Problem Statement

Matthew Baker
Fall 2018 CS461: Senior Design
Oregon State University Corvallis, OR

Abstract

Over the last 10 years reverse engineering has moved from the background to the foreground of cyber security. While products like the Interactive Disassembler (IDA Free/Pro) and Binary Ninja have continued to offer more accurate disassembly and decompilation, there is still a significant need for improvement. This document will describe in detail the current problem reverse engineers are facing, a potential solution, and several measurable metrics concerning the end implementation. The problem is straightforward; it is simply too time consuming to search for common library functions within disassembled binaries, an engineer's time is better spent elsewhere. The solution consists of a program which pairs IDA's scripting language and a sophisticated python C parser, to create a more efficient way of relabeling obfuscated functions within IDA. The measurable metrics consist of performance, usability, and cost. The interface must run quickly on decent hardware, be simple and extensible for anyone to use, and it must be free and open source.

I. Problem Definition

Everyday, thousands of malicious programs are compiled into executable binaries. They are sent out across the internet for a wide variety of purposes: adware, bots, ransomware, spyware, rootkits, trojan horse, worms, and viruses. Just recently, in May of 2017, the WannaCry ransomware attack infected more than 300,000 machines and was only briefly halted by the fast work of one reverse engineer. Eventually, these binaries make their way onto the desk of a malware analyst at a company like Symantec or McAfee or to a reverse engineering enthusiast, who's task is ultimately to determine the original functionality, end goal, and risk of the program.

Renaming functions appropriately within a program like IDA Pro or Binary Ninja is often one of the first tasks a reverse engineer will tackle, despite it being tedious and slow. This is because properly named methods have the potential to provide a foundation upon which most other analysis can stand. Libraries like OpenSSL, Boost, ZMQ, and Qt, in the case of C++, are often used in conjunction with the core functionality of the malware. Properly understanding how the malware utilizes a library like OpenSSL can teach the researcher a great deal about how other computers are infected, how data is stolen, or even how control commands are received.

The process of renaming library functions used within the binary is made worse by the fact that most malware developers strip, obfuscate, pack or otherwise hide the true names of their methods. A function named ”SSL use certificate” in the original C source file may end up named as ”subroutine01203” in the IDA Pro disassembler, which offers no indication, at first glance, as to what the method originally did.

II. Proposed Solution

To address the problem defined above, an interface between the C library source file and its disassembly is required. Moreover, the interface must be fully compatible with IDA as IDA stands as king in the contemporary reverse engineering community. Other disassemblers like Binary Ninja, Hopper, and Relyze simply cannot compare in terms of functionality and market share.

Therefore, because compatibility with both IDA Free and IDA Pro is required the interface must consist of an IDC script paired with a python-based program instead of the more common IDAPy Plugin. IDAPy support is a feature of IDA Pro, and while IDAPy is historically very helpful, IDA Pro can cost upwards of 10,000 dollars for a full floating license. Keeping barriers to entry as low as possible for use of the interface must continue to be a priority.

Thus, the interface will exist of two different parts. On one hand will be a python-based script which will parse C source files and create the map between unique strings and the function which uses it. On the other hand will be a IDA script which will interface with the python parser, accept the map as input, and then search for those same strings within the malicious binary. Upon finding an instance of that string, the IDA script will appropriately rename the function based on the map from the python parser. This kind of automated solution to a previously painstaking manual process will cut down on hours of wasted time.

III. Performance Metrics

The best way to determine when the project has been completed will be if the final product satisfies the following three metrics for performance, usability, and cost. Performance is the first, and arguably the most important of the three categories. The final program should be able to identify all of the unique strings within 10 OpenSSL library source files, find those same strings within the binary, and relabel those appropriately in under 20 seconds of execution time on average user hardware.

The second important performance metric is usability. The IDC script and linked python program should be compatible with all currently supported versions of both IDA Free and IDA Pro. Specifically versions 6.0 through 7.0. Continuing, user interaction with the interface should be minimal beyond specifying files to parse and setting up which binary to disassemble. The burden of work should be on the developers of this project, not on the user to specify esoteric command line arguments. Installation and use should take no longer than 30 minutes, start to finish. Lastly, the project will be open source and available for anyone to add, remove, or suggest changes to the code base.

The final performance metric to consider for this project is cost. There are other types of scripts, plugins, and software that do a similar sort of relabeling as this project will do, but all that software is either proprietary or built only for fully licensed versions of other intellectual property. This project shall be entirely free for everyone to use, regardless of whether or not its for commercial or personal reasons.