Problem Statement:
Reverse Engineering in Two Parts

Part one: parser; Part two: labeler

Sharlena Luyen *IDA Free Reverse Engineering Team*

CS 462 Senior Design Fall 2018

Corvallis, OR USA

luyens@oregonstate.edu

Abstract

By searching and organizing binary files via reverse engineering the strings within the files, the team will create a purely organic software model that will efficiently detect malware or corrupted files. A python based parser will be developed to detect either unique or repeated strings within files or directories. Subsequently, a script will then organize and map the function names to corresponding strings, uniquely pairing the two. This report dictates the necessary steps in order to increase the efficiency of malware compilation.

Index Terms

Security, binary program analysis, malware, python parser, iDC script labeler

I. Introduction

This document is a description of our team's senior project for Oregon State University's CS 462 course during Fall 2018; this document details our problem statement, our proposed solution, and metrics to measure success. The goal is to use this document as a standardization for expectations in the future. While the client will continue to guide our project based on the client's needs, this document's purpose is to provide a defined and clear problem statement that will ease communication between the team and the client.

II. Existing Problem Overview

At its essence, binary files are most commonly used as instructions that can only be interpreted by a machine. Malware is often created to invade and steal sensitive data. Binary malware files, however, are machine-read-only files with the intent to attack a machine from the inside. While machines are able to read binary files easily, it is very difficult for humans to understand binary files since there is no set ”interpreter.” Instead, currently existing programs like IDA Pro are meant to finesse through a binary file to rename functions and decipher strings within the file. Doing so will allow readability and proper analyzation of how exactly the binary file functions.

The purpose of our senior project is to make this process easier and more efficient. If we could create and develop a program or algorithm to decipher strings and match them to their corresponding function, then we will be able to decode binary malware files much quicker. By doing so, the hope is that malware can then be more easily understood and then anti-malware solutions can come about quicker as well. Our goal is to find a solution of the problem from the foundation of the issue.

A. Malware and Binary Handling

Often times, malware binaries are compiled inefficiently. The functionality of the binary file is questionable due to the miscommunicated strings and functions. Figuring out the exact intent and/or function of the malware can be challenging when deciphering the binary file.

B. Strings within Files

Because strings within binary files are often mislabeled or miscommunicated when malware is compiled, mapping the correct string to the proper function will allow insight to what the binary is actually trying to achieve, which engineers can then interpret and properly fight against.

C. IDA Free Plugin

Overall, the python parser and the iDC labeler will be used as a plugin for the reverse engineering software IDA Free. Running the plugin on multiple source files in various batches will allow us to see consistency within the developed plugin as well as the interpretation of the binary files.

III. Solution Overview

Develop a python parser to search binary files as well as an iDC script to organize and map aforementioned strings to corresponding functions.

A. Develop a python parser

Taking in a few files or directories as inputs, the python parser will read the file and determine strings.

B. Develop an iDC script

After determining the strings within the files, the iDC script will match the strings with the corresponding function names.

C. Reverse engineer the binary with outputs

Presumably matching the correct string to the proper function, the program will then be able to tangibly and visibly see the functionality of the file.

D. Analyze insight to functionality

The success of the entire project depends on the analyzation of the full functionality of the entire binary file.

IV. Metrics of Success

A. Time complexity for parsing and mapping

When the python parser splits the binary files based on strings within the files, we can measure the time complexity of the parser. The goal is to maximize the efficiency of the parser. In this case, the amount of time to parse a few binary files should not drastically differ from that of parsing hundreds of binary files. This distinction will allow us to measure the success rate of the python based parser.

Worst case scenario, the performance of our program should be able to parse the files in 15-30 seconds for 1 mb binary. Then, the script should be able to match the file names to the functions in less than 5 seconds.

B. Successful matching of corresponding string and function

In order to measure success of both outputs from the python parser and the iDC script, the corresponding string will have to match the function. While this is an obvious metric, it is important to realize that this is a measure of success.

C. Successful analyzation of functionality

Once the entirety of the program executes, we should be able to completely analyze the function of the binary file as a whole. Successfully analyzing the binary file in an efficient manner is the overarching goal that we hope to achieve.