12
13     return 0;
14 }
15
16 std::string base64_decode(const std::string &s)
17 {
18     // ...
19 }
20
21 std::string DecryptB64(std::string s) {
22     s = s.erase(7, 1);
23     s = s.erase(1, 1);
24     s = base64_decode(s);
25     s = s.substr(SALT2.length() + SALT3.length());
26     s = s.substr(0, s.length() - SALT1.length());
27     s = base64_decode(s);
28     s = s.substr(0, s.length() - SALT1.length());
29     s = s.erase(7, SALT3.length());
30     s = base64_decode(s);
31     s = s.substr(SALT1.length());
32     s = s.substr(0, s.length() - SALT2.length() - SALT3.length());
33
34     return s;
35 }
36
37 }
```

## Buffer Overflow

Buffer overflow occurs when more data than the allocated buffer can store is written to the buffer, causing the extra bytes to overflow. Attackers can exploit this vulnerability to write and execute malicious instructions to a system's memory.

### How Buffer Overflow Works

A program that requires user input has to reserve a certain number of bytes in memory in order to process that data. If the input is not validated and exceeds the allocated buffer size, the program will simply continue to write the extra bytes to memory, even if it means overwriting data stored in adjacent memory locations.

### How Buffer Overflow is Exploited

Attackers can exploit a buffer overflow vulnerability to inject malicious code into a program and its memory. If a program is vulnerable to buffer overflow, attackers can attempt to inject malicious code, jump to previously inserted malicious code, or write malicious code (shellcode).

- Injecting malicious code
  - Attackers can exploit a buffer overflow vulnerability by injecting binary code into the memory space reserved for the buffer. The malicious instructions are read and executed, compromising the program and system.
- Jumping to previously inserted malicious code
  - If there is already a piece of malicious code in memory, the attacker can inject a jump instruction to its address, executing the malicious code and overflowing the buffer.
- Writing malicious code



- Shellcode is a type of malicious code written to memory. Shellcode typically launches a command shell from which the attacker can run any command they want.

### Famous Buffer Overflow Attacks

- SQL Slammer Worm (2003)
  - A computer worm that caused a DoS attack on multiple internet hosts, slowing down the internet traffic by a considerable amount. A huge amount of UDP packets were arriving at UDP ports at a fast rate. The worm exploited a buffer overflow weakness to achieve this.
- Conficker (2008)
  - A computer worm that targets the Windows operating system. It uses a combination of advanced malware techniques, making it harder to stop. Most versions of this worm rely on the exploitation of a vulnerable computer that wasn't patched for the MS 08-067 vulnerability. They took a known vulnerability and then they exploited it. It would send a specifically crafted remote procedure call request to force a buffer overflow and execute shell code on the victim machine. Once that shell code was run, it would join a larger botnet and be ready to receive its payload from a daily list of 250 domain names. These payloads would update the worm from a new variant and have the ability to install additional malware, like keyloggers, rootkits, and more.

### Common Terms

- Reverse engineering
  - The process of taking a program apart and discovering how it works in order to replicate it or identify and exploit its vulnerabilities.
- IDE
  - An Integrated Development Environment (IDE) is an application used to write and compile code.
- Keylogger
  - Keystroke logging (keylogging) is a software surveillance method that records every keystroke made on the target system.
- Payload
  - A segment of a computer virus that is in charge of executing the malicious activities of that virus, or generally any malicious code that is deployed on the target device.
- Hash
  - A number that has been derived from a text string and once the string has been encoded, it is much harder to decrypt it than a regular encryption.
- Signature



- Also called a DAT file or Definition file. In computing, a signature is a hash or algorithm that is used to identify a specific virus or malware in a unique way.
- Encryption
  - The process of altering sensitive information in a way that makes it unreadable to everyone except the intended recipient.
- Decryption
  - The process of decoding the encrypted data and returning it into its legible form. The decryption process is identical to the encryption process performed on the data, but executed in a reversed order.
- Buffer
  - A buffer is an area of memory where data is temporarily stored. When programs require user input, that input is stored temporarily in memory before it is processed. To store the input, a portion of memory of fixed size—the buffer for that input—is allocated to the program.
- Worm
  - A worm is a type of malware that replicates itself to programs and documents on a victim's machine by exploiting vulnerabilities on the machine and then spreading to other computers as the infected files are transferred.
- Cheat Engine
  - An open-source memory scanner, hex editor, and debugger. It can be used in many ways, among which one of them are browsing through the RAM space of a program.